axiom™

The 30 Year Horizon

Manuel Bronstein   William Burge   Timothy Daly
James Davenport   Michael Dewar   Martin Dunstan
Albrecht Fortenbacher   Patrizia Gianni   Johannes Grabmeier
Jocelyn Guidry   Richard Jenks   Larry Lambe
Michael Monagan   Scott Morrison   William Sit
Jonathan Steinbach   Robert Sutor   Barry Trager
Stephen Watt   Jim Wen   Clifton Williamson

Volume 8: Axiom Graphics

Inclusion of names in the list of credits is based on historical information and is as accurate as possible. Inclusion of names does not in any way imply an endorsement but represents historical influence on Axiom development.

Cyril Alberga Roy Adler Richard Anderson
George Andrews Henry Baker Stephen Balzac
Yurij Baransky David R. Barton Gerald Baumgartner
Gilbert Baumslag Fred Blair Vladimir Bondarenko
Mark Botch Alexandre Bouyer Peter A. Broadbery
Martin Brock Manuel Bronstein Florian Bundschuh
William Burge Quentin Carpent Bob Caviness
Bruce Char Cheekai Chin David V. Chudnovsky
Gregory V. Chudnovsky Josh Cohen Christophe Conil
Don Coppersmith George Corliss Robert Corless
Gary Cornell Meino Cramer Claire Di Crescenzo
Timothy Daly Sr. Timothy Daly Jr. James H. Davenport
Jean Della Dora Gabriel Dos Reis Michael Dewar
Claire DiCrescendo Sam Dooley Lionel Ducos
Martin Dunstan Brian Dupee Dominique Duval
Robert Edwards Heow Eide-Goodman Lars Erickson
Richard Fateman Bertfried Fauser Stuart Feldman
Brian Ford Albrecht Fortenbacher George Frances
Constantine Frangos Timothy Freeman Korrinn Fu
Marc Gaetano Rudiger Gebauer Kathy Gerber
Patricia Gianni Holger Gollan Teresa Gomez-Diaz
Laureano Gonzalez-Vega Stephen Gortler Johannes Grabmeier
Matt Grayson James Griesmer Vladimir Grinberg
Oswald Gschnitzer Jocelyn Guidry Steve Hague
Vilya Harvey Satoshi Hamaguchi Martin Hassner
Ralf Hemmecke Henderson Antoine Hersen
Pietro Iglio Richard Jenks Kai Kaminski
Grant Keady Tony Kennedy Paul Kosinski
Klaus Kusche Bernhard Kutzler Larry Lambe
Frederic Lehobey Michel Levaud Howard Levy
Rudiger Loos Michael Lucks Richard Luczak
Camm Maguire Bob McElrath Michael McGettrick
Ian Meikle David Mentre Victor S. Miller
Gerard Milmeister Mohammed Mobarak H. Michael Moeller
Michael Monagan Marc Moreno-Maza Scott Morrison
Mark Murray William Naylor C. Andrew Neff
John Nelder Godfrey Nolan Arthur Norman
Jinzhong Niu Michael O'Connor Kostas Oikonomou
Julian A. Padget Bill Page Jaap Weel
Susan Pelzel Michel Petitot Didier Pinchon
Claude Quitte Norman Ramsey Michael Richardson
Renaud Rioboo Jean Rivlin Nicolas Robidoux
Simon Robinson Michael Rothstein Martin Rubey
Philip Santas Alfred Scheerhorn William Schelter
Gerhard Schneider Martin Schoenert Marshall Schor
Fritz Schwarz Nick Simicich William Sit
Elena Smirnova Jonathan Steinbach Christine Sundaresan
Robert Sutor Moss E. Sweedler Eugene Surowitz
James Thatcher Baldir Thomas Mike Thomas
Dylan Thurston Barry Trager Themos T. Tsikas
Gregory Vanuxem Bernhard Wall Stephen Watt
Juergen Weiss M. Weller Mark Wegman
James Wen Thorsten Werther Michael Wester
John M. Wiley Berhard Will Clifton J. Williamson
Stephen Wilson Shmuel Winograd Robert Wisbauer
Sandra Wityak Waldemar Wiwianka Knut Wolf
Clifford Yapp David Yun Richard Zippel
Evelyn Zoernack Bruno Zuercher Dan Zwillinger

Contents

New Foreword

On October 1, 2001 Axiom was withdrawn from the market and ended life as a commercial product. On September 3, 2002 Axiom was released under the Modified BSD license, including this document. On August 27, 2003 Axiom was released as free and open source software available for download from the Free Software Foundation's website, Savannah.

Work on Axiom has had the generous support of the Center for Algorithms and Interactive Scientific Computation (CAISS) at City College of New York. Special thanks go to Dr. Gilbert Baumslag for his support of the long term goal.

The online version of this documentation is roughly 1000 pages. In order to make printed versions we've broken it up into three volumes. The first volume is tutorial in nature. The second volume is for programmers. The third volume is reference material. We've also added a fourth volume for developers. All of these changes represent an experiment in print-on-demand delivery of documentation. Time will tell whether the experiment succeeded.

Axiom has been in existence for over thirty years. It is estimated to contain about three hundred man-years of research and has, as of September 3, 2003, 143 people listed in the credits. All of these people have contributed directly or indirectly to making Axiom available. Axiom is being passed to the next generation. I'm looking forward to future milestones.

With that in mind I've introduced the theme of the "30 year horizon". We must invent the tools that support the Computational Mathematician working 30 years from now. How will research be done when every bit of mathematical knowledge is online and instantly available? What happens when we scale Axiom by a factor of 100, giving us 1.1 million domains? How can we integrate theory with code? How will we integrate theorems and proofs of the mathematics with space-time complexity proofs and running code? What visualization tools are needed? How do we support the conceptual structures and semantics of mathematics in effective ways? How do we support results from the sciences? How do we teach the next generation to be effective Computational Mathematicians?

The "30 year horizon" is much nearer than it appears.

Tim Daly
CAISS, City College of New York
November 10, 2003 ((iHy))

Chapter 1

Overview

This book contains 5 programs, all related to the graphics subsystem.

The primary 2D graphics routines live in the program "view2d" (See Section 6 on page 101). The primary 3d graphics routines live in the program "view3d" (See Section 7 on page 203). These two programs can be run under the control of sman using the program "viewman" (See Section 4 on page 45). They can also be run standalone using the program "viewalone" (See Section 5 on page 81).

- viewman/viewman.c.pamphlet – int main (void)

- viewalone/viewalone.c.pamphlet – int main (int argc,char *argv[])

- view2d/main2d.c.pamphlet – int main(void)

- view3d/main3d.c.pamphlet – int main(void)

- view3d/testcol.c.pamphlet – void main(void)

The section "gdraws" (see 8 on page 465) contains a set of functions for handling postscript generation. This is handled by defining a set of cover functions for the X routines, as in GDrawArc versus XDrawArc. When the Xoption is set the X routine is called. When the PSoption is set the postscript routines are generated.

1.1 Standard Curves and Surfaces

In order to have an organized and thorough evaluation of the Axiom graphics code we turn to the CRC Standard Curves and Surfaces[8] (SCC). This volume was written years after the Axiom graphics code was written so there was no attempt to match the two until now. However, the SCC volume will give us a solid foundation to both evaluate the features of the current code and suggest future directions.

According to the SCC we can organize the various curves by the taxonomy:

1

1 **random**

    1.1 **fractal**

    1.2 **gaussian**

    1.3 **non-gaussian**

2 **determinate**

    2.1 **algebraic** – A polynomial is defined as a summation of terms composed of integral powers of x and y. An algebraic curve is one whose implicit function

$$f(x, y) = 0$$

    is a polynomial in x and y (after rationalization, if necessary). Because a curve is often defined in the explicit form

$$y = f(x)$$

    there is a need to distinguish rational and irrational functions of x.

       2.1.1 **irrational** – An irrational function of x is a quotient of two polynomials, one or both of which has a term (or terms) with power p/q, where p and q are integers.

       2.1.2 **rational** – A rational function of x is a quotient of two polynomials in x, both having only integer powers.

          2.1.2.1 **polynomial**

          2.1.2.2 **non-polynomial**

    2.2 **integral** – Certain continuous functions are not expressible in algebraic or transcendental forms but are familiar mathematical tools. These curves are equal to the integrals of algebraic or transcendental curves by definition; examples include Bessel functions, Airy integrals, Fresnel integrals, and the error function.

    2.3 **transcendental** – The transcendental curves cannot be expressed as polynomials in x and y. These are curves containing one or more of the following forms: exponential (e^x), logarithmic ($\log(x)$), or trigonometric ($\sin(x)$, $\cos(x)$).

       2.3.1 **exponential**

       2.3.2 **logarithmic**

       2.3.3 **trigonometric**

    2.4 **piecewise continuous** – Other curves, except at a few singular points, are smooth and differentiable. The class of nondifferentiable curves have discontinuity of the first derivative as a basic attribute. They are often composed of straight-line segments. Simple polygonal forms, regular fractal curves, and turtle tracks are examples.

       2.4.1 **periodic**

       2.4.2 **non-periodic**

       2.4.3 **polygonal**

          2.4.3.1 **regular**

          2.4.3.2 **irregular**

          2.4.3.3 **fractal**

1.2 CRC graphs

```
)clear all
)set mes auto off
vp:=makeViewport2D()
lineColorDefault(green())
f(c,x,n) == c*x^n
viewport1:=draw(f(1,x,1), x=-2..2, adaptive==true, unit==[1.0,1.0], title=="p27 2.1.1")
graph2111:=getGraph(viewport1,1)

lineColorDefault(blue())
viewport2:=draw(f(1,x,3), x=-2..2, adaptive==true, unit==[1.0,1.0])
graph2112:=getGraph(viewport2,1)

lineColorDefault(red())
viewport3:=draw(f(1,x,5), x=-2..2, adaptive==true, unit==[1.0,1.0])
graph2113:=getGraph(viewport3,1)

putGraph(viewport1,graph2112,2)
putGraph(viewport1,graph2113,3)
units(viewport1,1,"on")
points(viewport1,1,"off")
points(viewport1,2,"off")
points(viewport1,3,"off")

makeViewport2D(viewport1)
write(viewport1,"p27-2.1.1")
```
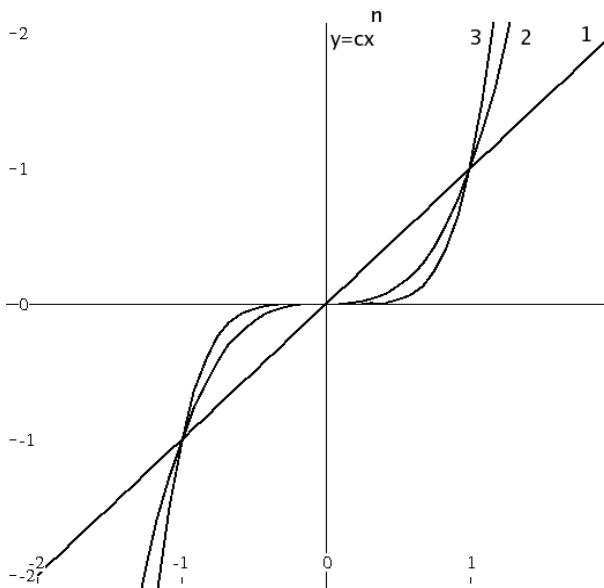
1.3 Environment Settings

1.3.1 X11 .Xdefaults

- Axiom.2D.Font
- Axiom.2D.messageFont (default: Axiom.2D.Font)
- Axiom.2D.postscriptFile (default: "axiom2d.ps")
- Axiom.2D.messageFont (default: Axiom.2D.Font)
- Axiom.2D.buttonFont (default: Axiom.2D.Font)
- Axiom.2D.headerFont (default: Axiom.2D.Font)
- Axiom.2D.titleFont (default: Axiom.2D.Font)
- Axiom.2D.graphFont (default: Axiom.2D.Font)
- Axiom.2D.unitFont (default: Axiom.2D.Font)
- Axiom.2D.monochrome (default: off)
- Axiom.2D.inverse (default: off)
- Axiom.3D.Font
- Axiom.3D.postscriptFile (default: "axiom3d.ps")
- Axiom.3D.messageFont (default: Axiom.3D.Font)
- Axiom.3D.buttonFont (default: Axiom.3D.Font)
- Axiom.3D.headerFont (default: Axiom.3D.Font)
- Axiom.3D.titleFont (default: Axiom.3D.Font)
- Axiom.3D.lightingFont (default: Axiom.3D.Font)
- Axiom.3D.volumeFont (default: Axiom.3D.Font)
- Axiom.3D.monochrome (default: off)
- Axiom.3D.inverse (default: off)

1.3.2 Shell Variables

- DISPLAY

- AXIOM

- $AXIOM/bli/view2d

- $AXIOM/bli/view3d

- HOME

- HOME/.Xdefaults

- XENVIRONMENT

- XENVIRONMENT/.Xdefaults-

- DEVE

1.4 Pre-release change history

```
This directory contains the C source code for the Scratchpad's
graphics. Comments given to the maintenance program refer to
version numbers. The versions are documented below, starting with
version23.
\begin{verbatim}
version23: (first version after returning from Europe)
   added a $DEVE environment variable - if it exists, the viewport manager
   tries those executable files first; if it was unsuccessful or, if the
   $DEVE variable was not defined, the $AXIOM variable is used tube.c:
   changed order of crossedLines(p,q... to crossedLines(q,p... in order to
   use segNum info (which segment of the second polygon the first polygon
   crosses) and using splitPolygon() rather than splitAndMove()
   tube.c: changed crossedLines() to generalPartialSplit() - an algorithms
   i hope would work. it is not taken from any literature as all the
   literature i have looked into and tried out had design flaws or lacked
   precise specifications (including Foley & van Dam volume I, Newman &
   Sprouill and others)
   viewport3D.c: xPts now malloced as a global variable in makeView3D and
   not freed up at the end. quadMesh is also not freed anymore (shouldn't
   be as it was changed to a global some time ago).
   made eyeDistance (3D stuff) into a float (from integer)
   added outlineRender (outlineOnOff in actions.h) field. note that this
   and the previous change both affected the following files:
      view3d/: main.c, process.c, viewport3D.c, spadAction.c, write.c
      viewman/: fun3d.c make3d.c
      viewalone/: spoon3D.c spoonTube.c
      spad/: view3d.spad (viewport3D, makeViewport3D and the Rep)
   3D: ability to write Stellar recognizable data files for Tube types
```

```
    spad additions: write(v,fn,listOfTypes)
    fixed perspective, added min/max range of eyeDistance

version24: view3d/tube.c: put in calculations for xmin,xmax,ymin,ymax
    >>> took them out again - doesn't the viewport manager do that?
    polygon list saved if previous render/opaque (hidden surface)
    was completed so that subsequent draws would be faster
    view3d: added NIL(pType) macro which checks for a null pointer
    >>> need same change in view2d (after testing)
    totalShades is now a global variable (to be decremented if color trouble)
    >>> need same change in view2d (after testing)
    cleaning up: added exitWithAck(info,size,i) macro to send spad
    acknowledge right before exit of a troubled program. this
    should avoid some of the causes of entering spad break loops
    not enough colors could be allocated; now, only a message is given.
    viewman: adding checks for abnormal exits of child processes (from
    people who use the "Cancel" command from the X server).
    deadBaby() and brokenPipe() in the works.
    view3d: hey! there was a bug in the projections: with perspective on,
    it turns out the Z-axis was rotating opposite of the rest of
    the system...?...
    fixed perspective
    added box (with back face removal)
    function of 2 variables now has it's own substructure, like tube, and
    shares the (now) general hidden surface subroutine used for the
    tube stuff when the perspective is turned on (when perspective is
    off, a simple back to front routine is sufficient and fast but the
    property that allows that is not preserved in the perspective projection)
    seems like (in tube.c) the overlap list property is not preserved (see
    notes) so now, routine needs to check, for polygon i, polygons
    i+1 through N, always.
    affects ALL: added deltaZ to all the stuff
    (spad, viewman, viewalone, view3d) - though not used yet
    error messages: if the .Xdefault not found (or .Xdefault wasn't defined)
    then view2d and view3d will a predefined default set of fonts. it that
    still doesn't work, a (more or less useful) error message is displayed
    and an error signal is sent to Scratchpad to prevent a freeze-up situation.
    viewpack.spad (package VIEW) and view2d.spad (domain GRIMAGE) now check for
    lists that contain illegal empty lists (i.e. point list contains nothing).
    warnings are issued for lists containing both legal point lists and empty
    point lists, errors are issued for lists containing nothing but empty lists.
    made spadcolors30.c into an archived library by the name of libColors.a,
    source file changed to spadcolors.c; makeColors' arguments have changed
    and now returns the number of colors allocated - not a pointer to the
    pixel map
    saymem, check, etc. moved to util.c for library archive
    added a view.h file, with macros to be used by all view programs
    monoColor(x) macro (in view.h) replaces spadColors[x] calls in case
    display is monochromatic (in global variable mono)
    tube.c: connecting of slices - polygon points changed to outline the
```

```
     rectangular regions without the diagonals (may be prettier when outlines
     are sketched...slightly, if no split occurs).
     clipping model: both against the hither plane as well as with a
     clipping volume
     viewport3D.c: made polygon list for functions of two variables so
     that it could call the general hidden surface algorithm (in tube.c) as
     well (turns out that back to front property of 3D explicit equations is
     not preserved when viewed with perspective)
     added volume.c, volume.h for the frustrum (perspective), projected clipping
     and clip volume interface

version25:
  view3d: added long jump to address signals that arrive while in the
          XNextEvent call. spadSignalHandler() now calls spadAction()
          if a signal is received.
  view2d: added query button and messages for each graph image
          (viewport2D.c, control.c, process.c)
  view3d: tube.c: improved speed of drawPolygon by creating overlapped
          list for unmoved polygon, and list for moved polygon that
          may be smaller than the entire list.
          see "Notes on the Depth Sort Algorithm" for strategy, etc.
          tube.c: moved the resetting of the variables relevant to recalc
          from process.c to tube.c (rotated, switchedPerspective,
          changedEyeDistance)
  GraphImage now supports 2D primitives like point(), component(), addPoint()
  ViewportDefaultsPackage now exports viewXPos() and stuff; all references
  to integers have been replaced by the more restrictive subdomains
  (e.g. PositiveInteger, NonNegativeInteger)
  ViewportPackage has dwindled to just drawCurves() and graphCurves()
  view2d, view3d: put in more robust signal handling routines so that
  signals from the viewport manager (Scratchpad) are all processed properly.
  the condition where the user is not allowed to use the control panel of the
  viewport that Scratchpad is sending commands to no longer exists!!!! wow!!!
  simultaneous processing without a race condition occuring (sorta) should
  not occur anymore.
  view3d: modification to keepDrawingViewport() so that signals also causes
  a return of no. this allows Scratchpad input files to be indistinguishable
  from interactive commands from a control panel! (that is, drawViewport()
  no longer need to complete the drawing if it was called from Scratchpad.)
  view2d: spadAction(): now only redraws viewport if the info was received for
  a graph image that is being shown
  view2d: fixed up pick/drop hangup problem. the "dodat" variable in process.c
  needed to be reset earlier, and in each separate routine (pick, drop and
  query) that required sequential button clicks (e.g.
  "Drop" + graph number "1").
  view2d: added global variable queriedGraph - so that the last queried graph
  will always be the one displayed.
  view2d.spad: default to points off
  added inverse and monochrome fields in .Xdefaults
  (e.g. view3d*monochrome : on)
```

```
   BUG FIXED: clipping of wire mesh for tubes
   view3d.spad: function of three variables for color specifications ==> changes
   in viewman, viewalone, and view3d to accept additional information.
   structure of fun2VarModel in view3d.h changes *** not backwards
   compatible with version24.
   BUG FOUND: viewport3D.c still drawing the function of 2 variables
   without perspective (painter's algorithm without processing) wrong!
   BUG FIXED: this time, for shur. flipped quadrants II and IV to remedy bug
   found in painter's algorithm in viewport3D.c.
   tube.c (view3d): changed routine that redraws quickly from the saved up list
   of processed polygons from the hidden surface algorithm so that each polygon

version 26:
  view3d: switched over to a generalized subspace points definition.
  so far, wire meshes work for existing types. code in viewport3D.c and tube.c
  are replaced by one more general in component.c; size reduced in half.
  include: modified=[view3d.h] new=[component.h]
  view3d: modified=[viewport3D.c, tube.c] new=[component.c]
  viewman: modified=[fun3d.c, make3d.c]
  representation should also handle future 3D things - like space curves,
  points and stuff without new data structures.
  NEED: take out unused code
  component.spad there temporarily to handle the new representation on the
  algebra side point.spad deals with the new representation in more
  generality on the algebra side
  NEED: interface to rest of algebra world
  view2d: draw dashed line if ticks are too close. view2d:
  modified=[viewport2D.c]
  coord.spad added for coordinate transformation
  xspadfill.c in the src/ directory for shade dithering in color - affects:
  src: modified=[spadcolors.c, spadcolors.h] new=[xspadfill.c]
  view3d: modified=[globals.h, main.c, tube.c]
  view2d: added tick labels for 2D
  view2d: modified=[viewport2D.c]
  view3d: tube.c replaced by surface.c and project.c
  viewman: for hue and shade fields in 2D, spad is one based and the viewport
  stuff is 0 based. modified=[makeGraph.c]
  --- backed up on tape ---
  replaced sprintf of XGetDefault value with direct assignment since
  XGetDefault may  return a NULL value which causes sprintf to freak out
  (xDefault is now pointer to char rather than a character array)
  view2d: modified=[globals.h, main.c]
  view3d: modified=[globals.h, main.c]
  BUG FOUND: on the PS2, redraws of hidden surface with saved data (quickList)
  bombs.
  BUG FIXED: no more bombs of quick draws
  view3d: modified=[surface.c (previously, tube.c)]
  put in SIGTERM handlers so that a kill signal to viewman would cause it
  exit cleanly (that is, kill all the child processes and then exit)
  viewman: modified=[viewman.h, viewman.c, cleanup.c]
```

```
  view3d: modified=[main.c]
  view2d: modified=[main.c]
  viewWoman: modified=[viewWoman.c]

version27:
  3D primitives: added type flag to polygon structure (for 3D primitives) -
  may or may not actually be used
  include: modified=[tube.c]
  added "#define smwattDefaultToFast" which, when defined, defaults to the
  simple back-to-front draw rather than the full depth sort processes
  (click middle button to switch)
  BUG FIXED: title reading in viewalone (to add \0 on top of the \newline
  fgets reads in)
  viewalone: modified=[spoon2D.c, spoonComp.c]
  points in 3D stored as references (indices) into a pool of points
  include: modified=[tube.h, component.h]
  view3d: modified=[main.c, project.c, surface.c, component.c]
  added (maybe last version...?) window manager override flag in override.h
  file which sets to true or false (e.g. Presentation Manager may need
  override=false)
  BUG FIXED: after the 3D stuff saves the ordering of polygons, the quick draw
  misses the last polygon...had to change doNotStopDraw flag to affect the
  subsequent polygon.
  view3d: modified=[surface.c]
  added a development header file for temporary defines
  include: added=[DEVE.h]
  part II:
  BUG FOUND: 3D color bar goes off the positive end
  BUG FIXED: the color bar error
  view3d: modified=[process.c]
  put XMapWindow after the drawViewport in make3DComponents - fixes the
  problem of having an empty viewport window come up with no well defined data
  view3d: modified=[component.c]
  view3d: initialize the numOfPolygons and polygons field right before they're
  used (as opposed to whereever i had them before)
  view3d: modified=[component.c]
```

Chapter 2

Graphics File Formats

2.1 The viewFile data file format

The viewFile is a control file for graph information. It contains settings for particular graphs. There are some general window settings that apply to the whole graph window followed by 9 graph settings, one for each possible graph shown.

2.1.1 The viewType

The viewType (A) is a switch used to decide what tool will be used to display the graph. At present there are 4 values defined (in the file **src/graph/include/action.h**) which are:

```
/* Viewport Types */
#define view3DType     1
#define viewGraphType  2
#define view2DType     3
#define viewTubeType   4
```

In the example below the integer value is '3', at (A), therefore it is of 'view2DType' meaning a 2D graph.

This value is read in **src/graph/viewalone/viewalone.c**

2.1.2 The title

The title, at (B), is read in **src/graph/viewalone/spoon2d.c**. It can be a maximum of 256 characters.

2.1.3 The window boundaries

There are 4 integers, at (C), the X, Y, Width, and Height which represent the window size in pixels.

2.1.4 The graph specifications

There are, at (D), a series of specifications for the 9 possible graphs. They are all in groups of 6 lines, one per graph. These lines are stored in a data structure called the `graphArray` or the `graphStateArray`. The lines are:

D1 is either an integer value 0 or 1 (%d format). If the value is zero the rest of the information is ignored and the graph is not displayed. If the value is 1 then the graph is displayed. This information is stored in `graphArray[i].key`.

D2 are 2 C general numbers (%g format). They represent the scale factors in X and Y. This information is stored in `graphStateArray[i].scaleX` and `graphStateArray[i].scaleY`.

D3 are 2 C general numbers (%g format). They represent the change in X and Y increments. This information is stored in `graphStateArray[i].deltaX` and `graphStateArray[i].d`

D4 are 2 C general numbers (%g format). They represent the center for the X and Y axes. This information is stored in `graphStateArray[i].centerX` and `graphStateArray[i].c`

D5 are 7 integers (%d format). They represent:

pointsOn - 0 means no points, 1 means plot points. Stored in `graphStateArray[i].points`

connectOn - 0 means isolated points, 1 means connected points. Stored in `graphStateArra`

splineOn - 0 means no spline, 1 means spline. Stored in `graphStateArray[i].splineOn`.

axesOn - 0 means no axes, 1 means draw axes. Stored in `graphStateArray[i].axesOn`.

axesColor - 0 means black and white, 1 means color. Stored in `graphStateArray[i].axesC`

unitsOn - 0 means no tick marks (units), 1 means units. Stored in `graphStateArray[i].un`

unitsColor - 0 means black and white, 1 means color. Stored in `graphStateArray[i].unit`

D6 are 2 C integers (%d format).

showing - 0 means the graph is hidden, 1 means showing. Stored in `graphStateArray[i].s`

selected - 0 means not selected, 1 means selected. Stored in `graphStateArray[i].selecte`

```
3                                          (A)
x*x                                        (B)
0 0 400 400                                (C)
1                                          (D1) (graph0)
0.867014 0.575432                          (D2)
0 0                                        (D3)
0 0                                        (D4)
```

```
1 1 1 1 61 1 68                          (D5)
1 1                                      (D6)
0                                        (graph1)
0.9 0.9
0 0
0 0
1 1 0 1 0 0 0
0 0
0                                        (graph2)
0.9 0.9
0 0
0 0
1 1 0 1 0 0 0
0 0
0                                        (graph3)
0.9 0.9
0 0
0 0
1 1 0 1 0 0 0
0 0
0                                        (graph4)
0.9 0.9
0 0
0 0
1 1 0 1 0 0 0
0 0
0                                        (graph5)
0.9 0.9
0 0
0 0
1 1 0 1 0 0 0
0 0
0                                        (graph6)
0.9 0.9
0 0
0 0
1 1 0 1 0 0 0
0 0
0                                        (graph7)
0.9 0.9
0 0
0 0
1 1 0 1 0 0 0
0 0
0                                        (graph8)
0.9 0.9
0 0
0 0
1 1 0 1 0 0 0
0 0
```

2.2   The graph file format

There are up to 9 files, named [`graph0` .. `graph8`]. There is one file per graph as indicated by the `data` file above.

2.2.1   The bounding values

There are 4 integers (%d format) at (A) giving the xmin, ymin, xmax, and ymax values for the graph. These are stored in `graphArray[i].xmin`, `graphArray[i].ymin`, `graphArray[i].xmax`, and `graphArray[i].ymax`.

There are 2 general numbers (%g format) at (B) giving the xNorm and yNorm values. These are stored in `graphArray[i].xNorm` and `graphArray[i].yNorm`.

There are 2 general numbers (%g format) at (C) giving the X origin and Y origin values. These are stored in `graphArray[i].originX` and `graphArray[i].originY`.

There are 2 general numbers (%g format) at (D) giving the X units and Y units from Axiom. These are stored in `graphArray[i].spadUnitX` and `graphArray[i].spadUnitY`.

There are 2 general numbers (%g format) at (E) giving the X units and Y units in graph coordinates. These are stored in `graphArray[i].unitX` and `graphArray[i].unitY`.

There is 1 integer (%d format) at (F) giving the number of lists that make up the graph. This is stored in `graphArray[i].numberOfLists`.

For each list, and in this case there is only 1 list. Each list is stored in a `pointListStruct` pointed to by `aList`. In this case we have:

> (G) is the number of points in the list. This is 1 integer (%d format). It is stored in `aList->numberOfPoints`.

> (H) is 3 integers (%d format) which represent the point color, the line color, and the point size. These are stored in `aList->pointColor`, `aList->lineColor`, and `aList->pointSize`.

> (I) is 4 general numbers (%g format) which represent the x, y, hue, and shade of a point. These are stored in a structure called `aPoint` which consists of `aPoint->x,aPoint->y,aPoint->h` and `aPoint->shade`. These are pointed to by the `aList` structure. There are as many copies of this data as there are points in the graph (G)

```
-3 0 3 9                             (A)
0.166667 0.111111                    (B)
1.49012e-08 -0.5                     (C)
1.2 1.8                              (D)
0.2 0.2                              (E)
1                                    (F)
49                                   (G)
91 135 3                             (H)
-0.5 0.5 0 2                         (I) repeated (G) times
```

```
-0.479167 0.418403 0 2
-0.458333 0.340278 0 2
-0.4375 0.265625 0 2
-0.416667 0.194444 0 2
-0.395833 0.126736 0 2
-0.375 0.0625 0 2
-0.354167 0.00173611 0 2
-0.333333 -0.0555556 0 2
-0.3125 -0.109375 0 2
-0.291667 -0.159722 0 2
-0.270833 -0.206597 0 2
-0.25 -0.25 0 2
-0.229167 -0.289931 0 2
-0.208333 -0.326389 0 2
-0.1875 -0.359375 0 2
-0.166667 -0.388889 0 2
-0.145833 -0.414931 0 2
-0.125 -0.4375 0 2
-0.104167 -0.456597 0 2
-0.0833333 -0.472222 0 2
-0.0625 -0.484375 0 2
-0.0416667 -0.493056 0 2
-0.0208333 -0.498264 0 2
1.49012e-08 -0.5 0 2
0.0208333 -0.498264 0 2
0.0416667 -0.493056 0 2
0.0625 -0.484375 0 2
0.0833334 -0.472222 0 2
0.104167 -0.456597 0 2
0.125 -0.4375 0 2
0.145833 -0.414931 0 2
0.166667 -0.388889 0 2
0.1875 -0.359375 0 2
0.208333 -0.326389 0 2
0.229167 -0.289931 0 2
0.25 -0.25 0 2
0.270833 -0.206597 0 2
0.291667 -0.159722 0 2
0.3125 -0.109375 0 2
0.333333 -0.0555556 0 2
0.354167 0.00173611 0 2
0.375 0.0625 0 2
0.395833 0.126736 0 2
0.416667 0.194444 0 2
0.4375 0.265625 0 2
0.458333 0.340278 0 2
0.479167 0.418403 0 2
0.5 0.5 0 2
```

2.3   The parabola

⟨*parabola.view/data*⟩≡
```
3
x*x
0 0 400 400
1
0.867014 0.575432
0 0
0 0
1 1 1 1 61 1 68
1 1
0
0.9 0.9
0 0
0 0
1 1 0 1 0 0 0
0 0
0
0.9 0.9
0 0
0 0
1 1 0 1 0 0 0
0 0
0
0.9 0.9
0 0
0 0
1 1 0 1 0 0 0
0 0
0
0.9 0.9
0 0
0 0
1 1 0 1 0 0 0
0 0
0
0.9 0.9
0 0
0 0
1 1 0 1 0 0 0
0 0
0
0.9 0.9
0 0
0 0
```

```
1 1 0 1 0 0 0
0 0
0
0.9 0.9
0 0
0 0
1 1 0 1 0 0 0
0 0
0
0.9 0.9
0 0
0 0
1 1 0 1 0 0 0
0 0
```

⟨*parabola.view/graph0*⟩≡
```
 -3 0 3 9
 0.166667 0.111111
 1.49012e-08 -0.5
 1.2 1.8
 0.2 0.2
 1
 49
 91 135 3
 -0.5 0.5 0 2
 -0.479167 0.418403 0 2
 -0.458333 0.340278 0 2
 -0.4375 0.265625 0 2
 -0.416667 0.194444 0 2
 -0.395833 0.126736 0 2
 -0.375 0.0625 0 2
 -0.354167 0.00173611 0 2
 -0.333333 -0.0555556 0 2
 -0.3125 -0.109375 0 2
 -0.291667 -0.159722 0 2
 -0.270833 -0.206597 0 2
 -0.25 -0.25 0 2
 -0.229167 -0.289931 0 2
 -0.208333 -0.326389 0 2
 -0.1875 -0.359375 0 2
 -0.166667 -0.388889 0 2
 -0.145833 -0.414931 0 2
 -0.125 -0.4375 0 2
 -0.104167 -0.456597 0 2
 -0.0833333 -0.472222 0 2
 -0.0625 -0.484375 0 2
 -0.0416667 -0.493056 0 2
 -0.0208333 -0.498264 0 2
 1.49012e-08 -0.5 0 2
 0.0208333 -0.498264 0 2
 0.0416667 -0.493056 0 2
 0.0625 -0.484375 0 2
 0.0833334 -0.472222 0 2
 0.104167 -0.456597 0 2
 0.125 -0.4375 0 2
 0.145833 -0.414931 0 2
 0.166667 -0.388889 0 2
 0.1875 -0.359375 0 2
 0.208333 -0.326389 0 2
 0.229167 -0.289931 0 2
 0.25 -0.25 0 2
```

```
0.270833 -0.206597 0 2
0.291667 -0.159722 0 2
0.3125 -0.109375 0 2
0.333333 -0.0555556 0 2
0.354167 0.00173611 0 2
0.375 0.0625 0 2
0.395833 0.126736 0 2
0.416667 0.194444 0 2
0.4375 0.265625 0 2
0.458333 0.340278 0 2
0.479167 0.418403 0 2
0.5 0.5 0 2
```

2.4   3D graph information

For the 3D graph information we have a table view of this information:

1	typeOf3D							
2	xmin	xmax	ymin	ymax	zmin	zmax		
3	title							
4	deltaX	deltaY	scale	scaleX	scaleY	scaleZ	theta	phi
5	window x	window y	window width	window height				
6	haveontrol	style	axesOn	hue Offset	number of Hues	diag- onals	outline render on	
7	light Vec[0]	light Vec[1]	light Vec[2]	trans- lucent				
8	persepective	eyeDistance						
9	numOfPoints							
10-	aPt− >x	aPt− >y	aPt− >z	aPt− >c				
10+	number of Components							
	prop closed	prop solid						
	numOfLists							
	prop closed	prop solid						
	numOfPoints							
	anIndex							

A data file, with uninteresting lines snipped is:

```
1
-1 1 -1 1 -0.998628 0.998628
-1*y^2+x^2
0 0 1.2 1 1 1 0.785398 -0.785398
0 0 400 400
0 9 1 0 27 0 0
-0.5 0.5 0.5 1
1 500
784
-1 -1 0 0.5
-0.925926 -1 -0.142661 0.428571
-0.851852 -1 -0.274348 0.362637
..<snip>..
0.851852 1 -0.274348 0.362637
0.925926 1 -0.142661 0.428571
1 1 0 0.5
1
0 0
28
0 0
```

```
28
0
1
...<snip>...
26
27
0 0
28
28
..<snip>..
54
55
0 0
28
56
..<snip>..
782
783
```

Chapter 3

include

3.1 actions.h

⟨include/actions.h⟩≡
```
#define makeAViewport -1


/* Viewport Types */
#define view3DType     1
#define viewGraphType  2
#define view2DType     3
#define viewTubeType   4

/* 2D Viewport */

#define translate2D    0
#define scale2D        1
#define pointsOnOff    2
#define connectOnOff   3
#define spline2D       4
#define reset2D        5
#define hideControl2D  6
#define closeAll2D     7
#define axesOnOff2D    8
#define unitsOnOff2D   9
#define pick2D         10
#define drop2D         11
#define clear2D        12
#define ps2D           13
#define graph1         14
```

```
#define graph2        15
#define graph3        16
#define graph4        17
#define graph5        18
#define graph6        19
#define graph7        20
#define graph8        21
#define graph9        22
#define graphSelect1  23
#define graphSelect2  24
#define graphSelect3  25
#define graphSelect4  26
#define graphSelect5  27
#define graphSelect6  28
#define graphSelect7  29
#define graphSelect8  30
#define graphSelect9  31
#define query2D       32
#define zoom2Dx       33
#define zoom2Dy       34
#define translate2Dx  35
#define translate2Dy  36

#define maxButtons2D 37

#define graphStart     14  /* the index of graph1 */
#define graphSelectStart (graphStart+maxGraphs)

/* 3D Viewport */

#define controlButtonsStart3D 0

#define rotate        0
#define zoom          1
#define translate     2
#define render        3
#define hideControl   4
#define closeAll      5
#define axesOnOff     6
#define opaqueMesh    7
#define resetView     8
#define transparent   9

#define lighting      10
#define viewVolume    11
#define region3D      12
```

```
#define outlineOnOff 13

#define zoomx        14
#define zoomy        15
#define zoomz        16
#define originr      17
#define objectr      18
#define xy           19
#define xz           20
#define yz           21
#define smooth       22
#define saveit       23
#define bwColor      24


#define maxControlButtons3D 25
#define controlButtonsEnd3D (controlButtonsStart3D + maxControlButtons3D)

#define graphStart3D  25  /* the index of g1 */
#define graphSelectStart3D (graphStart3D+maxGraphs)

/* these should be maxControlButtons3D+1.. (be sure to modify view3d.spad) */
#define diagOnOff           (maxControlButtons3D+1)
#define perspectiveOnOff    (maxControlButtons3D+2)
#define clipRegionOnOff     66
#define clipSurfaceOnOff    67

#define query        11


/* misc */

#define spadPressedAButton 100  /* used for communications with the .AXIOM file */
#define colorDef          101
#define moveViewport      102
#define resizeViewport    103
#define changeTitle       104
#define showing2D         105
#define putGraph          106 /* for 2D */
#define getGraph          107 /* for 2D */
#define lightDef          108 /* for 3D */
#define translucenceDef   109 /* for 3D */
#define writeView         110 /* for both */
#define eyeDistanceData   111 /* for 3D */
#define axesColor2D       112 /* for 2D */
#define unitsColor2D      113 /* for 2D */
#define modifyPOINT       114 /* for 3D */
```

```
#define hitherPlaneData     116 /* for 3D */
```

3.2 colors.h

This include file appears not to be used. However, the moColor macro IS used but not included.

⟨*include/colors.h*⟩≡

```
/*
   colors.h
   created on 25 November 1992, Jim Wen
   (same as the browser/src/color.h file - maybe should share?)
   */

/*  The Hues */
#define red0            0
#define red1            1
#define red2            2
#define orange0         3
#define orange1         4
#define orange2         5
#define tan0            6
#define tan1            7
#define tan2            8
#define yellow0         9
#define yellow1         10
#define yellow2         11
#define green0          12
#define green1          13
#define green2          14
#define cyan0           15
#define cyan1           16
#define cyan2           17
#define blue0           18
#define blue1           19
#define blue2           20
#define indigo0         21
#define indigo1         22
#define indigo2         23
#define violet0         24
#define violet1         25
#define violet2         26

/*
  The Shades
  */
#define dark    0
#define dim     1
```

```
#define normal  2
#define bright  3
#define pastel  3
#define light   4


/*
  The macros
  */
#define moColor(h,s) ((mono)?foregroundColor:XSolidColor(h,s))
#define moColor_BG(h,s) ((mono)?backgroundColor:XSolidColor(h,s))
```

3.3   component.h

⟨*include/component.h*⟩≡

```
/*
    This file contains the definitions for the generalized point
    structures in 3D.
    */
```

⟨*include/component.h*⟩+≡

```
⟨include/tube.h⟩


/* viewman's and viewAlone's refPt */
#define refPt(v,x) ((v).points + (x))
/* view3d's refPt - allows reference into new, dynamically generated points
    a function called traverse(n) is expected - it returns the nth point in
    the resevoir. note that x should be zero based (if numOfPoints is 10,
    then x=10 would access the first point on the resevoir list).
    */
#define refPt3D(v,x) ( (x)>(v).numOfPoints?traverse(resMax - ((x)-((v).numOfPoint
```

⟨*include/component.h*⟩+≡

```
typedef struct _componentProp {
  int closed,
     solid;
} componentProp;
```

⟨*include/component.h*⟩+≡
```
typedef struct _LPoint { /* meaning list of points */
  componentProp prop;
  int numOfPoints;
  int *indices;
} LPoint;
```

⟨*include/component.h*⟩+≡
```
typedef struct _LLPoint { /* meaning list of list of points */
  /* for the current 3D stuff:
      functions of 2 variables - closed is false (xmax does not close
      back to xmin) parametric surfaces of one variable (tubes) - closed
      is user defined (from Axiom)
      */
  componentProp prop;
  int numOfLists;
  LPoint *lp;
  int meshColor;   /* not used */
} LLPoint;
```

⟨*include/component.h*⟩+≡
```
typedef struct _LLLPoint { /* meaning list of list of list of points */
  /* for the current 3D stuff -- that is functions of 2 variables and
      parametric surfaces of one variable (tubes) -- there would be
      only one component
      */
  int numOfComponents;
  LLPoint *llp;
} LLLPoint;
```

3.4   g.h

⟨*include/g.h*⟩≡

```
  #define Xoption          0         /* Gdraw routine option */
  #define PSoption               1        /* Gdraw routine option */

  #define psError          -1        /* error return status */

  /* Black and white definitions of PS */

  #define psBlack          0.0       /* def for black in PS */
  #define psWhite          1.0       /* def for white in PS */

  /* Foreground and background definition */

  #define psForeground     psBlack /* foreground color: black */
  #define psBackground     psWhite /* background color: white */

  /* Gray scale defintions -- same as that in src/XShade.h for XShadeMax */

  #define psShadeMax       17.0     /* same as XShadeMax */
  #define psShadeMul       (1.0/(psShadeMax-1.0))  /* white and 16 gray shades */

  #define psNormalWidth    1        /* def for line width */

  /* These are all the line join styles available in PS */

  #define psMiterJoin      0
  #define psRoundJoin      1
  #define psBevelJoin      2

  /* These are all the line cap styles available in PS */

  #define psButtCap        0
  #define psRoundCap       1
  #define psPSqCap         2

  /*
   * Structures
   */

  /*
   * This is used to keep track of GC name in character and in unsigned long
   */
```

⟨*include/g.h*⟩+≡
```
typedef struct _GCstruct {
  GC      GCint;
  char            GCchar[10];
  struct _GCstruct *next;
} GCstruct, *GCptr;
```

3.5 nox10.h

⟨*include/nox10.h*⟩≡

```
/* Used in XDraw and XDrawFilled */
```

⟨*include/nox10.h*⟩+≡
```
typedef struct _Vertex {
  short x, y;
  unsigned short flags;
} Vertex;

/* The meanings of the flag bits.  If the bit is 1 the predicate is true */

#define VertexRelative          0x0001          /* else absolute */
#define VertexDontDraw          0x0002          /* else draw */
#define VertexCurved            0x0004          /* else straight */
#define VertexStartClosed       0x0008          /* else not */
#define VertexEndClosed         0x0010          /* else not */

/*
  The VertexDrawLastPoint option has not been implemented in XDraw and
  XDrawFilled so it shouldn't be defined.
*/
```

`XAssoc` - Associations used in the XAssocTable data structure. The associations are used as circular queue entries in the association table which is contains an array of circular queues (buckets).

⟨*include/nox10.h*⟩+≡

```
typedef struct _XAssoc {
  struct _XAssoc *next; /* Next object in this bucket. */
  struct _XAssoc *prev; /* Previous obejct in this bucket. */
  Display *display;     /* Display which ownes the id. */
  XID x_id;             /* X Window System id. */
  char *data;           /* Pointer to untyped memory. */
} XAssoc;
```

`XAssocTable` - X Window System id to data structure pointer association table. An XAssocTable is a hash table whose buckets are circular queues of XAssoc's. The XAssocTable is constructed from an array of XAssoc's which are the circular queue headers (bucket headers). An XAssocTable consists an XAssoc pointer that points to the first bucket in the bucket array and an integer that indicates the number of buckets in the array.

⟨*include/nox10.h*⟩+≡

```
typedef struct _XAssocTable {
  XAssoc **buckets;              /* Pointer to first bucket in bucket array.*/
  int size;                      /* Table size (number of buckets). */
} XAssocTable;
```

3.6   override.h

⟨*include/override.h*⟩≡

```
#define overrideManager False
/* override_redirect setting for overriding the window manager's directives.
   the window manager tends to stick its nose into too much - to the point
   where you can't even say where to put a new window. overriding it allows
   predictable placements of things like the control panel but also loses all
   features all the window manager (possibly things like resizing). there is
   no good solution to this, because certain window managers go as far as
   not allowing placement of windows on top of other windows while others
   do not allow windows to be resized unless they have window manager given
   title bars. */
```

3.7 rgb.h

⟨*include/rgb.h*⟩≡
```
typedef struct _RGB {
  float r,g,b;
} RGB ;
```

⟨*include/rgb.h*⟩+≡
```
typedef struct _HSV {
  float h,s,v;
} HSV ;
```

⟨*include/rgb.h*⟩+≡
```
typedef struct _HLS {
  float h,l,s;
} HLS ;
```

3.8   spadcolors.h

⟨*include/spadcolors.h*⟩≡

```
#define numOfColors 240
#define totalHuesConst   27
#define totalShadesConst 5
#define hueEnd   360
#define hueStep 12     /* hueEnd/totalHuesConst */

#define numPlanes 1
#define numColors 10
#define startColor 0
#define endColor    startColor+numColors

#define colorStep (maxColors+1)/numColors

#define yes 1
#define no 0

#define smoothConst   50
#define saymem(a,b,c) saymemWithLine(a,b,c,0)
#define Colorcells 256   /* KF number of elements in permutation vector */
#define shade 5
#define saturation   0.8

extern int        smoothHue;
extern Colormap   colorMap;
extern int        num;

#define maxColors DisplayCells(dsply,scrn)-1
```

⟨*include/spadcolors.h*⟩+≡
  ⟨*include/rgb.h*⟩

3.9   tube.h

⟨*include/tube.h*⟩≡

```
#define openTube 1
#define closedTube 0
```

⟨*include/tube.h*⟩+≡
```
typedef struct _triple {     /* used for normals */
  float x,y,z;
} triple;
```

⟨*include/tube.h*⟩+≡
  ⟨*include/rgb.h*⟩

⟨*include/tube.h*⟩+≡
```
typedef struct _viewTriple { /* used for points in 3 space */
  float x,y,z,c,sc;          /* c is color component */
  float wx,wy,wz;            /* world space coords */
  float px,py,pz;            /* as projected on the screen */
  float norm[3];
  struct _viewTriple *next;  /* for new points allocated by splits,
                                keep in list for freeing */
} viewTriple, *viewTriplePtr;

/* the xxxPRIM's are primitiveType's as deduced from the
   components received from Axiom. the info may be
   used in the hidden surface section */

#define stillDontKnow     0
#define pointComponent    1
#define lineComponent     2
#define polygonComponent 3
#define surfaceComponent 4
```

⟨*include/tube.h*⟩+≡
```
  typedef struct _poly {
    int num, sortNum,
      split;                         /* how many times the polygon's been split */
    int numpts;
    int primitiveType;
    int *indexPtr;          /* the index referring to the offset from the
                               beginning of the points field in the view3DStruct
                               in view3d.h */
    float N[3],planeConst,color;    /* planeConst - for plane equ'n, N has
                                       other 3 coeffs */
    float pxmin,pxmax,pymin,pymax,pzmin,pzmax;
    float xmin,xmax,ymin,ymax,zmin,zmax;
    int moved;                      /* moved - for depth sort */
    struct _poly *next;
    int doNotStopDraw;              /* for the quickDraw if depth info remains
                                       the same between draws */
    float normalFacingOut;
    int partialClip, totalClip,
      partialClipPz, totalClipPz;
  } poly;
```

⟨*include/tube.h*⟩+≡
```
  typedef struct _polyList {
    int  numPolys;
    poly *polyIndx;
    struct _polyList *next;
  } polyList;
```

⟨*include/tube.h*⟩+≡
```
  typedef struct _slice {
    int keyoffset;
    viewTriple *points;
    struct _slice *next;
  } slice;
```

⟨*include/tube.h*⟩+≡
```
typedef struct _tubeModel {
  /* numslices are the number of pts on the curve */
  int numslices, slicepts, numPolygons;
  int openLoop;  /* open or closed loop */
  slice *slices;
  poly  *polygons;
} tubeModel;
```

⟨*include/tube.h*⟩+≡
```
typedef struct _pointInfo {
  viewTriple *theVT;
  int        onVertex,segmentNum;
  int        indexNum;
} pointInfo;
```

3.10 view2d.h

⟨*include/view2d.h*⟩≡
```
#include <X11/Xlib.h>
#define maxGraphs 9
```

⟨*include/view2d.h*⟩+≡
```
typedef struct _viewManager {
  int viewType,          /* specifies view3d, view2d, etc... */
    PID,                 /* unique integer greater than zero */
    processID,           /* processID of child (PID could be the window ID) */
    viewIn,viewOut;      /* connection to viewport process */
  char propertyName[14]; /* string pointing to the property name */
  Window viewWindow;
  struct _viewManager *nextViewport;
} viewManager;
```

⟨*include/view2d.h*⟩+≡
```
typedef struct _viewsWithThisGraph {
  viewManager *viewGr;
  struct _viewsWithThisGraph *nextViewthing;
} viewsWithThisGraph;
```

⟨*include/view2d.h*⟩+≡
```
typedef struct _pointStruct {
  float x,y,hue,shade;
} pointStruct;
```

⟨*include/view2d.h*⟩+≡
```
typedef struct _pointListStruct {
  pointStruct            *listOfPoints;
  float                  hue, shade;
  int                    pointColor, lineColor, pointSize,
    numberOfPoints;
} pointListStruct;
```

⟨*include/view2d.h*⟩+≡
```
typedef struct _graphStruct {
  int                key;
  float              xmin,xmax,ymin,ymax;
  float              xNorm,yNorm;
  float              spadUnitX,spadUnitY;
  float              unitX,unitY;
  float              originX,originY;
  int                numberOfLists;
  pointListStruct    *listOfListsOfPoints;
  viewsWithThisGraph  *views;
  struct _graphStruct *nextGraph;
} graphStruct;
```

⟨*include/view2d.h*⟩+≡
```
typedef struct _view2DStruct {
  char      *title;
  int       vX,vY,vW,vH,
    showCP,
    axesOn,unitsOn,pointsOn,linesOn,splineOn,
    axesColor,unitsColor;
  int       graphKeyArray[maxGraphs];
} view2DStruct;
```

⟨*include/view2d.h*⟩+≡
```
  typedef struct _graphStateStruct {
    float scaleX, scaleY, deltaX, deltaY, centerX, centerY;
    int   pointsOn, connectOn, splineOn, axesOn, unitsOn,
      axesColor,unitsColor;
    int   showing, selected;  /* these fields are not passed from Spad;
                                  View2D initializes them */
  } graphStateStruct;

  /* need spline color, axes color, units color... */
```

3.11 view3d.h

⟨*include/view3d.h*⟩≡
```
  ⟨include/component.h⟩

  /* we now have two substructures (in the union, kind):
     tubeModel (in tube.h) and fun2VarModel (below)
     */
  #define maxGraphs 9
```

⟨*include/view3d.h*⟩+≡
```
  typedef struct _fun2VarModel {
    float *zArray,*cArray;
    viewTriple *pointList;
  } fun2VarModel;
```

⟨*include/view3d.h*⟩+≡
```
  union kindOf {
    /*  float *zArray; */
    fun2VarModel fun2Var;
    tubeModel tube;
  };
```

⟨*include/view3d.h*⟩+≡
```
  typedef struct _view3DStruct {
    int typeOf3D;
    float xmin,xmax,ymin,ymax,zmin,zmax;
    float cmin,cmax;
    float scaleToView;
    union kindOf kind;
    int xnumber, ynumber, zSize;
    char *title;
    float deltaX,deltaY,scale,theta,phi;
    float deltaZ;                    /***** not yet used *****/
    float scaleX,scaleY,scaleZ;
    float transX,transY,transZ;   /* translate so that rotation can be done
                                       about center of object volume */
    int vX,vY,vW,vH;
    int showCP,style,AxesOn,
      hueOff,numOfHues,
      diagonals;
    float lightVec[3],translucency;
    int scaleDown;
    int perspective;
    float eyeDistance;
    int outlineRenderOn,box,clipbox,
      clipStuff;       /* actually clip the stuff outside the clip boundaries */
    int numOfPoints;
    viewTriple *points;
    poly *polygons;
    LLLPoint lllp;
    int numPolygons;
    int pointSize;
    float distortX,distortY,distortZ;
    float clipXmin,clipXmax,      /* for object space clipping */
      clipYmin,clipYmax,
      clipZmin,clipZmax;
    float clipPlane;              /* for (frustrum hither plane) image space
                                      clipping note that there is already a
                                      clipOffset variable that is read in as a
                                      global variable
                                      */
  } view3DStruct;
```

for drawing the region box

⟨*include/view3d.h*⟩+≡

```
typedef struct _boxSideStruct {
  viewTriplePtr pointsPtr[4];   /* see notes for definition of box */
  int inside;
} boxSideStruct;
```

3.12   viewcommand.h

⟨*include/viewcommand.h*⟩≡

```
/* Commands that the viewports could send to the viewport manager */

#define viewportClosing 1
```

3.13   view.h

⟨*include/view.h*⟩≡
```
/* This file is to be included by all the viewport files */

#define check(code)      checker(code,__LINE__,"")
#define saymem(a,b,c)    saymemWithLine(a,b,c,__LINE__)
#define exitWithAck(ACK,ACKsize,i) \
                check(write(Socket,&(ACK),sizeof(ACKsize)));  exit(i);
#define NIL(type)        ((type *)NULL)

#define oldNum 8  /* in the old system, we assumed an eight shade palette */
#define oldOff 2

#define monoColor(x) ((mono)?foregroundColor:XSolidColor((int)x/oldNum,(int)(x%ol
#define monoDither(x,y) ((mono)?foregroundColor:XSolidColor(x,y))
#define notANumber (0.0/0.0)

/* error messages */
#define fontErrMess "        Try getting the font or changing the .Xdefaults entry

/* opening fonts */

/* getDef(v,s,d,x):
   v, the character pointer for the default value
   s, the .Xdefaults field
   d, the value in case the field is undefined in .Xdefaults
   x, a string specifying the prefix field (in .Xdefaults)
   */

#define getDef(v,s,d,x) {v=XGetDefault(dsply,x,s); if (v==NIL(char)) v=d;}

/* getFont(daFont,daDefault,whichView):
   assignTo, the font variable that will hold the font (globalFont)
   daFont,   the .Xdefault field name  ("buttonFont")
   daDefault, the default font (string) ("Rom12.500")
   whichView, the .Xdefault prefix name ("view2d")
   this is to be used in the files view2d/main.c and view3d/main.c where the
   appropriate variables are already defined
   */

#define getFont(assignTo,daFont,daDefault,whichView) \
getDef(xDefault,daFont,daDefault,whichView); \
if ((assignTo = XLoadQueryFont(dsply,xDefault)) == NULL) \
if ((assignTo = XLoadQueryFont(dsply,daDefault)) == NULL) { \
if (strcmp(xDefault,daDefault))   /* strcmp returns 0 if equal */ \
```

```
if (xDefault[0] == '\0') \
fprintf(stderr, \
"   >>> Font error: No .Xdefault entry for %s.%s and could not get the %s font\n%s\n", \
whichView,daFont,daDefault,fontErrMess); \
else \
fprintf(stderr, \
"   >>> Font error: Could get neither the %s nor the %s font\n%s\n",xDefault, \
daDefault,fontErrMess); \
else \
fprintf(stderr, \
"   >>> Font error: Could not get the %s font.\n%s\n",xDefault,fontErrMess); \
exitWithAck(RootWindow(dsply,scrn),Window,-1); \
}
```

3.14 write.h

⟨*include/write.h*⟩≡

```
/* These are types of files that the viewports would
   be able to write out upon a command from Axiom.
   Note that the numbers in this list is also the order
   in which they should appear in the Axiom file (e.g. view3D.spad) */

#define aPixmap 1
#define aBitmap 2
#define aPostscript 3
#define anImage 4
```

3.15   xdefs.h

⟨*include/xdefs.h*⟩≡

```
/***      default fonts      ***/
#ifdef  RTplatform
#define messageFontDefault    "Rom14.500"
#define buttonFontDefault     "vtbold"
#define headerFontDefault     "Itl14.500"
#define titleFontDefault      "Rom14.500"
#define lightingFontDefault   "6x10"
#define volumeFontDefault     "Rom8.500"
#define graphFontDefault      "fg-22"
#define unitFontDefault       "6x10"
#endif

#if defined(PS2platform) || defined(RIOSplatform) || defined(AIX370platform)
#define messageFontDefault  "Rom14"
#define buttonFontDefault   "Rom11"
#define headerFontDefault   "Itl14"
#define titleFontDefault    "Rom14"
#define lightingFontDefault "Rom10"
#define volumeFontDefault   "Rom8"
#define graphFontDefault    "Rom22"
#define unitFontDefault     "6x10"
#else
#define messageFontDefault  "9x15"
#define buttonFontDefault   "8x13"
#define headerFontDefault   "9x15"
#define titleFontDefault    "9x15"
#define lightingFontDefault "6x13"
#define volumeFontDefault   "6x10"
#define unitFontDefault     "6x10"
#define graphFontDefault    "9x15"
#endif
```

Chapter 4

viewman

4.1 viewman Call Graph

This was generated by the GNU cflow program with the argument list. Note that the line:NNNN numbers refer to the line in the code after it has been tangled from this file.

```
cflow --emacs -l -n -b -T --omit-arguments viewman.c
```

```
;; This file is generated by GNU cflow 1.3. -*- cflow -*-
    2 {   0} +-main() <int main () line:1342>
    3 {   1}   +-bsdSignal()
    4 {   1}   +-brokenPipe() <void brokenPipe () line:1337>
    5 {   2}     \-fprintf()
    6 {   1}   +-endChild() <void endChild () line:474>
    7 {   1}   +-goodbye() <void goodbye () line:557>
    8 {   2}     +-kill()
    9 {   2}     +-wait()
   10 {   2}     \-exit()
   11 {   1}   +-connect_to_local_server()
   12 {   1}   +-fprintf()
   13 {   1}   +-exit()
   14 {   1}   +-FD_ZERO()
   15 {   1}   +-FD_SET()
   16 {   1}   +-check()
   17 {   1}   +-superSelect() <int superSelect () line:1307>
   18 {   2}     +-select()
   19 {   2}     +-wait()
   20 {   2}     +-bsdSignal()
   21 {   2}     \-endChild() <void endChild () line:474> [see 6]
   22 {   1}   +-FD_ISSET()
   23 {   1}   +-readViewport() <int readViewport () line:1299>
   24 {   2}     \-read()
```

45

```
25 {   1}   +-write()
26 {   1}   +-sendGraphToView2D() <void sendGraphToView2D () line:760>
27 {   2}     +-fprintf()
28 {   2}     +-exit()
29 {   2}     +-malloc()
30 {   2}     \-write()
31 {   1}   +-closeChildViewport() <void closeChildViewport () line:551>
32 {   2}     +-rmViewMgr() <void rmViewMgr () line:478>
33 {   3}     | +-assert()
34 {   3}     | +-readViewport() <int readViewport () line:1299> [see 23]
35 {   3}     | +-free()
36 {   3}     | +-discardGraph() <void discardGraph () line:1289>
37 {   4}     | | \-free()
38 {   3}     | \-close()
39 {   2}     \-wait()
40 {   1}   +-get_int()
41 {   1}   +-forkView3D() <void forkView3D () line:978>
42 {   2}   | +-fprintf()
43 {   2}   | +-check()
44 {   2}   | +-pipe()
45 {   2}   | +-fork()
46 {   2}   | +-printf()
47 {   2}   | +-dup2()
48 {   2}   | +-close()
49 {   2}   | +-sprintf()
50 {   2}   | +-getenv()
51 {   2}   | +-execl()
52 {   2}   | +-exit()
53 {   2}   | +-malloc()
54 {   2}   | +-readViewport() <int readViewport () line:1299> [see 23]
55 {   2}   | +-makeView3DFromSpadData()
           |   <void makeView3DFromSpadData () line:1127>
56 {   3}   |   +-get_string()
57 {   3}   |   +-get_float()
58 {   3}   |   +-get_int()
59 {   3}   |   +-malloc()
60 {   3}   |   \-refPt()
61 {   2}   | +-write()
62 {   2}   | +-strlen()
63 {   2}   | +-refPt()
64 {   2}   | +-sleep()
65 {   2}   | \-send_int()
66 {   1}   +-funView3D() <void funView3D () line:836>
67 {   2}     +-get_int()
68 {   2}     +-send_int()
69 {   2}     +-write()
70 {   2}     +-get_float()
71 {   2}     +-get_string()
72 {   2}     +-strlen()
73 {   2}     \-readViewport() <int readViewport () line:1299> [see 23]
```

```
 74 {   1}    +-makeGraphFromSpadData()
                  <graphStruct *makeGraphFromSpadData () line:1230>
 75 {   2}    +-malloc()
 76 {   2}    +-fprintf()
 77 {   2}    +-exit()
 78 {   2}    +-get_float()
 79 {   2}    +-get_int()
 80 {   2}    \-send_int()
 81 {   1}  +-forkView2D() <void forkView2D () line:664>
 82 {   2}  | +-fprintf()
 83 {   2}  | +-check()
 84 {   2}  | +-pipe()
 85 {   2}  | +-fork()
 86 {   2}  | +-dup2()
 87 {   2}  | +-close()
 88 {   2}  | +-sprintf()
 89 {   2}  | +-getenv()
 90 {   2}  | +-execl()
 91 {   2}  | +-exit()
 92 {   2}  | +-malloc()
 93 {   2}  | +-readViewport() <int readViewport () line:1299> [see 23]
 94 {   2}  | +-makeView2DFromSpadData()
            |     <void makeView2DFromSpadData () line:1098>
 95 {   3}  |   +-get_string()
 96 {   3}  |   +-get_int()
 97 {   3}  |   \-get_float()
 98 {   2}  | +-write()
 99 {   2}  | +-strlen()
100 {   2}  | +-sendGraphToView2D()
            |     <void sendGraphToView2D () line:760> [see 26]
101 {   2}  | +-sleep()
102 {   2}  | \-send_int()
103 {   1}  \-funView2D() <void funView2D () line:568>
104 {   2}    +-get_int()
105 {   2}    +-send_int()
106 {   2}    +-write()
107 {   2}    +-sendGraphToView2D() <void sendGraphToView2D () line:760> [see 26]
108 {   2}    +-get_float()
109 {   2}    +-get_string()
110 {   2}    +-strlen()
111 {   2}    \-readViewport() <int readViewport () line:1299> [see 23]
```

4.2 Constants and Headers

4.2.1 defines

⟨*viewman*⟩≡
```
      /* Viewport Commands */
```

```
#define makeViewport -1
#define makeGraph    -1
#define check(code) checker(code,__LINE__,"")
#define maxConnect 40
#define intSize sizeof(int)
#define floatSize sizeof(float)
#define yes 1
#define no 0
#define writeEach
#define components
#define spadActionMode
```

4.2.2   System includes

⟨*viewman*⟩+≡

```
#include <assert.h>
#ifdef SGIplatform
#include <bstring.h>
#endif
#include <errno.h>
#if !defined(BSDplatform)
#include <malloc.h>
#endif
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/time.h>
#include <sys/wait.h>
#include <unistd.h>
```

4.2.3 Local includes

⟨*viewman*⟩+≡
  ⟨*include/actions.h*⟩
  ⟨*include/view2d.h*⟩
  ⟨*include/view3d.h*⟩
  ⟨*include/viewcommand.h*⟩

```
  #include "bsdsignal.h"
  #include "bsdsignal.h1"
  #include "com.h"
  #include "sockio-c.h1"
  #include "util.h1"
```

4.2.4 extern references

⟨*viewman*⟩+≡
```
  extern int acknow;
  extern int checkClosedChild;
  extern int currentGraph;
  extern graphStateStruct currentGraphState;
  extern int defDsply;
  extern Display *dsply;
  extern fd_set filedes;
  extern int foundBrokenPipe;
  extern int graphKey;
  extern graphStruct *graphList;
  extern int picked;
  extern char propertyBuffer[];
  extern Window  root;
  extern char *s1;
  extern viewManager *slot;
  extern Sock *spadSock;
  extern viewManager *stepSlot;
  extern int viewCommand;
  extern XEvent viewmanEvent;
  extern int viewError;
  extern int viewOkay;
  extern viewManager *viewports;
  extern int viewType;
```

4.2.5   forward references

⟨*viewman*⟩+≡
```
  int readViewport(viewManager *viewPort,void *info,int size);
  void discardGraph(graphStruct *theGraph);
  void sendGraphToView2D(int i,int there,viewManager *viewport,
                         graphStateStruct *doGraphStateArray);
  void makeView2DFromSpadData(view2DStruct *viewdata,
                               graphStateStruct graphState[]);
  void makeView3DFromSpadData(view3DStruct *viewdata,int typeOfViewport);
```

4.2.6   global variables

⟨*viewman*⟩+≡
```
  Display *dsply;
  Window  root;
  XEvent  viewmanEvent;
  viewManager *viewports,
    *slot,
    *stepSlot;
  Sock        *spadSock;
  int         viewType,
    viewCommand,
    acknow,
    graphKey = 1,
    defDsply,
    currentGraph,
    picked = no,
    viewOkay  = 0,
    viewError = -1,
    checkClosedChild = no,
    foundBrokenPipe = no;
  fd_set      filedes;
  graphStruct *graphList;
  graphStateStruct currentGraphState;
  char        *s1,
    propertyBuffer[256];/* XProperty buffer */
```

4.3 Code

4.3.1 endChild

⟨*viewman*⟩+≡

```
void endChild(int sig) {
  checkClosedChild = yes;
}
```

4.3.2   rmViewMgr

Given a pointer to a viewManager, this procedure removes it from the viewport list.

⟨*viewman*⟩+≡
```
void rmViewMgr(viewManager *slotPtr) {
  int i,throwAway,code;
  viewManager *somePort, *someOtherPort;
  graphStruct *someGraph,*someOtherGraph;   /* used in discarding graphs */
  viewsWithThisGraph *someView,*someOtherView;
  for (somePort=someOtherPort=viewports;
       (somePort != 0) && (somePort != slotPtr);
       somePort=(someOtherPort=somePort)->nextViewport)
    {}
  assert ((somePort == 0) ||
          (somePort == viewports) ||
          (somePort == someOtherPort->nextViewport));
  if (somePort) {
    if (somePort == viewports) viewports=viewports->nextViewport;
    else someOtherPort->nextViewport = somePort->nextViewport;
  }
  /*** if view2d, then clean up after the graphs as well ***/
  if (slotPtr->viewType == view2DType) {
    for (i=0; i<maxGraphs; i++) {
      /* get the graph to discard */
      code=readViewport(slotPtr,&throwAway,intSize);
      if (code == -1) break; /* read failure - give up */
      if (throwAway) {                          /* zero means no graph */

        for (someGraph = someOtherGraph = graphList;
             (someGraph != 0) && (someGraph->key != throwAway);
             someGraph=(someOtherGraph=someGraph)->nextGraph)
          {
          }
        /* someGraph is 0 if not found */
        /* someGraph == graphList if found at first */
        /* otherwise someGraph == someOtherGraph->nextGraph */
        assert( (someGraph == 0) ||
                (someGraph == graphList) ||
                (someGraph == someOtherGraph->nextGraph));
        if (someGraph) {              /* if found (should always be true) */
          for(someView=someOtherView=someGraph->views;
              (someView !=0 ) && (someView->viewGr != slotPtr);
              someView=(someOtherView=someView)->nextViewthing)
            {
            }
```

```
              /* similarly */
              assert( (someView == 0) ||
                      (someView == someGraph->views) ||
                      (someView == someOtherView->nextViewthing));

              if (someView) {      /* if found (should always be true) */
                if (someView == someGraph->views)
                  /* first */
                  someGraph->views = someGraph->views->nextViewthing;
                else
                  someOtherView->nextViewthing = someView->nextViewthing;
                free(someView);                      /* remove this viewport
                                                        from list */
              }
              /* if now nothing is pointing  to this graph */
              /* remove the graph from the list*/
              if (someGraph->views == 0) {
                if (someGraph == graphList)
                  graphList = graphList->nextGraph;
                else
                  someOtherGraph->nextGraph = someGraph->nextGraph;
                discardGraph(someGraph);           /* free the graph */
              }
            }  /* if someGraph */
          } /* if throwAway */
        } /* for i */
      } /* if type is view2D */
    close(slotPtr->viewIn);
    close(slotPtr->viewOut);
    free(slotPtr);
  } /* rmViewMgr() */
```

4.3.3   closeChildViewport

Given a pointer to a viewport structure (viewManager) this procedure first waits for the actual process to die and then removes it from the list of viewports via rmViewMgr().

⟨*viewman*⟩+≡

```
  void closeChildViewport(viewManager *slotPtr) {
    int status;
    rmViewMgr(slotPtr);
    wait(&status);
  }  /* closeChildViewport */
```

4.3.4   goodbye

Kill all children (how mean) and then kill self.

⟨*viewman*⟩+≡

```
void goodbye(int sig) {
  viewManager *v;
  v = viewports;
  while (v) {
    kill(v->PID,SIGTERM);
    while (wait(NULL) == -1);
    v = v->nextViewport;
  }
  exit(0);
}
```

4.3.5 funView2D

⟨*viewman*⟩+≡

```
void funView2D(int viewCommand) {
  int code;
  int viewPID;
  float f1,f2;
  int i1,i2,i3;
  viewManager *viewport;
  viewPID = get_int(spadSock);
  viewport = viewports;
  while ((viewport) && (viewport->PID != viewPID)) {
    viewport = viewport->nextViewport;
  }
  if (viewport) {
    send_int(spadSock,1);  /* acknowledge to spad */
    code = write(viewport->viewOut,&viewCommand,intSize);
    switch (viewCommand) {
    case putGraph:
      i1 = get_int(spadSock);  /* graph key */
      i2 = get_int(spadSock);  /* viewport slot 1..9 */
      i2--; /* 0..8*/
      code = write(viewport->viewOut,&i1,intSize);
      code = write(viewport->viewOut,&i2,intSize);
      i3 = 1; /* continue*/
      code = write(viewport->viewOut,&i3,intSize);
      sendGraphToView2D(0,i1,viewport,&currentGraphState);

      break;
    case translate2D:
      i1 = get_int(spadSock);   /* graph index */
      f1 = get_float(spadSock); /* translate in the x direction */
      f2 = get_float(spadSock); /* translate in the y direction */
      code = write(viewport->viewOut,&i1,intSize);
      code = write(viewport->viewOut,&f1,floatSize);
      code = write(viewport->viewOut,&f2,floatSize);
      break;
    case scale2D:
      i1 = get_int(spadSock);   /* graph index */
      f1 = get_float(spadSock); /* scale in the x direction */
      f2 = get_float(spadSock); /* scale in the y direction */
      code = write(viewport->viewOut,&i1,intSize);
      code = write(viewport->viewOut,&f1,floatSize);
      code = write(viewport->viewOut,&f2,floatSize);
      break;
    case hideControl2D:
```

```
    i1 = get_int(spadSock);
    code = write(viewport->viewOut,&i1,intSize);
    break;
case axesOnOff2D:
case unitsOnOff2D:
case connectOnOff:
case pointsOnOff:
case spline2D:
case showing2D:
  i1 = get_int(spadSock);   /* graph index */
  i2 = get_int(spadSock);    /* axes status */
  code = write(viewport->viewOut,&i1,intSize);
  code = write(viewport->viewOut,&i2,intSize);
  break;
case moveViewport:
case resizeViewport:
  i1 = get_int(spadSock);
  i2 = get_int(spadSock);
  code = write(viewport->viewOut,&i1,intSize);
  code = write(viewport->viewOut,&i2,intSize);
  break;
case changeTitle:
  s1 = get_string(spadSock);
  i1 = strlen(s1);
  code = write(viewport->viewOut,&i1,intSize);
  code = write(viewport->viewOut,s1,i1);
  break;
case writeView:
  s1 = get_string(spadSock);
  i1 = strlen(s1);
  code = write(viewport->viewOut,&i1,intSize);
  code = write(viewport->viewOut,s1,i1);
    /* write out the types of things to be written */
  i2 = get_int(spadSock);
  code = write(viewport->viewOut,&i2,intSize);
  while (i2) {
    i2 = get_int(spadSock);
    code = write(viewport->viewOut,&i2,intSize);
  }
  break;
case spadPressedAButton:
  i1 = get_int(spadSock);
  code = write(viewport->viewOut,&i1,intSize);
  break;
}  /* switch */
    /*** get acknowledge from viewport */
```

```
      code = readViewport(viewport,&acknow,intSize);
      send_int(spadSock,1);  /* acknowledge to spad */
  } else {
      send_int(spadSock,-1);  /* send error value in acknowledge to spad */
  }
}
```

4.3.6   forkView2D

⟨*viewman*⟩+≡

```
  void forkView2D(void) {
    viewManager      *viewport;
    int              childPID, code;
    int              i;
    view2DStruct      doView2D;
    graphStateStruct doGraphStateArray[maxGraphs];
    int                there;
    int  pipe0[2], pipe1[2];
    char envAXIOM[100],runView[100];
#ifdef DEBUG
    fprintf(stderr,"fun2d:Pipe calls for 2D\n");
#endif
    check(pipe(pipe0));
    check(pipe(pipe1));
#ifdef DEBUG
    fprintf(stderr,"Fork routine for 2D\n");
#endif
    childPID = check(fork());
    switch(childPID) {
    case -1:
      fprintf(stderr,
              "The viewport manager cannot open a viewport window.\nTry closing som
      return;
    case 0:
      /*****************************
       *       child process       *
       *****************************/
         /* map pipes from viewport manager to standard input and output */
#ifdef DEBUG
      fprintf(stderr,"Mapping pipes to standard I/O in 2D\n");
#endif
      check(dup2(pipe0[0],0));
      check(dup2(pipe1[1],1));
      close(pipe0[0]);
      close(pipe0[1]);
      close(pipe1[0]);
      close(pipe1[1]);
#ifdef DEBUG
      fprintf(stderr,"Executing TwoDimensionalViewport process\n");
#endif
      sprintf(envAXIOM,"%s",getenv("AXIOM"));
      sprintf(runView,"%s%s",envAXIOM,"/lib/view2d");
      check(execl(runView,runView,NULL));
```

```
   fprintf(stderr,"The viewport manager could not execute view2d.\nCheck that view2d is on
   exit(-1);
 default:
   /*****************************
    *        parent process        *
    *****************************/
   if (!(viewport = (viewManager *)malloc(sizeof(viewManager)))) {
     fprintf(stderr,"The viewport manager ran out of memory trying to create a new viewport
     return;
   }
   viewport->viewType = view2DType;
   viewport->PID = childPID;
       /* set up pipes to child process */
   close(pipe0[0]);
   close(pipe1[1]);
   viewport->viewIn  = pipe1[0];
   viewport->viewOut = pipe0[1];
       /* add new viewport to global list */
   viewport->nextViewport = viewports;
   viewports = viewport;
   if (viewport->viewIn <0) {
     fprintf(stderr,
 "viewman could not create connection to a 2D viewport window. Try again.\n");
     return;
   } else {
     code = readViewport(viewport,&acknow,intSize);
     if (code < 0) {
       fprintf(stderr,
        "viewman could not read from a 2D viewport window\ncode=%d\nack=%d\n",
        code,acknow);
       return;
     }
   }
   makeView2DFromSpadData(&doView2D,doGraphStateArray);
     /* tell the child that mother is a viewport manager */
   i = no;
   write(viewport->viewOut,&i,sizeof(int));
   write(viewport->viewOut,&doView2D,sizeof(view2DStruct));
   i = strlen(doView2D.title)+1;
   write(viewport->viewOut,&i,intSize); /* send length of the title child */
   write(viewport->viewOut,doView2D.title,i);  /* send title to the child */
   for (i=0; i<maxGraphs; i++) {
     there = doView2D.graphKeyArray[i];
     write(viewport->viewOut,&there,intSize);
     sendGraphToView2D(i,there,viewport,doGraphStateArray);
   }; /* for i in graphs */
```

```
        /*** get acknowledge from viewport */
    code = readViewport(viewport,&(viewport->viewWindow),sizeof(Window));
    sleep(1);  /* wait a second...*/
    send_int(spadSock,viewport->PID);  /* acknowledge to spad */
  }   /* switch */
}    /* forkView2D() */
```

4.3.7 sendGraphToView2D

⟨*viewman*⟩+≡

```
  void sendGraphToView2D(int i,int there,viewManager *viewport,
                         graphStateStruct *doGraphStateArray) {
    graphStruct      *gPtr;
    pointListStruct  *llPtr;
    pointStruct      *p;
    viewsWithThisGraph *oneView;
    int j,k;
    if (there) {
      gPtr = graphList;
       /** find the right graph (same key) in graph list **/
      while ( gPtr != NULL  &&   gPtr->key != there)
        gPtr = gPtr->nextGraph;
      if ((gPtr==NULL) ||(gPtr->key != there) ){
        fprintf(stderr,"The viewport manager cannot find the requested graph\n");
        fprintf(stderr,"and will quit and restart.\n");
        exit(-1);
      }
    /* Before sending off the data, insert a pointer to viewport from graph */
    if (!(oneView = (viewsWithThisGraph *)malloc(sizeof(viewsWithThisGraph)))) {
      fprintf(stderr,"The viewport manager ran out of memory trying to \n");
      fprintf(stderr,"create a new graph (viewsWithThisGraph).\n");
      return;
    }
    oneView->viewGr          = viewport;
    oneView->nextViewthing   = gPtr->views;
    gPtr->views              = oneView;

  #ifdef writeEach
      write(viewport->viewOut,&(gPtr->xmin),floatSize);
      write(viewport->viewOut,&(gPtr->xmax),floatSize);
      write(viewport->viewOut,&(gPtr->ymin),floatSize);
      write(viewport->viewOut,&(gPtr->ymax),floatSize);
      write(viewport->viewOut,&(gPtr->xNorm),floatSize);
      write(viewport->viewOut,&(gPtr->yNorm),floatSize);
      write(viewport->viewOut,&(gPtr->spadUnitX),floatSize);
      write(viewport->viewOut,&(gPtr->spadUnitY),floatSize);
      write(viewport->viewOut,&(gPtr->unitX),floatSize);
      write(viewport->viewOut,&(gPtr->unitY),floatSize);
      write(viewport->viewOut,&(gPtr->originX),floatSize);
      write(viewport->viewOut,&(gPtr->originY),floatSize);
      write(viewport->viewOut,&(gPtr->numberOfLists),intSize);
  #else
      write(viewport->viewOut,gPtr,sizeof(graphStruct));
```

```
#endif
    llPtr = gPtr->listOfListsOfPoints;
    for (j=0; j<(gPtr->numberOfLists); j++) {
      write(viewport->viewOut,&(llPtr->numberOfPoints),intSize);
      p = llPtr->listOfPoints;
      for (k=0; k<(llPtr->numberOfPoints); k++) {
        write(viewport->viewOut,&(p->x),floatSize);
        write(viewport->viewOut,&(p->y),floatSize);
        write(viewport->viewOut,&(p->hue),floatSize);
        write(viewport->viewOut,&(p->shade),floatSize);
        p++;
      }    /* for k in list of points */
      write(viewport->viewOut,&(llPtr->pointColor),intSize);
      write(viewport->viewOut,&(llPtr->lineColor),intSize);
      write(viewport->viewOut,&(llPtr->pointSize),intSize);
      llPtr++;
    }   /* for j in list of lists of points */
    /* a graph state is defined for a graph if graph is there */
    write(viewport->viewOut,&(doGraphStateArray[i].scaleX),floatSize);
    write(viewport->viewOut,&(doGraphStateArray[i].scaleY),floatSize);
    write(viewport->viewOut,&(doGraphStateArray[i].deltaX),floatSize);
    write(viewport->viewOut,&(doGraphStateArray[i].deltaY),floatSize);
    write(viewport->viewOut,&(doGraphStateArray[i].pointsOn),intSize);
    write(viewport->viewOut,&(doGraphStateArray[i].connectOn),intSize);
    write(viewport->viewOut,&(doGraphStateArray[i].splineOn),intSize);
    write(viewport->viewOut,&(doGraphStateArray[i].axesOn),intSize);
    write(viewport->viewOut,&(doGraphStateArray[i].axesColor),intSize);
    write(viewport->viewOut,&(doGraphStateArray[i].unitsOn),intSize);
    write(viewport->viewOut,&(doGraphStateArray[i].unitsColor),intSize);
    write(viewport->viewOut,&(doGraphStateArray[i].showing),intSize);
  }   /* if graph is there */
}
```

4.3.8 funView3D

⟨*viewman*⟩+≡

```
void funView3D(int viewCommand) {
  int code;
  int viewPID;
  float f1,f2,f3,f4;
  int i1,i2;
  viewManager *viewport;
  viewPID = get_int(spadSock);
  viewport = viewports;
  while ((viewport) && (viewport->PID != viewPID))
    viewport = viewport->nextViewport;
  if (viewport) {
    send_int(spadSock,1);  /* acknowledge to spad */
    viewmanEvent.xclient.window = viewport->viewWindow;
    code = write(viewport->viewOut,&viewCommand,intSize);
    switch (viewCommand) {
    case rotate:
      f1 = get_float(spadSock);
      f2 = get_float(spadSock);
      code = write(viewport->viewOut,&f1,floatSize);
      code = write(viewport->viewOut,&f2,floatSize);
      break;
    case zoom:
      f1 = get_float(spadSock);
      code = write(viewport->viewOut,&f1,floatSize);
      break;
    case zoomx:
      f1 = get_float(spadSock);
      f2 = get_float(spadSock);
      f3 = get_float(spadSock);
      code = write(viewport->viewOut,&f1,floatSize);
      code = write(viewport->viewOut,&f2,floatSize);
      code = write(viewport->viewOut,&f3,floatSize);
      break;
    case translate:
      f1 = get_float(spadSock);
      f2 = get_float(spadSock);
      code = write(viewport->viewOut,&f1,floatSize);
      code = write(viewport->viewOut,&f2,floatSize);
      break;
    case modifyPOINT:
      i1 = get_int(spadSock);
      f1 = get_float(spadSock);
      f2 = get_float(spadSock);
```

```
          f3 = get_float(spadSock);
          f4 = get_float(spadSock);
          code = write(viewport->viewOut,&i1,intSize);
          code = write(viewport->viewOut,&f1,floatSize);
          code = write(viewport->viewOut,&f2,floatSize);
          code = write(viewport->viewOut,&f3,floatSize);
          code = write(viewport->viewOut,&f4,floatSize);
          break;
        case hideControl:
          i1 = get_int(spadSock);
          code = write(viewport->viewOut,&i1,intSize);
          break;
        case axesOnOff:
        case perspectiveOnOff:
        case region3D:
        case clipRegionOnOff:
        case clipSurfaceOnOff:
          i1 = get_int(spadSock);
          code = write(viewport->viewOut,&i1,intSize);
          break;
        case eyeDistanceData:
        case hitherPlaneData:
          f1 = get_float(spadSock);
          code = write(viewport->viewOut,&f1,floatSize);
          break;
        case colorDef:
          i1 = get_int(spadSock);
          i2 = get_int(spadSock);
          code = write(viewport->viewOut,&i1,intSize);
          code = write(viewport->viewOut,&i2,intSize);
          break;
        case moveViewport:
          i1 = get_int(spadSock);
          i2 = get_int(spadSock);
          code = write(viewport->viewOut,&i1,intSize);
          code = write(viewport->viewOut,&i2,intSize);
          break;
        case resizeViewport:
          i1 = get_int(spadSock);
          i2 = get_int(spadSock);
          code = write(viewport->viewOut,&i1,intSize);
          code = write(viewport->viewOut,&i2,intSize);
          break;
        case transparent:
        case opaqueMesh:
        case render:
```

```
      break;
    case lightDef:
      f1 = get_float(spadSock);
      f2 = get_float(spadSock);
      f3 = get_float(spadSock);
      code = write(viewport->viewOut,&f1,floatSize);
      code = write(viewport->viewOut,&f2,floatSize);
      code = write(viewport->viewOut,&f3,floatSize);
      break;
    case translucenceDef:
      f1 = get_float(spadSock);
      code = write(viewport->viewOut,&f1,floatSize);
      break;
    case changeTitle:
      s1 = get_string(spadSock);
      i1 = strlen(s1);
      code = write(viewport->viewOut,&i1,intSize);
      code = write(viewport->viewOut,s1,i1);
      break;
    case writeView:
      s1 = get_string(spadSock);
      i1 = strlen(s1);
      code = write(viewport->viewOut,&i1,intSize);
      code = write(viewport->viewOut,s1,i1);
      /* write out the types of things to be written */
      i2 = get_int(spadSock);
      code = write(viewport->viewOut,&i2,intSize);
      while (i2) {
        i2 = get_int(spadSock);
        code = write(viewport->viewOut,&i2,intSize);
      }
      break;
    case diagOnOff:
      i1 = get_int(spadSock);
      code = write(viewport->viewOut,&i1,intSize);
      break;
    case outlineOnOff:
      i1 = get_int(spadSock);
      code = write(viewport->viewOut,&i1,intSize);
      break;
    case spadPressedAButton:
      i1 = get_int(spadSock);
      code = write(viewport->viewOut,&i1,intSize);
      break;
    }  /* switch */
    /*** get acknowledge from viewport */
```

```
      code = readViewport(viewport,&acknow,intSize);
      send_int(spadSock,1);  /* acknowledge to spad */
  } else {  /* if (viewport) */
      send_int(spadSock,-1);  /* send error value in acknowledge to spad */
  }
}
```

4.3.9 forkView3D

⟨*viewman*⟩+≡

```
void forkView3D(int typeOfViewport) {
  viewManager *viewport;
  int         childPID, code;
  int         i;
  view3DStruct doView3D;
  int  pipe0[2],pipe1[2];
  int *anIndex;
  char envAXIOM[100],runView[100];
  int j,k;
  LLPoint *anLLPoint;
  LPoint *anLPoint;
#ifdef DEBUG
  fprintf(stderr,"Pipe calls for 3D\n");
#endif
  check(pipe(pipe0));
  check(pipe(pipe1));
#ifdef DEBUG
  fprintf(stderr,"Fork routine for 3D\n");
#endif
  switch(childPID = check(fork())) {
  case -1:
    printf("Cannot create a new process - \n");
    printf("you probably have too many things running already.\n");
    return;
  case 0:
    /*****************************
     *       child process       *
     *****************************/
        /* map pipes from viewport manager to standard input and output */
#ifdef DEBUG
    fprintf(stderr,"Mapping pipes to standard I/O in 3D\n");
#endif
    check(dup2(pipe0[0],0));
    check(dup2(pipe1[1],1));
    close(pipe0[0]);
    close(pipe0[1]);
    close(pipe1[0]);
    close(pipe1[1]);
#ifdef DEBUG
    fprintf(stderr,"Executing ThreeDimensionalViewport process\n");
#endif
    sprintf(envAXIOM,"%s",getenv("AXIOM"));
    sprintf(runView,"%s%s",envAXIOM,"/lib/view3d");
```

```
    check(execl(runView,runView,NULL));
    fprintf(stderr,"The viewport manager could not execute view3d.\nCheck that vi
    exit(-1);
default:
  /*****************************
   *        parent process       *
   *****************************/
  if (!(viewport = (viewManager *)malloc(sizeof(viewManager)))) {
    printf("Ran out of memory trying to create a new viewport process.\n");
     return;
  }
  viewport->viewType = typeOfViewport;
  viewport->PID = childPID;
        /* set up pipes to child process */
  close(pipe0[0]);
  close(pipe1[1]);
  viewport->viewIn  = pipe1[0];
  viewport->viewOut = pipe0[1];
        /* add new viewport to global list */
  viewport->nextViewport = viewports;
  viewports = viewport;
  if (viewport->viewIn <0) {
    fprintf(stderr,"The viewport manager could not create connection to\n");
    fprintf(stderr," a 3D viewport window. Try again.\n");
    return;
  } else {
    code = readViewport(viewport,&acknow,intSize);
    if (code < 0) {
      fprintf(stderr,"The viewport manager could not read from a 3D \n");
      fprintf(stderr,"viewport window\ncode=%d\nack=%d\n",code,acknow);
      return;
    }
  }
  makeView3DFromSpadData(&doView3D,typeOfViewport);
    /* tell the child that parent is a viewport manager */
  i = no;
  write(viewport->viewOut,&i,sizeof(int));
  write(viewport->viewOut,&doView3D,sizeof(view3DStruct));
  i = strlen(doView3D.title)+1;
  write(viewport->viewOut,&i,intSize); /*tell the length of title to child */
  write(viewport->viewOut,doView3D.title,i);  /* tell the title to child */
  write(viewport->viewOut,&(doView3D.lightVec[0]),floatSize);
  write(viewport->viewOut,&(doView3D.lightVec[1]),floatSize);
  write(viewport->viewOut,&(doView3D.lightVec[2]),floatSize);
  /* send generalized 3D components */
  write(viewport->viewOut,&(doView3D.numOfPoints),intSize);
```

```
    for (i=0; i<doView3D.numOfPoints; i++) {
      write(viewport->viewOut,&(refPt(doView3D,i)->x),floatSize);
      write(viewport->viewOut,&(refPt(doView3D,i)->y),floatSize);
      write(viewport->viewOut,&(refPt(doView3D,i)->z),floatSize);
      write(viewport->viewOut,&(refPt(doView3D,i)->c),floatSize);
    }
    write(viewport->viewOut,&(doView3D.lllp.numOfComponents),intSize);
    anLLPoint = doView3D.lllp.llp;
    for (i=0; i<doView3D.lllp.numOfComponents; i++,anLLPoint++) {
      write(viewport->viewOut,&(anLLPoint->prop.closed),intSize);
      write(viewport->viewOut,&(anLLPoint->prop.solid),intSize);
      write(viewport->viewOut,&(anLLPoint->numOfLists),intSize);
      anLPoint = anLLPoint->lp;
      for (j=0; j<anLLPoint->numOfLists; j++,anLPoint++) {
        write(viewport->viewOut,&(anLPoint->prop.closed),intSize);
        write(viewport->viewOut,&(anLPoint->prop.solid),intSize);
        write(viewport->viewOut,&(anLPoint->numOfPoints),intSize);
        anIndex = anLPoint->indices;
        for (k=0; k<anLPoint->numOfPoints; k++,anIndex++)
          write(viewport->viewOut,anIndex,intSize);
      } /* for LPoints in LLPoints (j) */
    } /* for LLPoints in LLLPoints (i) */
        /*** get acknowledge from viewport */
    code = readViewport(viewport,&(viewport->viewWindow),sizeof(Window));
    sleep(1);  /* wait a second...*/
    send_int(spadSock,viewport->PID);  /* acknowledge to spad */

  }   /* switch */

}   /* forkView3D() */
```

4.3.10   makeView2DFromSpadData

⟨*viewman*⟩+≡
```
  void makeView2DFromSpadData(view2DStruct *viewdata,
                             graphStateStruct graphState[])
 { int i;
   viewdata->title = get_string(spadSock);
   viewdata->vX = get_int(spadSock);
   viewdata->vY = get_int(spadSock);
   viewdata->vW = get_int(spadSock);
   viewdata->vH = get_int(spadSock);
   viewdata->showCP = get_int(spadSock);
   for (i=0; i<maxGraphs; i++) {
     viewdata->graphKeyArray[i] = get_int(spadSock);
     if (viewdata->graphKeyArray[i]) {
       graphState[i].scaleX     = get_float(spadSock);
       graphState[i].scaleY     = get_float(spadSock);
       graphState[i].deltaX     = get_float(spadSock);
       graphState[i].deltaY     = get_float(spadSock);
       graphState[i].pointsOn   = get_int(spadSock);
       graphState[i].connectOn  = get_int(spadSock);
       graphState[i].splineOn   = get_int(spadSock);
       graphState[i].axesOn      = get_int(spadSock);
       graphState[i].axesColor  = get_int(spadSock);
       graphState[i].unitsOn    = get_int(spadSock);
       graphState[i].unitsColor = get_int(spadSock);
       graphState[i].showing    = get_int(spadSock);
       graphState[i].selected   = 1;        /* always default to selected? */
     }
   }
 }
```

4.3.11   makeView3DFromSpadData

⟨*viewman*⟩+≡

```
void makeView3DFromSpadData(view3DStruct *viewdata,int typeOfViewport) {
  int i,j,k;
  LLPoint *anLLPoint;
  LPoint *anLPoint;
  int *anIndex;
  int firstPieceOfData = yes;
  int constantColor;
  double cMin = 0;
  double cMax = 0;
  double cNorm = 0;
  viewdata->typeOf3D = typeOfViewport;
  viewdata->title = get_string(spadSock);
  viewdata->deltaX = get_float(spadSock);
  viewdata->deltaY = get_float(spadSock);
  viewdata->scale  = get_float(spadSock);
  viewdata->scaleX = get_float(spadSock);
  viewdata->scaleY = get_float(spadSock);
  viewdata->scaleZ = get_float(spadSock);
  viewdata->theta  = get_float(spadSock);
  viewdata->phi    = get_float(spadSock);
  viewdata->vX = get_int(spadSock);
  viewdata->vY = get_int(spadSock);
  viewdata->vW = get_int(spadSock);
  viewdata->vH = get_int(spadSock);
  viewdata->showCP    = get_int(spadSock);
  viewdata->style     = get_int(spadSock);
  viewdata->AxesOn    = get_int(spadSock);
  viewdata->diagonals = get_int(spadSock);
  viewdata->outlineRenderOn = get_int(spadSock);
  viewdata->box = get_int(spadSock);
  viewdata->clipbox = get_int(spadSock);
  viewdata->clipStuff = get_int(spadSock);
  viewdata->hueOff    = get_int(spadSock);
  viewdata->numOfHues = get_int(spadSock);
  viewdata->lightVec[0]  = get_float(spadSock);
  viewdata->lightVec[1]  = get_float(spadSock);
  viewdata->lightVec[2]  = get_float(spadSock);
  viewdata->translucency = get_float(spadSock);
  viewdata->perspective = get_int(spadSock);
  viewdata->eyeDistance = get_float(spadSock);
  viewdata->numOfPoints = get_int(spadSock);
  viewdata->points =
      (viewTriple *)malloc(viewdata->numOfPoints * sizeof(viewTriple));
```

```
for (i=0; i<viewdata->numOfPoints; i++) {
  refPt(*viewdata,i)->x = get_float(spadSock);
  refPt(*viewdata,i)->y = get_float(spadSock);
  refPt(*viewdata,i)->z = get_float(spadSock);
  refPt(*viewdata,i)->c = get_float(spadSock);
        /* set min/max values */
  if (firstPieceOfData) {
    firstPieceOfData = no;
    viewdata->xmin = viewdata->xmax = refPt(*viewdata,i)->x;
    viewdata->ymin = viewdata->ymax = refPt(*viewdata,i)->y;
    viewdata->zmin = viewdata->zmax = refPt(*viewdata,i)->z;
    cMin = cMax = refPt(*viewdata,i)->c;
  } else {
    if (refPt(*viewdata,i)->x < viewdata->xmin)
      viewdata->xmin = refPt(*viewdata,i)->x;
    else if (refPt(*viewdata,i)->x > viewdata->xmax)
      viewdata->xmax = refPt(*viewdata,i)->x;
    if (refPt(*viewdata,i)->y < viewdata->ymin)
      viewdata->ymin = refPt(*viewdata,i)->y;
    else if (refPt(*viewdata,i)->y > viewdata->ymax)
      viewdata->ymax = refPt(*viewdata,i)->y;
    if (refPt(*viewdata,i)->z < viewdata->zmin)
      viewdata->zmin = refPt(*viewdata,i)->z;
    else if (refPt(*viewdata,i)->z > viewdata->zmax)
      viewdata->zmax = refPt(*viewdata,i)->z;
    if (refPt(*viewdata,i)->c < cMin) cMin = refPt(*viewdata,i)->c;
    else if (refPt(*viewdata,i)->c > cMax) cMax = refPt(*viewdata,i)->c;
  } /* if (firstPieceOfData) else */
} /* for i (point data) */
viewdata->lllp.numOfComponents = get_int(spadSock);
anLLPoint = viewdata->lllp.llp =
  (LLPoint *)malloc(viewdata->lllp.numOfComponents*sizeof(LLPoint));
for (i=0; i<viewdata->lllp.numOfComponents; i++,anLLPoint++) {
  anLLPoint->prop.closed = get_int(spadSock);
  anLLPoint->prop.solid = get_int(spadSock);
  anLLPoint->numOfLists = get_int(spadSock);
  anLPoint = anLLPoint->lp =
    (LPoint *)malloc(anLLPoint->numOfLists*sizeof(LPoint));
  for (j=0; j<anLLPoint->numOfLists; j++,anLPoint++) {
    anLPoint->prop.closed = get_int(spadSock);
    anLPoint->prop.solid = get_int(spadSock);
    anLPoint->numOfPoints = get_int(spadSock);
    anIndex = anLPoint->indices =
      (int *)malloc(anLPoint->numOfPoints*sizeof(int));
    for (k=0; k<anLPoint->numOfPoints; k++,anIndex++)
      *anIndex = get_int(spadSock);
```

```
      } /* for LPoints in LLPoints (j) */
    } /* for LLPoints in LLLPoints (i) */
        /* now normalize the colors */
    cNorm = cMax - cMin;
          /*** new fields - cmin, cmax ***/
    viewdata->cmin = cMin;
    viewdata->cmax = cMax;
    constantColor = (cNorm < 0.0001);
    for (i=0; i<viewdata->numOfPoints; i++)
      if (constantColor) refPt(*viewdata,i)->c = 0.5;
      else refPt(*viewdata,i)->c = (refPt(*viewdata,i)->c - cMin)/cNorm;
    viewdata->scaleDown = yes;
}
```

4.3.12   makeGraphFromSpadData

⟨*viewman*⟩+≡

```
graphStruct *makeGraphFromSpadData(void) {
  graphStruct     *graphData;
  pointListStruct *pL;
  pointStruct     *p;
  int i,j;
  if (!(graphData = (graphStruct *)malloc(sizeof(graphStruct)))) {
    fprintf(stderr,"The viewport manager ran out of memory trying to \n");
    fprintf(stderr, "create a new graph (graphStruct).\n");
    exit(-1);
  }
  graphData->xmin = get_float(spadSock);    /* after everything is normalized */
  graphData->xmax = get_float(spadSock);
  graphData->ymin = get_float(spadSock);    /* view2d */
  graphData->ymax = get_float(spadSock);
  graphData->xNorm = 1/(graphData->xmax - graphData->xmin);
  graphData->yNorm = 1/(graphData->ymax - graphData->ymin);
  graphData->spadUnitX = get_float(spadSock);
  graphData->spadUnitY = get_float(spadSock);
  graphData->unitX = graphData->spadUnitX * graphData->xNorm;
  graphData->unitY = graphData->spadUnitY * graphData->yNorm;
  graphData->originX = -graphData->xmin * graphData->xNorm - 0.5;
  graphData->originY = -graphData->ymin * graphData->yNorm - 0.5;
  graphData->numberOfLists = get_int(spadSock);
  if (!(pL = (pointListStruct *)
          malloc(graphData->numberOfLists * sizeof(pointListStruct)))) {
    fprintf(stderr,"The viewport manager ran out of memory trying to \n");
    fprintf(stderr,"create a new graph (pointListStruct).\n");
    exit(-1);
  }
  graphData->listOfListsOfPoints = pL;
  for (i=0; i<graphData->numberOfLists; i++) {
    pL->numberOfPoints = get_int(spadSock);
    if (!(p=(pointStruct *)malloc(pL->numberOfPoints*sizeof(pointStruct)))) {
      fprintf(stderr,"The viewport manager ran out of memory trying to \n");
      fprintf(stderr,"create a new graph (pointStruct).\n");
      exit(-1);
    }
    pL->listOfPoints = p;                 /** point to current point list **/
    for (j=0; j<pL->numberOfPoints; j++) {
      p->x     = get_float(spadSock);          /* get numbers from Axiom */
      p->y     = get_float(spadSock);
      p->hue   = get_float(spadSock) - 1;    /* make zero based */
      p->shade = get_float(spadSock) - 1;
```

```
        /* normalize to range [-0.5..0.5] */
      p->x = (p->x - graphData->xmin) * graphData->xNorm - 0.5;
      p->y = (p->y - graphData->ymin) * graphData->yNorm - 0.5;
      p++;
    }
        /* for now, getting hue, shade - do hue * totalHues + shade */
    pL->pointColor = get_int(spadSock);
    pL->lineColor = get_int(spadSock);
    pL->pointSize = get_int(spadSock);
    pL++;                            /** advance to next point list **/
  }
  graphData->key = graphKey++;
  send_int(spadSock,(graphKey-1));          /* acknowledge to spad */
  return(graphData);
}
```

4.3.13   discardGraph

⟨*viewman*⟩+≡

```
  void discardGraph(graphStruct *theGraph) {
    pointListStruct *pL;
    int j;
    for (j=0, pL=theGraph->listOfListsOfPoints;
           j<theGraph->numberOfLists; j++,pL++)
      free(pL->listOfPoints);
    free(theGraph->listOfListsOfPoints);
    free(theGraph);
  }
```

4.3.14   readViewport

⟨*viewman*⟩+≡

```
  int readViewport(viewManager *viewPort,void *info,int size) {
    int canRead;
  again:
    if ((canRead=read(viewPort->viewIn,info,size)) > 0)   return(canRead);
    if (errno==EINTR || errno==EAGAIN) goto again;
    return(-1);
  }
```

4.3.15   superSelect

The function superselect!, if select returns a -1 due to an interrupt (EINTR), this routine checks to see if it's a child viewport that has closed. Expected global variables:
`checkClosedChild`

⟨*viewman*⟩+≡
```
  int superSelect(int n,int *rd,int *wr,int *ex,char *timeout) {
    int waiting;
    viewManager *viewport;
    int ret_val;
    ret_val = select(n, (void *)rd, (void *)wr, (void *)ex, (void *)timeout);
    while (ret_val == -1 && errno == EINTR) {
      /* checkClosedChild gets set by the SIGCHLD handler */
      if (checkClosedChild) {
        while ((waiting = wait(0)) == -1 );
        viewport = viewports;
        while ((viewport) && (viewport->PID != waiting))
          viewport = viewport->nextViewport;
        if (viewport) {
          /* we shouldn't really be doing this since child is dead */
          /* rmViewMgr(viewport); */
          /* flush(spadSock); */
          /* send_int(spadSock,1);   acknowledge to spad */
          checkClosedChild = no;
      #if defined(BSDplatform) || defined(MACOSXplatform)
          bsdSignal(SIGCHLD,endChild,DontRestartSystemCalls);
      #else
          bsdSignal(SIGCLD,endChild,DontRestartSystemCalls);
      #endif
        }
      }
      ret_val = select(n, (void *)rd, (void *)wr, (void *)ex, (void *)timeout);
    }
    return ret_val;
  }
```

4.3.16   brokenPipe

⟨*viewman*⟩+≡
```
  void brokenPipe(int sig) {
    fprintf(stderr,
            "The viewport manager tried to write to a non-existing pipe.\n");
  }
```

4.3.17 main

The function superselect!, if select returns a -1 due to an interrupt (EINTR), this routine checks to see if it's a child viewport that has closed. Expected global variables: checkClosedChild

⟨*viewman*⟩+≡
```
  int main(void) {
    graphStruct *aGraph;
    int keepLooking,code;
    bsdSignal(SIGPIPE,brokenPipe,DontRestartSystemCalls);
 #if defined(BSDplatform) || defined(MACOSXplatform)
    bsdSignal(SIGCHLD,endChild,RestartSystemCalls);
 #else
    bsdSignal(SIGCLD,endChild,RestartSystemCalls);
 #endif
    bsdSignal(SIGTERM,goodbye,DontRestartSystemCalls);
    /* Connect up to Axiom server */
    spadSock = connect_to_local_server(SpadServer,ViewportServer,Forever);
    if (spadSock == NULL) {
      fprintf(stderr,"The viewport manager couldn't connect to Axiom\n");
      exit(-1);
    }
 #ifdef DEBUG
    else
      fprintf(stderr,"viewman: Connected to Axiom\n");
 #endif
    /******** initialize ********/
    viewports = 0;
    graphList = 0;
    /******** getting stuff from spad and viewports ********
    ********   the viewports have priority over     ****
    ***   Axiom.                                ***/
    while (1) {
      FD_ZERO(&filedes); /* zero out file descriptor */
      FD_SET(spadSock->socket,&filedes);
      slot = viewports;
      while (slot) {
        FD_SET(slot->viewIn,&filedes);
        slot = slot->nextViewport;
      }
 #ifdef DEBUG
      fprintf(stderr,"Selection for filedes of %x \n",filedes);
 #endif
      code = check(superSelect(FD_SETSIZE,(void *) &filedes,0,0,0));
      for (;code<=0;)
```

```
        code = check(superSelect(FD_SETSIZE,(void *)&filedes,0,0,0));
     slot = viewports;
     keepLooking = 1;
     while (keepLooking && slot) {
       if (FD_ISSET(slot->viewIn,&filedes)) {
         keepLooking = 0;
#ifdef DEBUG
         fprintf(stderr,"Reading child viewport...\n");
#endif
         viewCommand = viewportClosing; /* close if read failed  */
         readViewport(slot,&viewCommand,intSize);
         switch (viewCommand) {
         case pick2D:
#ifdef DEBUG
           fprintf(stderr,"viewman: Doing 2D pick\n");
#endif
           picked = yes;
           readViewport(slot,&currentGraph,intSize); /* get the graph to pick */
           readViewport(slot,&currentGraphState,sizeof(graphStateStruct));
           break;
         case drop2D:
#ifdef DEBUG
           fprintf(stderr,"viewman: Doing 2D drop\n");
#endif
           if (picked) {
             write(slot->viewOut,&viewOkay,intSize);
             write(slot->viewOut,&currentGraph,intSize);
             sendGraphToView2D(0,currentGraph,slot,&currentGraphState);
           } else {
             write(slot->viewOut,&viewError,intSize);
             fprintf(stderr,"The viewport manager cannot drop a graph \n");
             fprintf(stderr,"because nothing has been picked yet.\n");
           }
           break;
         case viewportClosing:
#ifdef DEBUG
           fprintf(stderr,"viewman: closing viewport\n");
#endif
           closeChildViewport(slot);
           break;
         };  /* switch */
       };  /* if reading slot->viewIn */
       stepSlot = slot;
       slot = slot->nextViewport;
     };  /* while */
     if (keepLooking) {   /* if  1 => slots not read, read from spad */
```

```
#ifdef DEBUG
      fprintf(stderr,"viewman: still looking\n");
#endif
      viewType = get_int(spadSock);
      if (viewType == -1) goodbye(-1);
      viewCommand = get_int(spadSock);
      switch (viewType) {
      case view3DType:
#ifdef DEBUG
        fprintf(stderr,"viewman: making 3D viewport\n");
#endif
        if (viewCommand == makeViewport)
          forkView3D(view3DType);
        else
          funView3D(viewCommand);
        break;
      case viewTubeType:
#ifdef DEBUG
        fprintf(stderr,"viewman: viewing a tube\n");
#endif
        if (viewCommand == makeViewport)
          forkView3D(viewTubeType);
        else
          funView3D(viewCommand);
        break;
      case viewGraphType:
#ifdef DEBUG
        fprintf(stderr,"viewman: making a graph\n");
#endif
        if (viewCommand == makeGraph) {
          aGraph            = makeGraphFromSpadData();
          aGraph->nextGraph = graphList;
          graphList         = aGraph;
        }
        break;
      case view2DType:
#ifdef DEBUG
        fprintf(stderr,"viewman: forking 2D\n");
#endif
        if (viewCommand == makeViewport) {
          forkView2D();
        } else {
          funView2D(viewCommand);
        }
        break;
      }   /* switch on viewType */
```

```
    }   /* if (keepLooking) */
  }   /* while (1) */
}
```

Chapter 5

viewalone

5.1   viewalone Call Graph

This was generated by the GNU cflow program with the argument list. Note that the line:NNNN numbers refer to the line in the code after it has been tangled from this file.

```
cflow --emacs -l -n -b -T --omit-arguments viewalone.c

;; This file is generated by GNU cflow 1.3. -*- cflow -*-
    2 {   0} +-main() <int main () line:924>
    3 {   1}   +-printf()
    4 {   1}   +-sprintf()
    5 {   1}   +-fopen()
    6 {   1}   +-fprintf()
    7 {   1}   +-exit()
    8 {   1}   +-fscanf()
    9 {   1}   +-spoonView3D() <void spoonView3D () line:830>
   10 {   2}     +-sprintf()
   11 {   2}     +-check()
   12 {   2}     +-pipe()
   13 {   2}     +-fork()
   14 {   2}     +-fprintf()
   15 {   2}     +-exit()
   16 {   2}     +-dup2()
   17 {   2}     +-close()
   18 {   2}     +-getenv()
   19 {   2}     +-execl()
   20 {   2}     +-read()
   21 {   2}     +-makeView3DFromFileData()
                    <void makeView3DFromFileData () line:654>
   22 {   3}       +-fscanf()
   23 {   3}       +-fgets()
```

```
24 {   3}        +-malloc()
25 {   3}        +-strlen()
26 {   3}        +-fprintf()
27 {   3}        +-exit()
28 {   3}        +-sprintf()
29 {   3}        \-fclose()
30 {   2}     +-write()
31 {   2}     +-strlen()
32 {   2}     +-refPt()
33 {   2}     \-sleep()
34 {   1}  \-spoonView2D() <void spoonView2D () line:751>
35 {   2}     +-sprintf()
36 {   2}     +-check()
37 {   2}     +-pipe()
38 {   2}     +-fork()
39 {   2}     +-fprintf()
40 {   2}     +-exit()
41 {   2}     +-printf()
42 {   2}     +-dup2()
43 {   2}     +-close()
44 {   2}     +-getenv()
45 {   2}     +-execl()
46 {   2}     +-read()
47 {   2}     +-makeView2DFromFileData()
               <void makeView2DFromFileData () line:498>
48 {   3}       +-printf()
49 {   3}       +-fgets()
50 {   3}       +-malloc()
51 {   3}       +-strlen()
52 {   3}       +-fprintf()
53 {   3}       +-exit()
54 {   3}       +-sprintf()
55 {   3}       +-fscanf()
56 {   3}       +-fclose()
57 {   3}       +-fopen()
58 {   3}       \-perror()
59 {   2}     +-write()
60 {   2}     +-strlen()
61 {   2}     +-sendGraphToView2D() <void sendGraphToView2D () line:444>
62 {   3}       +-printf()
63 {   3}       \-write()
64 {   2}     \-sleep()
```

5.2   Constants and Headers

5.2.1   System includes

⟨*viewalone*⟩≡

```
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <stdio.h>
```

5.2.2 Local includes

⟨*viewalone*⟩+≡
  ⟨*include/view3d.h*⟩
  ⟨*include/view2d.h*⟩
  ⟨*include/actions.h*⟩
  ⟨*include/viewcommand.h*⟩
```
#include "util.h1"
```

5.2.3 defines

⟨*viewalone*⟩+≡
```
      /* Viewport Commands */
#define makeViewport -1
#define makeGraph    -1

      /* Assorted Junk */
#define check(code) checker(code,__LINE__,"")
#define components
#define maxConnect 40
#define intSize sizeof(int)
#define floatSize sizeof(float)
#define no 0
#define spadActionMode
#define yes 1
```

5.2.4   extern references

⟨*viewalone*⟩+≡

```
extern viewManager viewP;
extern view3DStruct doView3D;
extern view2DStruct doView2D;
extern graphStruct graphArray[maxGraphs];
extern graphStateStruct graphStateArray[maxGraphs];
extern graphStateStruct graphStateBackupArray[maxGraphs];
extern tubeModel doViewTube;
extern int viewType;
extern int filedes;
extern int ack;
extern char errorStr[80];
extern int viewOkay;
extern int viewError;
extern FILE *viewFile;
extern char filename[256];
extern char pathname[256];
```

5.2.5 global variables

⟨*viewalone*⟩+≡

```
viewManager viewP;   /* note that in viewman, this is called viewports */

/* 3D stuff */
view3DStruct doView3D;

/* 2D stuff */
view2DStruct doView2D;
graphStruct      graphArray[maxGraphs];
graphStateStruct graphStateArray[maxGraphs];

/* tube stuff */
tubeModel doViewTube;

int ack;
int filedes;
char filename[256];
char errorStr[80];
char pathname[256];
int viewError = -1;
FILE *viewFile;
int viewOkay  = 0;
int viewType;
```

5.3   Code

5.3.1   sendGraphToView2D

⟨*viewalone*⟩+≡

```
void sendGraphToView2D(int i,int there,viewManager *viewP) {
  graphStruct      *gPtr;
  pointListStruct  *llPtr;
  pointStruct      *p;
  int j,k;
  printf("(spoon2d sendGraphToView2D) i=%d there=%d\n",i,there);
  if (there) {
    gPtr = &(graphArray[i]);
    printf("(spoon2d sendGraphToView2D) graph %d is there\n",i);
    write(viewP->viewOut,&(gPtr->xmin),floatSize);
    write(viewP->viewOut,&(gPtr->xmax),floatSize);
    write(viewP->viewOut,&(gPtr->ymin),floatSize);
    write(viewP->viewOut,&(gPtr->ymax),floatSize);
    write(viewP->viewOut,&(gPtr->xNorm),floatSize);
    write(viewP->viewOut,&(gPtr->yNorm),floatSize);
    write(viewP->viewOut,&(gPtr->spadUnitX),floatSize);
    write(viewP->viewOut,&(gPtr->spadUnitY),floatSize);
    write(viewP->viewOut,&(gPtr->unitX),floatSize);
    write(viewP->viewOut,&(gPtr->unitY),floatSize);
    write(viewP->viewOut,&(gPtr->originX),floatSize);
    write(viewP->viewOut,&(gPtr->originY),floatSize);
    write(viewP->viewOut,&(gPtr->numberOfLists),intSize);
    llPtr = gPtr->listOfListsOfPoints;
    for (j=0; j<(gPtr->numberOfLists); j++) {
      write(viewP->viewOut,&(llPtr->numberOfPoints),intSize);
      p = llPtr->listOfPoints;
      for (k=0; k<(llPtr->numberOfPoints); k++) {
        write(viewP->viewOut,&(p->x),floatSize);
        write(viewP->viewOut,&(p->y),floatSize);
        write(viewP->viewOut,&(p->hue),floatSize);
        write(viewP->viewOut,&(p->shade),floatSize);
        p++;
      }    /* for k in list of points */
      write(viewP->viewOut,&(llPtr->pointColor),intSize);
      write(viewP->viewOut,&(llPtr->lineColor),intSize);
      write(viewP->viewOut,&(llPtr->pointSize),intSize);
      llPtr++;
    }   /* for j in list of lists of points */
    /* a state is defined for a graph if it is there */
    write(viewP->viewOut,&(graphStateArray[i].scaleX),floatSize);
    write(viewP->viewOut,&(graphStateArray[i].scaleY),floatSize);
```

```
        write(viewP->viewOut,&(graphStateArray[i].deltaX),floatSize);
        write(viewP->viewOut,&(graphStateArray[i].deltaY),floatSize);
        write(viewP->viewOut,&(graphStateArray[i].pointsOn),intSize);
        write(viewP->viewOut,&(graphStateArray[i].connectOn),intSize);
        write(viewP->viewOut,&(graphStateArray[i].splineOn),intSize);
        write(viewP->viewOut,&(graphStateArray[i].axesOn),intSize);
        write(viewP->viewOut,&(graphStateArray[i].axesColor),intSize);
        write(viewP->viewOut,&(graphStateArray[i].unitsOn),intSize);
        write(viewP->viewOut,&(graphStateArray[i].unitsColor),intSize);
        write(viewP->viewOut,&(graphStateArray[i].showing),intSize);
    }   /* if graph is there */
}
```

5.3.2   makeView2DFromFileData

⟨*viewalone*⟩+≡

```
void makeView2DFromFileData(view2DStruct *doView2D) {
  int i,j,k;
  char title[256];
  FILE *graphFile;
  char graphFilename[256];
  pointListStruct *aList;
  pointStruct     *aPoint;
  printf("(spoon2d makeView2DFromFileData)\n");
  fgets(title,256,viewFile);
  printf("(spoon2d) title=%s\n",title);
  if (!(doView2D->title =
        (char *)malloc((strlen(title)+1) * sizeof(char)))) {
    fprintf(stderr,
            "Ran out of memory (malloc) trying to get the title.\n");
    exit(-1);
  }
  sprintf(doView2D->title,"%s",title);
  /* put in a null terminator over the newline that the fgets reads */
  doView2D->title[strlen(doView2D->title)-1] = '\0';
  fscanf(viewFile,"%d %d %d %d\n",
         &(doView2D->vX),
         &(doView2D->vY),
         &(doView2D->vW),
         &(doView2D->vH));
  printf("(spoon2d) X=%d Y=%d W=%d H=%d \n",
         doView2D->vX,doView2D->vY,doView2D->vW,doView2D->vH);
  for (i=0; i<maxGraphs; i++) {
    fscanf(viewFile,"%d\n",
           &(graphArray[i].key));
    printf("(spoon2d) i=%d key=%d\n",
           i,graphArray[i].key);
    fscanf(viewFile,"%g %g\n",
           &(graphStateArray[i].scaleX),
           &(graphStateArray[i].scaleY));
    printf("(spoon2d) scaleX=%g scaleY=%g\n",
             graphStateArray[i].scaleX,graphStateArray[i].scaleY);
    fscanf(viewFile,"%g %g\n",
           &(graphStateArray[i].deltaX),
           &(graphStateArray[i].deltaY));
    printf("(spoon2d) deltaX=%g deltaY=%g\n",
             graphStateArray[i].deltaX,graphStateArray[i].deltaY);
    fscanf(viewFile,"%g %g\n",
           &(graphStateArray[i].centerX),
```

```
                  &(graphStateArray[i].centerY));
          printf("(spoon2d) centerX=%g centerY=%g\n",
                  graphStateArray[i].centerX,graphStateArray[i].centerY);
          fscanf(viewFile,"%d %d %d %d %d %d %d\n",
                  &(graphStateArray[i].pointsOn),
                  &(graphStateArray[i].connectOn),
                  &(graphStateArray[i].splineOn),
                  &(graphStateArray[i].axesOn),
                  &(graphStateArray[i].axesColor),
                  &(graphStateArray[i].unitsOn),
                  &(graphStateArray[i].unitsColor));
          printf("(spoon2d) pointsOn=%d connectOn=%d splineOn=%d\n",
                  graphStateArray[i].pointsOn,graphStateArray[i].connectOn,
                  graphStateArray[i].splineOn);
          printf("(spoon2d) axesOn=%d axesColor=%d unitsOn=%d unitsColor=%d\n",
                  graphStateArray[i].axesOn,
                  graphStateArray[i].axesColor,graphStateArray[i].unitsOn,
                  graphStateArray[i].unitsColor);
          fscanf(viewFile,"%d %d\n",
                  &(graphStateArray[i].showing),
                  &(graphStateArray[i].selected));
          printf("(spoon2d) showing=%d selected=%d\n",
                  graphStateArray[i].showing,graphStateArray[i].selected);
        }
        fclose(viewFile);
        for (i=0; i<maxGraphs; i++) {
          if (graphArray[i].key) {
            /** OPEN FILE FOR GRAPHS **/
            sprintf(graphFilename,"%s%s%d",pathname,"/graph",i);
            if ((graphFile = fopen(graphFilename,"r")) == NULL) {
              fprintf(stderr,"   Error: Cannot find the file %s\n",graphFilename);
              perror("fopen");
              return;
            } else {
              printf("(spoon2d) \n\nGRAPH%i\n",i);
              fscanf(graphFile,"%g %g %g %g\n",
                      &(graphArray[i].xmin),
                      &(graphArray[i].ymin),
                      &(graphArray[i].xmax),
                      &(graphArray[i].ymax));
              printf("(spoon2d) xmin=%g ymin=%g xmax=%g ymax=%g\n",
                      graphArray[i].xmin,graphArray[i].ymin,
                      graphArray[i].xmax,graphArray[i].ymax);
              fscanf(graphFile,"%g %g\n",
                      &(graphArray[i].xNorm),
                      &(graphArray[i].yNorm));
```

```
        printf("(spoon2d) xNorm=%g yNorm=%g\n",
                graphArray[i].xNorm,graphArray[i].yNorm);
        fscanf(graphFile,"%g %g\n",
                &(graphArray[i].originX),
                &(graphArray[i].originY));
        printf("(spoon2d) originX=%g originY=%g\n",
                graphArray[i].originX,graphArray[i].originY);
        fscanf(graphFile,"%g %g\n",
                &(graphArray[i].spadUnitX),
                &(graphArray[i].spadUnitY));
        printf("(spoon2d) spadUnitX=%g spadUnitY=%g\n",
                graphArray[i].spadUnitX,graphArray[i].spadUnitY);
        fscanf(graphFile,"%g %g\n",
                &(graphArray[i].unitX),
                &(graphArray[i].unitY));
        printf("(spoon2d) unitX=%g unitY=%g\n",
                graphArray[i].unitX,graphArray[i].unitY);
        fscanf(graphFile,"%d\n",
                &(graphArray[i].numberOfLists));
        printf("(spoon2d) numberOfLists=%d\n",
                 graphArray[i].numberOfLists);
        if (!(aList =
                (pointListStruct *)malloc(graphArray[i].numberOfLists *
                                          sizeof(pointListStruct)))) {
          fprintf(stderr,"viewalone: Fatal Error>> Out of memory trying\n");
          fprintf(stderr," to receive a graph.\n");
          exit(-1);
        }
        graphArray[i].listOfListsOfPoints = aList;
        for (j=0;
             j<graphArray[i].numberOfLists;
             j++, aList++) {
          printf("(spoon2d) list number %d\n",j);
          fscanf(graphFile,"%d\n",&(aList->numberOfPoints));
          printf("(spoon2d) number of points %d\n",
                   aList->numberOfPoints);
          fscanf(graphFile,"%d %d %d\n",
                  &(aList->pointColor),
                  &(aList->lineColor),
                  &(aList->pointSize));
          printf("(spoon2d) pointColor=%d lineColor=%d pointSize=%d\n",
                  aList->pointColor,aList->lineColor,aList->pointSize);
          if (!(aPoint = (pointStruct *)malloc(aList->numberOfPoints *
                                            sizeof(pointStruct)))) {
            fprintf(stderr,"viewalone: Fatal Error>> Out of memory trying\n");
            fprintf(stderr," to receive a graph.\n");
```

```
            exit(-1);
          }
          aList->listOfPoints = aPoint;    /** point to current point list **/
          for (k=0;
               k<aList->numberOfPoints;
               k++,aPoint++)
          {  fscanf(graphFile,"%g %g %g %g\n",
                  &(aPoint->x),
                  &(aPoint->y),
                  &(aPoint->hue),
                  &(aPoint->shade));
             printf("(spoon2d)k=%d x=%g y=%g hue=%g shade=%g\n",
                  k,aPoint->x,aPoint->y,aPoint->hue,aPoint->shade);
          }
        } /* for j, aList */
        fclose(graphFile);
      } /* else, opened up a file */
    } /* if graph.key */
  } /* for i */
} /* makeView2DFromFileData */
```

5.3.3 makeView3DFromFileData

⟨*viewalone*⟩+≡

```
  void makeView3DFromFileData(int type) {
    int i,j,k;
    char title[256];
    LLPoint *anLLPoint;
    LPoint *anLPoint;
    viewTriple *aPoint;
    int *anIndex;
    /* fscanf(doView3D,""); */
    /* read in the view3DStruct stuff */
    /* &view3DType already read */
    doView3D.typeOf3D = type;
    fscanf(viewFile,"%f %f %f %f %f %f\n",
           &(doView3D.xmin),
           &(doView3D.xmax),
           &(doView3D.ymin),
           &(doView3D.ymax),
           &(doView3D.zmin),
           &(doView3D.zmax));
    fgets(title,256,viewFile);
    if (!(doView3D.title = (char *)malloc((strlen(title)+1) *
                                          sizeof(char)))) {
      fprintf(stderr,"Ran out of memory (malloc) trying to get the title.\n");
      exit(-1);
    }
    sprintf(doView3D.title,"%s",title);
    /* put in a null terminator over the newline that the fgets reads */
    doView3D.title[strlen(doView3D.title)-1] = '\0';
    fscanf(viewFile,"%f %f %f %f %f %f %f %f\n",
           &(doView3D.deltaX),
           &(doView3D.deltaY),
           &(doView3D.scale),
           &(doView3D.scaleX),
           &(doView3D.scaleY),
           &(doView3D.scaleZ),
           &(doView3D.theta),
           &(doView3D.phi));
    fscanf(viewFile,"%d %d %d %d\n",
           &(doView3D.vX),
           &(doView3D.vY),
           &(doView3D.vW),
           &(doView3D.vH));
    fscanf(viewFile,"%d %d %d %d %d %d %d\n",
           &(doView3D.showCP),
```

```
            &(doView3D.style),
            &(doView3D.AxesOn),
            &(doView3D.hueOff),
            &(doView3D.numOfHues),
            &(doView3D.diagonals),
            &(doView3D.outlineRenderOn));
    fscanf(viewFile,"%f %f %f %f\n",
            &(doView3D.lightVec[0]),
            &(doView3D.lightVec[1]),
            &(doView3D.lightVec[2]),
            &(doView3D.translucency));
    fscanf(viewFile,"%d %f\n",
            &(doView3D.perspective),
            &(doView3D.eyeDistance));
    /* get generalized 3D components */
    fscanf(viewFile,"%d\n",
            &(doView3D.numOfPoints));
    aPoint = doView3D.points = (viewTriple *)malloc(doView3D.numOfPoints*
                                                  sizeof(viewTriple));
    for (i=0; i<doView3D.numOfPoints; i++, aPoint++)
      fscanf(viewFile,"%g %g %g %g\n",
             &(aPoint->x),
             &(aPoint->y),
             &(aPoint->z),
             &(aPoint->c));
    fscanf(viewFile,"%d\n",
            &(doView3D.lllp.numOfComponents));
    anLLPoint = doView3D.lllp.llp =
      (LLPoint *)malloc(doView3D.lllp.numOfComponents*sizeof(LLPoint));
    for (i=0; i<doView3D.lllp.numOfComponents; i++,anLLPoint++) {
      fscanf(viewFile,"%d %d\n",
             &(anLLPoint->prop.closed),
             &(anLLPoint->prop.solid));
      fscanf(viewFile,"%d\n",
             &(anLLPoint->numOfLists));
      anLPoint = anLLPoint->lp =
        (LPoint *)malloc(anLLPoint->numOfLists*sizeof(LPoint));
      for (j=0; j<anLLPoint->numOfLists; j++,anLPoint++) {
        fscanf(viewFile,"%d %d\n",
               &(anLPoint->prop.closed),
               &(anLPoint->prop.solid));
        fscanf(viewFile,"%d\n",
               &(anLPoint->numOfPoints));
        anIndex = anLPoint->indices =
          (int *)malloc(anLPoint->numOfPoints*sizeof(int));
        for (k=0; k<anLPoint->numOfPoints; k++,anIndex++) {
```

```
        fscanf(viewFile,"%dn",anIndex);
      } /* for points in LPoints (k) */
    } /* for LPoints in LLPoints (j) */
  } /* for LLPoints in LLLPoints (i) */
  fclose(viewFile);
  doView3D.scaleDown = no ;
}
```

5.3.4 spoonView2D

⟨*viewalone*⟩+≡

```c
void spoonView2D(void) {
  int  i,code,pipe0[2],pipe1[2],there;
  char envAXIOM[100],runView[100];
  sprintf(errorStr,"%s","creating pipes");
  check(pipe(pipe0));
  check(pipe(pipe1));
  switch(fork()) {
  case -1:
    fprintf(stderr,"Cannot create a new process - ");
    fprintf(stderr,"probably have too many things running already.\n");
    exit(-1);
  case 0:
    /************
     *   Child  *
     ************/
    printf("(spoon2d child) mapping of pipes to standard I/O for view2d\n");
    sprintf(errorStr,"%s",
             "(viewalone) mapping of pipes to standard I/O for view2d");
    check(dup2(pipe0[0],0));
    check(dup2(pipe1[1],1));
    close(pipe0[0]);
    close(pipe0[1]);
    close(pipe1[0]);
    close(pipe1[1]);
    printf("(spoon2d child) start the TwoDimensionalViewport process\n");
    sprintf(errorStr,"%s",
             "(viewalone) execution of the TwoDimensionalViewport process");
    sprintf(envAXIOM,"%s",getenv("AXIOM"));
    sprintf(runView,"%s%s",envAXIOM,"/lib/view2d");
    check(execl(runView,runView,NULL));
    fprintf(stderr,
     "Could not execute view2d! Check that view2d is on your path.\n");
    exit(-1);
  default:
    /*************
     *   Parent  *
     *************/
    viewP.viewType = view2DType;
    /* set up pipes to child process */
    close(pipe0[0]);
    close(pipe1[1]);
    viewP.viewIn  = pipe1[0];
    viewP.viewOut = pipe0[1];
```

```
      printf("(spoon2d parent) pipes created\n");
      if (viewP.viewIn <0) {
        fprintf(stderr,"Could not connect from Viewport manager to viewport\n");
        fprintf(stderr," process. Try again.\n");
        return;
      } else {
        code = read(viewP.viewIn,&ack,intSize);
        if (code < 0) {
         fprintf(stderr,"Could not connect from Viewport manager to viewport\n");
         fprintf(stderr," process. Try again.\n");
          return;
        }
      }
      printf("(spoon2d parent) making View2D data\n");
      makeView2DFromFileData(&doView2D);
      /* tell child it is to be a stand alone program */
      i = yes;
      fprintf(stderr,"  Transmitting data to viewport...\n");
      write(viewP.viewOut,&i,intSize);
      write(viewP.viewOut,&doView2D,sizeof(view2DStruct));
      i = strlen(doView2D.title)+1;
      write(viewP.viewOut,&i,intSize); /* tell the length of title to child */
      write(viewP.viewOut,doView2D.title,i);  /* tell the title to child */
      for (i=0; i<maxGraphs; i++) {
        there = graphArray[i].key;
        write(viewP.viewOut,&there,intSize);
        sendGraphToView2D(i,there,&viewP);
      };  /* for i in graphs */
      fprintf(stderr,"  Done.\n");
      /*** get acknowledge from viewport */
      code = read(viewP.viewIn,&(viewP.viewWindow),sizeof(Window));
      sleep(1);  /* wait a second...*/
      exit(0);
  }   /* switch */
}   /* forkView2D() */
```

5.3.5 spoonView3D

This file forks a child process and exits the parent. It has the same general form as ../view-man/funView3D() and so changes there may require similar changes here.

⟨*viewalone*⟩+≡

```
void spoonView3D(int type) {
  int  i,j,k,code,pipe0[2],pipe1[2];
  char envAXIOM[100],runView[100];
  LLPoint *anLLPoint;
  LPoint *anLPoint;
  int *anIndex;
  sprintf(errorStr,"%s","creating pipes");
  check(pipe(pipe0));
  check(pipe(pipe1));
  switch(fork()) {
  case -1:
    fprintf(stderr,"can't create a child process\n");
    fprintf(stderr,"you may have too many processes running\n");
    exit(-1);
  case 0:
    /*  Child  */
    sprintf(errorStr,"%s",
            "(viewalone) mapping of pipes to standard I/O for view3d");
    check(dup2(pipe0[0],0));
    check(dup2(pipe1[1],1));
    close(pipe0[0]);
    close(pipe0[1]);
    close(pipe1[0]);
    close(pipe1[1]);
    sprintf(errorStr,"%s",
            "(viewalone) execution of the ThreeDimensionalViewport process");
    sprintf(envAXIOM,"%s",getenv("AXIOM"));
    sprintf(runView,"%s%s",envAXIOM,"/lib/view3d");
    check(execl(runView,runView,NULL));
    fprintf(stderr,"Could not execute view3d!\n");
    exit(-1);
  default:
    /*  Parent  */
    viewP.viewType = type;
    /* set up pipes to child process */
    close(pipe0[0]);
    close(pipe1[1]);
    viewP.viewIn  = pipe1[0];
    viewP.viewOut = pipe0[1];
    if (viewP.viewIn <0) {
```

```
      fprintf(stderr,
              "can't set up pipes to viewport process. Try again.\n");
      return;
  } else {
    code = read(viewP.viewIn,&ack,intSize);
    if (code < 0) {
      fprintf(stderr,"can't read from viewport process pipe. Try again.\n");
      return;
    }
  }
  makeView3DFromFileData(type);
  /* tell child it is to be a stand alone program */
  i = yes;
  fprintf(stderr,"  Transmitting data to viewport...\n");
  write(viewP.viewOut,&i,intSize);
  write(viewP.viewOut,&doView3D,sizeof(view3DStruct));
  i = strlen(doView3D.title)+1;
  write(viewP.viewOut,&i,intSize);  /* tell the length of title to child */
  write(viewP.viewOut,doView3D.title,i);  /* tell the title to */
  write(viewP.viewOut,&(doView3D.lightVec[0]),floatSize);
  write(viewP.viewOut,&(doView3D.lightVec[1]),floatSize);
  write(viewP.viewOut,&(doView3D.lightVec[2]),floatSize);
  write(viewP.viewOut,&(doView3D.numOfPoints),intSize);
  for (i=0; i<doView3D.numOfPoints; i++) {
    write(viewP.viewOut,&(refPt(doView3D,i)->x),floatSize);
    write(viewP.viewOut,&(refPt(doView3D,i)->y),floatSize);
    write(viewP.viewOut,&(refPt(doView3D,i)->z),floatSize);
    write(viewP.viewOut,&(refPt(doView3D,i)->c),floatSize);
  }
  /* send generalized 3D components */
  write(viewP.viewOut,&(doView3D.lllp.numOfComponents),intSize);
  anLLPoint = doView3D.lllp.llp;
  for (i=0; i<doView3D.lllp.numOfComponents; i++,anLLPoint++) {
    write(viewP.viewOut,&(anLLPoint->prop.closed),intSize);
    write(viewP.viewOut,&(anLLPoint->prop.solid),intSize);
    write(viewP.viewOut,&(anLLPoint->numOfLists),intSize);
    anLPoint = anLLPoint->lp;
    for (j=0; j<anLLPoint->numOfLists; j++,anLPoint++) {
      write(viewP.viewOut,&(anLPoint->prop.closed),intSize);
      write(viewP.viewOut,&(anLPoint->prop.solid),intSize);
      write(viewP.viewOut,&(anLPoint->numOfPoints),intSize);
      anIndex = anLPoint->indices;
      for (k=0; k<anLPoint->numOfPoints; k++,anIndex++)
        write(viewP.viewOut,anIndex,intSize);
    } /* for LPoints in LLPoints (j) */
  } /* for LLPoints in LLLPoints (i) */
```

```
    fprintf(stderr,"    Done.\n");
    /*** get acknowledge from viewport */
    code = read(viewP.viewIn,&(viewP.viewWindow),sizeof(Window));
    sleep(1);  /* wait a second...*/
    exit(0);
  }   /* switch */
}    /* spoonView3D() */
```

5.3.6 main

⟨*viewalone*⟩+≡

```
  int main(int argc,char *argv[])
  {       printf("viewalone called with argc=%d\n",argc);
          printf("viewalone called with argv[1]=%s\n",argv[0]);
          printf("viewalone called with argv[2]=%s\n",argv[1]);
 /******** Open files and Figure out the viewport type ********/
          sprintf(filename,"%s%s",argv[1],".view/data");
          if((viewFile = fopen(filename,"r")) == NULL ) {
             sprintf(filename,"%s%s",argv[1],"/data");
             if((viewFile = fopen(filename,"r")) == NULL ){
               fprintf(stderr,"Error: Cannot find the file %s%s or %s%s\n",
                   argv[1],".view/data",argv[1],"/data");
               exit(-1);
             }
             sprintf(pathname,"%s",argv[1]);
          }
          else{
               sprintf(pathname,"%s%s",argv[1],".view");
             }
          fscanf(viewFile,"%d\n",&viewType);
          printf("filename = %s\n",filename);
          printf("viewType = %d\n",viewType);
          switch (viewType) {
          case view3DType:
          case viewTubeType:
                  printf("calling spoonView3D\n");
                  spoonView3D(viewType);
                  break;
          case view2DType:
                  printf("calling spoonView2D\n");
                  spoonView2D();
                  break;
          }  /* switch */
        printf("The first number in the file, %d, called the ",viewType);
        printf("viewType, not a valid value. It must be a number in the ");
        printf("range of [1..4]\n");
        return(0);
        }
```

The TESTFILE is created in the mnt directory to provide an example file to use for viewalone. The parabola example is detailed in the chapter on Graphics File Formats. The directory parabola.view will contain the data and graph0 files and is autogenerated from the documentation in that chapter.

Chapter 6

view2d

6.1   view2d Call Graph

This was generated by the GNU cflow program with the argument list. Note that the line:NNNN numbers refer to the line in the code after it has been tangled from this file.

```
cflow --emacs -l -n -b -T --omit-arguments view2d.c

;; This file is generated by GNU cflow 1.3. -*- cflow -*-
    2 {   0} +-main() <int main () line:4640>
    3 {   1}   +-XOpenDisplay()
    4 {   1}   +-getenv()
    5 {   1}   +-fprintf()
    6 {   1}   +-DefaultScreen()
    7 {   1}   +-RootWindow()
    8 {   1}   +-XCreateAssocTable()
              | <HashTable *XCreateAssocTable () line:1851>
    9 {   2}     +-malloc()
   10 {   2}     +-hash_init()
   11 {   2}     +-TrivEqual() <int TrivEqual () line:1841>
   12 {   2}     \-TrivHashCode() <int TrivHashCode () line:1846>
   13 {   1}   +-XInitSpadFill()
   14 {   1}   +-exitWithAck()
   15 {   1}   +-mergeDatabases() <void mergeDatabases () line:2980>
   16 {   2}   | +-XrmInitialize()
   17 {   2}   | +-strcpy()
   18 {   2}   | +-strcat()
   19 {   2}   | +-XrmGetFileDatabase()
   20 {   2}   | +-XrmMergeDatabases()
   21 {   2}   | +-XResourceManagerString()
   22 {   2}   | +-XrmGetStringDatabase()
   23 {   2}   | +-getenv()
```

```
24 {   2}    | +-strlen()
25 {   2}    | \-gethostname()
26 {   1}    +-XrmGetResource()
27 {   1}    +-strncpy()
28 {   1}    +-strcpy()
29 {   1}    +-strcmp()
30 {   1}    +-WhitePixel()
31 {   1}    +-BlackPixel()
32 {   1}    +-malloc()
33 {   1}    +-strlen()
34 {   1}    +-XQueryFont()
35 {   1}    +-XGContextFromGC()
36 {   1}    +-DefaultGC()
37 {   1}    +-XLoadQueryFont()
38 {   1}    +-PSGlobalInit() <int PSGlobalInit () line:1375>
39 {   2}    | +-tempnam()
40 {   2}    | +-sprintf()
41 {   2}    | +-free()
42 {   2}    | +-getenv()
43 {   2}    | \-fprintf()
44 {   1}    +-monoColor()
45 {   1}    +-XCreateGC()
46 {   1}    +-carefullySetFont()
47 {   1}    +-PSCreateContext() <int PSCreateContext () line:1497>
48 {   2}      +-malloc()
49 {   2}      +-fprintf()
50 {   2}      +-exit()
51 {   2}      +-sprintf()
52 {   2}      +-fopen()
53 {   2}      \-fclose()
54 {   1}    +-centerX() <int centerX () line:1686>
55 {   2}    | +-XGContextFromGC()
56 {   2}    | +-XQueryFont()
57 {   2}    | +-XTextWidth()
58 {   2}    | \-XFreeFontInfo()
59 {   1}    +-centerY() <int centerY () line:1700>
60 {   2}      +-XGContextFromGC()
61 {   2}      +-XQueryFont()
62 {   2}      \-XFreeFontInfo()
63 {   1}    +-check()
64 {   1}    +-write()
65 {   1}    +-readViewman() <int readViewman () line:3743>
66 {   2}      +-sprintf()
67 {   2}      +-check()
68 {   2}      \-read()
69 {   1}    +-getGraphFromViewman() <void getGraphFromViewman () line:2882>
70 {   2}      +-readViewman() <int readViewman () line:3743> [see 65]
71 {   2}      +-malloc()
72 {   2}      +-fprintf()
73 {   2}      +-exitWithAck()
```

```
 74 {   2}       \-RootWindow()
 75 {   1}     +-makeView2D() <viewPoints *makeView2D () line:4483>
 76 {   2}       +-makeViewport() <viewPoints *makeViewport () line:4394>
 77 {   3}         +-fprintf()
 78 {   3}         +-malloc()
 79 {   3}         +-sleep()
 80 {   3}         +-exitWithAck()
 81 {   3}         +-RootWindow()
 82 {   3}         +-strcpy()
 83 {   3}         +-XCreateBitmapFromData()
 84 {   3}         +-XQueryColor()
 85 {   3}         +-XCreatePixmapCursor()
 86 {   3}         +-XCreateWindow()
 87 {   3}         +-XInternAtom()
 88 {   3}         +-XSetWMProtocols()
 89 {   3}         +-XSetNormalHints()
 90 {   3}         +-XSetStandardProperties()
 91 {   3}         +-makeControlPanel()
                   | <controlPanelStruct *makeControlPanel () line:2777>
 92 {   4}         | +-malloc()
 93 {   4}         | +-fprintf()
 94 {   4}         | +-exitWithAck()
 95 {   4}         | +-RootWindow()
 96 {   4}         | +-getControlXY() <controlXY getControlXY () line:2722>
 97 {   5}         | | +-XQueryTree()
 98 {   5}         | | +-XFree()
 99 {   5}         | | \-XGetWindowAttributes()
100 {   4}         | +-XCreateBitmapFromData()
101 {   4}         | +-XQueryColor()
102 {   4}         | +-XCreatePixmapCursor()
103 {   4}         | +-XCreateWindow()
104 {   4}         | +-XSetNormalHints()
105 {   4}         | +-XSetStandardProperties()
106 {   4}         | +-initButtons() <int initButtons () line:1874>
107 {   5}         |   \-strcpy()
108 {   4}         | +-XMakeAssoc() <void XMakeAssoc () line:1860>
109 {   5}         |   \-hash_insert()
110 {   4}         | \-XMapWindow()
111 {   3}         +-putControlPanelSomewhere()
                     | <void putControlPanelSomewhere () line:2859>
112 {   4}           +-getControlXY()
                     | <controlXY getControlXY () line:2722> [see 96]
113 {   4}           +-XRaiseWindow()
114 {   4}           +-XMoveWindow()
115 {   4}           +-drawControlPanel() <void drawControlPanel () line:2542>
116 {   5}           | +-GSetForeground() <int GSetForeground () line:1548>
117 {   6}           |   +-XSetForeground()
118 {   6}           |   +-fopen()
119 {   6}           |   +-fprintf()
120 {   6}           |   +-PSfindGC() <char *PSfindGC () line:1534>
```

```
121 {   7}          |   | \-fprintf()
122 {   6}          |   \-fclose()
123 {   5}          | +-GSetLineAttributes()
                    |   <int GSetLineAttributes () line:1608>
124 {   6}          |   +-XSetLineAttributes()
125 {   6}          |   +-fprintf()
126 {   6}          |   +-fopen()
127 {   6}          |   +-PSfindGC() <char *PSfindGC () line:1534> [see 120]
128 {   6}          |   \-fclose()
129 {   5}          | +-GDrawLine() <int GDrawLine () line:1124>
130 {   6}          |   +-XDrawLine()
131 {   6}          |   +-fopen()
132 {   6}          |   +-fprintf()
133 {   6}          |   +-PSfindGC() <char *PSfindGC () line:1534> [see 120]
134 {   6}          |   \-fclose()
135 {   5}          | +-GDrawRectangle() <int GDrawRectangle () line:1223>
136 {   6}          |   +-XDrawRectangle()
137 {   6}          |   +-fopen()
138 {   6}          |   +-fprintf()
139 {   6}          |   +-PSfindGC() <char *PSfindGC () line:1534> [see 120]
140 {   6}          |   \-fclose()
141 {   5}          | +-writeControlTitle()
                    |   <void writeControlTitle () line:2483>
142 {   6}          |   +-strlen()
143 {   6}          |   +-XClearArea()
144 {   6}          |   +-GSetForeground()
                    |   | <int GSetForeground () line:1548> [see 116]
145 {   6}          |   +-GDrawImageString()
                    |   | <int GDrawImageString () line:1059>
146 {   7}          |   | +-XDrawImageString()
147 {   7}          |   | +-fopen()
148 {   7}          |   | +-fprintf()
149 {   7}          |   | +-PSfindGC() <char *PSfindGC () line:1534> [see 120]
150 {   7}          |   | \-fclose()
151 {   6}          |   \-centerX() <int centerX () line:1686> [see 54]
152 {   5}          | +-strlen()
153 {   5}          | +-GDrawString() <int GDrawString () line:1310>
154 {   6}          | | +-XDrawString()
155 {   6}          | | +-fopen()
156 {   6}          | | +-fprintf()
157 {   6}          | | +-PSfindGC() <char *PSfindGC () line:1534> [see 120]
158 {   6}          | | \-fclose()
159 {   5}          | +-centerX() <int centerX () line:1686> [see 54]
160 {   5}          | +-monoColor()
161 {   5}          | +-GDrawLines() <int GDrawLines () line:1154>
162 {   6}          |   +-XDrawLines()
163 {   6}          |   +-fopen()
164 {   6}          |   +-fprintf()
165 {   6}          |   +-PSfindGC() <char *PSfindGC () line:1534> [see 120]
166 {   6}          |   \-fclose()
```

```
167 {   5}              | +-GSetBackground() <int GSetBackground () line:1578>
168 {   6}              |   +-XSetBackground()
169 {   6}              |   +-fopen()
170 {   6}              |   +-fprintf()
171 {   6}              |   +-PSfindGC() <char *PSfindGC () line:1534> [see 120]
172 {   6}              |   \-fclose()
173 {   5}              | +-GDrawImageString()
                        | | <int GDrawImageString () line:1059> [see 145]
174 {   5}              | +-centerY() <int centerY () line:1700> [see 59]
175 {   5}              | +-strcpy()
176 {   5}              | +-GDrawPushButton() <int GDrawPushButton () line:1285>
177 {   6}              |   +-strlen()
178 {   6}              |   +-XClearArea()
179 {   6}              |   +-GSetForeground()
                        |   | <int GSetForeground () line:1548> [see 116]
180 {   6}              |   +-GDraw3DButtonIn() <int GDraw3DButtonIn () line:1269>
181 {   7}              |   | +-GDrawRectangle()
                        |   | | <int GDrawRectangle () line:1223> [see 135]
182 {   7}              |   | \-GDrawLine() <int GDrawLine () line:1124> [see 129]
183 {   6}              |   +-GDraw3DButtonOut()
                        |   | <int GDraw3DButtonOut () line:1254>
184 {   7}              |   | +-GDrawRectangle()
                        |   | | <int GDrawRectangle () line:1223> [see 135]
185 {   7}              |   | \-GDrawLine() <int GDrawLine () line:1124> [see 129]
186 {   6}              |   +-GDrawString()
                        |   | <int GDrawString () line:1310> [see 153]
187 {   6}              |   +-centerX() <int centerX () line:1686> [see 54]
188 {   6}              |   \-centerY() <int centerY () line:1700> [see 59]
189 {   5}              | +-makeMessageFromData()
                        | | <void makeMessageFromData () line:2494>
190 {   6}              | | +-strcpy()
191 {   6}              | | +-sprintf()
192 {   6}              | | +-centerX() <int centerX () line:1686> [see 54]
193 {   6}              | | \-centerY() <int centerY () line:1700> [see 59]
194 {   5}              | +-writeControlMessage()
                        | | <void writeControlMessage () line:2529>
195 {   6}              | | +-XGetWindowAttributes()
196 {   6}              | | +-strlen()
197 {   6}              | | +-GDrawImageString()
                        | | | <int GDrawImageString () line:1059> [see 145]
198 {   6}              | | \-centerX() <int centerX () line:1686> [see 54]
199 {   5}              | \-XFlush()
200 {   4}              \-XMapWindow()
201 {   3}            \-XSync()
202 {   2}        +-clearControlMessage()
                  | <void clearControlMessage () line:2876>
203 {   3}        | +-strcpy()
204 {   3}        | \-XClearArea()
205 {   2}        +-writeTitle() <void writeTitle () line:3984>
206 {   3}        | +-XGetWindowAttributes()
```

```
207 {   3}      | +-GSetForeground()
                | | <int GSetForeground () line:1548> [see 116]
208 {   3}      | +-XClearWindow()
209 {   3}      | +-strlen()
210 {   3}      | +-GDrawImageString()
                | | <int GDrawImageString () line:1059> [see 145]
211 {   3}      | \-centerX() <int centerX () line:1686> [see 54]
212 {   2}      +-XMapWindow()
213 {   2}      +-XSync()
214 {   2}      \-drawViewport()
215 {   1}   +-bsdSignal()
216 {   1}   +-goodbye() <void goodbye () line:3952>
217 {   2}      +-fprintf()
218 {   2}      +-PSClose() <int PSClose () line:1673>
219 {   3}      | +-free()
220 {   3}      | \-unlink()
221 {   2}      +-XFreeGC()
222 {   2}      +-XFreeFont()
223 {   2}      +-XFreeColormap()
224 {   2}      +-check()
225 {   2}      +-write()
226 {   2}      +-close()
227 {   2}      +-XCloseDisplay()
228 {   2}      \-exit()
229 {   1}   \-processEvents() <void processEvents () line:3380>
230 {   2}      +-ConnectionNumber()
231 {   2}      +-malloc()
232 {   2}      +-fprintf()
233 {   2}      +-exitWithAck()
234 {   2}      +-RootWindow()
235 {   2}      +-FD_ZERO()
236 {   2}      +-FD_SET()
237 {   2}      +-XEventsQueued()
238 {   2}      +-select()
239 {   2}      +-FD_ISSET()
240 {   2}      +-XPending()
241 {   2}      +-XNextEvent()
242 {   2}      +-goodbye() <void goodbye () line:3952> [see 216]
243 {   2}      +-XCheckWindowEvent()
244 {   2}      +-writeTitle() <void writeTitle () line:3984> [see 205]
245 {   2}      +-XGetWindowAttributes()
246 {   2}      +-XResizeWindow()
247 {   2}      +-XSync()
248 {   2}      +-drawViewport()
249 {   2}      +-XMapWindow()
250 {   2}      +-drawControlPanel()
                | <void drawControlPanel () line:2542> [see 115]
251 {   2}      +-XCheckMaskEvent()
252 {   2}      +-getPotValue() <mouseCoord getPotValue () line:3016>
253 {   2}      +-sprintf()
```

```
254 {   2}      +-projX()
255 {   2}      +-projY()
256 {   2}      +-writeControlMessage()
                | <void writeControlMessage () line:2529> [see 194]
257 {   2}      +-XFlush()
258 {   2}      +-XUnmapWindow()
259 {   2}      +-putControlPanelSomewhere()
                | <void putControlPanelSomewhere () line:2859> [see 111]
260 {   2}      +-XLookUpAssoc() <int *XLookUpAssoc () line:1865>
261 {   3}        \-hash_find()
262 {   2}      +-clearControlMessage()
                | <void clearControlMessage () line:2876> [see 202]
263 {   2}      +-drawControlPushButton()
                | <void drawControlPushButton () line:3120>
264 {   3}        +-GDrawPushButton()
                  | <int GDrawPushButton () line:1285> [see 176]
265 {   3}        +-monoColor()
266 {   3}        \-XSync()
267 {   2}      +-centerX() <int centerX () line:1686> [see 54]
268 {   2}      +-centerY() <int centerY () line:1700> [see 59]
269 {   2}      +-buttonAction() <void buttonAction () line:3134>
270 {   3}      | +-sprintf()
271 {   3}      | +-writeControlMessage()
                | | <void writeControlMessage () line:2529> [see 194]
272 {   3}      | +-XSync()
273 {   3}      | +-strcpy()
274 {   3}      | +-drawControlPushButton()
                | | <void drawControlPushButton () line:3120> [see 263]
275 {   3}      | +-drawViewport()
276 {   3}      | +-PSInit() <int PSInit () line:1469>
277 {   4}      |   +-fprintf()
278 {   4}      |   +-sprintf()
279 {   4}      |   +-tmpnam()
280 {   4}      |   \-GdrawsSetDimension()
                |     | <int GdrawsSetDimension () line:1025>
281 {   5}      |     +-fopen()
282 {   5}      |     +-XGetWindowAttributes()
283 {   5}      |     +-fprintf()
284 {   5}      |     \-fclose()
285 {   3}      | +-PSCreateFile() <int PSCreateFile () line:956>
286 {   4}      |   +-fopen()
287 {   4}      |   +-fprintf()
288 {   4}      |   +-fclose()
289 {   4}      |   +-filecopy() <void filecopy () line:949>
290 {   5}      |   | +-getc()
291 {   5}      |   | \-putc()
292 {   4}      |   \-unlink()
293 {   3}      | +-clearControlMessage()
                | | <void clearControlMessage () line:2876> [see 202]
294 {   3}      | +-strcat()
```

```
295 {   3}     | +-XUnmapWindow()
296 {   3}     | +-clickedOnGraphSelect()
               | | <void clickedOnGraphSelect () line:3070> (R)
297 {   4}     |   +-doPick() <void doPick () line:3027>
298 {   5}     |     +-check()
299 {   5}     |     +-write()
300 {   5}     |     +-sprintf()
301 {   5}     |     \-writeControlMessage()
               |       <void writeControlMessage () line:2529> [see 194]
302 {   4}     |   +-doDrop() <void doDrop () line:3043> (R)
303 {   5}     |     +-check()
304 {   5}     |     +-write()
305 {   5}     |     +-readViewman() <int readViewman () line:3743> [see 65]
306 {   5}     |     +-sprintf()
307 {   5}     |     +-writeControlMessage()
               |     | <void writeControlMessage () line:2529> [see 194]
308 {   5}     |     +-freeGraph() <void freeGraph () line:2966>
309 {   6}     |       \-free()
310 {   5}     |     +-getGraphFromViewman()
               |     | <void getGraphFromViewman () line:2882> [see 69]
311 {   5}     |     +-clickedOnGraph()
               |     | <void clickedOnGraph () line:3687> (R)
312 {   6}     |       +-doPick() <void doPick () line:3027> [see 297]
313 {   6}     |       +-doDrop() <void doDrop () line:3043>
               |       | (recursive: see 302) [see 302]
314 {   6}     |       +-makeMessageFromData()
               |       | <void makeMessageFromData () line:2494> [see 189]
315 {   6}     |       +-writeControlMessage()
               |       | <void writeControlMessage () line:2529> [see 194]
316 {   6}     |       +-GSetForeground()
               |       | <int GSetForeground () line:1548> [see 116]
317 {   6}     |       +-GSetBackground()
               |       | <int GSetBackground () line:1578> [see 167]
318 {   6}     |       +-GDrawImageString()
               |       | <int GDrawImageString () line:1059> [see 145]
319 {   6}     |       +-centerX() <int centerX () line:1686> [see 54]
320 {   6}     |       +-centerY() <int centerY () line:1700> [see 59]
321 {   6}     |       +-GDrawString()
               |       | <int GDrawString () line:1310> [see 153]
322 {   6}     |       \-drawViewport()
323 {   5}     |     \-clickedOnGraphSelect()
               |       <void clickedOnGraphSelect () line:3070>
               |       (recursive: see 296) [see 296]
324 {   4}     |   +-makeMessageFromData()
               |   | <void makeMessageFromData () line:2494> [see 189]
325 {   4}     |   +-writeControlMessage()
               |   | <void writeControlMessage () line:2529> [see 194]
326 {   4}     |   +-GSetForeground()
               |   | <int GSetForeground () line:1548> [see 116]
327 {   4}     |   +-strcpy()
```

```
328 {   4}    |   +-strlen()
329 {   4}    |   +-GSetBackground()
              |   | <int GSetBackground () line:1578> [see 167]
330 {   4}    |   +-GDrawImageString()
              |   | <int GDrawImageString () line:1059> [see 145]
331 {   4}    |   +-centerX() <int centerX () line:1686> [see 54]
332 {   4}    |   +-centerY() <int centerY () line:1700> [see 59]
333 {   4}    |   +-GSetLineAttributes()
              |   | <int GSetLineAttributes () line:1608> [see 123]
334 {   4}    |   \-GDrawLine() <int GDrawLine () line:1124> [see 129]
335 {   3}    | \-clickedOnGraph()
              |   <void clickedOnGraph () line:3687> (R) [see 311]
336 {   2}    \-spadAction() <int spadAction () line:3751>
337 {   3}      +-close()
338 {   3}      +-readViewman() <int readViewman () line:3743> [see 65]
339 {   3}      +-XUnmapWindow()
340 {   3}      +-putControlPanelSomewhere()
              | <void putControlPanelSomewhere () line:2859> [see 111]
341 {   3}      +-writeTitle() <void writeTitle () line:3984> [see 205]
342 {   3}      +-writeControlTitle()
              | <void writeControlTitle () line:2483> [see 141]
343 {   3}      +-XFlush()
344 {   3}      +-sprintf()
345 {   3}      +-writeViewport() <int writeViewport () line:4497>
346 {   4}      | +-XGetWindowAttributes()
347 {   4}      | +-sprintf()
348 {   4}      | +-system()
349 {   4}      | +-fprintf()
350 {   4}      | +-fopen()
351 {   4}      | +-perror()
352 {   4}      | +-centerX() <int centerX () line:1686> [see 54]
353 {   4}      | +-centerY() <int centerY () line:1700> [see 59]
354 {   4}      | +-fclose()
355 {   4}      | +-write_pixmap_file()
356 {   4}      | +-XWriteBitmapFile()
357 {   4}      | +-XResizeWindow()
358 {   4}      | +-drawViewport()
359 {   4}      | +-writeTitle() <void writeTitle () line:3984> [see 205]
360 {   4}      | +-PSInit() <int PSInit () line:1469> [see 276]
361 {   4}      | \-PSCreateFile() <int PSCreateFile () line:956> [see 285]
362 {   3}      +-fprintf()
363 {   3}      +-check()
364 {   3}      +-write()
365 {   3}      +-goodbye() <void goodbye () line:3952> [see 216]
366 {   3}      +-buttonAction() <void buttonAction () line:3134> [see 269]
367 {   3}      +-clickedOnGraph()
              | <void clickedOnGraph () line:3687> (R) [see 311]
368 {   3}      +-centerX() <int centerX () line:1686> [see 54]
369 {   3}      +-centerY() <int centerY () line:1700> [see 59]
370 {   3}      +-XMoveWindow()
```

```
371 {   3}          +-XSync()
372 {   3}          +-XResizeWindow()
373 {   3}          +-writeControlMessage()
                    | <void writeControlMessage () line:2529> [see 194]
374 {   3}          +-freeGraph() <void freeGraph () line:2966> [see 308]
375 {   3}          +-getGraphFromViewman()
                    | <void getGraphFromViewman () line:2882> [see 69]
376 {   3}          \-clickedOnGraphSelect()
                      <void clickedOnGraphSelect () line:3070> (R) [see 296]
```

6.2   Constants and Headers

6.2.1   System includes

⟨*view2d*⟩≡

```
#include <limits.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <X11/X.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xresource.h>
#include <setjmp.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/time.h>
#ifdef RIOSplatform
#include <sys/select.h>
#endif
```

6.2.2   local includes

⟨*view2d*⟩+≡
```
#include "bsdsignal.h"
#include "bsdsignal.h1"
#include "hash.h"
#include "hash.h1"
#include "pixmap.h1"
#include "util.h1"
#include "xspadfill.h1"
```

⟨*include/actions.h*⟩
⟨*include/g.h*⟩
⟨*include/nox10.h*⟩
⟨*include/override.h*⟩
⟨*include/view.h*⟩
⟨*include/viewcommand.h*⟩
⟨*include/view2d.h*⟩
⟨*include/write.h*⟩
⟨*include/xdefs.h*⟩

6.2.3   static variables

⟨*view2d*⟩+≡
```
static void drawControlPushButton(int  , int );

static int doit=0;  /* globish variable for picking/dropping/clearing */
     /* - all sorts of 2 button sequence events (command & graph #). */
```

6.2.4   structs

⟨*view2d*⟩+≡
```
typedef struct _buttonStruct {
  int buttonKey, pot, mask, graphNum, graphSelect;
  short buttonX,buttonY,buttonWidth,buttonHeight,xHalf,yHalf;
  Window self;
  char text[40];
  int textColor, textHue, textShade;
} buttonStruct;
```

⟨*view2d*⟩+≡
```
typedef struct _controlPanelStruct {
  int                 numOfButtons;
  Window              controlWindow,messageWindow,colormapWindow;
  char                message[40];
  struct _buttonStruct buttonQueue[maxButtons2D];
} controlPanelStruct;
```

⟨*view2d*⟩+≡
```
typedef struct _mouseCoord {
    float x,y;
} mouseCoord;
```

⟨*view2d*⟩+≡
```
typedef struct _viewPoints {
  int                 viewportKey;
  char                title[80];
  Window              viewWindow,titleWindow;
  controlPanelStruct  *controlPanel;
  int                 justMadeControl,haveControl,
                      axesOn,unitsOn,pointsOn,linesOn,splineOn,closing,
                      allowDraw;
  struct _viewPoints  *prevViewport,*nextViewport;
} viewPoints;
```

⟨*view2d*⟩+≡
```
typedef struct _controlXY {
  int putX,putY;
} controlXY;
```

⟨*view2d*⟩+≡
```
typedef struct _xPointStruct {
  XPoint *xPoint;
  Vertex *x10Point;
  XArc   *arc;
} xPointStruct;
```

6.2.5   defines

⟨*view2d*⟩+≡

```
#define numBits (8*sizeof(int))

/* for xdefs */
#define view2d

#define bColor          98
#define graphColor      138

#define NotPoint        (SHRT_MAX)
#define eqNANQ(x)       (x == NotPoint)

#define mouseBitmap_width 16
#define mouseBitmap_height 16
#define mouseBitmap_x_hot 8
#define mouseBitmap_y_hot 0
static char mouseBitmap_bits[] = {
   0x00, 0x01, 0x00, 0x01, 0x80, 0x02, 0x40, 0x04, 0xc0, 0x06, 0x20, 0x08,
   0x20, 0x08, 0x30, 0x18, 0x50, 0x14, 0x58, 0x34, 0x90, 0x12, 0x20, 0x08,
   0xc0, 0x47, 0x00, 0x21, 0x80, 0x10, 0x00, 0x0f};
#define mouseMask_width 16
#define mouseMask_height 16
static char mouseMask_bits[] = {
   0x00, 0x01, 0x00, 0x01, 0x80, 0x03, 0xc0, 0x07, 0xc0, 0x07, 0xe0, 0x0f,
   0xe0, 0x0f, 0xf0, 0x1f, 0xf0, 0x1f, 0xf8, 0x3f, 0xf0, 0x1f, 0xe0, 0x0f,
   0xc0, 0x47, 0x00, 0x21, 0x80, 0x10, 0x00, 0x0f};

#define spadBitmap_width 34
#define spadBitmap_height 20
#define spadBitmap_x_hot 15
#define spadBitmap_y_hot 10
static char spadBitmap_bits[] = {
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x80, 0x00, 0x00, 0x00, 0x00, 0xc0, 0x01, 0x00,
   0x00, 0x00, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xf8,
   0xe6, 0xf8, 0x76, 0x00, 0x84, 0x98, 0x44, 0x49, 0x00, 0xc0, 0x98, 0x42,
   0x49, 0x00, 0xb8, 0x98, 0x42, 0x49, 0x00, 0x84, 0x95, 0x42, 0x49, 0x00,
   0x44, 0xa5, 0x22, 0x49, 0x00, 0x78, 0x63, 0x1d, 0xdb, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x80, 0x00, 0x00, 0x00, 0x00, 0xc0, 0x01, 0x00,
   0x00, 0x00, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00};

#define spadMask_width 34
#define spadMask_height 20
```

```
#define spadMask_x_hot 15
#define spadMask_y_hot 10
static char spadMask_bits[] = {
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xc0,
   0x01, 0x00, 0x00, 0x00, 0xe0, 0x03, 0x00, 0x00, 0x00, 0xe0, 0x03, 0x00,
   0x00, 0x00, 0xe0, 0x03, 0x00, 0x00, 0xfc, 0xff, 0xff, 0xff, 0x01, 0xfe,
   0xff, 0xff, 0xff, 0x01, 0xfe, 0xff, 0xff, 0xff, 0x01, 0xfe, 0xff, 0xff,
   0xff, 0x01, 0xfe, 0xff, 0xff, 0xff, 0x01, 0xfe, 0xff, 0xff, 0xff, 0x01,
   0xfe, 0xff, 0xff, 0xff, 0x01, 0xfe, 0xff, 0xff, 0xff, 0x01, 0xfc, 0xff,
   0xff, 0xff, 0x01, 0x00, 0xe0, 0x03, 0x00, 0x00, 0x00, 0xe0, 0x03, 0x00,
   0x00, 0x00, 0xe0, 0x03, 0x00, 0x00, 0x00, 0xc0, 0x01, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00};

/* Defines the pixmap for the arrow displayed in the scale window */

#define scaleArrowN 11
static XPoint scaleArrow[scaleArrowN] = {
  {55,14},{64,23},{59,23},
  {66,45},{79,45},
  {55,69},
  {31,45},{44,45},
  {51,23},{46,23},{55,14} };

/* Defines the pixmap for the arrows displayed in the translate window */

#define translateArrowN 25
static XPoint translateArrow[translateArrowN] = {
  {55,2},{60,10},{58,10},{58,37},
  {85,37},{85,35},{93,40},{85,45},{85,43},{58,43},
  {58,70},{60,70},{55,78},{50,70},{52,70},{52,43},
  {25,43}, {25,45}, {17,40},  {25,35}, {25,37}, {52,37},
  {52,10},{50,10},{55,2} };


#define carefullySetFont(gc,font) if (font != serverFont) XSetFont(dsply,gc,font-

#define controlMASK (ButtonPressMask + ExposureMask)
#define potMASK      (ButtonPressMask + ButtonReleaseMask + ButtonMotionMask + Lea
#define buttonMASK  (ButtonPressMask + ButtonReleaseMask + LeaveWindowMask)
#define colorMASK   (ButtonPressMask + ButtonReleaseMask + LeaveWindowMask)

/* make mouse grab for stationery mouse on a potentiometer slower */
#define mouseWait 50

#define controlCreateMASK (CWBackPixel | CWBorderPixel | CWEventMask | CWCursor |
#define buttonCreateMASK    CWEventMask
```

```
#define messageCreateMASK    0
#define colormapCreateMASK   CWEventMask

#define controlWidth   236
#define controlHeight  400

#define closeLeft cp->buttonQueue[closeAll2D].buttonX - 5
#define closeTop  cp->buttonQueue[closeAll2D].buttonY - 5

#define controlBackground WhitePixel(dsply,scrn)
#define controlCursorForeground monoColor(4)
#define controlCursorBackground monoColor(44)
#define controlTitleColor        monoColor(36)
#define controlPotHeaderColor    monoColor(52)
#define controlColorColor        monoColor(13)
#define controlColorSignColor    monoColor(22)

#define controlMessageHeight globalFont->max_bounds.ascent + globalFont->max_bounds.descent
#define messageBot controlMessageY + controlMessageHeight

#define headerHeight headerFont->max_bounds.ascent
#define graphHeaderHeight messageBot + headerHeight

#define graphBarTop    graphHeaderHeight + 12
#define graphBarLeft   66
#define graphBarWidth  graphFont->max_bounds.width
#define graphBarHeight graphFont->max_bounds.ascent + graphFont->max_bounds.descent

#define colormapX 10
#define colormapY 235
#define colormapW 280
#define colormapH 60

#define colorWidth   8
#define colorHeight  12

#define colorOffset  3
#define colorOffsetX 24
#define colorOffsetY 20
#define colorPointer 18

#define buttonColor monoColor(105)

#define graphBarDefaultColor    monoColor(15)
#define graphBarShowingColor    monoColor(15)
#define graphBarHiddenColor     monoColor(138)
```

```
#define graphBarSelectColor      monoColor(15)
#define graphBarNotSelectColor  monoColor(138)

#define rint(z) ((int)(z))

#define viewportCreateMASK (CWBackPixel|CWBorderPixel|CWEventMask|CWCursor|CWColo
#define viewportTitleCreateMASK (CWBackPixel|CWBorderPixel|CWCursor|CWColormap|CW

#define viewportMASK (KeyPressMask + ButtonPressMask + ExposureMask)
#define titleMASK    ExposureMask

#define lineWidth  1
#define lineHeight 1

#define titleColor      monoColor(36)
#define titleHeight     24
#define appendixHeight 0

#define viewWidth  400
#define viewHeight 400

#define viewBorderWidth 0
#define borderWidth     22
#define borderHeight    45

#define initDeltaX 0.0
#define initDeltaY 0.0
#define initScale  1.3

#define minScale  0.01
#define maxScale  1000.0
#define maxDelta  1000.0

#define scaleFactor     0.5
#define translateFactor 10

#define viewCursorForeground monoColor(166)
#define viewCursorBackground monoColor(5)

#define axisLength 100.0

#define axesColorDefault  35
#define labelColor 22

#define meshOutline      monoColor(132)
#define opaqueOutline    monoColor(53)
```

```
#define opaqueForeground monoColor(236)

#define drawWireFrame 0
#define drawOpaque    1
#define drawRendered  2


#define numOfColors 240

#define totalHuesConst   27
#define totalShadesConst 8
#define hueEnd 360
#define hueStep hueEnd/totalHuesConst

#define numPlanes 1
#define numColors 10
#define startColor 0
#define endColor   startColor+numColors
#define maxColors DisplayCells(dsply,scrn)-1

#define colorStep (maxColors+1)/numColors



#define physicalWidth  DisplayWidth(dsply,scrn)
#define physicalHeight DisplayHeight(dsply,scrn)
#define deep           DisplayPlanes(dsply,scrn)

#define basicScreen 19

#define yes 1
#define no  0

#define potA  25   /* line dividing potentiometers from stuff above it */
#define potB 173   /* line dividing potentiometers from title */
#define butA 260   /* line dividing buttons from stuff above it */

#define controlMessageY      181
#define controlMessageColor monoColor(29)

#define pi_half          1.5707963267948966192313216916397514420999
#define pi               3.1415926535897932384626433832795028841971
#define three_pi_halves 4.7123889803846898576939650749192543262966
#define two_pi          6.2831853071795864769252867665590057683994

#define degrees_in_two_pi 57
#define d2Pi 57
```

```
#define viewBackground 0

#define nbuckets         128

#define anywhere 0


#define intSize sizeof(int)
#define floatSize sizeof(float)

/* type is X, PS,... */

#define drawViewport(type) drawTheViewport(type);
#define spadDrawViewport() spadMode++; drawTheViewport(X); spadMode--;

#define calcUnitX(ii) (vwInfo.width * \
                       ((graphArray[0].unitX * ii + \
                        graphArray[0].originX - graphStateArray[0].centerX) *\
                        graphStateArray[0].scaleX + 0.5))
#define calcUnitY(ii) (vwInfo.height * aspectR * \
                       (1 - ((graphArray[0].unitY*aspectR * ii + \
                             graphArray[0].originY*aspectR - \
                             graphStateArray[0].centerY) * \
                            graphStateArray[0].scaleY + 0.5*aspectR )))

#define projX(x,w,i) ((((float)x/w-0.5)/graphStateArray[i].scaleX + \
                       graphStateArray[i].centerX + 0.5) /        \
                      graphArray[i].xNorm + graphArray[i].xmin)

#define projY(y,h,i) (((0.5-(float)y/h*aspectR)/graphStateArray[i].scaleY + \
                       graphStateArray[i].centerY + 0.5) /        \
                      graphArray[i].yNorm + graphArray[i].ymin)

#define isNaN(v) (v != v)
```

6.2.6 extern references

⟨*view2d*⟩+≡
```
extern Display       *dsply;
extern XFontStruct   *globalFont,*buttonFont,*headerFont,*titleFont,
                     *graphFont,*unitFont,*serverFont;
extern XrmDatabase     rDB;
extern char          scaleXReport[5],scaleYReport[5],deltaXReport[5],
                     deltaYReport[5];
extern unsigned long *spadColors;
extern int           followMouse,viewportKeyNum;
extern Window        rtWindow,viewman;
extern GC            globalGC1,globalGC2,anotherGC,globGC,trashGC,
                     controlMessageGC,graphGC,unitGC;
extern HashTable     *table;
extern Colormap      colorMap;
extern int           Socket,ack;
extern  GC  processGC;
extern  viewPoints *viewport;
extern  controlPanelStruct *control;
extern  XGCValues gcVals;
extern  char *s;
extern  int someInt;
extern unsigned long foregroundColor, backgroundColor;
extern int drawMore;
extern int spadMode,spadDraw;
extern jmp_buf jumpFlag;
extern graphStruct graphArray[maxGraphs];
extern graphStateStruct graphStateArray[maxGraphs],
                        graphStateBackupArray[maxGraphs];
extern xPointStruct xPointsArray[maxGraphs];
extern int pointsON, connectON, splineON, axesON, unitsON, zoomXON, zoomYON;
extern int transXON, transYON;
extern char errorStr[80];
extern int currentGraph;
extern int queriedGraph;
extern int picking,dropping;
extern char filename[256];
extern char *xDefault;
extern int viewAloned;
extern int mono, totalColors,
           totalHues, totalSolidShades, totalDitheredAndSolids,totalShades;
extern float aspectR;
```

6.2.7   forward references

⟨*view2d*⟩+≡

```
  extern int initButtons(buttonStruct * );
  extern void writeControlTitle(void );
  extern void makeMessageFromData(int );
  extern void writeControlMessage(void );
  extern void drawControlPanel(void );
  extern controlXY getControlXY(int );
  extern controlPanelStruct * makeControlPanel(void );
  extern void putControlPanelSomewhere(int );
  extern void clearControlMessage(void );
  extern void getGraphFromViewman(int );
  extern void freeGraph(int );
  extern int main(void);
  extern void mergeDatabases(void);
  extern mouseCoord getPotValue(short  , short  , short  , short );
  extern void doPick(int   , int );
  extern void doDrop(int   , int );
  extern void clickedOnGraph(int   , int );
  extern void buttonAction(int );
  extern void processEvents(void);
  extern void clickedOnGraphSelect(int   , int );
  extern int readViewman(void *   , int );
  extern int spadAction(void);
  extern float absolute(float);
  extern void goodbye(int);
  extern void writeTitle(void);
  extern void drawTheViewport(int );
  extern viewPoints * makeViewport(char *  , int  , int  , int  , int  , int );
  extern viewPoints * makeView2D(view2DStruct * );
  extern int writeViewport(int );
  extern int PSCreateFile(int , Window , Window , char * );
  extern int GdrawsDrawFrame(int , Window , Window , char * );
  extern int GdrawsSetDimension(Window , Window );
  extern int GDrawImageString(GC , Window , int , int , char * , int , int );
  extern int GDrawArc(GC , Window , int , int , unsigned int , unsigned int , int ,
  extern int GDrawLine(GC , Window , int , int , int , int , int );
  extern int GDrawLines(GC , Window , XPoint * , int , int , int );
  extern int GDrawPoint(Window , GC , int , int , int );
  extern int GDrawString(GC , Window , int , int , char * , int , int );
  extern int GFillArc(GC , Window , int , int , unsigned int , unsigned int , int ,
  extern int PSGlobalInit(void );
  extern int PSInit(Window , Window );
  extern int PSCreateContext(GC , char * , int , int , int , float , float );
  extern char * PSfindGC(GC );
```

```
extern int GSetForeground(GC , float , int );
extern int GSetBackground(GC , float , int );
extern int GSetLineAttributes(GC , int , int , int , int , int );
extern int PSClose(void );
extern int centerX(GC , char * , int , int );
extern int centerY(GC , int );
extern int PSColorPolygon(float , float , float , XPoint * , int );
extern int PSColorwOutline(float , float , float , XPoint * , int );
extern int PSDrawColor(float , float , float , XPoint * , int );
extern int PSFillPolygon(GC , XPoint * , int );
extern int PSFillwOutline(GC , XPoint * , int );
extern HashTable * XCreateAssocTable(int );
extern void XMakeAssoc(Display * , HashTable * , Window , int * );
extern int * XLookUpAssoc(Display * , HashTable * , Window );
extern void XDeleteAssoc(Display * , HashTable * , Window );
extern int GDrawRectangle(GC , Window , short , short , short , short , int );
extern int GDraw3DButtonOut(GC , Window , short , short , short , short , int );
extern int GDraw3DButtonIn(GC , Window , short , short , short , short , int );
extern int GDrawPushButton(Display * , GC , GC , GC , Window , short , short , short , short
#ifdef _GFUN_C
static void filecopy(FILE * , FILE * );
static int TrivEqual(Window , Window );
static int TrivHashCode(Window , int );
#endif
```

6.2.8   global variables

⟨*view2d*⟩+≡
```
  Window        rtWindow,viewman;
  Display       *dsply;
  XFontStruct   *globalFont,
    *buttonFont,
    *headerFont,
    *titleFont,
    *graphFont,
    *unitFont,
    *serverFont;
  GC            globalGC1,
    globalGC2,
    anotherGC,
    globGC,
    trashGC,
    controlMessageGC,
    graphGC,
    unitGC,
    processGC;
  XGCValues     gcVals;
  HashTable     *table;
  Colormap      colorMap;

  Atom wm_delete_window;

  XrmDatabase rDB; /* Resource database */


  char          scaleXReport[5],
    scaleYReport[5],
    deltaXReport[5],
    deltaYReport[5],
    *s,
    errorStr[80],
    filename[256], /** For writing viewport data out to a file **/
    *xDefault;     /** used for accessing .XDefaults **/


  unsigned long *spadColors;
  unsigned long foregroundColor, backgroundColor;

  int           followMouse = no,
    viewportKeyNum = 0,
    scrn,
```

```
      Socket = 1,
      ack = 1,
      someInt,
      drawMore,
      spadMode=no, /* yes if receiving Axiom command and calling drawViewport */
      spadDraw=no, /* yes if drawing viewport because of a Axiom command */
      pointsON  = yes,  /* these would affect the choices in buttons.c */
      connectON = yes,
      splineON  = no,
      axesON    = yes,
      unitsON   = no,
      zoomXON   = yes,
      zoomYON   = yes,
      transXON  = yes,
      transYON  = yes,
      currentGraph = 0,  /* last graph selected */
      queriedGraph = 0,  /* current graph queried */
      picking=0,
      dropping=0,
      viewAloned,  /** if not connected to Axiom **/
      mono,
      totalColors,
      totalSolid,
      totalDithered,
      maxGreyShade,
      totalHues,
      totalSolidShades,
      totalDitheredAndSolids,
      totalShades;
  /* totalShades is initially set to totalShadesConst (probably 8).
     If X cannot allocate 8 shades for each hue, totalShades is
     decremented. There is currently only a check for this value
     to be positive. ---> something to add: change over to monochrome
     if totalShades=0. Just modify the spadcolors.c file.
     spadcolors.c has been modified so that it returns the value for
     totalShades. Since the return value had previously been unused,
     a modification in this way ensures continued support of other
     routines calling this function (e.g. hyperDoc stuff). */

  viewPoints     *viewport;
  controlPanelStruct *control;
  jmp_buf jumpFlag;
  graphStruct       graphArray[maxGraphs];
  graphStateStruct graphStateArray[maxGraphs],
    graphStateBackupArray[maxGraphs];
  xPointStruct      xPointsArray[maxGraphs];
```

```
float aspectR = 1.0;

/* global ps variables */
int     psInit=no;      /* need to call globalInitPs() each run */
GCptr   GChead=NULL;    /* ptr to head of ps GC linked list */
char    *PSfilename;    /* output file name used in user directory */
char *envAXIOM;    /* environment variable AXIOM or DEVE */
```

6.3 Code

6.3.1 initButtons

Creates the fields for each button window in the two dimensional control panel, and returns the number of buttons created.

⟨*view2d*⟩+≡

```
  ⟨gfun.c⟩
  int initButtons(buttonStruct *buttons) {
    int ii, num = 0;
  /*********** Scale(Zoom) and Translate Potentiometer Buttons **********/
    /* Title:  "Scale" */
    ii = scale2D;
    buttons[ii].buttonX      = 5;
    buttons[ii].buttonY      = 85;
    buttons[ii].buttonWidth  = 110;
    buttons[ii].buttonHeight = 80;
    buttons[ii].buttonKey    = ii;
    buttons[ii].pot          = yes;          /* scale is a potentiometer */
    buttons[ii].graphNum     = no;
    buttons[ii].graphSelect  = no;
    buttons[ii].mask         = potMASK;
    buttons[ii].textColor    = 164;
    buttons[ii].xHalf        = buttons[ii].buttonWidth/2;
    buttons[ii].yHalf        = buttons[ii].buttonHeight/2;
    ++num;
    /* Title:  "Translate" */
    ii = translate2D;
    buttons[ii].buttonX      = 121;
    buttons[ii].buttonY      = 85;
    buttons[ii].buttonWidth  = 110;
    buttons[ii].buttonHeight = 80;
    buttons[ii].buttonKey    = ii;
    buttons[ii].pot          = yes;          /* translate is a potentiometer */
    buttons[ii].graphNum     = no;
    buttons[ii].graphSelect  = no;
    buttons[ii].mask         = potMASK;
    buttons[ii].textColor    = 21;           /* line color of translate arrow */
    buttons[ii].xHalf        = buttons[ii].buttonWidth/2;
    buttons[ii].yHalf        = buttons[ii].buttonHeight/2;
    ++num;
    /* Scale potentiometer buttons */
    /* Scale along X axis */
    ii = zoom2Dx;
    buttons[ii].buttonX      = 5;
```

```
buttons[ii].buttonY       = 55;
buttons[ii].buttonWidth  = 53;
buttons[ii].buttonHeight = 25;
buttons[ii].buttonKey     = ii;
buttons[ii].pot           = no;
buttons[ii].graphNum      = no;
buttons[ii].graphSelect  = no;
buttons[ii].mask          = buttonMASK;
strcpy(buttons[ii].text,"X On ");
buttons[ii].textColor     = bColor;
buttons[ii].xHalf         = buttons[ii].buttonWidth/2;
buttons[ii].yHalf         = buttons[ii].buttonHeight/2;
++num;
/* Scale along Y axis */
ii = zoom2Dy;
buttons[ii].buttonX       = 62;
buttons[ii].buttonY       = 55;
buttons[ii].buttonWidth  = 53;
buttons[ii].buttonHeight = 25;
buttons[ii].buttonKey     = ii;
buttons[ii].pot           = no;
buttons[ii].graphNum      = no;
buttons[ii].graphSelect  = no;
buttons[ii].mask          = buttonMASK;
strcpy(buttons[ii].text,"Y On ");
buttons[ii].textColor     = bColor;
buttons[ii].xHalf         = buttons[ii].buttonWidth/2;
buttons[ii].yHalf         = buttons[ii].buttonHeight/2;
++num;
/* Translate along X axis */
ii = translate2Dx;
buttons[ii].buttonX       = 121;
buttons[ii].buttonY       = 55;
buttons[ii].buttonWidth  = 53;
buttons[ii].buttonHeight = 25;
buttons[ii].buttonKey     = ii;
buttons[ii].pot           = no;
buttons[ii].graphNum      = no;
buttons[ii].graphSelect  = no;
buttons[ii].mask          = buttonMASK;
strcpy(buttons[ii].text,"X On ");
buttons[ii].textColor     = bColor;
buttons[ii].xHalf         = buttons[ii].buttonWidth/2;
buttons[ii].yHalf         = buttons[ii].buttonHeight/2;
++num;
/* Translate along Y axis */
```

```
ii = translate2Dy;
buttons[ii].buttonX      = 179;
buttons[ii].buttonY      = 55;
buttons[ii].buttonWidth  = 53;
buttons[ii].buttonHeight = 25;
buttons[ii].buttonKey    = ii;
buttons[ii].pot          = no;
buttons[ii].graphNum     = no;
buttons[ii].graphSelect  = no;
buttons[ii].mask         = buttonMASK;
strcpy(buttons[ii].text,"Y On ");
buttons[ii].textColor    = bColor;
buttons[ii].xHalf        = buttons[ii].buttonWidth/2;
buttons[ii].yHalf        = buttons[ii].buttonHeight/2;
++num;
/* Axes Turned On/Off */
ii = axesOnOff2D;
buttons[ii].buttonX      = 5;
buttons[ii].buttonY      = 292;
buttons[ii].buttonWidth  = 90;
buttons[ii].buttonHeight = 30;
buttons[ii].buttonKey    = ii;
buttons[ii].pot          = no;
buttons[ii].graphNum     = no;
buttons[ii].graphSelect  = no;
buttons[ii].mask         = buttonMASK;
strcpy(buttons[ii].text,"Axes On ");
buttons[ii].textColor    = 75;
buttons[ii].textHue      = 10;
buttons[ii].textShade    = 3;
buttons[ii].xHalf        = buttons[ii].buttonWidth/2;
buttons[ii].yHalf        = buttons[ii].buttonHeight/2;
++num;
/* Units Turned On/Off */
ii = unitsOnOff2D;
buttons[ii].buttonX      = 100;
buttons[ii].buttonY      = 292;
buttons[ii].buttonWidth  = 90;
buttons[ii].buttonHeight = 30;
buttons[ii].buttonKey    = ii;
buttons[ii].pot          = no;
buttons[ii].graphNum     = no;
buttons[ii].graphSelect  = no;
buttons[ii].mask         = buttonMASK;
strcpy(buttons[ii].text,"Units Off");
buttons[ii].textColor    = 75;
```

```
buttons[ii].textHue      = 10;
buttons[ii].textShade    = 3;
buttons[ii].xHalf        = buttons[ii].buttonWidth/2;
buttons[ii].yHalf        = buttons[ii].buttonHeight/2;
++num;
/* Generate a Postscript file */
ii = ps2D;
buttons[ii].buttonX      = 195;
buttons[ii].buttonY      = 292;
buttons[ii].buttonWidth  = 36;
buttons[ii].buttonHeight = 30;
buttons[ii].buttonKey    = ii;
buttons[ii].pot          = no;
buttons[ii].graphNum     = no;
buttons[ii].graphSelect  = no;
buttons[ii].mask         = buttonMASK;
strcpy(buttons[ii].text,"PS");
buttons[ii].textColor    = 35;
buttons[ii].textHue      = 5;
buttons[ii].textShade    = 2;
buttons[ii].xHalf        = buttons[ii].buttonWidth/2;
buttons[ii].yHalf        = buttons[ii].buttonHeight/2;
++num;
/* Bounding Rectangle On/Off */
ii = spline2D;
buttons[ii].buttonX      = 5;
buttons[ii].buttonY      = 329;
buttons[ii].buttonWidth  = 66;
buttons[ii].buttonHeight = 30;
buttons[ii].buttonKey    = ii;
buttons[ii].pot          = no;
buttons[ii].graphNum     = no;
buttons[ii].graphSelect  = no;
buttons[ii].mask         = buttonMASK;
strcpy(buttons[ii].text,"Box Off");
buttons[ii].textColor    = 7;
buttons[ii].textHue      = 26;
buttons[ii].textShade    = 3;
buttons[ii].xHalf        = buttons[ii].buttonWidth/2;
buttons[ii].yHalf        = buttons[ii].buttonHeight/2;
++num;
/* Graph points On/Off */
ii = pointsOnOff;
buttons[ii].buttonX      = 75;
buttons[ii].buttonY      = 329;
buttons[ii].buttonWidth  = 67;
```

```
buttons[ii].buttonHeight = 30;
buttons[ii].buttonKey    = ii;
buttons[ii].pot          = no;
buttons[ii].graphNum     = no;
buttons[ii].graphSelect  = no;
buttons[ii].mask         = buttonMASK;
strcpy(buttons[ii].text,"Pts On ");
buttons[ii].textColor    = 7;
buttons[ii].textHue      = 26;
buttons[ii].textShade    = 3;
buttons[ii].xHalf        = buttons[ii].buttonWidth/2;
buttons[ii].yHalf        = buttons[ii].buttonHeight/2;
++num;
/* Graph lines On/Off */
ii = connectOnOff;
buttons[ii].buttonX      = 147;
buttons[ii].buttonY      = 329;
buttons[ii].buttonWidth  = 84;
buttons[ii].buttonHeight = 30;
buttons[ii].buttonKey    = ii;
buttons[ii].pot          = no;
buttons[ii].graphNum     = no;
buttons[ii].graphSelect  = no;
buttons[ii].mask         = buttonMASK;
strcpy(buttons[ii].text,"Lines On ");
buttons[ii].textColor    = 7;
buttons[ii].textHue      = 26;
buttons[ii].textShade    = 3;
buttons[ii].xHalf        = buttons[ii].buttonWidth/2;
buttons[ii].yHalf        = buttons[ii].buttonHeight/2;
++num;
/* Reset View Position Button */
ii = reset2D;
buttons[ii].buttonX      = 5;
buttons[ii].buttonY      = 364;
buttons[ii].buttonWidth  = 60;
buttons[ii].buttonHeight = 30;
buttons[ii].buttonKey    = ii;
buttons[ii].pot          = no;
buttons[ii].graphNum     = no;
buttons[ii].graphSelect  = no;
buttons[ii].mask         = buttonMASK;
strcpy(buttons[ii].text,"Reset");
buttons[ii].textColor    = bColor;
buttons[ii].textHue      = 5;
buttons[ii].textShade    = 2;
```

```
buttons[ii].xHalf         = buttons[ii].buttonWidth/2;
buttons[ii].yHalf         = buttons[ii].buttonHeight/2;
++num;
/* Hide Control Panel */
ii = hideControl2D;
buttons[ii].buttonX       = 70;
buttons[ii].buttonY       = 364;
buttons[ii].buttonWidth   = 88;
buttons[ii].buttonHeight  = 30;
buttons[ii].buttonKey     = ii;
buttons[ii].pot           = no;
buttons[ii].graphNum      = no;
buttons[ii].graphSelect   = no;
buttons[ii].mask          = buttonMASK;
strcpy(buttons[ii].text,"Hide Panel");
buttons[ii].textColor     = bColor;
buttons[ii].textHue       = 5;
buttons[ii].textShade     = 2;
buttons[ii].xHalf         = buttons[ii].buttonWidth/2;
buttons[ii].yHalf         = buttons[ii].buttonHeight/2;
++num;
/* Exits from the viewport running */
ii = closeAll2D;
buttons[ii].buttonX       = 169;
buttons[ii].buttonY       = 370;
buttons[ii].buttonWidth   = 61;
buttons[ii].buttonHeight  = 24;
buttons[ii].buttonKey     = ii;
buttons[ii].pot           = no;
buttons[ii].graphNum      = no;
buttons[ii].graphSelect   = no;
buttons[ii].mask          = buttonMASK;
strcpy(buttons[ii].text,"Quit");
buttons[ii].textColor     = 13;
buttons[ii].textHue       = 29;
buttons[ii].textShade     = 2;
buttons[ii].xHalf         = buttons[ii].buttonWidth/2;
buttons[ii].yHalf         = buttons[ii].buttonHeight/2;
++num;
/* Indicates that the graph from a viewport is to be picked up. */
ii = pick2D;
buttons[ii].buttonX       = 190;
buttons[ii].buttonY       = 217;
buttons[ii].buttonWidth   = 40;
buttons[ii].buttonHeight  = 24;
buttons[ii].buttonKey     = ii;
```

```
buttons[ii].pot          = no;
buttons[ii].graphNum     = no;
buttons[ii].graphSelect  = no;
buttons[ii].mask         = buttonMASK;
strcpy(buttons[ii].text,"Pick");
buttons[ii].textColor    = 123;
buttons[ii].textHue      = 19;
buttons[ii].textShade    = 3;
buttons[ii].xHalf        = buttons[ii].buttonWidth/2;
buttons[ii].yHalf        = buttons[ii].buttonHeight/2;
++num;
/* Indicates that the graph from a viewport is to be dropped into a slot. */
ii = drop2D;
buttons[ii].buttonX      = 190;
buttons[ii].buttonY      = 245;
buttons[ii].buttonWidth  = 40;
buttons[ii].buttonHeight = 24;
buttons[ii].buttonKey    = ii;
buttons[ii].pot          = no;
buttons[ii].graphNum     = no;
buttons[ii].graphSelect  = no;
buttons[ii].mask         = buttonMASK;
strcpy(buttons[ii].text,"Drop");
buttons[ii].textColor    = 123;
buttons[ii].textHue      = 19;
buttons[ii].textShade    = 3;
buttons[ii].xHalf        = buttons[ii].buttonWidth/2;
buttons[ii].yHalf        = buttons[ii].buttonHeight/2;
++num;
/* Indicates that the status of the graphs being displayed in the viewport
   is to be cleared. */
ii = clear2D;
buttons[ii].buttonX      = 5;
buttons[ii].buttonY      = 217;
buttons[ii].buttonWidth  = 49;
buttons[ii].buttonHeight = 24;
buttons[ii].buttonKey    = ii;
buttons[ii].pot          = no;
buttons[ii].graphNum     = no;
buttons[ii].graphSelect  = no;
buttons[ii].mask         = buttonMASK;
strcpy(buttons[ii].text,"Clear");
buttons[ii].textColor    = 123;
buttons[ii].textHue      = 19;
buttons[ii].textShade    = 3;
buttons[ii].xHalf        = buttons[ii].buttonWidth/2;
```

```
buttons[ii].yHalf          = buttons[ii].buttonHeight/2;
++num;
/* Asks for the scale and translation information for the specified graph. */
ii = query2D;
buttons[ii].buttonX        = 5;
buttons[ii].buttonY        = 245;
buttons[ii].buttonWidth    = 49;
buttons[ii].buttonHeight = 24;
buttons[ii].buttonKey      = ii;
buttons[ii].pot            = no;
buttons[ii].graphNum       = no;
buttons[ii].graphSelect    = no;
buttons[ii].mask           = buttonMASK;
strcpy(buttons[ii].text,"Query");
buttons[ii].textColor      = 123;
buttons[ii].textHue        = 19;
buttons[ii].textShade      = 3;
buttons[ii].xHalf          = buttons[ii].buttonWidth/2;
buttons[ii].yHalf          = buttons[ii].buttonHeight/2;
++num;
/* These buttons indicate the 9 available slot numbers into which
   a 2D graph can be placed, and the status of the graph, i.e. whether
   it is displayed or not. */
ii = graph1;
buttons[ii].buttonX        = graphBarLeft;
buttons[ii].buttonY        = graphBarTop;
buttons[ii].buttonWidth  = graphBarWidth;
buttons[ii].buttonHeight = graphBarHeight;
buttons[ii].buttonKey      = ii;
buttons[ii].pot            = no;
buttons[ii].graphNum       = yes;
buttons[ii].graphSelect    = no;
buttons[ii].mask           = buttonMASK;
strcpy(buttons[ii].text,"1");
buttons[ii].textColor      = graphColor;
buttons[ii].xHalf          = buttons[ii].buttonWidth/2;
buttons[ii].yHalf          = buttons[ii].buttonHeight/2;
++num;
ii = graphSelect1;
buttons[ii].buttonX        = graphBarLeft;
buttons[ii].buttonY        = graphBarTop + graphBarHeight;
buttons[ii].buttonWidth  = graphBarWidth;
buttons[ii].buttonHeight = graphBarHeight-2;
buttons[ii].buttonKey      = ii;
buttons[ii].pot            = no;                    /* this is a regular button */
buttons[ii].graphNum       = no;
```

```
buttons[ii].graphSelect  = yes;
buttons[ii].mask         = buttonMASK;
strcpy(buttons[ii].text,"^");
buttons[ii].textColor    = graphColor;
buttons[ii].xHalf        = buttons[ii].buttonWidth/2;
buttons[ii].yHalf        = buttons[ii].buttonHeight/2;
++num;
ii = graph2;
buttons[ii].buttonX      = graphBarLeft + (graphBarWidth);
buttons[ii].buttonY      = graphBarTop;
buttons[ii].buttonWidth  = graphBarWidth;
buttons[ii].buttonHeight = graphBarHeight;
buttons[ii].buttonKey    = ii;
buttons[ii].pot          = no;              /* this is a regular button */
buttons[ii].graphNum     = yes;
buttons[ii].graphSelect  = no;
buttons[ii].mask         = buttonMASK;
strcpy(buttons[ii].text,"2");
buttons[ii].textColor    = graphColor;
buttons[ii].xHalf        = buttons[ii].buttonWidth/2;
buttons[ii].yHalf        = buttons[ii].buttonHeight/2;
++num;
ii = graphSelect2;
buttons[ii].buttonX      = graphBarLeft + (graphBarWidth);
buttons[ii].buttonY      = graphBarTop + graphBarHeight;
buttons[ii].buttonWidth  = graphBarWidth;
buttons[ii].buttonHeight = graphBarHeight-2;
buttons[ii].buttonKey    = ii;
buttons[ii].pot          = no;              /* this is a regular button */
buttons[ii].graphNum     = no;
buttons[ii].graphSelect  = yes;
buttons[ii].mask         = buttonMASK;
strcpy(buttons[ii].text,"-");
buttons[ii].textColor    = graphColor;
buttons[ii].xHalf        = buttons[ii].buttonWidth/2;
buttons[ii].yHalf        = buttons[ii].buttonHeight/2;
++num;
ii = graph3;
buttons[ii].buttonX      = graphBarLeft + 2*(graphBarWidth);
buttons[ii].buttonY      = graphBarTop;
buttons[ii].buttonWidth  = graphBarWidth;
buttons[ii].buttonHeight = graphBarHeight;
buttons[ii].buttonKey    = ii;
buttons[ii].pot          = no;              /* this is a regular button */
buttons[ii].graphNum     = yes;
buttons[ii].graphSelect  = no;
```

```
buttons[ii].mask          = buttonMASK;
strcpy(buttons[ii].text,"3");
buttons[ii].textColor     = graphColor;
buttons[ii].xHalf         = buttons[ii].buttonWidth/2;
buttons[ii].yHalf         = buttons[ii].buttonHeight/2;
++num;
ii = graphSelect3;
buttons[ii].buttonX       = graphBarLeft + 2*(graphBarWidth);
buttons[ii].buttonY       = graphBarTop + graphBarHeight;
buttons[ii].buttonWidth   = graphBarWidth;
buttons[ii].buttonHeight  = graphBarHeight-2;
buttons[ii].buttonKey     = ii;
buttons[ii].pot           = no;   /**** blend these three together ***/
buttons[ii].graphNum      = no;
buttons[ii].graphSelect   = yes;
buttons[ii].mask          = buttonMASK;
strcpy(buttons[ii].text,"-");
buttons[ii].textColor     = graphColor;
buttons[ii].xHalf         = buttons[ii].buttonWidth/2;
buttons[ii].yHalf         = buttons[ii].buttonHeight/2;
++num;
ii = graph4;
buttons[ii].buttonX       = graphBarLeft + 3*(graphBarWidth);
buttons[ii].buttonY       = graphBarTop;
buttons[ii].buttonWidth   = graphBarWidth;
buttons[ii].buttonHeight  = graphBarHeight;
buttons[ii].buttonKey     = ii;
buttons[ii].pot           = no;                /* this is a regular button */
buttons[ii].graphNum      = yes;
buttons[ii].graphSelect   = no;
buttons[ii].mask          = buttonMASK;
strcpy(buttons[ii].text,"4");
buttons[ii].textColor     = graphColor;
buttons[ii].xHalf         = buttons[ii].buttonWidth/2;
buttons[ii].yHalf         = buttons[ii].buttonHeight/2;
++num;
ii = graphSelect4;
buttons[ii].buttonX       = graphBarLeft + 3*(graphBarWidth);
buttons[ii].buttonY       = graphBarTop + graphBarHeight;
buttons[ii].buttonWidth   = graphBarWidth;
buttons[ii].buttonHeight  = graphBarHeight-2;
buttons[ii].buttonKey     = ii;
buttons[ii].pot           = no;                /* this is a regular button */
buttons[ii].graphNum      = no;
buttons[ii].graphSelect   = yes;
buttons[ii].mask          = buttonMASK;
```

```
strcpy(buttons[ii].text,"-");
buttons[ii].textColor    = graphColor;
buttons[ii].xHalf        = buttons[ii].buttonWidth/2;
buttons[ii].yHalf        = buttons[ii].buttonHeight/2;
++num;
ii = graph5;
buttons[ii].buttonX      = graphBarLeft + 4*(graphBarWidth);
buttons[ii].buttonY      = graphBarTop;
buttons[ii].buttonWidth  = graphBarWidth;
buttons[ii].buttonHeight = graphBarHeight;
buttons[ii].buttonKey    = ii;
buttons[ii].pot          = no;                  /* this is a regular button */
buttons[ii].graphNum     = yes;
buttons[ii].graphSelect  = no;
buttons[ii].mask         = buttonMASK;
strcpy(buttons[ii].text,"5");
buttons[ii].textColor    = graphColor;
buttons[ii].xHalf        = buttons[ii].buttonWidth/2;
buttons[ii].yHalf        = buttons[ii].buttonHeight/2;
++num;
ii = graphSelect5;
buttons[ii].buttonX      = graphBarLeft + 4*(graphBarWidth);
buttons[ii].buttonY      = graphBarTop + graphBarHeight;
buttons[ii].buttonWidth  = graphBarWidth;
buttons[ii].buttonHeight = graphBarHeight-2;
buttons[ii].buttonKey    = ii;
buttons[ii].pot          = no;                  /* this is a regular button */
buttons[ii].graphNum     = no;
buttons[ii].graphSelect  = yes;
buttons[ii].mask         = buttonMASK;
strcpy(buttons[ii].text,"-");
buttons[ii].textColor    = graphColor;
buttons[ii].xHalf        = buttons[ii].buttonWidth/2;
buttons[ii].yHalf        = buttons[ii].buttonHeight/2;
++num;
ii = graph6;
buttons[ii].buttonX      = graphBarLeft + 5*(graphBarWidth);
buttons[ii].buttonY      = graphBarTop;
buttons[ii].buttonWidth  = graphBarWidth;
buttons[ii].buttonHeight = graphBarHeight;
buttons[ii].buttonKey    = ii;
buttons[ii].pot          = no;                  /* this is a regular button */
buttons[ii].graphNum     = yes;
buttons[ii].graphSelect  = no;
buttons[ii].mask         = buttonMASK;
strcpy(buttons[ii].text,"6");
```

```
buttons[ii].textColor    = graphColor;
buttons[ii].xHalf        = buttons[ii].buttonWidth/2;
buttons[ii].yHalf        = buttons[ii].buttonHeight/2;
++num;
ii = graphSelect6;
buttons[ii].buttonX      = graphBarLeft + 5*(graphBarWidth);
buttons[ii].buttonY      = graphBarTop + graphBarHeight;
buttons[ii].buttonWidth  = graphBarWidth;
buttons[ii].buttonHeight = graphBarHeight-2;
buttons[ii].buttonKey    = ii;
buttons[ii].pot          = no;                 /* this is a regular button */
buttons[ii].graphNum     = no;
buttons[ii].graphSelect  = yes;
buttons[ii].mask         = buttonMASK;
strcpy(buttons[ii].text,"-");
buttons[ii].textColor    = graphColor;
buttons[ii].xHalf        = buttons[ii].buttonWidth/2;
buttons[ii].yHalf        = buttons[ii].buttonHeight/2;
++num;
ii = graph7;
buttons[ii].buttonX      = graphBarLeft + 6*(graphBarWidth);
buttons[ii].buttonY      = graphBarTop;
buttons[ii].buttonWidth  = graphBarWidth;
buttons[ii].buttonHeight = graphBarHeight;
buttons[ii].buttonKey    = ii;
buttons[ii].pot          = no;                 /* this is a regular button */
buttons[ii].graphNum     = yes;
buttons[ii].graphSelect  = no;
buttons[ii].mask         = buttonMASK;
strcpy(buttons[ii].text,"7");
buttons[ii].textColor    = graphColor;
buttons[ii].xHalf        = buttons[ii].buttonWidth/2;
buttons[ii].yHalf        = buttons[ii].buttonHeight/2;
++num;
ii = graphSelect7;
buttons[ii].buttonX      = graphBarLeft + 6*(graphBarWidth);
buttons[ii].buttonY      = graphBarTop + graphBarHeight;
buttons[ii].buttonWidth  = graphBarWidth;
buttons[ii].buttonHeight = graphBarHeight-2;
buttons[ii].buttonKey    = ii;
buttons[ii].pot          = no;                 /* this is a regular button */
buttons[ii].graphNum     = no;
buttons[ii].graphSelect  = yes;
buttons[ii].mask         = buttonMASK;
strcpy(buttons[ii].text,"-");
buttons[ii].textColor    = graphColor;
```

```
buttons[ii].xHalf        = buttons[ii].buttonWidth/2;
buttons[ii].yHalf        = buttons[ii].buttonHeight/2;
++num;
ii = graph8;
buttons[ii].buttonX      = graphBarLeft + 7*(graphBarWidth);
buttons[ii].buttonY      = graphBarTop;
buttons[ii].buttonWidth  = graphBarWidth;
buttons[ii].buttonHeight = graphBarHeight;
buttons[ii].buttonKey    = ii;
buttons[ii].pot          = no;                    /* this is a regular button */
buttons[ii].graphNum     = yes;
buttons[ii].graphSelect  = no;
buttons[ii].mask         = buttonMASK;
strcpy(buttons[ii].text,"8");
buttons[ii].textColor    = graphColor;
buttons[ii].xHalf        = buttons[ii].buttonWidth/2;
buttons[ii].yHalf        = buttons[ii].buttonHeight/2;
++num;
ii = graphSelect8;
buttons[ii].buttonX      = graphBarLeft + 7*(graphBarWidth);
buttons[ii].buttonY      = graphBarTop + graphBarHeight;
buttons[ii].buttonWidth  = graphBarWidth;
buttons[ii].buttonHeight = graphBarHeight-2;
buttons[ii].buttonKey    = ii;
buttons[ii].pot          = no;                    /* this is a regular button */
buttons[ii].graphNum     = no;
buttons[ii].graphSelect  = yes;
buttons[ii].mask         = buttonMASK;
strcpy(buttons[ii].text,"-");
buttons[ii].textColor    = graphColor;
buttons[ii].xHalf        = buttons[ii].buttonWidth/2;
buttons[ii].yHalf        = buttons[ii].buttonHeight/2;
++num;
ii = graph9;
buttons[ii].buttonX      = graphBarLeft + 8*(graphBarWidth);
buttons[ii].buttonY      = graphBarTop;
buttons[ii].buttonWidth  = graphBarWidth;
buttons[ii].buttonHeight = graphBarHeight;
buttons[ii].buttonKey    = ii;
buttons[ii].pot          = no;                    /* this is a regular button */
buttons[ii].graphNum     = yes;
buttons[ii].graphSelect  = no;
buttons[ii].mask         = buttonMASK;
strcpy(buttons[ii].text,"9");
buttons[ii].textColor    = graphColor;
buttons[ii].xHalf        = buttons[ii].buttonWidth/2;
```

```
    buttons[ii].yHalf        = buttons[ii].buttonHeight/2;
    ++num;
    ii = graphSelect9;
    buttons[ii].buttonX      = graphBarLeft + 8*(graphBarWidth);
    buttons[ii].buttonY      = graphBarTop + graphBarHeight;
    buttons[ii].buttonWidth  = graphBarWidth;
    buttons[ii].buttonHeight = graphBarHeight-2;
    buttons[ii].buttonKey    = ii;
    buttons[ii].pot          = no;                /* this is a regular button */
    buttons[ii].graphNum     = no;
    buttons[ii].graphSelect  = yes;
    buttons[ii].mask         = buttonMASK;
    strcpy(buttons[ii].text,"*");
    buttons[ii].textColor    = graphColor;
    buttons[ii].xHalf        = buttons[ii].buttonWidth/2;
    buttons[ii].yHalf        = buttons[ii].buttonHeight/2;
    ++num;
    return(num);
  }
```

6.3.2   writeControlTitle

⟨*view2d*⟩+≡

```
  void writeControlTitle(void) {
    int strlength;
    s = viewport->title;
    strlength = strlen(s);
    XClearArea(dsply,control->controlWindow,0,0,controlWidth,potA,False);
    GSetForeground(anotherGC,(float)controlTitleColor,Xoption);
    GDrawImageString(anotherGC,control->controlWindow,
                     centerX(anotherGC,s,strlength,controlWidth),
                     15,s,strlength,Xoption);
  } /* writeControlTitle() */
```

6.3.3 makeMessageFromData

⟨*view2d*⟩+≡

```c
void makeMessageFromData(int whichGraph) {
  if (viewport->haveControl) {
    if ((graphStateArray[whichGraph].scaleX) > 99.0) {
      strcpy(scaleXReport,"big");
    } else {
      sprintf(scaleXReport,"%4.1f",graphStateArray[whichGraph].scaleX);
    }
    if ((graphStateArray[whichGraph].scaleY) > 99.0) {
      strcpy(scaleYReport,"big");
    } else {
      sprintf(scaleYReport,"%4.1f",graphStateArray[whichGraph].scaleY);
    }
    if ((graphStateArray[whichGraph].centerX) > 999.0) {
      strcpy(deltaXReport,"+big");
    } else if ((graphStateArray[whichGraph].centerX) < -999.0) {
      strcpy(deltaXReport,"-big");
    } else {
      sprintf(deltaXReport,"%4.0f",
              -graphStateArray[whichGraph].centerX /
              graphArray[whichGraph].unitX);
    }
    if ((graphStateArray[whichGraph].centerY) > 999.0) {
      strcpy(deltaYReport,"+big");
    } else if ((graphStateArray[whichGraph].centerY) < -999.0) {
      strcpy(deltaYReport,"-big");
    } else {
      sprintf(deltaYReport,"%4.0f",
              -graphStateArray[whichGraph].centerY /
              graphArray[whichGraph].unitY);
    }
    sprintf(viewport->controlPanel->message,"[%s,%s] >%d< [%s,%s]",
            scaleXReport,scaleYReport,whichGraph+1,deltaXReport,deltaYReport);
  }  /* if haveControl */
} /* makeMessageFromData() */
```

6.3.4   writeControlMessage

⟨*view2d*⟩+≡

```
void writeControlMessage(void) {
  int               strlength;
  controlPanelStruct *cp;
  XWindowAttributes  cwInfo;
  cp = viewport->controlPanel;
  XGetWindowAttributes(dsply,cp->controlWindow,&cwInfo);
  strlength = strlen(cp->message);
  GDrawImageString(controlMessageGC,cp->controlWindow,
                   centerX(globalGC1,cp->message,strlength,controlWidth),
                   controlMessageY + globalFont->max_bounds.ascent - 2,
                   cp->message,strlength,Xoption);
}
```

6.3.5 drawControlPanel

⟨*view2d*⟩+≡

```
void drawControlPanel(void) {
  controlPanelStruct *cp;
  int i,strlength;
  char *s;
  cp = viewport->controlPanel;
  /* Draw border lines to separate the potentiometer, message, graph select
     and button regions of the control panel. */
  GSetForeground(trashGC,(float)foregroundColor,Xoption);
  GSetLineAttributes(trashGC,3,LineSolid,CapButt,JoinMiter,Xoption);
  GDrawLine(trashGC, cp->controlWindow, 0, potA, controlWidth, potA, Xoption);
  GSetLineAttributes(trashGC,2,LineSolid,CapButt,JoinMiter,Xoption);
  GDrawLine(trashGC, cp->controlWindow, 0, potB, controlWidth, potB, Xoption);
  GDrawLine(trashGC, cp->controlWindow, 0, messageBot,
            controlWidth, messageBot, Xoption);
  GDrawLine(trashGC, cp->controlWindow, 0, 286, controlWidth, 286, Xoption);
  /** put the line width as 1 last because used below as well **/
  GSetLineAttributes(trashGC,1,LineSolid,CapButt,JoinMiter,Xoption);
  GDrawRectangle(trashGC,cp->controlWindow,closeLeft,closeTop,
        (controlWidth-closeLeft+8),(controlHeight-closeTop+8),Xoption);
  /* Write potentiometer titles on the control panel. */
  writeControlTitle();
  GSetForeground(globGC,(float)controlPotHeaderColor,Xoption);
  s = "Scale";
  strlength = strlen(s);
  GDrawString(globGC,cp->controlWindow,
      centerX(globGC,s,strlength,
              cp->buttonQueue[scale2D].buttonWidth) +
      cp->buttonQueue[scale2D].buttonX, 31+headerHeight,s,strlength,Xoption);
  s = "Translate";
  strlength = strlen(s);
  GDrawString(globGC,cp->controlWindow,
              centerX(globGC,s,strlength,
                      cp->buttonQueue[translate2D].buttonWidth) +
              cp->buttonQueue[translate2D].buttonX,
              31+headerHeight,s,strlen(s),Xoption);
  GSetForeground(globGC,(float)controlColorColor,Xoption);
  /* Write title of the graph selection window. */
  s = "Graphs";
  strlength = strlen(s);
  GDrawString(globGC,cp->controlWindow,
              centerX(globGC,s,strlength,controlWidth),graphHeaderHeight,
              s,strlength,Xoption);
  /* Write titles on regular buttons and draw pixmaps on potentiometers. */
```

```
      for (i=0; i<(maxButtons2D); i++) {
        if ((cp->buttonQueue[i]).pot) {
          GSetForeground(globalGC1,(float)buttonColor,Xoption);
          GDrawRectangle(globalGC1,cp->controlWindow,
                  (cp->buttonQueue[i]).buttonX,
                  (cp->buttonQueue[i]).buttonY,
                  (cp->buttonQueue[i]).buttonWidth,
                  (cp->buttonQueue[i]).buttonHeight,Xoption);
          GSetForeground(trashGC,
                  (float)monoColor((cp->buttonQueue[i]).textColor),Xoption);
          GDrawLine(globalGC1,cp->controlWindow, /* trashGC, */
            (cp->buttonQueue[i]).buttonX + (cp->buttonQueue[i]).xHalf,
            (cp->buttonQueue[i]).buttonY,
            (cp->buttonQueue[i]).buttonX + (cp->buttonQueue[i]).xHalf,
            (cp->buttonQueue[i]).buttonY + 2*(cp->buttonQueue[i]).yHalf,Xoption);
          GDrawLine(globalGC1,cp->controlWindow, /* trashGC, */
            (cp->buttonQueue[i]).buttonX,
            (cp->buttonQueue[i]).buttonY + (cp->buttonQueue[i]).yHalf,
            (cp->buttonQueue[i]).buttonX + 2*(cp->buttonQueue[i]).xHalf,
            (cp->buttonQueue[i]).buttonY + (cp->buttonQueue[i]).yHalf,Xoption);
          switch (i) {
          case scale2D:
            GDrawLines(trashGC,cp->controlWindow,scaleArrow,
                      scaleArrowN,CoordModeOrigin,Xoption);
            break;
          case translate2D:
            GDrawLines(trashGC,cp->controlWindow,translateArrow,
                      translateArrowN,CoordModeOrigin,Xoption);
            break;
          } /* switch i */
        } else if (cp->buttonQueue[i].graphNum) {
          if (mono) {
            if (graphStateArray[i-graphStart].showing) {
              GSetForeground(graphGC,(float)backgroundColor,Xoption);
              GSetBackground(graphGC,(float)foregroundColor,Xoption);
            } else {
              GSetForeground(graphGC,(float)foregroundColor,Xoption);
              GSetBackground(graphGC,(float)backgroundColor,Xoption);
            }
            strlength = strlen((cp->buttonQueue[i]).text);
            GDrawImageString(graphGC,cp->controlWindow,
                    (cp->buttonQueue[i]).buttonX +
                    centerX(graphGC,cp->buttonQueue[i].text,
                          strlength,(cp->buttonQueue[i]).buttonWidth),
                    (cp->buttonQueue[i]).buttonY +
                    centerY(graphGC,(cp->buttonQueue[i]).buttonHeight),
```

```
                      cp->buttonQueue[i].text,strlength,Xoption);
        } else {
          if (graphStateArray[i-graphStart].showing)
            GSetForeground(graphGC,(float)graphBarShowingColor,Xoption);
          else
            GSetForeground(graphGC,(float)graphBarHiddenColor,Xoption);
          strlength = strlen((cp->buttonQueue[i]).text);
          GDrawString(graphGC,cp->controlWindow,
              (cp->buttonQueue[i]).buttonX +
              centerX(graphGC,cp->buttonQueue[i].text,
                      strlength,(cp->buttonQueue[i]).buttonWidth),
              (cp->buttonQueue[i]).buttonY +
              centerY(graphGC,(cp->buttonQueue[i]).buttonHeight),
              cp->buttonQueue[i].text,strlength,Xoption);
        }
    } else if (cp->buttonQueue[i].graphSelect) {
      /* The select characters are defined as: "^" for on and "-" for off. */
      if (graphStateArray[i-graphSelectStart].selected) {
        GSetForeground(graphGC,(float)graphBarSelectColor,Xoption);
        strcpy((cp->buttonQueue[i]).text,"^");
      } else {
        GSetForeground(graphGC,(float)graphBarNotSelectColor,Xoption);
        *(cp->buttonQueue[i]).text = '-';
        strcpy((cp->buttonQueue[i]).text,"-");
      }
       GDrawString(graphGC,cp->controlWindow,
         (cp->buttonQueue[i]).buttonX +
         centerX(graphGC,cp->buttonQueue[i].text,
                 strlength,(cp->buttonQueue[i]).buttonWidth),
         (cp->buttonQueue[i]).buttonY +
         centerY(graphGC,(cp->buttonQueue[i]).buttonHeight),
         cp->buttonQueue[i].text,strlength,Xoption);
    }
    else {  /* a regular button */
      int isOn = 1;
      switch(i) {
        case pointsOnOff:
          isOn = pointsON = graphStateArray[0].pointsOn;
          if (graphStateArray[0].pointsOn)
            strcpy((cp->buttonQueue[i]).text,"Pts On ");
          else
            strcpy((cp->buttonQueue[i]).text,"Pts Off");
          break;
        case spline2D:
          isOn = splineON = graphStateArray[0].splineOn;
          if (graphStateArray[0].splineOn)
```

```
                           strcpy((cp->buttonQueue[i]).text, "Box On ");
                         else
                           strcpy((cp->buttonQueue[i]).text, "Box Off");
                         break;
                     case connectOnOff:
                         isOn = connectON = graphStateArray[0].connectOn;
                         if (graphStateArray[0].connectOn)
                           strcpy((cp->buttonQueue[i]).text, "Lines On ");
                         else
                           strcpy((cp->buttonQueue[i]).text, "Lines Off");
                         break;
                     case axesOnOff2D:
                         isOn = axesON = graphStateArray[0].axesOn;
                         if (graphStateArray[0].axesOn)
                           strcpy((cp->buttonQueue[i]).text , "Axes On ");
                         else
                           strcpy((cp->buttonQueue[i]).text , "Axes Off");
                         break;
                     case unitsOnOff2D:
                         isOn = unitsON = graphStateArray[0].unitsOn;
                         if (graphStateArray[0].unitsOn)
                          strcpy( (cp->buttonQueue[i]).text , "Units On ");
                         else
                          strcpy( (cp->buttonQueue[i]).text , "Units Off");
                         break;
                     case closeAll2D:
                         isOn = 0;
                     default:
                         break;
                 } /* switch i */
                 s = (cp->buttonQueue[i]).text;
                 strlength = strlen(s);
                 GDrawPushButton(dsply, globalGC1, trashGC, processGC, cp->controlWindow,
                       (cp->buttonQueue[i]).buttonX, (cp->buttonQueue[i]).buttonY,
                       (cp->buttonQueue[i]).buttonWidth, (cp->buttonQueue[i]).buttonHeight,
                       isOn, s,buttonColor,
                       monoColor((cp->buttonQueue[i]).textColor), Xoption);
             } /* else a regular button */
         } /* for each button */
         /* Refresh the latest message */
         makeMessageFromData(0);
         writeControlMessage();
         XFlush(dsply);
    }    /*** drawControlPanel ***/
```

6.3.6 getControlXY

⟨*view2d*⟩+≡

```
controlXY getControlXY(int whereDoYouWantPanel) {
  XWindowAttributes wAttr, wAttrib;
  controlXY        cXY = {0,0};
  int              tmp=1;
  Window           rootW, parentW, *childrenWs, tmpW;
  unsigned int     nChildren;
  tmpW = viewport->titleWindow;
  while(tmp) {
    XQueryTree(dsply,tmpW,&rootW,&parentW,&childrenWs,&nChildren);
    XFree(childrenWs);
    if (parentW == rtWindow) tmp = 0;
    else tmpW = parentW;
  }
  XGetWindowAttributes(dsply,tmpW,&wAttrib);
  XGetWindowAttributes(dsply,viewport->titleWindow,&wAttr);
  if (whereDoYouWantPanel) {
    switch (whereDoYouWantPanel) {
    case 1: /* right */
      cXY.putX = wAttrib.x + wAttrib.width;
      cXY.putY = wAttrib.y;
      break;
    case 2: /* bottom */
      cXY.putX = wAttrib.x + (wAttr.width - controlWidth)/2;  /* center it */
      cXY.putY = wAttrib.y + wAttrib.height;
      break;
    case 3: /* left */
      cXY.putX = wAttrib.x - controlWidth - borderWidth;
      cXY.putY = wAttrib.y;
      break;
    case 4: /* top */
      cXY.putX = wAttrib.x + (wAttr.width - controlWidth)/2;
      cXY.putY = wAttrib.y - controlHeight - borderHeight;
    }
  } else {
    if ((physicalWidth - (wAttrib.x + wAttr.width)) >= controlWidth) {
      cXY.putX = wAttrib.x + wAttrib.width;
      cXY.putY = wAttrib.y;
    } else if ((physicalHeight - (wAttrib.y + wAttr.height)) >=
               controlHeight) {
      cXY.putX = wAttrib.x + (wAttr.width - controlWidth)/2;
      cXY.putY = wAttrib.y + wAttrib.height;
    } else if (wAttrib.x >= controlWidth) {
      cXY.putX = wAttrib.x - controlWidth - borderWidth;
```

```
        cXY.putY = wAttrib.y;
    } else if (wAttrib.y >= controlHeight) {
        cXY.putX = wAttrib.x + (wAttr.width - controlWidth)/2;
        cXY.putY = wAttrib.y - controlHeight - borderHeight;
    } else {                            /* put inside of viewport */
        cXY.putX = wAttrib.x + wAttr.width - controlWidth;
        cXY.putY = wAttrib.y + wAttr.height - controlHeight;
    }
  }
  return(cXY);
}
```

6.3.7 makeControlPanel

⟨view2d⟩+≡

```
controlPanelStruct *makeControlPanel(void) {
  Window cw;
  int               i,num;
  controlPanelStruct   *control;
  buttonStruct          *buttons;
  controlXY             cXY = {0,0};
  XSetWindowAttributes cwAttrib, controlAttrib;
  XSizeHints            sizehints;
  Pixmap                mousebits,mousemask;
  XColor                foreColor, backColor;
  if (!(control = (controlPanelStruct *)malloc(sizeof(controlPanelStruct)))) {
    fprintf(stderr,"Ran out of memory trying to create a control panel.\n");
    exitWithAck(RootWindow(dsply,scrn),Window,-1);
  }
  cXY = getControlXY(0);
  /* Define and assign a mouse cursor. */
  mousebits = XCreateBitmapFromData(dsply,rtWindow,mouseBitmap_bits,
                                    mouseBitmap_width,mouseBitmap_height);
  mousemask = XCreateBitmapFromData(dsply,rtWindow,mouseMask_bits,
                                    mouseMask_width,mouseMask_height);
  cwAttrib.background_pixel = backgroundColor; /* controlBackground; */
  cwAttrib.border_pixel = foregroundColor;
  cwAttrib.backing_store = WhenMapped;
  cwAttrib.event_mask = controlMASK;
  cwAttrib.colormap = colorMap;
  cwAttrib.override_redirect = overrideManager;
  foreColor.pixel = controlCursorForeground;
  XQueryColor(dsply,colorMap,&foreColor);
  backColor.pixel = controlCursorBackground;
  XQueryColor(dsply,colorMap,&backColor);
  cwAttrib.cursor = XCreatePixmapCursor(dsply,mousebits,mousemask,
                                        &foreColor,&backColor,
                                        mouseBitmap_x_hot,mouseBitmap_y_hot);
  cw = XCreateWindow(dsply,rtWindow,
                     cXY.putX,cXY.putY,controlWidth,controlHeight,3,
                     CopyFromParent,InputOutput,CopyFromParent,
                     controlCreateMASK,&cwAttrib);
  sizehints.flags  = PPosition | PSize;
  sizehints.x      = cXY.putX;
  sizehints.y      = cXY.putY;
  sizehints.width  = controlWidth;
  sizehints.height = controlHeight;
  /*** the None stands for icon pixmap...change.... ***/
```

```
  XSetNormalHints(dsply,cw,&sizehints);
  XSetStandardProperties(dsply,cw,"2D Control Panel","2D Control Panel",
                         None,NULL,0,&sizehints);
  control->controlWindow = cw;
  num = initButtons(control->buttonQueue);
  control->numOfButtons = num;
  buttons = control->buttonQueue;
  for (i=0; i<num; i++) {
    controlAttrib.event_mask = (control->buttonQueue[i]).mask;
    (control->buttonQueue[i]).self = XCreateWindow(dsply,cw,
                                       (control->buttonQueue[i]).buttonX,
                                       (control->buttonQueue[i]).buttonY,
                                       (control->buttonQueue[i]).buttonWidth,
                                       (control->buttonQueue[i]).buttonHeight,
                                       0,0,InputOnly,CopyFromParent,
                                       buttonCreateMASK,&controlAttrib);
    XMakeAssoc(dsply,table,(control->buttonQueue[i]).self,
               &((control->buttonQueue[i]).buttonKey));
   /* Use buttonKey instead of i because buttonKey has a permanent address */
    XMapWindow(dsply,(control->buttonQueue[i]).self);
  }
          /* Create message window */
  control->messageWindow = XCreateWindow(dsply,cw,0,controlMessageY,
                                     controlWidth,controlMessageHeight,
                                     0,0,InputOnly,CopyFromParent,
                                     messageCreateMASK,&cwAttrib);
  XMapWindow(dsply,control->messageWindow);
  for (i=0; i<scaleArrowN; i++) {
    scaleArrow[i].x += buttons[scale2D].buttonX;
    scaleArrow[i].y += buttons[scale2D].buttonY;
  }
  for (i=0; i<translateArrowN; i++) {
    translateArrow[i].x += buttons[translate2D].buttonX;
    translateArrow[i].y += buttons[translate2D].buttonY;
  }
  viewport->justMadeControl = yes;
  return(control);
} /* makeControlPanel() */
```

6.3.8   putControlPanelSomewhere

This routine puts up the control panel associated with the viewport passed in. It first tries to put it to the right of the viewport. If there isn't enough room there, it tries the bottom and so on going clockwise. If the viewport is too big and there is no room to put the control panel outside of it, it placed the control panel in the bottom right hand corner of the viewport window.

⟨*view2d*⟩+≡
```
  void putControlPanelSomewhere(int whereDoesPanelGo) {
    controlPanelStruct *control;
    controlXY           whereControl= {0,0};
    control = viewport->controlPanel;
    whereControl = getControlXY(whereDoesPanelGo);
    viewport->haveControl = yes;
    XRaiseWindow(dsply,control->controlWindow);
    XMoveWindow(dsply,control->controlWindow,whereControl.putX,
                whereControl.putY);
    drawControlPanel();
    if (viewport->justMadeControl) {
      XMapWindow(dsply,control->controlWindow);
      viewport->justMadeControl = no;
    }
    XMapWindow(dsply,control->controlWindow);
  }
```

6.3.9   clearControlMessage

⟨*view2d*⟩+≡
```
  void clearControlMessage(void) {
    strcpy(viewport->controlPanel->message,"");
    XClearArea(dsply,viewport->controlPanel->controlWindow,
               0,controlMessageY-2,controlWidth,controlMessageHeight,False);
  }
```

6.3.10   getGraphFromViewman

This routine should be called right after a read of the graph key was made from the viewport
manager (or defined in some other way).

⟨*view2d*⟩+≡

```
  void getGraphFromViewman(int i) {
    int j,k,xPointsNeeded;
    pointListStruct *llPtr;
    pointStruct     *p;
    readViewman(&(graphArray[i].xmin),floatSize);
    readViewman(&(graphArray[i].xmax),floatSize);
    readViewman(&(graphArray[i].ymin),floatSize);
    readViewman(&(graphArray[i].ymax),floatSize);
    readViewman(&(graphArray[i].xNorm),floatSize);
    readViewman(&(graphArray[i].yNorm),floatSize);
    readViewman(&(graphArray[i].spadUnitX),floatSize);
    readViewman(&(graphArray[i].spadUnitY),floatSize);
    readViewman(&(graphArray[i].unitX),floatSize);
    readViewman(&(graphArray[i].unitY),floatSize);
    readViewman(&(graphArray[i].originX),floatSize);
    readViewman(&(graphArray[i].originY),floatSize);
    readViewman(&(graphArray[i].numberOfLists),intSize);
    if (!(llPtr = (pointListStruct *)malloc(graphArray[i].numberOfLists * sizeof(po
      fprintf(stderr,"VIEW2D: Fatal Error>> Ran out of memory trying to receive a g
      exitWithAck(RootWindow(dsply,scrn),Window,-1);
    }
    graphArray[i].listOfListsOfPoints = llPtr;
    xPointsNeeded = 0;
    for (j=0; j<graphArray[i].numberOfLists; j++) {
      readViewman(&(llPtr->numberOfPoints),intSize);
      if (!(p = (pointStruct *)
            malloc(llPtr->numberOfPoints * sizeof(pointStruct)))) {
        fprintf(stderr,"VIEW2D: (pointStruct) ran out of memory trying to \n");
        fprintf(stderr,"create a new graph.\n");
        exitWithAck(RootWindow(dsply,scrn),Window,-1);
      }
      llPtr->listOfPoints = p;                 /** point to current point list **/
      for (k=0; k<llPtr->numberOfPoints; k++) {
        readViewman(&(p->x),floatSize);
        readViewman(&(p->y),floatSize);
        readViewman(&(p->hue),floatSize);
        readViewman(&(p->shade),floatSize);
        p++;
      }   /* for k in list of points */
      readViewman(&(llPtr->pointColor),intSize);
```

```
      readViewman(&(llPtr->lineColor),intSize);
      readViewman(&(llPtr->pointSize),intSize);
      xPointsNeeded += llPtr->numberOfPoints;
      llPtr++;
  }   /* for j in list of lists of points */
  /* read in graph state for the existing graph (override default values) */
  readViewman(&(graphStateArray[i].scaleX),floatSize);
  readViewman(&(graphStateArray[i].scaleY),floatSize);
  readViewman(&(graphStateArray[i].deltaX),floatSize);
  readViewman(&(graphStateArray[i].deltaY),floatSize);
  readViewman(&(graphStateArray[i].pointsOn),intSize);
  readViewman(&(graphStateArray[i].connectOn),intSize);
  readViewman(&(graphStateArray[i].splineOn),intSize);
  readViewman(&(graphStateArray[i].axesOn),intSize);
  readViewman(&(graphStateArray[i].axesColor),intSize);
  readViewman(&(graphStateArray[i].unitsOn),intSize);
  readViewman(&(graphStateArray[i].unitsColor),intSize);
  readViewman(&(graphStateArray[i].showing),intSize);
  graphStateArray[i].selected = yes;
  graphStateBackupArray[i] = graphStateArray[i];
  graphStateArray[i].deltaX = graphStateArray[0].deltaX;
  graphStateArray[i].deltaY = graphStateArray[0].deltaY;
  graphStateArray[i].scaleX = graphStateArray[0].scaleX;
  graphStateArray[i].scaleY = graphStateArray[0].scaleY;
  /* allocate memory for xPoints (used in drawViewport) */
  if (!(xPointsArray[i].xPoint =
          (XPoint *)malloc(xPointsNeeded * sizeof(XPoint)))) {
    fprintf(stderr,"VIEW2D: (XPoint) Ran out of memory (malloc) trying \n");
    fprintf(stderr,"to create a new graph.\n");
    exitWithAck(RootWindow(dsply,scrn),Window,-1);
  }
  if (!(xPointsArray[i].x10Point =
          (Vertex *)malloc(xPointsNeeded * sizeof(Vertex)))) {
    fprintf(stderr,"VIEW2D: (X10Point) Ran out of memory (malloc) trying \n");
    fprintf(stderr,"to create a new graph.\n");
    exitWithAck(RootWindow(dsply,scrn),Window,-1);
  }
  if (!(xPointsArray[i].arc = (XArc *)malloc(xPointsNeeded * sizeof(XArc)))) {
    fprintf(stderr,"VIEW2D: (XArc) Ran out of memory (malloc) trying \n");
    fprintf(stderr,"to create a new graph.\n");
    exitWithAck(RootWindow(dsply,scrn),Window,-1);
  }
}   /* getGraphFromViewman */
```

6.3.11   freeGraph

⟨*view2d*⟩+≡

```
void freeGraph(int i) {
  int j;
  pointListStruct *llPtr;
  if (graphArray[i].key) {
    graphArray[i].key = 0;    /* 0 means no graph */
    for (j=0,llPtr=graphArray[i].listOfListsOfPoints;
         j<graphArray[i].numberOfLists; j++,llPtr++)
      free(llPtr->listOfPoints);
    free(llPtr);
    free(xPointsArray[i].xPoint);
  } else {
  }
}
```

6.3.12 mergeDatabases

⟨*view2d*⟩+≡

```
void mergeDatabases(void) {
  /* using global
     rDB
     dsply
     */
  XrmDatabase homeDB,serverDB,applicationDB;
  char filenamebuf[1024];
  char *filename = &filenamebuf[0];
  char *classname = "Axiom";
  char name[255];
  (void) XrmInitialize();
  (void) strcpy(name, "/usr/lib/X11/app-defaults/");
  (void) strcat(name, classname);
  applicationDB = XrmGetFileDatabase(name);
  (void) XrmMergeDatabases(applicationDB, &rDB);
  if (XResourceManagerString(dsply) != NULL)
    serverDB = XrmGetStringDatabase(XResourceManagerString(dsply));
  else {
    (void) strcpy(filename,getenv("HOME"));
    (void) strcat(filename,"/.Xdefaults");
    serverDB = XrmGetFileDatabase(filename);
  }
  XrmMergeDatabases(serverDB,&rDB);
  if ( getenv ("XENVIRONMENT") == NULL) {
    int len;
    (void) strcpy(filename,getenv("HOME"));
    (void) strcat(filename,"/.Xdefaults-");
    len = strlen(filename);
    (void) gethostname(filename+len,1024-len);
  }
  else
    (void) strcpy (filename,getenv ("XENVIRONMENT"));
  homeDB = XrmGetFileDatabase(filename);
  XrmMergeDatabases(homeDB,&rDB);
}
```

6.3.13   getPotValue

⟨*view2d*⟩+≡

```
mouseCoord getPotValue(short eX,short eY,short xH,short yH) {
  mouseCoord whereMouse;
  float x,y;
  x = (float)eX/xH - 1;
  y = -((float)eY/yH -1);
            /* make non-linear potentiometer */
  whereMouse.x = x*x*x;
  whereMouse.y = y*y*y;
  return(whereMouse);
} /* getPotValue() */
```

6.3.14   doPick

⟨*view2d*⟩+≡

```
void doPick(int i,int bKey) {
  int vCommand=pick2D;
  picking = no;
  /* reset indicator so that we're not in pick/drop/query mode anymore */
  doit = 0;
  if (graphArray[currentGraph].key) {
    check(write(Socket,&vCommand,intSize));
    check(write(Socket,&(graphArray[currentGraph].key),intSize));
    check(write(Socket,&(graphStateArray[currentGraph]),
                sizeof(graphStateStruct)));
    sprintf(control->message,"%s%d","Picked up graph ",currentGraph+1);
  } else
    sprintf(control->message,"%s","This graph is empty!");
  writeControlMessage();
}
```

6.3.15 doDrop

⟨*view2d*⟩+≡

```
void doDrop(int i,int bKey) {
  int vCommand=drop2D;
  int viewGoAhead;
  dropping = no;
  /* reset indicator so that we're not in pick/drop/query mode anymore */
  doit = 0;
  check(write(Socket,&(vCommand),intSize));
  readViewman(&viewGoAhead,intSize);
  if (viewGoAhead < 0) {
    sprintf(control->message,"%s%d","Couldn't drop to graph ",currentGraph+1);
    writeControlMessage();
  } else {
    sprintf(control->message,"%s%d","Dropped onto graph ",currentGraph+1);
    writeControlMessage();
    freeGraph(currentGraph);
    readViewman(&(graphArray[currentGraph].key),intSize);
    getGraphFromViewman(currentGraph);
    /* simulate a button press to turn display number on and select on */
    /* need !yes since it will be inverted */
    graphStateArray[currentGraph].selected = no;
    graphStateArray[currentGraph].showing =
      !(graphStateArray[currentGraph].showing);
    clickedOnGraph(currentGraph,currentGraph+graphStart);
    clickedOnGraphSelect(currentGraph,currentGraph+graphSelectStart);
  }
}
```

6.3.16   clickedOnGraphSelect

⟨*view2d*⟩+≡
```
  void clickedOnGraphSelect(int i,int bKey) {
    int strlength;
    switch (doit) {
    case pick2D:
      currentGraph = i;
      doPick(i,bKey);
      break;
    case drop2D:
      currentGraph = i;
      doDrop(i,bKey);
      break;
    case query2D:
      queriedGraph = i;
      makeMessageFromData(queriedGraph);
      writeControlMessage();
      /* reset indicator so that we're not in pick/drop/query mode anymore */
      doit = 0;
      break;
    default:
      graphStateArray[i].selected = !(graphStateArray[i].selected);
      if (graphStateArray[i].selected) {
        GSetForeground(graphGC,(float)graphBarSelectColor,Xoption);
        strcpy(control->buttonQueue[bKey].text,"^");
        strlength = strlen(control->buttonQueue[bKey].text);
      } else {
        GSetForeground(graphGC,(float)graphBarNotSelectColor,Xoption);
        strcpy(control->buttonQueue[bKey].text,"-");
        strlength = strlen(control->buttonQueue[bKey].text);
      }
      /* just to make sure the background is reset from clickedOnGraph */
      if (mono) {
        GSetForeground(graphGC,(float)foregroundColor,Xoption);
        GSetBackground(graphGC,(float)backgroundColor,Xoption);
      }
      GDrawImageString(graphGC,control->controlWindow,
                       control->buttonQueue[bKey].buttonX +
                       centerX(graphGC,
                               control->buttonQueue[bKey].text,strlength,
                               control->buttonQueue[bKey].buttonWidth),
                       control->buttonQueue[bKey].buttonY +
                       centerY(graphGC,
                               control->buttonQueue[bKey].buttonHeight),
                       control->buttonQueue[bKey].text,strlength,Xoption);
```

```
      GSetForeground(trashGC,(float)foregroundColor,Xoption);
      GSetLineAttributes(trashGC,2,LineSolid,CapButt,JoinMiter,Xoption);
      GDrawLine(trashGC,control->controlWindow,0,286,controlWidth,286,Xoption);
      break;
  } /* switch doit */
}
```

6.3.17   drawControlPushButton

⟨*view2d*⟩+≡

```
static void drawControlPushButton(int isOn, int index) {
  GDrawPushButton(dsply, processGC, processGC, processGC,
                  control->controlWindow,
                  (control->buttonQueue[index]).buttonX,
                  (control->buttonQueue[index]).buttonY,
                  (control->buttonQueue[index]).buttonWidth,
                  (control->buttonQueue[index]).buttonHeight,
                  isOn,
                  (control->buttonQueue[index]).text,
                  buttonColor,
                  monoColor((control->buttonQueue[index]).textColor), Xoption);
  XSync(dsply,False);
}
```

6.3.18   buttonAction

⟨*view2d*⟩+≡
```
  void buttonAction(int bKey) {
    int i;
    switch (bKey) {
    case pick2D:
      if (viewAloned) {
        sprintf(control->message,"%s","Cannot pick without Axiom!");
        writeControlMessage();
        XSync(dsply,False);
      }
      else {
        doit = pick2D;
        sprintf(control->message,"%s","Click on the graph to pick");
        writeControlMessage();
        XSync(dsply,False);
      }
      break;
    case drop2D:
      if (viewAloned) {
        sprintf(control->message,"%s","Cannot drop without Axiom!");
        writeControlMessage();
        XSync(dsply,False);
      } else {
        doit = drop2D;
        sprintf(control->message,"%s","Click on the graph to drop");
        writeControlMessage();
        XSync(dsply,False);
      }
      break;
    case query2D:
      doit = query2D;
      sprintf(control->message,"%s","Click on the graph to query");
      writeControlMessage();
      XSync(dsply,False);
      break;
    case zoom2Dx:
      if (zoomXON)
        strcpy((control->buttonQueue[zoom2Dx]).text ,"X Off");
      else
        strcpy((control->buttonQueue[zoom2Dx]).text ,"X On ");
      zoomXON = !zoomXON;
      drawControlPushButton(zoomXON, zoom2Dx);
      XSync(dsply,False);
      break;
```

```
case zoom2Dy:
  if (zoomYON) strcpy((control->buttonQueue[zoom2Dy]).text,
                      "Y Off");
  else strcpy((control->buttonQueue[zoom2Dy]).text ,
             "Y On ");
  zoomYON = !zoomYON;
  drawControlPushButton(zoomYON, zoom2Dy);
  XSync(dsply,False);
  break;
case translate2Dx:
  if (transXON) strcpy((control->buttonQueue[translate2Dx]).text,"X Off");
  else strcpy( (control->buttonQueue[translate2Dx]).text,"X On ");
  transXON = !transXON;
  drawControlPushButton(transXON, translate2Dx);
  XSync(dsply,False);
  break;
case translate2Dy:
  if (transYON) strcpy((control->buttonQueue[translate2Dy]).text,
                        "Y Off");
  else strcpy( (control->buttonQueue[translate2Dy]).text,
              "Y On");
  transYON = !transYON;
  drawControlPushButton(transYON, translate2Dy);
  XSync(dsply,False);
  break;
case pointsOnOff:
  if (pointsON) strcpy((control->buttonQueue[pointsOnOff]).text,
                        "Pts Off");
  else strcpy( (control->buttonQueue[pointsOnOff]).text,
              "Pts On ");
  pointsON = !pointsON;
  for (i=0; i<maxGraphs; i++)
    if (graphStateArray[i].showing && graphStateArray[i].selected)
      graphStateArray[i].pointsOn = pointsON;
  drawControlPushButton(pointsON, pointsOnOff);
  drawViewport(Xoption);
  break;
case connectOnOff:
  if (connectON)
    strcpy((control->buttonQueue[connectOnOff]).text,"Lines Off");
  else
    strcpy( (control->buttonQueue[connectOnOff]).text,"Lines On ");
  connectON = !connectON;
  for (i=0; i<maxGraphs; i++)
    if (graphStateArray[i].showing && graphStateArray[i].selected)
      graphStateArray[i].connectOn = connectON;
```

```
    drawControlPushButton(connectON, connectOnOff);
    drawViewport(Xoption);
    break;
  case spline2D:
    if (splineON) strcpy((control->buttonQueue[spline2D]).text,"Box Off");
    else strcpy ((control->buttonQueue[spline2D]).text ,"Box On ");
    splineON = !splineON;
    for (i=0; i<maxGraphs; i++)
      if (graphStateArray[i].showing && graphStateArray[i].selected)
        graphStateArray[i].splineOn = splineON;
    drawControlPushButton(splineON, spline2D);
    drawViewport(Xoption);
    break;
  case axesOnOff2D:
    if (axesON)
      strcpy ((control->buttonQueue[axesOnOff2D]).text,"Axes Off");
    else
      strcpy ((control->buttonQueue[axesOnOff2D]).text,"Axes On ");
    axesON = !axesON;
    for (i=0; i<maxGraphs; i++)
      if (graphStateArray[i].showing && graphStateArray[i].selected)
        graphStateArray[i].axesOn = axesON;
    drawControlPushButton(axesON, axesOnOff2D);
    drawViewport(Xoption);
    break;
  case unitsOnOff2D:
    if (unitsON)
      strcpy( (control->buttonQueue[unitsOnOff2D]).text,"Units Off");
    else
      strcpy ((control->buttonQueue[unitsOnOff2D]).text,"Units On ");
    unitsON = !unitsON;
    for (i=0; i<maxGraphs; i++)
      if (graphStateArray[i].showing && graphStateArray[i].selected)
        graphStateArray[i].unitsOn = unitsON;
    drawControlPushButton(unitsON, unitsOnOff2D);
    drawViewport(Xoption);
    break;
  case ps2D:
    strcpy(control->message,"Creating postscript now ...");
    writeControlMessage();
    drawControlPushButton(1, ps2D);
    if (PSInit(viewport->viewWindow, viewport->titleWindow) == psError) {
      strcpy(control->message,"Aborted: PSInit error.");
      writeControlMessage();
      drawControlPushButton(0, ps2D);
      XSync(dsply,False);
```

```
      return;    /* make new temp name for new file */
    }
    drawViewport(PSoption);      /* draw picture in PS; create ps script file */
    if (PSCreateFile(viewBorderWidth,
                     viewport->viewWindow,
                     viewport->titleWindow,
                     viewport->title) == psError) {
      strcpy(control->message,"Aborted: PSCreateFile error.");
      writeControlMessage();
      drawControlPushButton(0, ps2D);
      XSync(dsply,False);
      return;
    }
    clearControlMessage();
    strcpy(control->message,PSfilename);
    strcat(control->message," in working dir ");
    writeControlMessage();
    drawControlPushButton(0, ps2D);
    XSync(dsply,False);
    break;
  case hideControl2D:
    if (viewport->haveControl) {
      viewport->haveControl = no;
      XUnmapWindow(dsply,control->controlWindow);
      XSync(dsply,False);
    }
    break;
  case reset2D:
    /* reset view */
    for (i=0; i<maxGraphs; i++)
      if (graphStateArray[i].showing && graphStateArray[i].selected)
        graphStateArray[i] = graphStateBackupArray[i];
    unitsON = no;
    strcpy( (control->buttonQueue[unitsOnOff2D]).text,s  = "Units Off");
    for (i=0; i<maxGraphs; i++)
      if (graphStateArray[i].showing && graphStateArray[i].selected)
        graphStateArray[i].unitsOn = no;
    drawControlPushButton(unitsON, unitsOnOff2D);
    pointsON = yes;
    strcpy ((control->buttonQueue[pointsOnOff]).text ,"Pts On ");
    for (i=0; i<maxGraphs; i++)
      if (graphStateArray[i].showing && graphStateArray[i].selected)
        graphStateArray[i].pointsOn = yes;
    drawControlPushButton(pointsON, pointsOnOff);
    axesON = yes;
    strcpy ((control->buttonQueue[axesOnOff2D]).text,"Axes On ");
```

```
    for (i=0; i<maxGraphs; i++)
      if (graphStateArray[i].showing && graphStateArray[i].selected)
        graphStateArray[i].axesOn = yes;
    drawControlPushButton(axesON, axesOnOff2D);
    connectON = yes;
    strcpy((control->buttonQueue[connectOnOff]).text,"Lines On ");
    for (i=0; i<maxGraphs; i++)
      if (graphStateArray[i].showing && graphStateArray[i].selected)
        graphStateArray[i].connectOn = yes;
    drawControlPushButton(connectON, connectOnOff);
    splineON = no;
    strcpy( (control->buttonQueue[connectOnOff]).text ,"Box Off");
    for (i=0; i<maxGraphs; i++)
      if (graphStateArray[i].showing && graphStateArray[i].selected)
        graphStateArray[i].splineOn = splineON;
    drawControlPushButton(splineON, spline2D);
    drawViewport(Xoption);
    break;
  case closeAll2D:
    strcpy(control->message,"       Click again to confirm       ");
    writeControlMessage();
    drawControlPushButton(1, closeAll2D);
    XSync(dsply,False);
    viewport->closing = yes;
    break;
  case clear2D:
    for (i=0; i<maxGraphs; i++) graphStateArray[i].selected = 1;
    clickedOnGraphSelect(0,graphSelect1);
    clickedOnGraphSelect(1,graphSelect2);
    clickedOnGraphSelect(2,graphSelect3);
    clickedOnGraphSelect(3,graphSelect4);
    clickedOnGraphSelect(4,graphSelect5);
    clickedOnGraphSelect(5,graphSelect6);
    clickedOnGraphSelect(6,graphSelect7);
    clickedOnGraphSelect(7,graphSelect8);
    clickedOnGraphSelect(8,graphSelect9);
    XSync(dsply,False);
    break;
  case graph1:
  case graph2:
  case graph3:
  case graph4:
  case graph5:
  case graph6:
  case graph7:
  case graph8:
```

```
      case graph9:
        clickedOnGraph(bKey-graphStart,bKey);
        XSync(dsply,False);
        break;
      case graphSelect1:
      case graphSelect2:
      case graphSelect3:
      case graphSelect4:
      case graphSelect5:
      case graphSelect6:
      case graphSelect7:
      case graphSelect8:
      case graphSelect9:
        clickedOnGraphSelect(bKey-graphSelectStart,bKey);
        XSync(dsply,False);
        break;
    } /* switch (action) */
  }
```

6.3.19   processEvents

⟨*view2d*⟩+≡
```
void processEvents(void) {
  XEvent                 *event,
    tempEvent;
  Window                 whichWindow;
  XWindowAttributes      graphWindowAttrib;
  buttonStruct           *controlButton;
  mouseCoord             mouseXY = {0.0,0.0};
  int                    i,
    someInt,
    mouseW4,
    mouseH4,
    toggleReady,
    gotToggle = no,
    checkButton = no,
    firstTime = yes,
    gotEvent = 0,
    buttonTablePtr,
    Xcon,
    len,
    externalControl;
  fd_set                 rd;
  externalControl=0;
  Xcon = ConnectionNumber(dsply);
  if (!(event = (XEvent *)malloc(sizeof(XEvent)))) {
    fprintf(stderr,"Ran out of memory initializing event processing.\n");
    exitWithAck(RootWindow(dsply,scrn),Window,-1);
  }
  controlButton = control->buttonQueue;
  while(1) {
    len=0;
    while(len<=0) {
      FD_ZERO(&rd);
      if (externalControl==0) FD_SET(0, &rd);
      FD_SET(Xcon,&rd);
      if (XEventsQueued(dsply, QueuedAlready)) {
        len=1;
        break;
      }
      if (!followMouse)
        len=select(FD_SETSIZE,(void *) &rd,0,0,0);
      else
        len=1;
    }
```

```
    if (FD_ISSET(Xcon,&rd)||
        XEventsQueued(dsply, QueuedAfterFlush) ||
        followMouse) {

      if (followMouse) {
        if (XPending(dsply))
          XNextEvent(dsply,event);
        gotEvent++;
      } else {
        XNextEvent(dsply,event);
        gotEvent++;
      }
      if (gotToggle || !followMouse)
        checkButton = no;
      if (gotEvent) {
        whichWindow = ((XButtonEvent *)event)->window;
        firstTime = no;
        switch(((XEvent *)event)->type) {
        case ClientMessage:
          if (event->xclient.data.l[0] == wm_delete_window) {
                  goodbye(-1);
          }
          else {
            fprintf(stderr,"Unknown Client Message ...\n");
          }
          break;
        case Expose:
          if (whichWindow == viewport->titleWindow) {
            /* get rid of redundant events */
            XCheckWindowEvent(dsply,
                              viewport->titleWindow,
                              ExposureMask,
                              &tempEvent);
            writeTitle();
            XGetWindowAttributes(dsply,
                                 whichWindow,
                                 &graphWindowAttrib);
            XResizeWindow(dsply,
                          viewport->viewWindow,
                          graphWindowAttrib.width,
                          graphWindowAttrib.height-titleHeight);
            XSync(dsply,False);
            break;
          } else if (whichWindow == viewport->viewWindow) {
            XCheckWindowEvent(dsply,
                              viewport->viewWindow,
```

```
                                    ExposureMask,
                                    &tempEvent);
              XGetWindowAttributes(dsply,
                                   viewport->titleWindow,
                                   &graphWindowAttrib);
              XResizeWindow(dsply,
                            viewport->viewWindow,
                            graphWindowAttrib.width,
                            graphWindowAttrib.height-titleHeight);
              drawViewport(Xoption);
              XMapWindow(dsply,whichWindow);
              XSync(dsply,False);
              break;
            } else {        /* it's gotta be the control panel */
              XGetWindowAttributes(dsply,
                                   control->controlWindow,
                                   &graphWindowAttrib);
              /* do not allow resizing of control panel */
              if ((graphWindowAttrib.width != controlWidth) ||
                  (graphWindowAttrib.height != controlHeight)) {
                XResizeWindow(dsply,
                              control->controlWindow,
                              controlWidth,
                              controlHeight);
              }
              drawControlPanel();
              XSync(dsply,False);
              break;
            }
            break;
          case MotionNotify:
            if (followMouse) {
              while (XCheckMaskEvent(dsply,
                                     ButtonMotionMask,
                                     event));
              mouseXY = getPotValue(((XButtonEvent *)event)->x,
                                    ((XButtonEvent *)event)->y,
                                    controlButton->xHalf,
                                    controlButton->yHalf);
            }
            if (controlButton->pot) {
              gotToggle = no;
              checkButton = yes;
            }
            break;
          case ButtonRelease:
```

```
                       if (followMouse==yes) {
                         followMouse = no;
                         toggleReady = yes;
                         checkButton = no;
                         drawViewport(Xoption);
                       } else {
                         followMouse = no;
                         toggleReady = yes;
                         checkButton = no;
                       }
                       break;
                    case LeaveNotify:
                      /*
                        We still follow the mouse when we leave the pots.
                       */
                      /*
                              followMouse = no;
                              toggleReady = yes;
                              checkButton = no;
                              */
                      break;
                    case ButtonPress:
                      if (whichWindow == viewport->viewWindow) {
                        /* mouse clicked on viewport */
                        switch (((XButtonEvent *)event)->button) {
                        case Button3:
                          /* print out (x,y) object-space coordinates in message area */

                          XGetWindowAttributes(dsply,whichWindow,&graphWindowAttrib);
                          sprintf(viewport->controlPanel->message,
                                  "      >%d<: [%6.2f,%6.2f]      ",
                                  queriedGraph+1,
                                  projX((((XButtonEvent *)event)->x),
                                        graphWindowAttrib.width,queriedGraph),
                                  projY((((XButtonEvent *)event)->y),
                                        graphWindowAttrib.height,queriedGraph));
                          writeControlMessage();
                          XFlush(dsply);
                          break;
                        default:
                          /* Find where mouse is on the viewport => where to put the CP */
                          XGetWindowAttributes(dsply,whichWindow,&graphWindowAttrib);
                          mouseW4 = graphWindowAttrib.width/4;
                          if ((((XButtonEvent *)event)->x >
                               (graphWindowAttrib.width - mouseW4))
                            someInt = 1;
```

```
                              else {
                                mouseH4 = graphWindowAttrib.height/4;
                                if (((XButtonEvent *)event)->y >
                                    (graphWindowAttrib.height - mouseH4))
                                  someInt = 2;
                                else if (((XButtonEvent *)event)->x < mouseW4)
                                  someInt = 3;
                                else if (((XButtonEvent *)event)->y < mouseH4)
                                  someInt = 4;
                                else someInt = 0;
                              }
                              if (viewport->haveControl) {
                                XUnmapWindow(dsply,control->controlWindow);
                              }
                              putControlPanelSomewhere(someInt);
                              XMapWindow(dsply,control->controlWindow);
                              XSync(dsply,False);
                              break;
                            } /* switch on mouse button */
                          } else if (whichWindow == control->colormapWindow) {
                            /* mouse clicked on colormap */
                            followMouse = yes;
                            gotToggle   = no;
                            checkButton = yes;
                            firstTime   = yes;
                          } else if (whichWindow != control->controlWindow) {
                            /* mouse clicked on control window (not colormap) */
                            if (controlButton->self != whichWindow) {
                              buttonTablePtr = *((int *)XLookUpAssoc(dsply,table,whichWindow));
                              controlButton = &(control->buttonQueue[buttonTablePtr]);
                            }
                            if (controlButton->pot) {
                              /* figure out [x,y] for this button in the range [-1..1,-1..1] */
                              mouseXY = getPotValue(((XButtonEvent *)event)->x,
                                                    ((XButtonEvent *)event)->y,
                                                    controlButton->xHalf,
                                                    controlButton->yHalf);
                              followMouse = yes;
                              gotToggle = no;
                            } else {
                              followMouse = no;
                              gotToggle = yes;  /* auto-repeat on toggle buttons not allowed */
                              if (toggleReady) {
                                toggleReady = no;
                              }
                            }
```

```
          checkButton = yes;
          firstTime   = yes;
      }
      break;
  } /* switch */
  gotEvent--;
} /* if gotEvent */
/* Allow repeat polling when mouse button clicked on a potentiometer. */
if (followMouse && !firstTime && (followMouse++ > mouseWait)) {
  followMouse = yes;   /* reset for next timing loop */
  checkButton = yes;
}
if (checkButton) {
  if (viewport->closing && (controlButton->buttonKey == closeAll2D)) {
    goodbye(-1);
  } else {
    clearControlMessage();
    viewport->closing = no;
    drawControlPushButton(0, closeAll2D);
    if ((doit) &&
        ((controlButton->buttonKey < graphStart) &&
          (controlButton->buttonKey > (graphSelectStart + maxGraphs))))
      doit = 0;
    switch(controlButton->buttonKey) {
    case translate2D:
      for (i=0; i<maxGraphs; i++) {
        if (graphStateArray[i].showing && graphStateArray[i].selected) {
          if (transXON) {
            graphStateArray[i].centerX -= mouseXY.x * 0.1;
            if (graphStateArray[i].centerX > maxDelta)
              graphStateArray[i].centerX = maxDelta;
            else if (graphStateArray[i].centerX < -maxDelta)
              graphStateArray[i].centerX = maxDelta;
          }
          if (transYON) {
            graphStateArray[i].centerY -= mouseXY.y * 0.1;
            if (graphStateArray[i].centerY > maxDelta)
              graphStateArray[i].centerY = maxDelta;
            else if (graphStateArray[i].centerY < -maxDelta)
              graphStateArray[i].centerY = maxDelta;
          }
        } /* graph showing or selected */
      }  /* for graphs */
      drawViewport(Xoption);
      break;
    case scale2D:
```

```
                for (i=0; i<maxGraphs; i++) {
                  if (graphStateArray[i].showing && graphStateArray[i].selected) {
                    if (zoomXON) {
                      graphStateArray[i].scaleX *= (1 - mouseXY.y * 0.3);
                      if (graphStateArray[i].scaleX > maxScale)
                        graphStateArray[i].scaleX = maxScale;
                      else if (graphStateArray[i].scaleX < minScale)
                        graphStateArray[i].scaleX = minScale;
                    }
                    if (zoomYON) {
                      graphStateArray[i].scaleY *= (1 - mouseXY.y * 0.3);
                      if (graphStateArray[i].scaleY > maxScale)
                        graphStateArray[i].scaleY = maxScale;
                      else if (graphStateArray[i].scaleY < minScale)
                        graphStateArray[i].scaleY = minScale;
                    }
                  } /* graph showing or selected */
                } /* for graphs */
                drawViewport(Xoption);
                break;

            default:
              buttonAction(controlButton->buttonKey);
            } /* switch on buttonKey */
          } /* else - not closing */
        } /* if checkButton */
      } /* if FD_ISSET(Xcon.... */
      else if FD_ISSET(0,&rd) {
        externalControl=spadAction(); /* returns (-1) if broken ,0 if success */
        if (spadDraw && (externalControl==0)) drawViewport(Xoption);
      }
    } /* while */
  } /* processEvents() */
```

6.3.20 clickedOnGraph

⟨*view2d*⟩+≡

```
  void clickedOnGraph(int i,int bKey) {
    switch (doit) {
    case pick2D:
      currentGraph = queriedGraph = i;
      doPick(i,bKey);
      break;
    case drop2D:
      currentGraph = queriedGraph = i;
      doDrop(i,bKey);
      break;
    case query2D:
      queriedGraph = i;
      makeMessageFromData(queriedGraph);
      writeControlMessage();
      /* reset indicator so that we're not in pick/drop/query mode anymore */
      doit = 0;
      break;
    default:
      graphStateArray[i].showing = !(graphStateArray[i].showing);
      if (mono) {
        if (graphStateArray[i].showing) {
          GSetForeground(graphGC,(float)backgroundColor,Xoption);
          GSetBackground(graphGC,(float)foregroundColor,Xoption);
        } else {
          GSetForeground(graphGC,(float)foregroundColor,Xoption);
          GSetBackground(graphGC,(float)backgroundColor,Xoption);
        }
        GDrawImageString(graphGC,
                control->controlWindow,
                (control->buttonQueue[bKey]).buttonX +
                centerX(graphGC,(control->buttonQueue[bKey]).text,1,
                        (control->buttonQueue[bKey]).buttonWidth),
                (control->buttonQueue[bKey]).buttonY +
                centerY(graphGC,(control->buttonQueue[bKey]).buttonHeight),
                (control->buttonQueue[bKey]).text,
                1,
                Xoption);
      } else {
        if (graphStateArray[i].showing)
          GSetForeground(graphGC,(float)graphBarShowingColor,Xoption);
        else
          GSetForeground(graphGC,(float)graphBarHiddenColor,Xoption);
        GDrawString(graphGC,
```

```
                  control->controlWindow,
                  (control->buttonQueue[bKey]).buttonX +
                  centerX(graphGC,(control->buttonQueue[bKey]).text,1,
                        (control->buttonQueue[bKey]).buttonWidth),
                  (control->buttonQueue[bKey]).buttonY +
                  centerY(graphGC,(control->buttonQueue[bKey]).buttonHeight),
                  (control->buttonQueue[bKey]).text,1,Xoption);
      }
      drawViewport(Xoption);
      break;
    } /* switch doit */
  }
```

6.3.21   readViewman

⟨*view2d*⟩+≡

```
  int readViewman(void * info,int size) {
    int mold = 0;
    sprintf(errorStr,"%s %d %s","read of ",size,
            " bytes from viewport manager\n");
    mold = check(read(0,info,size));
    return(mold);
  }
```

6.3.22 spadAction

⟨*view2d*⟩+≡

```
int spadAction(void) {
  int code,viewCommand;
  float f1,f2;
  int i1,i2,i3,viewGoAhead;
  static int ack = 1;
  if (viewAloned==yes) {
     close(0);
     return(-1);
     }
  readViewman(&viewCommand,intSize);
  switch (viewCommand) {
  case hideControl2D:
    readViewman(&i1,intSize);
    if (i1) {                           /* show control panel */
      if (viewport->haveControl) XUnmapWindow(dsply,control->controlWindow);
      putControlPanelSomewhere(someInt);
    } else {    /* turn off control panel */
      if (viewport->haveControl) {
        viewport->haveControl = no;
        XUnmapWindow(dsply,control->controlWindow);
      }
    }
    break;
  case changeTitle:
    readViewman(&i1,intSize);
    readViewman(viewport->title,i1);
    viewport->title[i1] = '\0';
    writeTitle();
    writeControlTitle();
    XFlush(dsply);
    spadDraw=no;
    break;
  case writeView:
    readViewman(&i1,intSize);
    readViewman(filename,i1);
    filename[i1] = '\0';
    sprintf(errorStr,"writing of viewport data");
    i3 = 0;
    readViewman(&i2,intSize);
    while (i2) {
      i3 = i3 | (1<<i2);
      readViewman(&i2,intSize);
    }
```

```
      if (writeViewport(i3) < 0)
        fprintf(stderr,"          Nothing was written\n");
      break;
    case closeAll2D:
      code = check(write(Socket,&ack,intSize));
      goodbye(-1);
    case ps2D:
      readViewman(&i1,intSize);
      buttonAction(viewCommand);
      break;
    case axesOnOff2D:
      readViewman(&i1,intSize);
      i1--;
      readViewman(&i2,intSize);
      graphStateArray[i1].axesOn = i2;
      if (graphStateArray[i1].showing) spadDraw=yes;
      break;
    case axesColor2D:
      readViewman(&i1,intSize);
      i1--;
      readViewman(&i2,intSize);
      graphStateArray[i1].axesColor = i2;
      if (graphStateArray[i1].showing) spadDraw=yes;
      break;
    case unitsOnOff2D:
      readViewman(&i1,intSize);
      i1--;
      readViewman(&i2,intSize);
      graphStateArray[i1].unitsOn = i2;
      if (graphStateArray[i1].showing) spadDraw=yes;
      break;
    case unitsColor2D:
      readViewman(&i1,intSize);
      i1--;
      readViewman(&i2,intSize);
      graphStateArray[i1].unitsColor = i2;
      if (graphStateArray[i1].showing) spadDraw=yes;
      break;
    case connectOnOff:
      readViewman(&i1,intSize);
      i1--;
      readViewman(&i2,intSize);
      graphStateArray[i1].connectOn = i2;
      if (graphStateArray[i1].showing) spadDraw=yes;
      break;
    case pointsOnOff:
```

```
      readViewman(&i1,intSize);
      i1--;
      readViewman(&i2,intSize);
      graphStateArray[i1].pointsOn = i2;
      if (graphStateArray[i1].showing) spadDraw=yes;
      break;
    case spline2D:
      readViewman(&i1,intSize);
      i1--;
      readViewman(&i2,intSize);
      graphStateArray[i1].splineOn = i2;
      if (graphStateArray[i1].showing) spadDraw=yes;
      break;
    case showing2D:
      readViewman(&i1,intSize);
      i1--;
      readViewman(&i2,intSize);
      /* simulate a button press to turn display number on/off */
      graphStateArray[i1].showing = !i2;
      clickedOnGraph(i1,i1+graphStart);
      break;
    case scale2D:
      readViewman(&i1,intSize);
      i1--;    /* passed index is [1..9] but internal representation is [0..8] */
      readViewman(&f1,floatSize);
      readViewman(&f2,floatSize);
      graphStateArray[i1].scaleX = f1;
      graphStateArray[i1].scaleY = f2;
      if (graphStateArray[i1].scaleX > maxScale)
        graphStateArray[i1].scaleX = maxScale;
      else
        if (graphStateArray[i1].scaleX < minScale)
          graphStateArray[i1].scaleX = minScale;
      if (graphStateArray[i1].scaleY > maxScale)
        graphStateArray[i1].scaleY = maxScale;
      else
        if (graphStateArray[i1].scaleY < minScale)
          graphStateArray[i1].scaleY = minScale;
      if (graphStateArray[i1].showing) spadDraw=yes;
      break;    /* scale2D */
    case translate2D:
      readViewman(&i1,intSize);
      i1--;    /* passed index is [1..9] but internal representation is [0..8] */
      readViewman(&f1,floatSize);
      readViewman(&f2,floatSize);
      graphStateArray[i1].centerX = f1;
```

```
      graphStateArray[i1].centerY = f2;
      if (graphStateArray[i1].centerX > maxDelta)
        graphStateArray[i1].centerX = maxDelta;
      else if (graphStateArray[i1].centerX < -maxDelta)
            graphStateArray[i1].centerX = maxDelta;
      if (graphStateArray[i1].centerY > maxDelta)
        graphStateArray[i1].centerY = maxDelta;
      else if (graphStateArray[i1].centerY < -maxDelta)
            graphStateArray[i1].centerY = maxDelta;
      if (graphStateArray[i1].showing) spadDraw=yes;
      break;    /* translate2D */
   case moveViewport:
     readViewman(&i1,intSize);
     readViewman(&i2,intSize);
     XMoveWindow(dsply,viewport->titleWindow,i1,i2);
     XSync(dsply,False);
     break;
  case resizeViewport:
     readViewman(&i1,intSize);
     readViewman(&i2,intSize);
     XResizeWindow(dsply,viewport->titleWindow,i1,i2+titleHeight);
     XResizeWindow(dsply,viewport->viewWindow,i1,i2);
     spadDraw=yes;
     break;
   case putGraph:
     readViewman(&i1,intSize);                /* key of graph to get */
     readViewman(&i2,intSize);                /* slot to drop graph onto 0..8*/
     readViewman(&viewGoAhead,intSize);
     if (viewGoAhead < 0) {
       sprintf(control->message,"%s%d","Couldn't put into graph ",i2+1);
       writeControlMessage();
     } else {
       sprintf(control->message,"%s%d","Dropped onto graph ",i2+1);
       writeControlMessage();
       freeGraph(i2);
       graphArray[i2].key = i1;
       getGraphFromViewman(i2);
       /* simulate a button press to turn display number on and select on */
       /* need !yes since it will be inverted */
       graphStateArray[i2].selected = no;
       graphStateArray[i2].connectOn = yes;
       graphStateArray[i2].showing = !(graphStateArray[i2].showing);
       clickedOnGraph(i2,i2+graphStart);
       clickedOnGraphSelect(i2,i2+graphSelectStart);
     }
     break;
```

```
    case spadPressedAButton:
      readViewman(&i1,intSize);
      buttonAction(i1);
      break;
    default:
      return(-1);
    } /* switch */
    ack++;
    code = check(write(Socket,&ack,intSize));
    return(0);
  }
```

6.3.23   absolute

⟨*view2d*⟩+≡
```
  float absolute(float x) {
    if (x<0.0) {
      return(-x);
    } else {
      return(x);
    }
  }
```

6.3.24   goodbye

⟨*view2d*⟩+≡

```
  void goodbye(int sig) {
    int Command,i;
#ifdef DEBUG
    fprintf(stderr,"view2d: Tidying up and exiting\n");
#endif
    PSClose(); /* free PS file and data structure space */
    XFreeGC(dsply,globalGC1);
    XFreeGC(dsply,globalGC2);
    XFreeGC(dsply,globGC);
    XFreeGC(dsply,trashGC);
    XFreeGC(dsply,anotherGC);
    XFreeGC(dsply,controlMessageGC);
    XFreeGC(dsply,graphGC);
    XFreeGC(dsply,unitGC);
    XFreeFont(dsply,globalFont);
    XFreeFont(dsply,buttonFont);
    XFreeFont(dsply,headerFont);
    XFreeFont(dsply,titleFont);
    XFreeFont(dsply,graphFont);
    XFreeFont(dsply,unitFont);
    XFreeColormap(dsply,colorMap);
    /** send off the current graphs to viewport manager **/
    Command = viewportClosing;
    check(write(Socket,&Command,intSize));
    for (i=0; i<maxGraphs;i++) {
      check(write(Socket,&graphArray[i].key,intSize));
    }
    close(Socket);
    XCloseDisplay(dsply);
    exit(0);
  }
```

6.3.25 writeTitle

⟨*view2d*⟩+≡

```
void writeTitle(void) {
  int strlength;
  XWindowAttributes attribInfo;
  XGetWindowAttributes(dsply,viewport->titleWindow,&attribInfo);
  if (mono) GSetForeground(anotherGC,(float)foregroundColor,Xoption);
  else GSetForeground(anotherGC,(float)titleColor,Xoption);
  XClearWindow(dsply,viewport->titleWindow); /* it's behind the viewWindow */
  strlength = strlen(viewport->title);
  GDrawImageString(anotherGC,viewport->titleWindow,
            centerX(anotherGC,viewport->title,strlength,attribInfo.width),
            15,viewport->title,strlength,Xoption);
}
```

6.3.26   drawTheViewport

⟨*view2d*⟩+≡

```
void drawTheViewport(int dFlag) {
  Window              vw;
  XWindowAttributes   vwInfo;
  pointListStruct     *aList;
  pointStruct         *aPoint;
  XPoint              *anXPoint,*tempXpt;
  XArc                *anXarc;
  Vertex              *anX10Point;
  float               jj,diffX, diffY, tickStart,oneTickUnit;
  int                 i,j,k,ii,halfSize;
  int                 charlength,strlength,halflength,halfheight;
  int                 ptX,ptY,ptX1,ptY1,clipped, clipped1;
  int                 xAxis,yAxis,dummyInt, ascent, descent;
  int                 unitWidth,boxX,boxY,boxW,boxH;
  char                aunit[20];
  XCharStruct         overall;
  drawMore = yes;
  vw = viewport->viewWindow;
  XGetWindowAttributes(dsply,vw,&vwInfo);
  aspectR = (float)vwInfo.width/(float)vwInfo.height;
  XTextExtents(unitFont,"o",1,&dummyInt,&ascent,&descent,&overall);
  halfheight = (ascent + descent) / 2;
  /* Calculate various factors for use in projection. */
  /* Scale the plot, so that the scaling between the axes remains
     constant and fits within the smaller of the two dimensions. */
  charlength = overall.width;
  if (dFlag==Xoption) XClearWindow(dsply,vw);
  for (i=0; i<maxGraphs; i++) {
    if ((graphArray[i].key) && (graphStateArray[i].showing)) {
      /* Scale y coordinate dimensions relative to viewport aspect ratio. */
      graphArray[i].yNorm = 1.0/((graphArray[i].ymax-graphArray[i].ymin) *
                                 aspectR);
      graphArray[i].originY = -graphArray[i].ymin*graphArray[i].yNorm
        - 0.5/aspectR;
      graphArray[i].unitY = graphArray[i].spadUnitY*graphArray[i].yNorm;
      xAxis = rint(vwInfo.width *
                   ((graphArray[0].originX - graphStateArray[0].centerX) *
                    graphStateArray[0].scaleX + 0.5));
      yAxis= rint(vwInfo.height * aspectR *
                  (1 - ((graphArray[0].originY*aspectR -
                         graphStateArray[0].centerY) *
                        graphStateArray[0].scaleY + 0.5*aspectR )));
      if (graphStateArray[i].axesOn) {
```

```
                  if (dFlag==Xoption) /* do only for X, ps uses default of black */
                    GSetForeground(globalGC1,
                                   (float)monoColor(graphStateArray[i].axesColor),
                                   dFlag);
                  if ((yAxis >=0) && (yAxis <= vwInfo.height))
                    GDrawLine(globalGC1,vw,
                              0,yAxis,
                              vwInfo.width,yAxis,
                              dFlag);
                  if ((xAxis >=0) && (xAxis <= vwInfo.width))
                    GDrawLine(globalGC1,vw,
                              xAxis,0,
                              xAxis,vwInfo.height,
                              dFlag);
              }
              tempXpt   = anXPoint   = xPointsArray[i].xPoint;
              anX10Point = xPointsArray[i].x10Point;
              anXarc     = xPointsArray[i].arc;
              for (j=0,aList=graphArray[i].listOfListsOfPoints;
                   (j<graphArray[i].numberOfLists);
                   j++, aList++) {
                for (k=0,aPoint=aList->listOfPoints;
                     (k<aList->numberOfPoints);
                     k++,aPoint++) {
                  if (graphStateArray[i].scaleX > 99.0)
                    graphStateArray[i].scaleX = 99.0;
                  if (graphStateArray[i].scaleY > 99.0)
                    graphStateArray[0].scaleY = 99.0;
                  if (i > 0) {
                    if (isNaN(aPoint->x)) {
                      anXPoint->x = anX10Point->x = NotPoint;
                    }
                    else {
                      diffX = graphArray[i].xmax-graphArray[i].xmin;
                      anXPoint->x = anX10Point->x = vwInfo.width *
                        ((aPoint->x * diffX/(graphArray[0].xmax-graphArray[0].xmin)
                          + (graphArray[0].originX - graphArray[i].originX*diffX /
                             (graphArray[0].xmax-graphArray[0].xmin))
                          - graphStateArray[0].centerX)*graphStateArray[i].scaleX+0.5);
                    }
                    if (isNaN(aPoint->y)) {
                      anXPoint->y = anX10Point->y = NotPoint;
                    }
                    else {
                      diffY = graphArray[i].ymax-graphArray[i].ymin;
                      anXPoint->y = anX10Point->y = vwInfo.height * aspectR *
```

```
                            (1 - ((aPoint->y * diffY/(graphArray[0].ymax-graphArray[0].ymin)
                                + (graphArray[0].originY - graphArray[i].originY* diffY/
                                   (graphArray[0].ymax-graphArray[0].ymin))*aspectR
                                - graphStateArray[0].centerY) *
                            graphStateArray[i].scaleY + 0.5*aspectR));
                }
              } else {
                if (isNaN(aPoint->x)) {
                  anXPoint->x = anX10Point->x = NotPoint;
                }
                else {
                  anXPoint->x = anX10Point->x = vwInfo.width *
                    ((aPoint->x - graphStateArray[i].centerX) *
                     graphStateArray[i].scaleX + 0.5);
                }
                if (isNaN(aPoint->y)) {
                  anXPoint->y = anX10Point->y = NotPoint;
                }
                else {
                  anXPoint->y = anX10Point->y = vwInfo.height * aspectR *
                    (1 - ((aPoint->y - graphStateArray[i].centerY) *
                          graphStateArray[i].scaleY + 0.5*aspectR));
                }
              }

              /* first or last point */
              if (k == 0 || k == (aList->numberOfPoints - 1)) {
                anX10Point->flags = 0;
              } else {
                anX10Point->flags = VertexCurved;
              }

              anXPoint++;
              anX10Point++;
              anXarc++;
            }       /* for aPoint in pointList */

          aPoint--; /* make it legal, the last one*/
          if (graphStateArray[i].connectOn || graphStateArray[i].pointsOn) {
            halfSize = aList->pointSize/2;
            ptX = tempXpt->x;
            ptY = tempXpt->y;
            clipped = ptX > vwInfo.x && ptX < vwInfo.width &&
              ptY > 0 && ptY < vwInfo.height;
            if (graphStateArray[i].pointsOn) {
              if (dFlag==Xoption) {
```

```
          if (mono) {
            GSetForeground(globalGC1,
                          (float)monoColor((int)(aPoint->hue)),
                          dFlag);
          } else {
            GSetForeground(globalGC1,
                          (float)XSolidColor((int)(aPoint->hue),
                                            (int)(aPoint->shade)),
                          dFlag);
          }
        }
        if (clipped && !eqNANQ(ptX) && !eqNANQ(ptY))
          GFillArc(globalGC1,vw,ptX-halfSize,
                  ptY-halfSize,aList->pointSize,aList->pointSize,
                  0,360*64, dFlag);

    } /* if points on */
    for (ii=0, aPoint=aList->listOfPoints;
         ii<aList->numberOfPoints;
         ++ii, ++tempXpt, ++aPoint) {
      ptX1 = tempXpt->x;
      ptY1 = tempXpt->y;
      clipped1 = ptX1 > vwInfo.x && ptX1 < vwInfo.width &&
        ptY1 > 0 && ptY1 < vwInfo.height;
      if (graphStateArray[i].connectOn) {
        if (dFlag==Xoption) {
          if (mono) {
            GSetForeground(globalGC1,
                    (float)monoColor((int)(aList->lineColor-1)/5),
                    dFlag);
          } else {
            GSetForeground(globalGC1,
                    (float)XSolidColor((int)(aList->lineColor-1)/5,
                                      (int)((aList->lineColor-1)%5)/2),
                          dFlag);
          }
        } /* if X */
        if ((clipped || clipped1) && !eqNANQ(ptX) && !eqNANQ(ptY) &&
            !eqNANQ(ptX1) && !eqNANQ(ptY1))
          GDrawLine(globalGC1,vw,
                    ptX,ptY,ptX1,ptY1,
                    dFlag);
      } /* if lines on */
      if (graphStateArray[i].pointsOn) {
        if (dFlag==Xoption) {
          if (mono) {
```

```
                         GSetForeground(globalGC1,
                                        (float)monoColor((int)(aPoint->hue)),
                                        dFlag);
                       } else {
                         GSetForeground(globalGC1,
                                        (float)XSolidColor((int)(aPoint->hue),
                                                           (int)(aPoint->shade)),
                                        dFlag);
                       }
                     }
                     if (clipped1 && !eqNANQ(ptX1) && !eqNANQ(ptY1))
                       GFillArc(globalGC1,vw,ptX1-halfSize,
                                ptY1-halfSize,aList->pointSize,aList->pointSize,
                                0,360*64, dFlag);
                 } /* if points on */
                 ptX = ptX1;  ptY = ptY1;  clipped = clipped1;
               } /* for all points */
             } /* if points or lines on */

             if (graphStateArray[i].splineOn) {   /* need spline color as well */
               if (dFlag==Xoption)  /* do only for X, ps uses default of black */
                 GSetForeground(globalGC1,
                                (float)monoColor(148),
                                dFlag);
               boxX = vwInfo.width *
                 ((-0.5 - graphStateArray[i].centerX)*
                  graphStateArray[i].scaleX + 0.5);
               boxY = vwInfo.height * aspectR *
                 (1 - ((0.5 - graphStateArray[i].centerY)*
                       graphStateArray[i].scaleY + 0.5*aspectR));
               boxW = graphStateArray[i].scaleX * vwInfo.width + 1;
               boxH = graphStateArray[i].scaleY * vwInfo.height * aspectR + 1;
               GDrawRectangle(globalGC1,vw,
                              boxX,boxY,boxW,boxH,
                              dFlag);
             }
             tempXpt = anXPoint;
         }     /* for a aList in listofListsOfPoints */
         if (graphStateArray[i].unitsOn) {
           /* do only for X, ps uses default of black */
           if (dFlag==Xoption)
             GSetForeground(unitGC,
                            (float)monoColor(graphStateArray[i].unitsColor),
                            dFlag);
           tickStart  = calcUnitX(0);
           oneTickUnit = calcUnitX(1) - tickStart;
```

```
/* ticks along the positive X axis */
/* limit on acceptable separation : 5 chars */
unitWidth = 5*overall.width;
k = floor(unitWidth/oneTickUnit) +1;     /* get skipping integer */
for (ii=0, jj = tickStart;
      jj < vwInfo.width;
      ii=ii+k,jj =jj+k* oneTickUnit) {
  if (jj >= 0) {

    /* ticks stuck to viewport*/
    GDrawLine(unitGC,vw,
      (int)rint(jj),vwInfo.height-8,(int)rint(jj),vwInfo.height-4,
      dFlag);
    sprintf(aunit,"%0.3g",ii*graphArray[0].spadUnitX);
    strlength=strlen(aunit);
    halflength=XTextWidth(unitFont,aunit,strlength)/2;
    if (dFlag == Xoption)
      GDrawImageString(unitGC,vw,(int)rint(jj) - halflength,
                        vwInfo.height-8-descent,aunit,strlength,dFlag);
    if (dFlag == PSoption)
      GDrawImageString(unitGC,vw,(int)rint(jj) -(strlength*3),
                        vwInfo.height-14,aunit,strlength,dFlag);
    /* these are "eyeball" parameters for the given PS font */
  }
}
/* ticks along the negative X axis */
for (ii=-k,jj=tickStart - k*oneTickUnit;
      jj > 0;
      ii=ii-k,jj = jj-k*oneTickUnit) {
  if (jj <= vwInfo.width) {
    /* ticks stuck to viewport*/
    GDrawLine(unitGC,vw,(int)rint(jj),vwInfo.height-8,(int)rint(jj),
              vwInfo.height-4,dFlag);
    sprintf(aunit,"%0.3g",ii*graphArray[0].spadUnitX);
    strlength=strlen(aunit);
    halflength=XTextWidth(unitFont,aunit,strlength)/2;
    if (dFlag == Xoption)
      GDrawImageString(unitGC,vw,(int)rint(jj) - halflength,
                        vwInfo.height-8-descent,aunit,strlength,dFlag);
    if (dFlag == PSoption)
      GDrawImageString(unitGC,vw,(int)rint(jj) -(strlength*3),
                        vwInfo.height -14,aunit,strlength,dFlag);
    /* these are "eyeball" parameters for the given PS font */
  }
}
tickStart = calcUnitY(0);
```

```
oneTickUnit = calcUnitY(1) - tickStart;
/* ticks along the positive Y axis */
unitWidth = 2*(ascent+descent);  /* limit of acceptable separation */
k = floor(unitWidth/fabs(oneTickUnit)) +1;  /* get skipping integer */
for (ii=0,jj = tickStart;
     jj > 0;
     ii=ii+k,jj =jj+k*oneTickUnit ) {
  if  (jj < vwInfo.height) {
    /* ticks stuck to viewport*/
    /* on the right */
    /*
      GDrawLine(unitGC,vw,
      vwInfo.width-6,(int)rint(jj),
      vwInfo.width-2,(int)rint(jj),dFlag);
      */
    /* on the left */
    GDrawLine(unitGC,vw,
              2,(int)rint(jj),
              6,(int)rint(jj),
              dFlag);
    sprintf(aunit,"%0.3g",ii*graphArray[0].spadUnitY);
    strlength=strlen(aunit);
    XTextExtents(unitFont,aunit,strlength,&dummyInt,
                 &ascent,&descent,&overall);
    halflength=overall.width;              /* let's reuse that variable */
    if(dFlag == Xoption){
      /* on the right */
      /*
        GDrawImageString(unitGC, vw,
        vwInfo.width-halflength -6-descent,
        (int)rint(jj)+ascent/2 ,
        aunit, strlength, dFlag);
        */
      /* on the left */
      GDrawImageString(unitGC, vw,
                       8 + charlength/2,
                       (int)rint(jj)+ascent/2 ,
                       aunit, strlength, dFlag);
    }
    if(dFlag == PSoption){
      /* on the right */
      /*
        GDrawImageString(unitGC, vw,
        vwInfo.width - 6 - (strlength*6),
        (int)rint(jj)+4,
        aunit, strlength, dFlag);
```

```
                    */
                 /* on the left */
                 GDrawImageString(unitGC, vw,
                                  8,(int)rint(jj)+4,
                                  aunit, strlength, dFlag);
                 /* these are "eyeball" parameters for the given PS font */
              }
            }
          }
          /* ticks along the negative Y axis */
          for (ii=(-k),jj = tickStart - k*oneTickUnit;
               jj < vwInfo.height;
               ii=ii-k,jj =jj-k*oneTickUnit) {
            if (jj > 0) {

              /* ticks stuck to viewport*/
              /* on the right */
              /*
                GDrawLine(unitGC,vw,
                vwInfo.width-6,(int)rint(jj),
                vwInfo.width-2,(int)rint(jj),
                dFlag);
                */
              /* on the left */
              GDrawLine(unitGC,vw,
                        2,(int)rint(jj),
                        6,(int)rint(jj),
                        dFlag);

              sprintf(aunit,"%0.3g",ii*graphArray[0].spadUnitY);
              strlength=strlen(aunit);
              XTextExtents(unitFont,aunit,strlength,&dummyInt,
                           &ascent,&descent,&overall);
              halflength=overall.width;            /* let's reuse that variable */
              if(dFlag == Xoption){
                /* on the right */
                /*
                  GDrawImageString(unitGC, vw,
                  vwInfo.width-halflength -6-descent,
                  (int)rint(jj)+ascent/2 ,
                  aunit, strlength, dFlag);
                  */
                /* on the left */
                GDrawImageString(unitGC, vw,
                                 8 + charlength/2,
                                 (int)rint(jj)+ascent/2 ,
```

```
                                                 aunit, strlength, dFlag);
                }
                if(dFlag == PSoption){
                  /* on the right */
                  /*
                    GDrawImageString(unitGC, vw,
                    vwInfo.width -6 -(strlength*6),
                    (int)rint(jj)+4 ,
                    aunit, strlength, dFlag);
                    */
                  /* on the left */
                  GDrawImageString(unitGC, vw,
                                   8,
                                   (int)rint(jj)+4 ,
                                   aunit, strlength, dFlag);
                  /* these are "eyeball" parameters for the given PS font */
                }
              }
            }
        }  /* if unitsOn */
      }    /* if graph i exists and is showing */
    }   /* for i in graphs */
    if (dFlag==Xoption) {
      if (!followMouse) {
        /* no need to do this while autorepeating */
        makeMessageFromData(queriedGraph);
        writeControlMessage();
      }
      XFlush(dsply);
    }
}             /* drawViewport() */
```

6.3.27 makeViewport

⟨*view2d*⟩+≡

```
viewPoints *makeViewport(char *title,int vX,int vY,int vW,int vH,int showCP) {
  Pixmap              spadbits,spadmask;
  XSetWindowAttributes viewAttrib;
  XSizeHints          titleSizeHints,viewSizeHints;
  Window              viewTitleWindow,viewGraphWindow;
  XColor              foreColor, backColor;
#ifdef DEBUG
  fprintf(stderr,"view2d: About to make a viewport\n");
#endif
  /* Create a viewport */
  if (!(viewport = (viewPoints *)malloc(sizeof(viewPoints)))) {
   fprintf(stderr,"Ran out of memory (malloc) trying to create a viewport.\n");
   sleep(5);
   exitWithAck(RootWindow(dsply,scrn),Window,-1);
  }
#ifdef DEBUG
  fprintf(stderr,"view2d: Made a viewport\n");
#endif
  strcpy(viewport->title,title);
  viewport->closing     = no;
  viewport->allowDraw   = yes;   /* just draw axes the first time around */
  viewport->axesOn    = axesON;
  viewport->unitsOn   = unitsON;
  viewport->pointsOn  = pointsON;
  viewport->linesOn   = connectON;
  viewport->splineOn  = splineON;
  /**** Make the windows for the viewport ****/
  spadbits = XCreateBitmapFromData(dsply,rtWindow,
                                   spadBitmap_bits,
                                   spadBitmap_width,spadBitmap_height);
  spadmask = XCreateBitmapFromData(dsply,rtWindow,
                                   spadMask_bits,
                                   spadMask_width,spadMask_height);
  viewAttrib.background_pixel = backgroundColor;
  viewAttrib.border_pixel = foregroundColor;
  viewAttrib.override_redirect = overrideManager;
  viewAttrib.colormap = colorMap;
  foreColor.pixel = foregroundColor;
  backColor.pixel = backgroundColor;
  XQueryColor(dsply,colorMap,&foreColor);
  XQueryColor(dsply,colorMap,&backColor);
  viewAttrib.cursor = XCreatePixmapCursor(dsply,spadbits,spadmask,
                &foreColor,&backColor,spadBitmap_x_hot,spadBitmap_y_hot);
```

```
    viewAttrib.event_mask = titleMASK;
    if (vW) {
      titleSizeHints.flags  = PPosition | PSize;
      titleSizeHints.x       = vX;
      titleSizeHints.y       = vY;
      titleSizeHints.width   = vW;
      titleSizeHints.height = vH;
    } else {
      titleSizeHints.flags  = PSize;
      titleSizeHints.width   = viewWidth;
      titleSizeHints.height = viewHeight;
    }
    viewTitleWindow = XCreateWindow(dsply,rtWindow,vX,vY,vW,vH,
                                        viewBorderWidth,
                                        CopyFromParent,InputOutput,CopyFromParent,
                                        viewportTitleCreateMASK,&viewAttrib);
    wm_delete_window = XInternAtom(dsply, "WM_DELETE_WINDOW", False);
    (void) XSetWMProtocols(dsply, viewTitleWindow, &wm_delete_window, 1);
    XSetNormalHints(dsply,viewTitleWindow,&titleSizeHints);
    XSetStandardProperties(dsply,viewTitleWindow,"Axiom 2D",viewport->title,
                            None,NULL,0,&titleSizeHints);
    viewport->titleWindow = viewTitleWindow;
    viewAttrib.event_mask = viewportMASK;
    viewSizeHints.flags   = PPosition | PSize;
    viewSizeHints.x        = -viewBorderWidth;
    viewSizeHints.y        = titleHeight;
    viewSizeHints.width    = titleSizeHints.width;
    viewSizeHints.height  = titleSizeHints.height -
                            (titleHeight + appendixHeight);
    viewGraphWindow = XCreateWindow(dsply,viewTitleWindow,
                                        viewSizeHints.x,viewSizeHints.y,
                                        viewSizeHints.width,viewSizeHints.height,
                                        viewBorderWidth,
                                        CopyFromParent,InputOutput,CopyFromParent,
                                        viewportCreateMASK,&viewAttrib);
    XSetNormalHints(dsply,viewGraphWindow,&viewSizeHints);
    XSetStandardProperties(dsply,viewGraphWindow,"2D Viewport","2D Viewport",
                            None,NULL,0,&viewSizeHints);
    viewport->viewWindow = viewGraphWindow;
    /*Make the control panel for the viewport. */
    viewport->controlPanel = makeControlPanel();
    if ((viewport->haveControl = showCP)) putControlPanelSomewhere(anywhere);
    XSync(dsply,False);
    return(viewport);
}
```

6.3.28   makeView2D

⟨*view2d*⟩+≡

```
viewPoints *makeView2D(view2DStruct *viewdata) {
  viewPoints *vPoints;
  vPoints = makeViewport(viewdata->title, viewdata->vX,viewdata->vY,
                         viewdata->vW,viewdata->vH,viewdata->showCP);
  vPoints->allowDraw = yes;   /* draw everything from now on */
  if (viewdata->showCP) clearControlMessage();
  writeTitle();
  XMapWindow(dsply,vPoints->viewWindow);
  XMapWindow(dsply,vPoints->titleWindow);
  XSync(dsply,0);
  drawViewport(Xoption); /* draw viewport with X routines (as opposed to PS) */
  return(vPoints);
}    /* makeView2D */
```

6.3.29   writeViewport

⟨*view2d*⟩+≡

```
int writeViewport(int thingsToWrite) {
  FILE              *viewDataFile;
  char              viewDirName[80],
                    viewBitmapFilename[80],viewDataFilename[80],command[80];
  int               i,j,k,code,ii;
  pointListStruct   *aList;
  pointStruct       *aPoint;
  XWindowAttributes vwInfo;
  XGetWindowAttributes(dsply,viewport->titleWindow,&vwInfo);
  sprintf(viewDirName,"%s%s",filename,".view");
  sprintf(command,"%s%s%s","rm -r ",viewDirName," >  /dev/null 2>&1");
  code = system(command);
  sprintf(command,"%s%s%s","mkdir ",viewDirName," > /dev/null 2>&1");
  if (system(command)) {
    fprintf(stderr,"  Error: Cannot create %s\n",viewDirName);
    return(-1);
  } else {
          /*** Create the data file ***/
    sprintf(viewDataFilename,"%s%s",viewDirName,"/data");
    if ((viewDataFile = fopen(viewDataFilename,"w")) == NULL) {
      fprintf(stderr,"  Error: Cannot create %s\n",viewDataFilename);
      perror("fopen");
      return(-1);
    } else {
            /*** write out the view2DStruct stuff ***/
      fprintf(viewDataFile,"%d\n",view2DType);
      fprintf(viewDataFile,"%s\n",viewport->title);
      fprintf(viewDataFile,"%d %d %d %d\n",vwInfo.x,vwInfo.y,
            vwInfo.width,vwInfo.height);
      for (i=0; i<maxGraphs; i++) {
        fprintf(viewDataFile,"%d\n",graphArray[i].key);
        fprintf(viewDataFile,"%g %g\n",
              graphStateArray[i].scaleX,graphStateArray[i].scaleY);
        fprintf(viewDataFile,"%g %g\n",
              graphStateArray[i].deltaX,graphStateArray[i].deltaY);
        fprintf(viewDataFile,"%g %g\n",
              graphStateArray[i].centerX,graphStateArray[i].centerY);
        fprintf(viewDataFile,"%d %d %d %d %d %d %d\n",
              graphStateArray[i].pointsOn,graphStateArray[i].connectOn,
              graphStateArray[i].splineOn,
              graphStateArray[i].axesOn, graphStateArray[i].axesColor,
              graphStateArray[i].unitsOn, graphStateArray[i].unitsColor);
        fprintf(viewDataFile,"%d %d\n",
```

```
                graphStateArray[i].showing,graphStateArray[i].selected);
      }
      fclose(viewDataFile);
      for (i=0; i<maxGraphs; i++) {
        if (graphArray[i].key) {
          sprintf(viewDataFilename,"%s%s%d",viewDirName,"/graph",i);
          if ((viewDataFile = fopen(viewDataFilename,"w")) == NULL) {
            fprintf(stderr,"   Error: Cannot create %s\n",viewDataFilename);
            perror("fopen");
            return(-1);
          } else {
            fprintf(viewDataFile,"%g %g %g %g\n",
                    graphArray[i].xmin,graphArray[i].ymin,
                    graphArray[i].xmax,graphArray[i].ymax);
            fprintf(viewDataFile,"%g %g\n",
                    graphArray[i].xNorm,graphArray[i].yNorm);
            fprintf(viewDataFile,"%g %g\n",
                    graphArray[i].originX,graphArray[i].originY);
            fprintf(viewDataFile,"%g %g\n",
                    graphArray[i].spadUnitX,graphArray[i].spadUnitY);
            fprintf(viewDataFile,"%g %g\n",
                    graphArray[i].unitX,graphArray[i].unitY);
            fprintf(viewDataFile,"%d\n",graphArray[i].numberOfLists);
            for (j=0,aList=graphArray[i].listOfListsOfPoints;
                  j<graphArray[i].numberOfLists;
                  j++, aList++) {
              fprintf(viewDataFile,"%d\n",aList->numberOfPoints);
              fprintf(viewDataFile,"%d %d %d\n",
                      aList->pointColor,aList->lineColor,aList->pointSize);
              for (k=0,aPoint=aList->listOfPoints;
                    k<aList->numberOfPoints;
                    k++,aPoint++)
                fprintf(viewDataFile,"%g %g %g %g\n",
                        aPoint->x,aPoint->y,aPoint->hue,aPoint->shade);
            } /* for j, aList */
            fclose(viewDataFile);
          } /* else graph i */
        } /* if */
      } /* for */
  } /* else */
        /* write out special files */
  for (ii=1; ii<numBits; ii++) {   /* write.h is one-based */
    if (thingsToWrite & (1<<ii)) {
      switch (ii) {
      case aPixmap:
          /*** Create the pixmap (bitmaps need leaf name) ***/
```

```
          sprintf(viewBitmapFilename,"%s%s",viewDirName,"/image.xpm");
          XGetWindowAttributes(dsply,viewport->viewWindow,&vwInfo);
          write_pixmap_file(dsply,scrn,viewBitmapFilename,
                                   viewport->titleWindow,0,0,vwInfo.width,
                                   vwInfo.height+titleHeight);
          break;
      case aBitmap:
          /*** Create the bitmap (bitmaps need leaf name) ***/
          sprintf(viewBitmapFilename,"%s%s",viewDirName,"/image.bm");
          XGetWindowAttributes(dsply,viewport->viewWindow,&vwInfo);
          code = XWriteBitmapFile(dsply,viewBitmapFilename,
                                   viewport->titleWindow,vwInfo.width,
                                   vwInfo.height+vwInfo.border_width+20,-1,-1);
          break;
      case anImage:
          /*** Create the pixmap (bitmaps need leaf name) ***/
          sprintf(viewBitmapFilename,"%s%s",viewDirName,"/image.xpm");
          XResizeWindow(dsply,viewport->titleWindow,300,300+titleHeight);
          XResizeWindow(dsply,viewport->viewWindow,300,300);
          XGetWindowAttributes(dsply,viewport->viewWindow,&vwInfo);
          drawViewport(Xoption);
          writeTitle();
          write_pixmap_file(dsply,scrn,viewBitmapFilename,
                                     viewport->titleWindow,0,0,vwInfo.width,
                                     vwInfo.height+titleHeight);
          /*** Create the bitmap (bitmaps need leaf name) ***/
          mono = 1;
          drawViewport(Xoption);
          writeTitle();
          sprintf(viewBitmapFilename,"%s%s%s",viewDirName,"/","image.bm");
          code = XWriteBitmapFile(dsply,viewBitmapFilename,
                                   viewport->titleWindow,vwInfo.width,
                                   vwInfo.height+vwInfo.border_width+20,-1,-1);
          mono = 0;
          break;
      case aPostscript:
          /*** Create postscript output for viewport (in axiom2d.ps) ***/
        sprintf(PSfilename,"%s%s",viewDirName,"/axiom2d.ps");
        if (PSInit(viewport->viewWindow,viewport->titleWindow) == psError)
          return (-1);
        drawViewport(PSoption);  /* write new script file in /tmp */
        if (PSCreateFile(viewBorderWidth,viewport->viewWindow,
                        viewport->titleWindow, viewport->title) == psError)
          return(-1);              /* concat script & proc into axiom2d.ps */
        break;
```

```
            } /* switch on ii */
          }  /* if thingsToWrite >> ii */
        }  /* for ii */

      return(0);
    }    /* else create directory okay */
  }
```

6.3.30   main

The main function performs the following steps

1. calls XOpenDisplay (See 9.1.13 on page 523), using the DISPLAY variable from the environment, to choose the display.

2. uses the DefaultScreen macro (See 9.1.10 on page 523), to get the user's default screen.

3. uses the RootWindow macro (See 9.1.11 on page 523), to get the root window on the user's display and screen.

4. calls XCreateAssocTable to create an association table with `nbuckets` which is elsewhere defined to be 128. Note that we do not actually use the X10 definition of this function but use our own version. See 8.1.33 on page 498. This table is used to hold an association between the control panel buttons and the window they control.

⟨*view2d*⟩+≡

```
int main(void) {
  XGCValues    controlGCVals;
  int          i,code;
  view2DStruct viewData;
  char         property[256];
  char         *prop = &property[0];
  char         *str_type[20];
  XrmValue     value;
  if ((dsply = XOpenDisplay(getenv("DISPLAY"))) == NULL)
    fprintf(stderr,"Could not open the display.\n");
  scrn  = DefaultScreen(dsply);
  rtWindow  = RootWindow(dsply,scrn);
  /**** link Xwindows to viewports - X10 feature ****/
  table        = XCreateAssocTable(nbuckets);
  /**** Create Axiom color map ****/
  totalColors = XInitSpadFill(dsply,scrn,&colorMap,
                              &totalHues,&totalSolidShades,
                              &totalDitheredAndSolids,&totalShades);
  if (totalColors < 0) {
    fprintf(stderr,">>Error: Could not allocate all the necessary colors.\n");
    exitWithAck(RootWindow(dsply,scrn),Window,-1);
  }
  mergeDatabases();
  /*** Determine whether monochrome or color is used ***/
  if (XrmGetResource(rDB,"Axiom.2D.monochrome","",str_type,&value) == True)
    (void) strncpy(prop,value.addr,(int)value.size);
  else
    (void) strcpy(prop, "off");
  mono = ((totalSolid == 2) || (strcmp(prop,"on") == 0));
```

```
if (XrmGetResource(rDB,"Axiom.2D.inverse","",str_type,&value) == True)
  (void) strncpy(prop,value.addr,(int)value.size);
else
  (void) strcpy(prop, "off");

if (mono)
  if (strcmp(prop,"on") == 0) {               /* 0 if equal (inverse video) */
    foregroundColor = WhitePixel(dsply,scrn);
    backgroundColor = BlackPixel(dsply,scrn);
  } else {                                    /* off (no inverse video) */
    foregroundColor = BlackPixel(dsply,scrn);
    backgroundColor = WhitePixel(dsply,scrn);
  }
else   /* inverse of inverse in color (for some strange reason) */
  if (strcmp(prop,"on") == 0) {          /* 0 if equal (inverse video) */
    foregroundColor = WhitePixel(dsply,scrn);
    backgroundColor = BlackPixel(dsply,scrn);
  } else {                                    /* off (no inverse video) */
    foregroundColor = BlackPixel(dsply,scrn);
    backgroundColor = WhitePixel(dsply,scrn);
  }
/* read default file name for postScript output */
if (XrmGetResource(rDB,
                   "Axiom.2D.postscriptFile",
                   "",
                   str_type, &value) == True)
  (void) strncpy(prop,value.addr,(int)value.size);
else
  (void) strcpy(prop, "axiom2d.ps");
PSfilename = (char *)malloc(strlen(prop)+1);
strcpy(PSfilename,prop);
/**** Open global fonts ****/
serverFont = XQueryFont(dsply,XGContextFromGC(DefaultGC(dsply,scrn)));
if (XrmGetResource(rDB,
                   "Axiom.2D.messageFont",
                   "Axiom.2D.Font",
                   str_type, &value) == True)
  (void) strncpy(prop,value.addr,(int)value.size);
else
  (void) strcpy(prop,messageFontDefault);
if ((globalFont = XLoadQueryFont(dsply, prop)) == NULL) {
  fprintf(stderr,
          "Warning:  could not get the %s font for messageFont\n",prop);
  globalFont = serverFont;
}
if (XrmGetResource(rDB,
```

```
                    "Axiom.2D.buttonFont",
                    "Axiom.2D.Font",
                    str_type, &value) == True)
  (void) strncpy(prop,value.addr,(int)value.size);
else
  (void) strcpy(prop,buttonFontDefault);
if ((buttonFont = XLoadQueryFont(dsply, prop)) == NULL) {
  fprintf(stderr,
          "Warning:  could not get the %s font for buttonFont\n",prop);
  buttonFont = serverFont;
}
if (XrmGetResource(rDB,
                    "Axiom.2D.headerFont",
                    "Axiom.2D.Font",
                    str_type, &value) == True)
    (void) strncpy(prop,value.addr,(int)value.size);
else
  (void) strcpy(prop,headerFontDefault);
if ((headerFont = XLoadQueryFont(dsply, prop)) == NULL) {
  fprintf(stderr,
          "Warning:  could not get the %s font for headerFont\n",prop);
  headerFont = serverFont;
}
if (XrmGetResource(rDB,
                    "Axiom.2D.titleFont",
                    "Axiom.2D.Font",
                    str_type,&value) == True)
  (void) strncpy(prop,value.addr,(int)value.size);
else
  (void) strcpy(prop,titleFontDefault);
if ((titleFont = XLoadQueryFont(dsply, prop)) == NULL) {
  fprintf(stderr,
          "Warning:  could not get the %s font for titleFont\n",prop);
  titleFont = serverFont;
}
if (XrmGetResource(rDB,
                    "Axiom.2D.graphFont",
                    "Axiom.2D.Font",
                    str_type,&value) == True)
  (void) strncpy(prop,value.addr,(int)value.size);
else
  (void) strcpy(prop,graphFontDefault);
if ((graphFont = XLoadQueryFont(dsply, prop)) == NULL) {
  fprintf(stderr,
          "Warning:  could not get the %s font for graphFont\n",prop);
  graphFont = serverFont;
```

```
}
if (XrmGetResource(rDB,
                    "Axiom.2D.unitFont",
                    "Axiom.2D.Font",
                    str_type,&value) == True)
  (void) strncpy(prop,value.addr,(int)value.size);
else
  (void) strcpy(prop,unitFontDefault);
if ((unitFont = XLoadQueryFont(dsply, prop)) == NULL) {
  fprintf(stderr,
          "Warning:  could not get the %s font for unitFont\n",prop);
  unitFont = serverFont;
}

/**** Create widely used Graphic Contexts ****/
PSGlobalInit();
/* must initiate before using any G/PS functions
   need character name: used as postscript GC variable
   need to create ps GCs for all GCs used by drawings in viewWindow */
/* globalGC1 */
controlGCVals.foreground = monoColor(axesColorDefault);
controlGCVals.background = backgroundColor;
globalGC1 = XCreateGC(dsply,rtWindow,GCForeground | GCBackground ,
                      &controlGCVals);
carefullySetFont(globalGC1,globalFont);
/* create the equivalent GCs for ps */
PSCreateContext(globalGC1, "globalGC1", psNormalWidth, psButtCap,
                psMiterJoin, psWhite, psBlack);
/* controlMessageGC */
controlGCVals.foreground = controlMessageColor;
controlMessageGC = XCreateGC(dsply,rtWindow,GCForeground | GCBackground
                             ,&controlGCVals);
carefullySetFont(controlMessageGC,globalFont);
/* globalGC2 */
controlGCVals.foreground = monoColor(labelColor);
controlGCVals.background = backgroundColor;
globalGC2 = XCreateGC(dsply,rtWindow,GCForeground | GCBackground,
                      &controlGCVals);
carefullySetFont(globalGC2,buttonFont);
PSCreateContext(globalGC2, "globalGC2", psNormalWidth, psButtCap,
                psMiterJoin, psWhite, psBlack);
/* trashGC  */
trashGC = XCreateGC(dsply,rtWindow,0,&controlGCVals);
carefullySetFont(trashGC,buttonFont);
PSCreateContext(trashGC, "trashGC", psNormalWidth, psButtCap,
                psMiterJoin, psWhite, psBlack);
```

```
/* globGC */
globGC = XCreateGC(dsply,rtWindow,0,&controlGCVals);
carefullySetFont(globGC,headerFont);
PSCreateContext(globGC, "globGC", psNormalWidth, psButtCap,
                psMiterJoin, psWhite, psBlack);
/* anotherGC  */
controlGCVals.line_width = colorWidth;
anotherGC  = XCreateGC(dsply,rtWindow,GCBackground,&controlGCVals);
carefullySetFont(anotherGC,titleFont);
PSCreateContext(anotherGC, "anotherGC", psNormalWidth, psButtCap,
                psMiterJoin, psWhite, psBlack);
/* processGC */
gcVals.background = backgroundColor;
processGC        = XCreateGC(dsply,rtWindow,GCBackground ,&gcVals);
carefullySetFont(processGC,buttonFont);
/* graphGC */
graphGC          = XCreateGC(dsply,rtWindow,GCBackground,&gcVals);
carefullySetFont(graphGC,graphFont);
PSCreateContext(graphGC, "graphGC", psNormalWidth, psButtCap,
                psMiterJoin, psWhite, psBlack);
/* unitGC */
unitGC           = XCreateGC(dsply,rtWindow,GCBackground ,&gcVals);
carefullySetFont(unitGC,unitFont);
PSCreateContext(unitGC, "unitGC", psNormalWidth, psButtCap,
                psMiterJoin, psWhite, psBlack);
/**** Initialize Graph States ****/
for (i=0; i<maxGraphs; i++) {
  graphStateArray[i].scaleX = 0.9;
  graphStateArray[i].scaleY = 0.9;
  graphStateArray[i].deltaX = 0.0;
  graphStateArray[i].deltaY = 0.0;
  graphStateArray[i].centerX = 0.0;
  graphStateArray[i].centerY = 0.0;
  graphStateArray[i].pointsOn  = yes;
  graphStateArray[i].connectOn = yes;
  graphStateArray[i].splineOn  = no;
  graphStateArray[i].axesOn    = yes;
  graphStateArray[i].unitsOn   = no;
  graphStateArray[i].showing   = no;
  graphStateArray[i].selected  = no;
  graphStateBackupArray[i] = graphStateArray[i];
}
/**** Get Data from the Viewport Manager ****/
i = 123;
code=check(write(Socket,&i,intSize));
/* Check if I am getting stuff from Axiom or, if I am viewAlone. */
```

```
readViewman(&viewAloned,intSize);
readViewman(&viewData,sizeof(view2DStruct));
readViewman(&i,intSize);
if (!(viewData.title = (char *)malloc(i))) {
  fprintf(stderr,
          "ERROR: Ran out of memory trying to receive the title.\n");
  exitWithAck(RootWindow(dsply,scrn),Window,-1);
}
readViewman(viewData.title,i);
for (i=0; i<maxGraphs; i++) {
  readViewman(&(graphArray[i].key),intSize);
  if (graphArray[i].key) {              /** this graph slot has data **/
    getGraphFromViewman(i);
  } /* if graph exists (graphArray[i].key is not zero) */
} /* for i in graphs */
viewport = makeView2D(&viewData);
control = viewport->controlPanel;
bsdSignal(SIGTERM,goodbye,DontRestartSystemCalls);
/* send acknowledgement to viewport manager */
i = 345;
check(write(Socket,&(viewport->viewWindow),sizeof(Window)));
processEvents();
goodbye(-1);
return(0);  /* control never reaches here but compiler complains */
} /* main() */
```

Chapter 7

view3d

7.1   view3d Call Graph

This was generated by the GNU cflow program with the argument list. Note that the line:NNNN numbers refer to the line in the code after it has been tangled from this file.

```
cflow --emacs -l -n -b -T --omit-arguments view3d.c

;; This file is generated by GNU cflow 1.3. -*- cflow -*-
    2 {   0} +-main() <int main () line:10360>
    3 {   1}   +-NIL()
    4 {   1}   +-XOpenDisplay()
    5 {   1}   +-getenv()
    6 {   1}   +-fprintf()
    7 {   1}   +-exit()
    8 {   1}   +-DefaultScreen()
    9 {   1}   +-RootWindow()
   10 {   1}   +-XSetErrorHandler()
   11 {   1}   +-theHandler() <int theHandler () line:5139>
   12 {   2}     +-XGetErrorText()
   13 {   2}     \-fprintf()
   14 {   1}   +-XCreateAssocTable()
                  <HashTable *XCreateAssocTable () line:2722>
   15 {   2}     +-malloc()
   16 {   2}     +-hash_init()
   17 {   2}     +-TrivEqual() <int TrivEqual () line:2712>
   18 {   2}     \-TrivHashCode() <int TrivHashCode () line:2717>
   19 {   1}   +-XInitSpadFill()
   20 {   1}   +-exitWithAck()
   21 {   1}   +-mergeDatabases() <void mergeDatabases () line:5146>
   22 {   2}   | +-XrmInitialize()
   23 {   2}   | +-strcpy()
```

```
24 {   2}    | +-strcat()
25 {   2}    | +-XrmGetFileDatabase()
26 {   2}    | +-XrmMergeDatabases()
27 {   2}    | +-XResourceManagerString()
28 {   2}    | +-XrmGetStringDatabase()
29 {   2}    | +-getenv()
30 {   2}    | +-strlen()
31 {   2}    | \-gethostname()
32 {   1}    +-XrmGetResource()
33 {   1}    +-strncpy()
34 {   1}    +-strcpy()
35 {   1}    +-strcmp()
36 {   1}    +-XInitShades()
37 {   1}    +-malloc()
38 {   1}    +-strlen()
39 {   1}    +-XSync()
40 {   1}    +-XQueryFont()
41 {   1}    +-XGContextFromGC()
42 {   1}    +-DefaultGC()
43 {   1}    +-XLoadQueryFont()
44 {   1}    +-PSGlobalInit() <int PSGlobalInit () line:2246>
45 {   2}    | +-tempnam()
46 {   2}    | +-sprintf()
47 {   2}    | +-free()
48 {   2}    | +-getenv()
49 {   2}    | \-fprintf()
50 {   1}    +-monoColor()
51 {   1}    +-XCreateGC()
52 {   1}    +-carefullySetFont()
53 {   1}    +-PSCreateContext() <int PSCreateContext () line:2368>
54 {   2}      +-malloc()
55 {   2}      +-fprintf()
56 {   2}      +-exit()
57 {   2}      +-sprintf()
58 {   2}      +-fopen()
59 {   2}      \-fclose()
60 {   1}    +-check()
61 {   1}    +-write()
62 {   1}    +-readViewman() <int readViewman () line:7972>
63 {   2}      +-sprintf()
64 {   2}      +-check()
65 {   2}      \-read()
66 {   1}    +-saymem()
67 {   1}    +-make3DComponents() <viewPoints *make3DComponents () line:3365>
68 {   2}      +-readComponentsFromViewman()
                  <void readComponentsFromViewman () line:3270>
69 {   3}      +-readViewman() <int readViewman () line:7972> [see 62]
70 {   3}      +-saymem()
71 {   3}      \-fprintf()
72 {   2}      +-scaleComponents() <void scaleComponents () line:3095>
```

```
 73 {   3}      | \-absolute() <float absolute () line:8273>
 74 {   2}      +-NIL()
 75 {   2}      +-triangulate() <void triangulate () line:3184>
 76 {   3}      | +-makeTriangle() <void makeTriangle () line:3165>
 77 {   4}      |   +-samePoint()
 78 {   4}      |   \-saymem()
 79 {   3}      | \-saymem()
 80 {   2}      +-calcNormData() <void calcNormData () line:3324>
 81 {   3}      | +-NIL()
 82 {   3}      | +-refPt3D()
 83 {   3}      | \-getMeshNormal() <void getMeshNormal () line:5180>
 84 {   4}      |   \-sqrt()
 85 {   2}      +-makeViewport() <viewPoints *makeViewport () line:9303>
 86 {   3}        +-saymem()
 87 {   3}        +-fprintf()
 88 {   3}        +-exitWithAck()
 89 {   3}        +-RootWindow()
 90 {   3}        +-strcpy()
 91 {   3}        +-sin()
 92 {   3}        +-cos()
 93 {   3}        +-ROTATE() <void ROTATE () line:8884>
 94 {   3}        +-ROTATE1() <void ROTATE1 () line:8903>
 95 {   3}        +-SCALE() <void SCALE () line:8922>
 96 {   3}        +-TRANSLATE() <void TRANSLATE () line:8929>
 97 {   3}        +-XCreateBitmapFromData()
 98 {   3}        +-XQueryColor()
 99 {   3}        +-XCreatePixmapCursor()
100 {   3}        +-XCreateWindow()
101 {   3}        +-XInternAtom()
102 {   3}        +-XSetWMProtocols()
103 {   3}        +-XSetNormalHints()
104 {   3}        +-strlen()
105 {   3}        +-XSetStandardProperties()
106 {   3}        +-XSync()
107 {   3}        +-makeControlPanel()
                    <controlPanelStruct *makeControlPanel () line:4422>
108 {   4}        | +-saymem()
109 {   4}        | +-fprintf()
110 {   4}        | +-exitWithAck()
111 {   4}        | +-RootWindow()
112 {   4}        | +-getControlXY() <controlXY getControlXY () line:4360>
113 {   5}        | | +-XQueryTree()
114 {   5}        | | +-XFree()
115 {   5}        | | \-XGetWindowAttributes()
116 {   4}        | +-XCreateBitmapFromData()
117 {   4}        | +-XQueryColor()
118 {   4}        | +-XCreatePixmapCursor()
119 {   4}        | +-XCreateWindow()
120 {   4}        | +-XSetNormalHints()
121 {   4}        | +-XSetStandardProperties()
```

```
122 {   4}        | +-initButtons() <int initButtons () line:2745>
123 {   4}        | +-XMakeAssoc() <void XMakeAssoc () line:2731>
124 {   5}        |   \-hash_insert()
125 {   4}        | +-XMapWindow()
126 {   4}        | \-strcpy()
127 {   3}        +-makeLightingPanel() <int makeLightingPanel () line:4712>
128 {   4}        | +-XCreateBitmapFromData()
129 {   4}        | +-XQueryColor()
130 {   4}        | +-XCreatePixmapCursor()
131 {   4}        | +-XCreateWindow()
132 {   4}        | +-XSetNormalHints()
133 {   4}        | +-XSetStandardProperties()
134 {   4}        | +-XMapWindow()
135 {   4}        | +-initLightButtons() <int initLightButtons () line:4626>
136 {   4}        | +-XMakeAssoc() <void XMakeAssoc () line:2731> [see 123]
137 {   4}        | +-sin()
138 {   4}        | \-cos()
139 {   3}        +-makeVolumePanel() <void makeVolumePanel () line:9734>
140 {   4}        | +-XCreateBitmapFromData()
141 {   4}        | +-XQueryColor()
142 {   4}        | +-XCreatePixmapCursor()
143 {   4}        | +-XCreateWindow()
144 {   4}        | +-XSetNormalHints()
145 {   4}        | +-XSetStandardProperties()
146 {   4}        | +-initVolumeButtons() <int initVolumeButtons () line:9612>
147 {   4}        | +-XMakeAssoc() <void XMakeAssoc () line:2731> [see 123]
148 {   4}        | \-XMapWindow()
149 {   3}        +-makeSavePanel() <int makeSavePanel () line:7056>
150 {   4}        | +-XCreateBitmapFromData()
151 {   4}        | +-XQueryColor()
152 {   4}        | +-XCreatePixmapCursor()
153 {   4}        | +-XCreateWindow()
154 {   4}        | +-XSetNormalHints()
155 {   4}        | +-XSetStandardProperties()
156 {   4}        | +-initSaveButtons() <int initSaveButtons () line:7135>
157 {   4}        | +-XMakeAssoc() <void XMakeAssoc () line:2731> [see 123]
158 {   4}        | \-XMapWindow()
159 {   3}        +-makeQuitPanel() <int makeQuitPanel () line:6939>
160 {   4}        | +-XCreateBitmapFromData()
161 {   4}        | +-XQueryColor()
162 {   4}        | +-XCreatePixmapCursor()
163 {   4}        | +-XCreateWindow()
164 {   4}        | +-XSetNormalHints()
165 {   4}        | +-XSetStandardProperties()
166 {   4}        | +-initQuitButtons() <int initQuitButtons () line:7024>
167 {   4}        | +-XMakeAssoc() <void XMakeAssoc () line:2731> [see 123]
168 {   4}        | \-XMapWindow()
169 {   3}        \-putControlPanelSomewhere()
                    <void putControlPanelSomewhere () line:4509>
170 {   4}          +-getControlXY()
```

```
                              | <controlXY getControlXY () line:4360> [see 112]
171 {    4}                   +-XRaiseWindow()
172 {    4}                   +-XMoveWindow()
173 {    4}                   +-drawControlPanel() <void drawControlPanel () line:3835>
174 {    5}                   | +-GSetForeground() <int GSetForeground () line:2419>
175 {    6}                   |   +-XSetForeground()
176 {    6}                   |   +-fopen()
177 {    6}                   |   +-fprintf()
178 {    6}                   |   +-PSfindGC() <char *PSfindGC () line:2405>
179 {    7}                   |   | \-fprintf()
180 {    6}                   |   \-fclose()
181 {    5}                   | +-GSetLineAttributes()
                              |   <int GSetLineAttributes () line:2479>
182 {    6}                   |   +-XSetLineAttributes()
183 {    6}                   |   +-fprintf()
184 {    6}                   |   +-fopen()
185 {    6}                   |   +-PSfindGC() <char *PSfindGC () line:2405> [see 178]
186 {    6}                   |   \-fclose()
187 {    5}                   | +-GDrawLine() <int GDrawLine () line:1995>
188 {    6}                   |   +-XDrawLine()
189 {    6}                   |   +-fopen()
190 {    6}                   |   +-fprintf()
191 {    6}                   |   +-PSfindGC() <char *PSfindGC () line:2405> [see 178]
192 {    6}                   |   \-fclose()
193 {    5}                   | +-writeControlTitle()
                              |    <void writeControlTitle () line:3798>
194 {    6}                   |   +-strlen()
195 {    6}                   |   +-XClearArea()
196 {    6}                   |   +-GSetForeground()
                              |      <int GSetForeground () line:2419> [see 174]
197 {    6}                   |   +-GDrawString() <int GDrawString () line:2181>
198 {    7}                   |   | +-XDrawString()
199 {    7}                   |   | +-fopen()
200 {    7}                   |   | +-fprintf()
201 {    7}                   |   | +-PSfindGC() <char *PSfindGC () line:2405> [see 178]
202 {    7}                   |   | \-fclose()
203 {    6}                   |   \-centerX() <int centerX () line:2557>
204 {    7}                   |      +-XGContextFromGC()
205 {    7}                   |      +-XQueryFont()
206 {    7}                   |      +-XTextWidth()
207 {    7}                   |      \-XFreeFontInfo()
208 {    5}                   | +-GDrawString() <int GDrawString () line:2181> [see 197]
209 {    5}                   | +-strlen()
210 {    5}                   | +-monoColor()
211 {    5}                   | +-XFillRectangle()
212 {    5}                   | +-GDrawRectangle() <int GDrawRectangle () line:2094>
213 {    6}                   |   +-XDrawRectangle()
214 {    6}                   |   +-fopen()
215 {    6}                   |   +-fprintf()
216 {    6}                   |   +-PSfindGC() <char *PSfindGC () line:2405> [see 178]
```

```
217 {   6}           |   \-fclose()
218 {   5}           | +-XChangeShade()
219 {   5}           | +-XShadeRectangle()
220 {   5}           | +-GDrawImageString() <int GDrawImageString () line:1930>
221 {   6}           | | +-XDrawImageString()
222 {   6}           | | +-fopen()
223 {   6}           | | +-fprintf()
224 {   6}           | | +-PSfindGC() <char *PSfindGC () line:2405> [see 178]
225 {   6}           | | \-fclose()
226 {   5}           | +-centerX() <int centerX () line:2557> [see 203]
227 {   5}           | +-centerY() <int centerY () line:2571>
228 {   6}           |   +-XGContextFromGC()
229 {   6}           |   +-XQueryFont()
230 {   6}           |   \-XFreeFontInfo()
231 {   5}           | +-GDraw3DButtonOut() <int GDraw3DButtonOut () line:2125>
232 {   6}           |   +-GDrawRectangle()
                     |   | <int GDrawRectangle () line:2094> [see 212]
233 {   6}           |   \-GDrawLine() <int GDrawLine () line:1995> [see 187]
234 {   5}           | +-GDrawArc() <int GDrawArc () line:1963>
235 {   6}           |   +-XDrawArc()
236 {   6}           |   +-fopen()
237 {   6}           |   +-fprintf()
238 {   6}           |   +-PSfindGC() <char *PSfindGC () line:2405> [see 178]
239 {   6}           |   \-fclose()
240 {   5}           | +-GDrawLines() <int GDrawLines () line:2025>
241 {   6}           |   +-XDrawLines()
242 {   6}           |   +-fopen()
243 {   6}           |   +-fprintf()
244 {   6}           |   +-PSfindGC() <char *PSfindGC () line:2405> [see 178]
245 {   6}           |   \-fclose()
246 {   5}           | +-clearControlMessage()
                     | | <void clearControlMessage () line:3808>
247 {   6}           | | +-strcpy()
248 {   6}           | | +-strlen()
249 {   6}           | | +-GDrawImageString()
                     | | | <int GDrawImageString () line:1930> [see 220]
250 {   6}           | | \-centerX() <int centerX () line:2557> [see 203]
251 {   5}           | +-strcpy()
252 {   5}           | +-writeControlMessage()
                     | | | <void writeControlMessage () line:3819>
253 {   6}           | | +-strlen()
254 {   6}           | | +-XClearArea()
255 {   6}           | | +-GSetForeground()
                     | | | <int GSetForeground () line:2419> [see 174]
256 {   6}           | | +-GDrawImageString()
                     | | | <int GDrawImageString () line:1930> [see 220]
257 {   6}           | | +-centerX() <int centerX () line:2557> [see 203]
258 {   6}           | | \-XFlush()
259 {   5}           | +-drawColorMap() <void drawColorMap () line:3740>
260 {   6}           | | +-XClearArea()
```

```
261 {   6}            | | +-XChangeShade()
262 {   6}            | | +-XShadeRectangle()
263 {   6}            | | +-GDrawString()
                      | | |<int GDrawString () line:2181> [see 197]
264 {   6}            | | +-GSetForeground()
                      | | | <int GSetForeground () line:2419> [see 174]
265 {   6}            | | +-XSolidColor()
266 {   6}            | | +-GDrawLine() <int GDrawLine () line:1995> [see 187]
267 {   6}            | | +-monoColor()
268 {   6}            | | \-XSync()
269 {   5}            | \-XFlush()
270 {   4}            +-XSync()
271 {   4}            +-XMapWindow()
272 {   4}            \-XFlush()
273 {   2}        +-XCreateImage()
274 {   2}        +-DefaultVisual()
275 {   2}        +-DefaultDepth()
276 {   2}        +-writeTitle() <void writeTitle () line:8936>
277 {   3}        | +-XGetWindowAttributes()
278 {   3}        | +-GSetForeground()
                  | | <int GSetForeground () line:2419> [see 174]
279 {   3}        | +-XClearWindow()
280 {   3}        | +-strlen()
281 {   3}        | +-GDrawImageString()
                  | | <int GDrawImageString () line:1930> [see 220]
282 {   3}        | \-centerX() <int centerX () line:2557> [see 203]
283 {   2}        +-postMakeViewport() <void postMakeViewport () line:9510>
284 {   2}        +-drawViewport()
285 {   2}        +-XMapWindow()
286 {   2}        \-XFlush()
287 {   1}    +-bsdSignal()
288 {   1}    +-goodbye() <void goodbye () line:8301>
289 {   2}        +-PSClose() <int PSClose () line:2544>
290 {   3}          +-free()
291 {   3}          \-unlink()
292 {   2}        +-FreePixels()
293 {   2}        +-check()
294 {   2}        +-write()
295 {   2}        +-XCloseDisplay()
296 {   2}        \-exit()
297 {   1}    +-sprintf()
298 {   1}    +-XCreatePixmap()
299 {   1}    +-DisplayPlanes()
300 {   1}    \-processEvents() <void processEvents () line:6003>
301 {   2}        +-ConnectionNumber()
302 {   2}        +-saymem()
303 {   2}        +-fprintf()
304 {   2}        +-exitWithAck()
305 {   2}        +-RootWindow()
306 {   2}        +-XGetWindowAttributes()
```

```
307 {   2}      +-FD_ZERO()
308 {   2}      +-FD_SET()
309 {   2}      +-XEventsQueued()
310 {   2}      +-select()
311 {   2}      +-FD_ISSET()
312 {   2}      +-XPending()
313 {   2}      +-XNextEvent()
314 {   2}      +-goodbye() <void goodbye () line:8301> [see 288]
315 {   2}      +-XSync()
316 {   2}      +-XCheckWindowEvent()
317 {   2}      +-writeTitle() <void writeTitle () line:8936> [see 276]
318 {   2}      +-XResizeWindow()
319 {   2}      +-drawViewport()
320 {   2}      +-XMapWindow()
321 {   2}      +-drawLightingPanel() <void drawLightingPanel () line:4936>
322 {   3}        +-GSetForeground()
                  | <int GSetForeground () line:2419> [see 174]
323 {   3}        +-GSetLineAttributes()
                  | <int GSetLineAttributes () line:2479> [see 181]
324 {   3}        +-GDrawLine() <int GDrawLine () line:1995> [see 187]
325 {   3}        +-writeControlTitle()
                  | <void writeControlTitle () line:3798> [see 193]
326 {   3}        +-strlen()
327 {   3}        +-GDrawString() <int GDrawString () line:2181> [see 197]
328 {   3}        +-centerX() <int centerX () line:2557> [see 203]
329 {   3}        +-GDraw3DButtonOut()
                  | <int GDraw3DButtonOut () line:2125> [see 231]
330 {   3}        +-monoColor()
331 {   3}        +-drawLightTransArrow()
                    | <void drawLightTransArrow () line:4867>
332 {   4}        +-XClearArea()
333 {   4}        +-GDrawLine() <int GDrawLine () line:1995> [see 187]
334 {   4}        \-GSetForeground()
                      <int GSetForeground () line:2419> [see 174]
335 {   3}        +-centerY() <int centerY () line:2571> [see 227]
336 {   3}        \-drawLightingAxes() <void drawLightingAxes () line:4789>
337 {   4}          +-XGetWindowAttributes()
338 {   4}          +-XClearWindow()
339 {   4}          +-sin()
340 {   4}          +-cos()
341 {   4}          +-GSetForeground()
                    | <int GSetForeground () line:2419> [see 174]
342 {   4}          +-monoColor()
343 {   4}          +-proj2PX()
344 {   4}          +-proj2PY()
345 {   4}          \-GDrawLine() <int GDrawLine () line:1995> [see 187]
346 {   2}      +-drawVolumePanel() <void drawVolumePanel () line:10091>
347 {   3}        +-GSetForeground()
                  | <int GSetForeground () line:2419> [see 174]
348 {   3}        +-GSetLineAttributes()
```

```
                         | <int GSetLineAttributes () line:2479> [see 181]
  349 {   3}              +-GDrawLine() <int GDrawLine () line:1995> [see 187]
  350 {   3}              +-writeControlTitle()
                         | <void writeControlTitle () line:3798> [see 193]
  351 {   3}              +-strlen()
  352 {   3}              +-GDrawString() <int GDrawString () line:2181> [see 197]
  353 {   3}              +-centerX() <int centerX () line:2557> [see 203]
  354 {   3}              +-monoColor()
  355 {   3}              +-GDraw3DButtonOut()
                         | <int GDraw3DButtonOut () line:2125> [see 231]
  356 {   3}              +-centerY() <int centerY () line:2571> [see 227]
  357 {   3}              +-drawClipXBut() <void drawClipXBut () line:9789>
  358 {   4}                +-XClearArea()
  359 {   4}                +-GSetForeground()
                         | <int GSetForeground () line:2419> [see 174]
  360 {   4}                +-monoColor()
  361 {   4}                +-GDrawLine() <int GDrawLine () line:1995> [see 187]
  362 {   4}                +-GFillArc() <int GFillArc () line:2213>
  363 {   5}                  +-XFillArc()
  364 {   5}                  +-fopen()
  365 {   5}                  +-fprintf()
  366 {   5}                  +-PSfindGC() <char *PSfindGC () line:2405> [see 178]
  367 {   5}                  \-fclose()
  368 {   4}                \-GDrawString() <int GDrawString () line:2181> [see 197]
  369 {   3}              +-drawClipYBut() <void drawClipYBut () line:9846>
  370 {   4}                +-XClearArea()
  371 {   4}                +-GSetForeground()
                         | <int GSetForeground () line:2419> [see 174]
  372 {   4}                +-monoColor()
  373 {   4}                +-GDrawLine() <int GDrawLine () line:1995> [see 187]
  374 {   4}                +-GFillArc() <int GFillArc () line:2213> [see 362]
  375 {   4}                \-GDrawString() <int GDrawString () line:2181> [see 197]
  376 {   3}              +-drawClipZBut() <void drawClipZBut () line:9904>
  377 {   4}                +-XClearArea()
  378 {   4}                +-GSetForeground()
                         | <int GSetForeground () line:2419> [see 174]
  379 {   4}                +-monoColor()
  380 {   4}                +-GDrawLine() <int GDrawLine () line:1995> [see 187]
  381 {   4}                +-GFillArc() <int GFillArc () line:2213> [see 362]
  382 {   4}                \-GDrawString() <int GDrawString () line:2181> [see 197]
  383 {   3}              +-drawFrustrum() <void drawFrustrum () line:10064>
  384 {   4}              | +-XClearArea()
  385 {   4}              | +-GSetForeground()
                         | | <int GSetForeground () line:2419> [see 174]
  386 {   4}              | +-pow()
  387 {   4}              | +-GDrawLine() <int GDrawLine () line:1995> [see 187]
  388 {   4}              | +-frusX()
  389 {   4}              | +-frusY()
  390 {   4}              | +-drawHitherControl()
                         | | <void drawHitherControl () line:9981>
```

```
391 {   5}        | | +-GSetForeground()
                  | | | <int GSetForeground () line:2419> [see 174]
392 {   5}        | | +-GDraw3DButtonOut()
                  | | | <int GDraw3DButtonOut () line:2125> [see 231]
393 {   5}        | | +-frusY()
394 {   5}        | | +-fabs()
395 {   5}        | | +-GDrawLine() <int GDrawLine () line:1995> [see 187]
396 {   5}        | | +-frusX()
397 {   5}        | | \-GDrawString() <int GDrawString () line:2181> [see 197]
398 {   4}        | \-drawEyeControl() <void drawEyeControl () line:10026>
399 {   5}        |   +-GSetForeground()
                  |   | <int GSetForeground () line:2419> [see 174]
400 {   5}        |   +-GDraw3DButtonOut()
                  |   | <int GDraw3DButtonOut () line:2125> [see 231]
401 {   5}        |   +-frusX()
402 {   5}        |   +-frusY()
403 {   5}        |   +-GDrawLine() <int GDrawLine () line:1995> [see 187]
404 {   5}        |   +-pow()
405 {   5}        |   +-GSetLineAttributes()
                  |   | <int GSetLineAttributes () line:2479> [see 181]
406 {   5}        |   +-monoColor()
407 {   5}        |   +-GDrawString() <int GDrawString () line:2181> [see 197]
408 {   5}        |   \-strlen()
409 {   3}        \-drawClipVolume() <void drawClipVolume () line:9931>
410 {   4}          +-XClearArea()
411 {   4}          +-GSetForeground()
                    | <int GSetForeground () line:2419> [see 174]
412 {   4}          +-GSetLineAttributes()
                    | <int GSetLineAttributes () line:2479> [see 181]
413 {   4}          +-GDrawRectangle()
                    | <int GDrawRectangle () line:2094> [see 212]
414 {   4}          \-GDrawLine() <int GDrawLine () line:1995> [see 187]
415 {   2}      +-drawQuitPanel() <void drawQuitPanel () line:6993>
416 {   3}        +-strlen()
417 {   3}        +-GSetForeground()
                  | <int GSetForeground () line:2419> [see 174]
418 {   3}        +-GDrawString() <int GDrawString () line:2181> [see 197]
419 {   3}        +-centerX() <int centerX () line:2557> [see 203]
420 {   3}        +-centerY() <int centerY () line:2571> [see 227]
421 {   3}        +-GDraw3DButtonOut()
                  | <int GDraw3DButtonOut () line:2125> [see 231]
422 {   3}        \-monoColor()
423 {   2}      +-drawSavePanel() <void drawSavePanel () line:7110>
424 {   3}      | +-GSetForeground()
                | | <int GSetForeground () line:2419> [see 174]
425 {   3}      | +-GDraw3DButtonOut()
                | | <int GDraw3DButtonOut () line:2125> [see 231]
426 {   3}      | +-strlen()
427 {   3}      | +-monoColor()
428 {   3}      | +-GDrawString() <int GDrawString () line:2181> [see 197]
```

```
429 {   3}    | +-centerX() <int centerX () line:2557> [see 203]
430 {   3}    | \-centerY() <int centerY () line:2571> [see 227]
431 {   2}    +-drawControlPanel()
              | <void drawControlPanel () line:3835> [see 173]
432 {   2}    +-XFlush()
433 {   2}    +-XCheckTypedWindowEvent()
434 {   2}    +-XCheckMaskEvent()
435 {   2}    +-getPotValue() <mouseCoord getPotValue () line:5276>
436 {   2}    +-getLinearPotValue()
              | <mouseCoord getLinearPotValue () line:5291>
437 {   2}    +-XQueryPointer()
438 {   2}    +-XUnmapWindow()
439 {   2}    +-putControlPanelSomewhere()
              | <void putControlPanelSomewhere () line:4509> [see 169]
440 {   2}    +-writeControlMessage()
              | <void writeControlMessage () line:3819> [see 252]
441 {   2}    +-XLookUpAssoc() <int *XLookUpAssoc () line:2736>
442 {   3}      \-hash_find()
443 {   2}    +-drawColorMap() <void drawColorMap () line:3740> [see 259]
444 {   2}    +-clearControlMessage()
              | <void clearControlMessage () line:3808> [see 246]
445 {   2}    +-strcpy()
446 {   2}    +-drawLightingAxes()
              | <void drawLightingAxes () line:4789> [see 336]
447 {   2}    +-drawLightTransArrow()
              | <void drawLightTransArrow () line:4867> [see 331]
448 {   2}    +-inside()
449 {   2}    +-drawFrustrum() <void drawFrustrum () line:10064> [see 383]
450 {   2}    +-lessThan() <int lessThan () line:8813>
451 {   2}    +-greaterThan() <int greaterThan () line:8820>
452 {   2}    +-equal() <int equal () line:8835>
453 {   2}    +-drawClipXBut() <void drawClipXBut () line:9789> [see 357]
454 {   2}    +-drawClipVolume()
              |  <void drawClipVolume () line:9931> [see 409]
455 {   2}    +-drawClipYBut() <void drawClipYBut () line:9846> [see 369]
456 {   2}    +-drawClipZBut() <void drawClipZBut () line:9904> [see 376]
457 {   2}    +-GSetForeground() <int GSetForeground () line:2419> [see 174]
458 {   2}    +-monoColor()
459 {   2}    +-GDrawString() <int GDrawString () line:2181> [see 197]
460 {   2}    +-centerX() <int centerX () line:2557> [see 203]
461 {   2}    +-centerY() <int centerY () line:2571> [see 227]
462 {   2}    +-XClearArea()
463 {   2}    +-buttonAction() <void buttonAction () line:5298>
464 {   3}    | +-XUnmapWindow()
465 {   3}    | +-clearControlMessage()
              | |  <void clearControlMessage () line:3808> [see 246]
466 {   3}    | +-strcpy()
467 {   3}    | +-writeControlMessage()
              | |  <void writeControlMessage () line:3819> [see 252]
468 {   3}    | +-XChangeShade()
```

```
469 {   3}      | +-XShadeRectangle()
470 {   3}      | +-GSetForeground()
                | |   <int GSetForeground () line:2419> [see 174]
471 {   3}      | +-GDrawRectangle()
                | | <int GDrawRectangle () line:2094> [see 212]
472 {   3}      | +-XFillRectangle()
473 {   3}      | +-strlen()
474 {   3}      | +-monoColor()
475 {   3}      | +-GDrawImageString()
                | | <int GDrawImageString () line:1930> [see 220]
476 {   3}      | +-centerX() <int centerX () line:2557> [see 203]
477 {   3}      | +-centerY() <int centerY () line:2571> [see 227]
478 {   3}      | +-drawViewport()
479 {   3}      | +-XInitShades()
480 {   3}      | +-drawColorMap() <void drawColorMap () line:3740> [see 259]
481 {   3}      | +-writeTitle() <void writeTitle () line:8936> [see 276]
482 {   3}      | +-XMapWindow()
483 {   3}      | +-normalizeVector() <void normalizeVector () line:5204>
484 {   4}      |   \-sqrt()
485 {   3}      | +-equal() <int equal () line:8835> [see 452]
486 {   3}      | +-drawControlPanel()
                | | <void drawControlPanel () line:3835> [see 173]
487 {   3}      | +-PSInit() <int PSInit () line:2340>
488 {   4}      |   +-fprintf()
489 {   4}      |   +-sprintf()
490 {   4}      |   +-tmpnam()
491 {   4}      |   \-GdrawsSetDimension()
                |     <int GdrawsSetDimension () line:1896>
492 {   5}      |     +-fopen()
493 {   5}      |     +-XGetWindowAttributes()
494 {   5}      |     +-fprintf()
495 {   5}      |     \-fclose()
496 {   3}      | +-PSCreateFile() <int PSCreateFile () line:1827>
497 {   4}      |   +-fopen()
498 {   4}      |   +-fprintf()
499 {   4}      |   +-fclose()
500 {   4}      |   +-filecopy() <void filecopy () line:1820>
501 {   5}      |   | +-getc()
502 {   5}      |   | \-putc()
503 {   4}      |   \-unlink()
504 {   3}      | +-strcat()
505 {   3}      | +-XGetWindowAttributes()
506 {   3}      | +-write_pixmap_file()
507 {   3}      | \-fprintf()
508 {   2}      \-spadAction() <int spadAction () line:7989>
509 {   3}        +-close()
510 {   3}        +-readViewman() <int readViewman () line:7972> [see 62]
511 {   3}        +-refPt3D()
512 {   3}        +-scalePoint() <void scalePoint () line:7979>
513 {   3}        +-XUnmapWindow()
```

```
514 {   3}        +-putControlPanelSomewhere()
                  | <void putControlPanelSomewhere () line:4509> [see 169]
515 {   3}        +-drawControlPanel()
                  | <void drawControlPanel () line:3835> [see 173]
516 {   3}        +-drawColorMap() <void drawColorMap () line:3740> [see 259]
517 {   3}        +-check()
518 {   3}        +-write()
519 {   3}        +-goodbye() <void goodbye () line:8301> [see 288]
520 {   3}        +-XMoveWindow()
521 {   3}        +-XSync()
522 {   3}        +-XResizeWindow()
523 {   3}        +-normalizeVector()
                  | <void normalizeVector () line:5204> [see 483]
524 {   3}        +-drawLightingAxes()
                  | <void drawLightingAxes () line:4789> [see 336]
525 {   3}        +-drawLightTransArrow()
                  | <void drawLightTransArrow () line:4867> [see 331]
526 {   3}        +-writeTitle() <void writeTitle () line:8936> [see 276]
527 {   3}        +-writeControlTitle()
                  | <void writeControlTitle () line:3798> [see 193]
528 {   3}        +-XFlush()
529 {   3}        +-sprintf()
530 {   3}        +-writeViewport() <int writeViewport () line:10218>
531 {   4}        | +-XGetWindowAttributes()
532 {   4}        | +-sprintf()
533 {   4}        | +-system()
534 {   4}        | +-fprintf()
535 {   4}        | +-fopen()
536 {   4}        | +-perror()
537 {   4}        | +-refPt3D()
538 {   4}        | +-fclose()
539 {   4}        | +-XWriteBitmapFile()
540 {   4}        | +-write_pixmap_file()
541 {   4}        | +-XResizeWindow()
542 {   4}        | +-drawViewport()
543 {   4}        | +-writeTitle() <void writeTitle () line:8936> [see 276]
544 {   4}        | +-XInitShades()
545 {   4}        | +-PSInit() <int PSInit () line:2340> [see 487]
546 {   4}        | \-PSCreateFile() <int PSCreateFile () line:1827> [see 496]
547 {   3}        +-fprintf()
548 {   3}        +-strcpy()
549 {   3}        +-writeControlMessage()
                  | <void writeControlMessage () line:3819> [see 252]
550 {   3}        \-buttonAction() <void buttonAction () line:5298> [see 463]
```

7.2   Constants and Headers

7.2.1   System includes

⟨*view3d*⟩≡
```
#include <limits.h>
#include <math.h>
#include <setjmp.h>
#include <signal.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/time.h>
#include <unistd.h>
#include <X11/Xlib.h>
#include <X11/Xresource.h>
#include <X11/Xutil.h>
```

7.2.2   Local includes

⟨*view3d*⟩+≡
```
#include "bsdsignal.h"
#include "bsdsignal.h1"
#include "com.h"
#include "hash.h"
#include "hash.h1"
#include "pixmap.h1"
```
⟨*include/view3d.h*⟩
```
#include "spadcolors.h1"
#include "util.h1"
#include "xshade.h1"
#include "xspadfill.h1"
```

⟨*include/actions.h*⟩
⟨*include/g.h*⟩
⟨*include/override.h*⟩
⟨*include/view.h*⟩
⟨*include/viewcommand.h*⟩
⟨*include/write.h*⟩
⟨*include/xdefs.h*⟩

7.2.3 defines

⟨*view3d*⟩+≡
```
#define BH       31  /* button window height */
#define PH       80  /* potentiometer window height */
#define XEDGE    5   /* leftmost button starts here */

#define axisLength    1.0   /* use 100.0, if data is not to be normalized */

#define samePoint(a,b) ((refPt3D(viewData,a)->x == refPt3D(viewData,b)->x) &&\
                        (refPt3D(viewData,a)->y == refPt3D(viewData,b)->y) &&\
                        (refPt3D(viewData,a)->z == refPt3D(viewData,b)->z))
#define MAX_POINT     1000.0
#define MIN_POINT     -1000.0

#define spadBitmap_width 34
#define spadBitmap_height 20
#define spadBitmap_x_hot 15
#define spadBitmap_y_hot 10
static char spadBitmap_bits[] = {
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x80, 0x00, 0x00, 0x00, 0x00, 0xc0, 0x01, 0x00,
   0x00, 0x00, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xf8,
   0xe6, 0xf8, 0x76, 0x00, 0x84, 0x98, 0x44, 0x49, 0x00, 0xc0, 0x98, 0x42,
   0x49, 0x00, 0xb8, 0x98, 0x42, 0x49, 0x00, 0x84, 0x95, 0x42, 0x49, 0x00,
   0x44, 0xa5, 0x22, 0x49, 0x00, 0x78, 0x63, 0x1d, 0xdb, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x80, 0x00, 0x00, 0x00, 0x00, 0xc0, 0x01, 0x00,
   0x00, 0x00, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00};

#define spadMask_width 34
#define spadMask_height 20
#define spadMask_x_hot 15
#define spadMask_y_hot 10
static char spadMask_bits[] = {
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xc0,
   0x01, 0x00, 0x00, 0x00, 0xe0, 0x03, 0x00, 0x00, 0x00, 0xe0, 0x03, 0x00,
   0x00, 0x00, 0xe0, 0x03, 0x00, 0x00, 0xfc, 0xff, 0xff, 0xff, 0x01, 0xfe,
   0xff, 0xff, 0xff, 0x01, 0xfe, 0xff, 0xff, 0xff, 0x01, 0xfe, 0xff, 0xff,
   0xff, 0x01, 0xfe, 0xff, 0xff, 0xff, 0x01, 0xfe, 0xff, 0xff, 0xff, 0x01,
   0xfe, 0xff, 0xff, 0xff, 0x01, 0xfe, 0xff, 0xff, 0xff, 0x01, 0xfc, 0xff,
   0xff, 0xff, 0x01, 0x00, 0xe0, 0x03, 0x00, 0x00, 0x00, 0xe0, 0x03, 0x00,
   0x00, 0x00, 0xe0, 0x03, 0x00, 0x00, 0x00, 0xc0, 0x01, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00};

#define volumeBitmap_width 16
```

```
#define volumeBitmap_height 16
#define volumeBitmap_x_hot 5
#define volumeBitmap_y_hot 1
static char volumeBitmap_bits[] = {
   0x00, 0x00, 0x60, 0x00, 0x90, 0x00, 0x10, 0x01, 0x10, 0x07, 0x10, 0x09,
   0x0c, 0x39, 0x1a, 0x51, 0x32, 0x50, 0x22, 0x40, 0x22, 0x40, 0x04, 0x60,
   0x04, 0x30, 0xf8, 0x1f, 0x04, 0x20, 0xf8, 0x1f};
#define volumeMask_width 16
#define volumeMask_height 16
#define volumeMask_x_hot 5
#define volumeMask_y_hot 1
static char volumeMask_bits[] = {
   0x00, 0x00, 0x60, 0x00, 0xf0, 0x00, 0xf0, 0x01, 0xf0, 0x07, 0xf0, 0x0f,
   0xfc, 0x3f, 0xfe, 0x7f, 0xfe, 0x7f, 0xfe, 0x7f, 0xfe, 0x7f, 0xfc, 0x7f,
   0xfc, 0x3f, 0xf8, 0x1f, 0x04, 0x20, 0xf8, 0x1f};

#define mouseBitmap_width 16
#define mouseBitmap_height 16
#define mouseBitmap_x_hot 8
#define mouseBitmap_y_hot 0
static char mouseBitmap_bits[] = {
   0x00, 0x01, 0x00, 0x01, 0x80, 0x02, 0x40, 0x04, 0xc0, 0x06, 0x20, 0x08,
   0x20, 0x08, 0x30, 0x18, 0x50, 0x14, 0x58, 0x34, 0x90, 0x12, 0x20, 0x08,
   0xc0, 0x47, 0x00, 0x21, 0x80, 0x10, 0x00, 0x0f};
#define mouseMask_width 16
#define mouseMask_height 16
static char mouseMask_bits[] = {
   0x00, 0x01, 0x00, 0x01, 0x80, 0x03, 0xc0, 0x07, 0xc0, 0x07, 0xe0, 0x0f,
   0xe0, 0x0f, 0xf0, 0x1f, 0xf0, 0x1f, 0xf8, 0x3f, 0xf0, 0x1f, 0xe0, 0x0f,
   0xc0, 0x47, 0x00, 0x21, 0x80, 0x10, 0x00, 0x0f};

/* Defines the pixmap for the arrow displayed in the scale window */
#define zoomArrowN 11
static XPoint zoomArrow[zoomArrowN] = {
  {29,14},{38,23},{33,23},
  {40,45},{53,45},
  {29,69},
  {5,45},{18,45},
  {25,23},{20,23},{29,14} };

/* Defines the pixmap for the arrows displayed in the translate window */
#define translateArrowN 25
static XPoint translateArrow[translateArrowN] = {
  {55,2},{60,10},{58,10},{58,37},
  {85,37},{85,35},{93,40},{85,45},{85,43},{58,43},
  {58,70},{60,70},{55,78},{50,70},{52,70},{52,43},
```

```
   {25,43},{25,45},{17,40},{25,35},{25,37},{52,37},
   {52,10},{50,10},{55,2} };

#define controlMASK             (ButtonPressMask + ButtonReleaseMask + ExposureMask)
#define potMASK                 (ButtonPressMask + ButtonReleaseMask + ButtonMotionMask + Le
#define buttonMASK              (ButtonPressMask + ButtonReleaseMask + ButtonMotionMask + Le
#define colorMASK               (ButtonPressMask + ButtonReleaseMask + ButtonMotionMask + Le


#define mouseWait               50
        /* make mouse grab for stationary mouse on a potentiometer slower */

#define controlCreateMASK       CWBackPixel | CWBorderPixel | CWEventMask |CWCursor |CWColo
#define buttonCreateMASK        CWEventMask
#define messageCreateMASK       0
#define colormapCreateMASK      CWEventMask

#define controlWidth            300
#define controlHeight           400
#define quitWidth               63
#define quitHeight              107
#define saveWidth               63
#define saveHeight              107
#define borderWidth             22
#define borderHeight            45


#define controlCursorForeground monoColor(4)
#define controlCursorBackground monoColor(54)
#define controlTitleColor       monoColor(36)
#define controlPotHeaderColor   monoColor(52)
#define controlColorColor       monoColor(13)
#define controlColorSignColor   monoColor(22)

#define headerHeight            headerFont->max_bounds.ascent
#define controlMessageHeight    globalFont->max_bounds.ascent +globalFont->max_bounds.descen

#define potA                    25   /* y coordinate of line dividing
                                        potentiometers from stuff above it */
#define potB                    173  /* y coordinate of line dividing
                                        potentiometers from title */

#define cmapA                   233  /* y coordinate of line dividing
                                        colormap from stuff above it */

#define butA                    ((cp->buttonQueue[render]).buttonY - 5)
```

```
#define closeL                      ((cp->buttonQueue[closeAll]).buttonX - 5)
#define closeA                      ((cp->buttonQueue[closeAll]).buttonY - 5)

#define controlMessageY        181
#define controlMessageColor    monoColor(68)

#define offColor        13
#define onColor         98
#define modeColor       44

#define colormapX              21
#define colormapY              240
#define colormapW              290
#define colormapH              48
#define colorWidth             8
#define colorHeight            8
#define colorOffset            3
#define colorOffsetX           24
#define colorOffsetY           16
#define colorPointer           16

#define meshOutline        monoColor(140)
#define opaqueOutline      monoColor(85)
#define opaqueForeground   backgroundColor


#define clipOffset 500

/* recalculation occurs if any of these situations have occured */

#define recalc (rotated || zoomed || translated || !finishedList || \
                firstTime || switchedPerspective || changedEyeDistance)


/*** projection macros if matrices are not used  ***/
#define projPersp(z) (viewData.eyeDistance / (z+viewData.eyeDistance))

#define proj2PX(x,y)   -(x*cosTheta + y*sinTheta)
#define proj2PY(x,y,z) -(y*cosTheta*cosPhi - x*sinTheta*cosPhi + z*sinPhi)

/*** For clipping points ***/

#define behindClipPlane(pz) lessThan(pz,viewData.clipPlane)

#define outsideClippedBoundary(x,y,z) (lessThan(x,viewData.clipXmin) ||    \
```

```
                                        greaterThan(x,viewData.clipXmax) || \
                                        lessThan(y,viewData.clipYmin) ||     \
                                        greaterThan(y,viewData.clipYmax) || \
                                        lessThan(z,viewData.clipZmin) ||     \
                                        greaterThan(z,viewData.clipZmax) || \
                                        isNaNPoint(x,y,z))
#define NotPoint        (SHRT_MAX)
#define eqNANQ(x)       (x == NotPoint)


#define swap(a,b) {a^=b; b^=a; a^=b;}

#define viewportCreateMASK CWBackPixel | CWBorderPixel | CWEventMask | CWCursor | CWColorma
#define viewportTitleCreateMASK CWBackPixel | CWBorderPixel | CWCursor | CWColormap | CWEver
#define carefullySetFont(gc,font) if (font != serverFont) XSetFont(dsply,gc,font->fid)


#define viewportMASK    (KeyPressMask + ButtonPressMask + ExposureMask)
#define titleMASK       (ExposureMask)

#define lineWidth       1
#define lineHeight      1

#define titleColor      monoColor(36)
#define titleHeight     24
#define appendixHeight  0

#define viewWidth       400
#define viewHeight      400
#define viewYmax        vwInfo.height
#define viewYmin        vwInfo.y
#define viewXmax        vwInfo.width
#define viewXmin        vwInfo.x

#define GC9991 ((GC)9991)



/* For smooth shading buffers. Should be screen resolution size,
   and one for each of screen width and height may be needed, or
   it can be changed dynamically if desired. */

#ifdef RIOSplatform
#define ARRAY_WIDTH     1280 + 1  /* DisplayWidth(dsply,scrn) */
#define ARRAY_HEIGHT    1024 + 1  /* DisplayHeight(dsply,scrn) */
#else
#define ARRAY_WIDTH     1300  /* DisplayWidth(dsply,scrn) */
#define ARRAY_HEIGHT    1100  /* DisplayHeight(dsply,scrn) */
#endif
```

```
#define viewBorderWidth 0 /* make sure ps.h (postscript header) is the same */

#define initDeltaX        0.0
#define initDeltaY        0.0
#define initTheta         pi_half/2.0
#define initPhi           -pi_half/2.0

#define maxDeltaX         1500.0
#define maxDeltaY         1500.0
#define minScale          0.01
#define maxScale          1000.0

#define rotateFactor      0.2
#define scaleFactor       0.2
#define translateFactor 8

#define viewCursorForeground monoColor(166)
#define viewCursorBackground monoColor(5)

#define axesColor         52
#define buttonColor       120
#define labelColor        12

                    /*********************************/
                    /***        graph stuff        ***/
                    /*********************************/

#define graphBarLeft      76
#define graphBarTop       180
#define graphBarWidth     graphFont->max_bounds.width + 5
#define graphBarHeight    graphFont->max_bounds.ascent + graphFont->max_bounds.desc
#define graphBarDefaultColor    monoColor(85)
#define graphBarShowingColor    monoColor(45)
#define graphBarHiddenColor     monoColor(146)
#define graphBarSelectColor     monoColor(45)
#define graphBarNotSelectColor monoColor(145)

                    /*****************************/
                    /***        colors        ***/
                    /*****************************/

#define totalHuesConst  27

#define hueEnd            360
#define hueStep           (hueEnd/totalHuesConst)
```

```
#define black            BlackPixel(dsply,scrn)
#define white            WhitePixel(dsply,scrn)
#define numPlanes        1
#define numColors        10
#define startColor       0
#define endColor         (startColor+numColors)
#define maxColors        (DisplayCells(dsply,scrn)-1)
#define maxPlanes        (DefaultVisual((dpy),(scr))->bits_per_rgb)

#define colorStep        ((maxColors+1)/numColors)

                    /*********************************/
                    /***    Screen and Window Sizes   */
                    /*********************************/

#define physicalWidth   DisplayWidth(dsply,scrn)
#define physicalHeight  DisplayHeight(dsply,scrn)
#define deep            DisplayPlanes(dsply,scrn)

#define basicScreen      19

#define yes              1
#define no               0

#define pi_half          1.57079632
#define pi               3.14159265
#define three_pi_halves 4.71238898
#define two_pi           6.28318530
#define pi_sq            9.86960440

#define degrees_in_two_pi 57
#define d2Pi               57

#define nbuckets          128


#define anywhere         0

#ifdef DEBUG
#include "eventnames.h"
#endif

#define intSize          sizeof(int)
#define floatSize        sizeof(float)
```

```
/* Types so far are X, PS */
#define drawViewport(type) { drawPreViewport(type); drawTheViewport(type); }
#define spadDrawViewport()   spadMode++; drawTheViewport(X); spadMode--;



                      /*******************************/
                      /***       lighting panel    ***/
                      /*******************************/


/* These are the lighting panel buttons, they start at 101
   (numbers less than 101 are reserved for control panel buttons */

/* From ../include/actions.h */

#define lightingButtonsStart controlButtonsEnd3D

#define lightMove        (lightingButtonsStart)
#define lightMoveXY      (lightingButtonsStart+1)
#define lightMoveZ       (lightingButtonsStart+2)
#define lightAbort       (lightingButtonsStart+3)
#define lightReturn      (lightingButtonsStart+4)
#define lightTranslucent (lightingButtonsStart+5)

#define maxlightingButtons 6
#define lightingButtonsEnd (lightingButtonsStart + maxlightingButtons)

                      /*********************************/
                      /***       view volume panel   ***/
                      /*********************************/


/* These are the volume panel buttons, they start at 200
   (numbers less than 101 are reserved for control panel buttons */

#define volumeButtonsStart lightingButtonsEnd

#define volumeReturn     (volumeButtonsStart)
#define frustrumBut      (volumeButtonsStart+1)
#define clipXBut         (volumeButtonsStart+2)
#define clipYBut         (volumeButtonsStart+3)
#define clipZBut         (volumeButtonsStart+4)
#define perspectiveBut   (volumeButtonsStart+5)
#define clipRegionBut    (volumeButtonsStart+6)
#define clipSurfaceBut   (volumeButtonsStart+7)
#define volumeAbort      (volumeButtonsStart+8)

#define maxVolumeButtons 9
```

```
#define volumeButtonsEnd (volumeButtonsStart + maxVolumeButtons)

                        /**** quit panel ****/

#define quitButtonsStart volumeButtonsEnd

#define quitAbort        (quitButtonsStart)
#define quitReturn       (quitButtonsStart+1)
#define maxQuitButtons   2
#define quitButtonsEnd   (quitButtonsStart + maxQuitButtons)

                        /**** save panel ****/

#define saveButtonsStart quitButtonsEnd

#define saveExit         (saveButtonsStart)
#define pixmap           (saveButtonsStart+1)
#define ps               (saveButtonsStart+2)
#define maxSaveButtons   3
#define saveButtonsEnd   (saveButtonsStart + maxSaveButtons)

                   /*****************************************/
                   /***       buttons to be allocated     ***/
                   /*****************************************/

#define maxButtons3D    saveButtonsEnd

     /******* useful definitions *******/

#define CONTROLpanel 1
#define LIGHTpanel   2
#define VOLUMEpanel  3
#define CONTOURpanel 4
#define QUITpanel    5
#define SAVEpanel    6

#define machine0 0.0002

#define lightBitmap_width 16
#define lightBitmap_height 16
#define lightBitmap_x_hot 8
#define lightBitmap_y_hot 0
static char lightBitmap_bits[] = {
   0x00, 0x01, 0x00, 0x01, 0x00, 0x01, 0x04, 0x40, 0xc8, 0x27, 0x20, 0x08,
   0x10, 0x10, 0x16, 0x10, 0x10, 0xd0, 0x10, 0x10, 0x24, 0x08, 0x42, 0x44,
   0x40, 0x84, 0x80, 0x02, 0x80, 0x02, 0x00, 0x01};
```

```
#define lightMask_width 16
#define lightMask_height 16
#define lightMask_x_hot 8
#define lightMask_y_hot 0
static char lightMask_bits[] = {
   0x00, 0x01, 0x00, 0x01, 0x00, 0x01, 0x04, 0x40, 0xc8, 0x27, 0x20, 0x08,
   0x10, 0x11, 0x96, 0x12, 0x90, 0xd2, 0x90, 0x10, 0x24, 0x09, 0x42, 0x45,
   0x40, 0x85, 0x80, 0x03, 0x80, 0x02, 0x00, 0x01};

#define lightMASK ExposureMask
#define lightCursorForeground lightingTitleColor
#define lightCursorBackground foregroundColor

#define lightFontHeight (lightingFont->max_bounds.ascent+lightingFont->max_bounds

#define lightingAxesColor  monoColor(52)
#define lightingLabelColor monoColor(12)
#define lightingBoxColor   monoColor(138)
#define lightingLightColor monoColor(7)
#define lightingTitleColor monoColor(69)
#define lightingButtonColor monoColor(140)
#define lightingTransColor monoColor(140)
#define lightingTransArrowColor monoColor(100)
#define lightingTransLabelColor monoColor(207)

#define lightingAxesSize 175
#define lightingAxesX    61
#define lightingAxesY    28

#define lightAxesScale 110      /* the extent of the axes in object space */
#define lightScale 0.63         /* projected scale factor */

#define arrowHead (control->buttonQueue[lightTranslucent].buttonX + 5)
static viewTriple point0 = {0,0,0};

#define linkThing poly

#define spadActionMode
#define components

#define inside(A,B) (((XButtonEvent *)event)->x >= A && \
                     ((XButtonEvent *)event)->x <= B)

#define maxEyeDistance 2000.0
#define minEyeDistance 200.0
#define eyeIncrement   25.0
```

```
#define clipPlaneMin (-250.0)
#define clipPlaneMax 250.0
#define clipPlaneIncrement 10.0

#define quitMASK ExposureMask
#define quitCursorForeground monoColor(55)
#define quitCursorBackground monoColor(197)
#define quitTitleColor monoColor(69)
#define quitButtonColor monoColor(195)
#define quitFontHeight (quitFont->max_bounds.ascent+quitFont->max_bounds.descent)

#define saveMASK ExposureMask
#define saveCursorForeground monoColor(55)
#define saveCursorBackground monoColor(197)
#define saveTitleColor monoColor(70)
#define saveButtonColor monoColor(195)
#define saveFontHeight (saveFont->max_bounds.ascent+saveFont->max_bounds.descent)

#define SAFE_VALUE 892347

#define precisionFactor 1024

/* depthChecker turns on the extensive depth checking mechanisms
   for the depth sort algorithm. Without it, the hidden surface
   removal is just a sort by z which works remarkably well, but,
   is insufficient and, at times, may end up being incorrect */
#define depthChecker

#define axesOffset  5

#define eyeDistMessX (frusX(eyeWinX+27))
#define eyeDistMessY (frusY(eyeWinY-5))
#define hitherMessX (frusX(hitherWinX+15))
#define hitherMessY (frusY(hitherWinY))

#define clipXMessX (control->buttonQueue[clipXBut].buttonX + \
                    control->buttonQueue[clipXBut].xHalf)
#define clipXMessY (control->buttonQueue[clipXBut].buttonY + 2)
#define clipYMessX (control->buttonQueue[clipYBut].buttonX + \
                    control->buttonQueue[clipYBut].buttonWidth-2)
#define clipYMessY (control->buttonQueue[clipYBut].buttonY + \
                    control->buttonQueue[clipYBut].yHalf)
#define clipZMessX (control->buttonQueue[clipZBut].buttonX + \
                    control->buttonQueue[clipZBut].xHalf+4)
#define clipZMessY (control->buttonQueue[clipZBut].buttonY + \
```

```
                          control->buttonQueue[clipZBut].yHalf-4)

  #define volumeCursorForeground monoColor(68)
  #define volumeCursorBackground monoColor(197)


  #define hitherBoxColor  monoColor(141)
  #define hitherBoxTop    (frustrumMidY - 10)
  #define hitherBoxHeight 20


  #define clipButtonColor 144
  #define toggleColor 42
  #define arcColor 75


  #define arcSize   6
  #define tinyArc   5
  #define blank 4
  #define toggleX 190
  #define toggleY 280


  #define oldWay


  #define frusX(x) (control->buttonQueue[frustrumBut].buttonX + x)
  #define frusY(y) (control->buttonQueue[frustrumBut].buttonY + y)


  #define clipMessX 7
  #define clipMessY (control->buttonQueue[clipXBut].buttonY + 15)
     /* someotherFont holds title font (see main.c) */
  #define clipMessDy (globalFont->max_bounds.ascent/2 + \
                      globalFont->max_bounds.descent)
  static char *clipMess = "Clip Volume";


  #define eyeMess1Dy clipMessDy
  #define eyeMess1X 7
  #define eyeMess1Y (frustrumY + 40 + 3*eyeMess1Dy)
  static char *eyeMess1 = "Eye";


  #define eyeMess2X (globalFont->max_bounds.width + 14)
  #define eyeMess2Y (frustrumY + 40)
  #define eyeMess2Dy eyeMess1Dy


  #define leftRight
  #define newStuff


     /******* Define's *******/
        /*** box colors ***/
  #define boxInline       monoColor(140)
```

```
#define boxOutline      monoColor(140)
#define clipBoxInline   monoColor(148)
#define clipBoxOutline  monoColor(148)

#define lightB          205
#define lightPotA       (control->buttonQueue[lightMoveZ].buttonY - 15)
#define lightPotB       (control->buttonQueue[lightMoveZ].buttonY +\
                          control->buttonQueue[lightMoveZ].buttonHeight + 7)
#define lightTransL     (control->buttonQueue[lightTranslucent].buttonX - 20)

#define volumeTitleColor monoColor(77)
#define volumeTitleA    190
#define volumeTitleB    217


#define volumeMASK      ExposureMask


#define frustrumColor   monoColor(147)
#define frustrumX       30
#define frustrumY       20
#define frustrumLength  100
#define frustrumMidY    70    /* frustrumY + frustrumLength/2 */
#define frustrumBotY    (frustrumY + frustrumLength)
#ifdef  newStuff
#define frustrumMin     (control->buttonQueue[frustrumBut].xHalf)
#define frustrumMax     (frustrumMin + \
                          (control->buttonQueue[frustrumBut].xHalf))
#endif

#define hitherColor     monoColor(68)   /* clipping plane */
#define hitherMinX      (frustrumX + 5)
#define hitherMaxX      (frustrumMin - 30)
#define hitherWinX      (hitherMinX - 5)
#define hitherWinY      (frustrumBotY + 10)
#define hitherWidth     (hitherMaxX - hitherMinX + 10)
#define hitherHeight    20
#define hitherBarY      (hitherWinY + 10)   /* hitherWinY + hitherHeight/2 */

#ifdef  newStuff
#define eyeColor        monoColor(131)
#define eyeMinX         frustrumMin
#define eyeMaxX         frustrumMax
#define eyeWinX         (eyeMinX - 5)
#define eyeWinY         hitherWinY
#define eyeWidth        (eyeMaxX - eyeMinX + 10)
#define eyeHeight       hitherHeight
#define eyeBarY         hitherBarY
```

```
#endif

#define volumeButtonColor        monoColor(157)

#define frustrumWindowX          30
#define frustrumWindowY          28
#define frustrumWindowWidth      (controlWidth - 60)
#define frustrumWindowHeight     (frustrumBotY + 40)

/**** clip volume ****/
#define lengthFace      80
#ifdef  rightLeft
#define backFaceX       190
#endif
#ifdef  leftRight
#define backFaceX       33
#endif
#define backFaceY       255
#define deltaFace       25
#define zLength         35.355 /* sqrt(2*deltaFace^2) */
#ifdef  rightLeft
#define frontFaceX      (backFaceX - deltaFace)
#endif
#ifdef  leftRight
#define frontFaceX      (backFaceX + deltaFace)
#endif
#define frontFaceY      (backFaceY + deltaFace)


#define majorAxis       lengthFace    /* size of the potentiometers */
#define minorAxis       20
#define midAxis         40

#define clipXButX       backFaceX
#define clipXButY       (backFaceY-30)

#ifdef  rightLeft
#define clipYButX       (frontFaceX - minorAxis - 10)
#endif
#ifdef  leftRight
#define clipYButX       (frontFaceX + lengthFace + 10)
#endif
#define clipYButY       frontFaceY

#ifdef  rightLeft
#define clipZButX       clipYButX  /* align left side */
```

```
#endif
#ifdef  leftRight
#define clipZButX        (clipYButX+minorAxis-midAxis) /* align right side */
#endif
#define clipZButY        clipXButY

#define zFactor          0.6   /* ratio of clipZBut box & actual input area */
#define minDistXY        0.1   /* min distance between normalized clip faces */
#define minDistZ         0.06  /* 2/3 of XY */


#ifdef  rightLeft
#define AA               (clipZButX + midAxis)
#define BB               clipZButY
#define CC               backFaceX
#define DD               backFaceY
#define EE               frontFaceX
#define FF               frontFaceY
#define clipZButTopEndX ((AA+BB+CC-DD)/2)
#define clipZButTopEndY ((AA+BB-CC+DD)/2)
#define clipZButBotEndX ((AA+BB+EE-FF)/2)
#define clipZButBotEndY ((AA+BB-EE+FF)/2)
#endif

#ifdef  leftRight
#define AA               clipZButX
#define BB               clipZButY
#define CC               (backFaceX + majorAxis)
#define DD               backFaceY
#define EE               (frontFaceX + majorAxis)
#define FF               frontFaceY

#define clipZButTopEndX ((AA-BB+CC+DD)/2)
#define clipZButTopEndY ((BB-AA+CC+DD)/2)
#define clipZButBotEndX ((AA-BB+EE+FF)/2)
#define clipZButBotEndY ((BB-AA+EE+FF)/2)

#endif

  /* upper limit as to how many kinds of files could be written (numBits-1) */
#define numBits (8*sizeof(int))
#define StellarColors 9
```

7.2.4   static variables

```
static char *event_name[] = {
  "",                         /* 0  */
  "",                         /* 1  */
  "KeyPress",                 /* 2  */
  "KeyRelease",               /* 3  */
  "ButtonPress",              /* 4  */
  "ButtonRelease",            /* 5  */
  "MotionNotify",             /* 6  */
  "EnterNotify",              /* 7  */
  "LeaveNotify",              /* 8  */
  "FocusIn",                  /* 9  */
  "FocusOut",                 /* 10 */
  "KeymapNotify",             /* 11 */
  "Expose",                   /* 12 */
  "GraphicsExpose",           /* 13 */
  "NoExpose",                 /* 14 */
  "VisibilityNotify",         /* 15 */
  "CreateNotify",             /* 16 */
  "DestroyNotify",            /* 17 */
  "UnmapNotify",              /* 18 */
  "MapNotify",                /* 19 */
  "MapRequest",               /* 20 */
  "ReparentNotify",           /* 21 */
  "ConfigureNotify",          /* 22 */
  "ConfigureRequest",         /* 23 */
  "GravityNotify",            /* 24 */
  "ResizeRequest",            /* 25 */
  "CirculateNotify",          /* 26 */
  "CirculateRequest",         /* 27 */
  "PropertyNotify",           /* 28 */
  "SelectionClear",           /* 29 */
  "SelectionRequest",         /* 30 */
  "SelectionNotify",          /* 31 */
  "ColormapNotify",           /* 32 */
  "ClientMessage",            /* 33 */
  "MappingNotify"             /* 34 */
  };
```

This is a description of script character labels for the x, y, and z axes

⟨*view3d*⟩+≡

```
  static float axes[3][6] = {{-117,0,0,117,0,0},      /* x axis */
                             {0,-117,0,0,117,0},      /* y axis */
                             {0,0,-117,0,0,117}};     /* z axis */

  /* text labels are currently used */
  static float labels[basicScreen][7] = {
    {105,0,4,106,0,3,labelColor},      /* script x label - 4 segments */
```

```
    {106,0,3,112,0,10,labelColor},
    {112,0,10,114,0,9,labelColor},
    {106,0,10,113,0,3,labelColor},
    {0,106,9,0,107,10,labelColor},    /* script y label - 7 segments */
    {0,107,10,0,107,6,labelColor},
    {0,107,6,0,113,5,labelColor},
    {0,113,10,0,113,-3,labelColor},
    {0,113,-3,0,111,-5,labelColor},
    {0,111,-5,0,110,-1,labelColor},
    {0,110,-1,0,114,3,labelColor},
    {0,5,114,0,6,115,labelColor},     /* script z label - 8 segments */
    {0,6,115,0,11,116,labelColor},
    {0,11,116,0,12,113,labelColor},
    {0,12,113,0,10,111,labelColor},
    {0,10,111,0,11,110,labelColor},
    {0,11,110,0,11,103,labelColor},
    {0,11,103,0,9,102,labelColor},
    {0,9,102,0,9,105,labelColor}};
```

7.2.5   structs

⟨*view3d*⟩+≡
```
  typedef struct _buttonStruct {
    int           buttonKey, pot, mask;
    short         buttonX, buttonY, buttonWidth, buttonHeight, xHalf, yHalf;
    Window        self;
    char          *text;
    int           textColor,textHue,textShade;
  } buttonStruct;
```

⟨*view3d*⟩+≡
```
  typedef struct _controlPanelStruct {
    Window        controlWindow, messageWindow, colormapWindow;
    char          message[40];
    buttonStruct  buttonQueue[maxButtons3D];
  } controlPanelStruct;
```

⟨*view3d*⟩+≡
```
  typedef struct _mouseCoord {
      float       x, y;
  } mouseCoord;
```

⟨*view3d*⟩+≡
```
  typedef struct _meshStruct {
    float         N0[4], N1[4];   /* the fourth element is Zmin */
  } meshStruct;
```

⟨*view3d*⟩+≡
```
  typedef struct _points3D {
    float         xmin, xmax,
                  ymin, ymax,
                  xstep, ystep,
                  zmin, zmax,
                  scaleToView;
    float         *zPoints;
    int           xnum, ynum,
                  nextRow,
                  style;
    meshStruct    *normData;   /* list of normals */
  } points3D;
```

⟨*view3d*⟩+≡
```
  typedef struct _colorBuffer {
    int      indx;
    char     axes;
  } colorBuffer;
```

⟨*view3d*⟩+≡
```
  typedef struct _point {
    float         x, y, z;
    int           flag;
  } point;
```

One of the (many) sloppy things that need to be cleaned up is the viewPoints structure. a lot of stuff in it is used solely for the function of two variables stuff. they should be moved to the fun2Var substructure.

⟨*view3d*⟩+≡
```
typedef struct _viewPoints {
  int               viewportKey;
  char              title[80];
  Window            viewWindow, titleWindow;
  float             deltaX, deltaY,
                    scale, scaleX, scaleY, scaleZ,
                    theta, phi,
                    deltaX0, deltaY0,    /* initial values */
                    scale0, transX, transY, transZ, thetaObj, phiObj,
                    theta0, phi0, theta1, phi1, axestheta, axesphi;
  float             deltaZ, deltaZ0;
  controlPanelStruct *controlPanel;
  int               axesOn, regionOn, monoOn;
  int               zoomXOn, zoomYOn, zoomZOn;
  int               originrOn, objectrOn;
  int               xyOn, xzOn, yzOn;
  int               originFlag;
  int               justMadeControl, haveControl,
                    closing, allowDraw, needNorm;
  points3D          meshData;
  float             lightVector[3], translucency;
  int               hueOffset, numberOfHues, hueTop, diagonals;
  struct _viewPoints *prevViewport, *nextViewport;
} viewPoints;
```

⟨*view3d*⟩+≡
```
typedef struct _controlXY {
  int         putX, putY;
} controlXY;
```

7.2.6   extern references

⟨*view3d*⟩+≡

```
  extern Display           *dsply;
  extern XFontStruct       *globalFont, *buttonFont, *headerFont,
                           *titleFont, *graphFont,
                           *lightingFont, *volumeFont, *quitFont, *saveFont,
                           *serverFont;
  extern XrmDatabase       rDB;

  extern char              scaleReport[5], deltaXReport[5], deltaYReport[5];
  extern unsigned long     *spadColors;
  extern int               followMouse, gotToggle, viewportKeyNum;
  extern Window            rtWindow, quitWindow, saveWindow;
  extern GC                globalGC1, globalGC2, anotherGC, globGC, trashGC,
                           componentGC, opaqueGC, renderGC,
                           controlMessageGC, lightingGC, volumeGC, quitGC,
                           saveGC, graphGC;
  extern XSizeHints        viewSizeHints;
  extern HashTable         *table;
  extern Colormap          colorMap;
  extern int               Socket, ack;

  extern GC                processGC;
  extern viewPoints        *viewport;
  extern controlPanelStruct *control;
  extern XGCValues         gcVals;
  extern char              *s;
  extern int               someInt;

  extern unsigned long     foregroundColor, backgroundColor;
  extern int               mono, totalColors,
                           totalHues, totalSolidShades, totalSolid,
                           totalDitheredAndSolids,totalShades;

  extern int               drawMore;
  extern int               spadMode,spadDraw;
  extern int               spadSignalReceived;
  extern int               inNextEvent;
  extern jmp_buf           jumpFlag;

  extern char              errorStr[80];

  extern view3DStruct      viewData;

  extern Window            lightingWindow, lightingAxes;
```

```
extern float          lightPointer[3], tempLightPointer[3];
extern float          lightIntensity, tempLightIntensity;
extern float          backLightIntensity;

extern char           filename[256];


   /** stuff from draw viewport routines */
extern float          sinTheta, sinPhi, cosTheta, cosPhi,
                      viewScale, viewScaleX, viewScaleY, viewScaleZ, reScale;
extern int            xCenter, yCenter;
extern XWindowAttributes vwInfo;
extern XWindowAttributes graphWindowAttrib;
extern XPoint         *quadMesh;
extern int            *xPts;
extern XImage         *imageX;

extern float          eyePoint[3];

extern XPoint         polygonMesh[20];

extern int            saveFlag;
extern int            firstTime, noTrans, startup;
extern int            redrawView;
extern int            finishedList, redoSmooth, redoColor, zoomed,
                      rotated, switchedPerspective, changedEyeDistance,
                      translated, changedIntensity, movingLight, writeImage,
                      pixelSetFlag, redoDither, multiColorFlag;
extern poly           *quickList;

extern int            viewAloned;

extern viewTriple     corners[8], clipCorners[8];
extern boxSideStruct  box[6], clipBox[6];
extern int            axesXY[3][4];
extern float          axesZ[3][2];

extern viewTriple     *splitPoints;
extern int            resMax;

extern Window         volumeWindow;
extern int            frustrumVertex;
extern int            doingPanel;
extern int            doingVolume;
extern int            screenX;
extern float          xClipMinN, xClipMaxN,
```

```
                                   yClipMinN, yClipMaxN,
                                   zClipMinN, zClipMaxN,
                                   clipValue;

    extern float                   pzMin, pzMax;

    extern int                     maxGreyShade;

    extern char                    propertyName[];
    extern char                    propertyBuffer[];

    #undef R1
    #define R1 RR1

    extern float                   transform[4][4], transform1[4][4],
                                   R[4][4], R1[4][4], S[4][4], T[4][4], I[4][4];
    extern float                   vxmax,vxmin,vymax,vymin,
                                   wxmax,wxmin,wymax,wymin,wzmax,wzmin;

    extern polyList                *scanList[ARRAY_HEIGHT];
    extern int                     scanline, polyCount;
    extern float                   xleft, xright;

    extern colorBuffer             cBuffer[ARRAY_WIDTH];
    extern float                   zBuffer[ARRAY_WIDTH];

    extern float                   zC, dzdx;
    extern float                   intersectColor[2], dcolor;
    extern triple                  dpt, dnorm;

    extern float                   Cspec, Cdiff, Camb, coeff, lum, saturation;

    extern Pixmap                  viewmap;
    extern int                     viewmap_valid;
    extern int                     smoothHue;
    extern int                     smoothConst;
    extern int                     smoothError;
```

7.2.7   forward references

⟨*view3d*⟩+≡

```
  extern viewTriple * traverse(int );
  extern float absolute(float );
  extern float getRandom(void );
  extern triple normDist(void );
  extern void goodbye(int);
  extern int initButtons(buttonStruct * );
  extern int writeViewport(int );
  extern int initVolumeButtons(buttonStruct * );
  extern void makeVolumePanel(void );
  extern void drawClipXBut(void );
  extern void drawClipYBut(void );
  extern void drawClipZBut(void );
  extern void drawClipVolume(void );
  extern void drawHitherControl(void );
  extern void drawEyeControl(void );
  extern void drawFrustrum(void );
  extern void drawVolumePanel(void );
  extern void drawColorMap(void);
  extern void writeControlTitle(Window );
  extern void clearControlMessage(void);
  extern void writeControlMessage(void);
  extern void drawControlPanel(void);
  extern controlXY getControlXY(int );
  extern controlPanelStruct * makeControlPanel(void);
  extern void putControlPanelSomewhere(int );

  extern void matrixMultiply4x4(float [4][4] , float [4][4] , float [4][4]);
  extern void vectorMatrix4(float [4] , float [4][4] , float [4]);
  extern void ROTATE(float [4][4]);
  extern void ROTATE1(float [4][4]);
  extern void SCALE(float  , float  , float  , float [4][4]);
  extern void TRANSLATE(float  , float  , float  , float [4][4]);
  extern void closeViewport(void);
  extern float phong(triple  , float [3]);
  extern int hueValue(float );
  extern int getHue(float );
  extern float Value(float  , float  , float );
  extern RGB hlsTOrgb(float  , float  , float );
  extern poly * merge(poly *  , poly *  , int (*)(poly *  , poly * ));
  extern poly * msort(poly *  , int  , int  , int (*)(poly *  , poly * ));
  extern void drawLineComponent(poly *  , int );
  extern void drawOpaquePolygon(poly *  , GC  , GC  , int );
  extern poly * copyPolygons(poly * );
```

```
extern void minMaxPolygons(poly * );
extern int polyCompare(poly *  , poly * );
extern void calcEyePoint(void );
extern void drawRenderedPolygon(poly *  , int );
extern void freePointResevoir(void);
extern void freeListOfPolygons(poly * );
extern void drawPolygons(int );
extern int lessThan(float  , float );
extern int greaterThan(float  , float );
extern int isNaN(float );
extern int isNaNPoint(float  , float  , float );
extern int equal(float  , float );
extern void getMeshNormal(float, float, float, float, float, float,
                          float, float, float, float, float, float [3]);
extern void normalizeVector(float * );
extern float dotProduct(float *  , float *  , int );
extern void project(viewTriple *  , XPoint *  , int );
extern void projectAPoint(viewTriple * );
extern void projectAllPoints(void);
extern void projectAllPolys(poly * );
extern void projectAPoly(poly * );
extern void projectStuff(float, float, float, int *, int *, float *);
extern int makeLightingPanel(void);
extern void drawLightingAxes(void);
extern void drawLightTransArrow(void);
extern void drawLightingPanel(void);
extern int initLightButtons(buttonStruct * );
extern int readViewman(void *  , int );
extern void scalePoint(viewTriple * );
extern int spadAction(void);
extern void writeTitle(void);
extern void drawPreViewport(int );
extern void drawTheViewport(int );
extern int keepDrawingViewport(void);
extern viewPoints * makeViewport(void);
extern void postMakeViewport(void);
extern mouseCoord getPotValue(short  , short  , short  , short );
extern mouseCoord getLinearPotValue(short  , short  , short  , short );
extern void buttonAction(int );
extern void processEvents(void);
extern int initQuitButtons(buttonStruct * );
extern int makeQuitPanel(void);
extern void drawQuitPanel(void);
extern void scaleComponents(void);
extern void makeTriangle(int  , int  , int );
extern void triangulate(void);
```

```
extern void readComponentsFromViewman(void);
extern void calcNormData(void);
extern viewPoints * make3DComponents(void);
extern void draw3DComponents(int );

extern char getCBufferAxes(int );
extern void putCBufferAxes(int   , char );
extern int getCBufferIndx(int );
extern void putCBufferIndx(int   , int );
extern void putZBuffer(int   , float );
extern float getZBuffer(int );
extern void putImageX(int   , char );
extern void drawPhongSpan(triple  , float [3] , int );
extern void scanPhong(int );
extern void boxTObuffer(void );
extern void clipboxTObuffer(void );
extern void axesTObuffer(void );
extern void scanLines(int );
extern void freePolyList(void );
extern void showAxesLabels(int );
extern void changeColorMap(void );
extern void drawPhong(int );
extern int initSaveButtons(buttonStruct * );
extern int makeSavePanel(void);
extern void drawSavePanel(void);
extern int main(void);
extern void mergeDatabases(void);

extern int PSCreateFile(int , Window , Window , char * );
extern int GdrawsDrawFrame(int , Window , Window , char * );
extern int GdrawsSetDimension(Window , Window );
extern int GDrawImageString(GC , Window , int , int , char * , int , int );
extern int GDrawArc(GC , Window , int , int , unsigned int , unsigned int , int , int , int
extern int GDrawLine(GC , Window , int , int , int , int , int );
extern int GDrawLines(GC , Window , XPoint * , int , int , int );
extern int GDrawPoint(Window , GC , int , int , int );
extern int GDrawString(GC , Window , int , int , char * , int , int );
extern int GFillArc(GC , Window , int , int , unsigned int , unsigned int , int , int , int
extern int PSGlobalInit(void );
extern int PSInit(Window , Window );
extern int PSCreateContext(GC , char * , int , int , int , float , float );
extern char * PSfindGC(GC );
extern int GSetForeground(GC , float , int );
extern int GSetBackground(GC , float , int );
extern int GSetLineAttributes(GC , int , int , int , int , int );
extern int PSClose(void );
```

```
extern int centerX(GC , char * , int , int );
extern int centerY(GC , int );
extern int PSColorPolygon(float , float , float , XPoint * , int );
extern int PSColorwOutline(float , float , float , XPoint * , int );
extern int PSDrawColor(float , float , float , XPoint * , int );
extern int PSFillPolygon(GC , XPoint * , int );
extern int PSFillwOutline(GC , XPoint * , int );
extern HashTable * XCreateAssocTable(int );
extern void XMakeAssoc(Display * , HashTable * , Window , int * );
extern int * XLookUpAssoc(Display * , HashTable * , Window );
extern void XDeleteAssoc(Display * , HashTable * , Window );
extern int GDrawRectangle(GC , Window , short , short , short , short , int );
extern int GDraw3DButtonOut(GC , Window , short , short , short , short , int );
extern int GDraw3DButtonIn(GC , Window , short , short , short , short , int );
extern int GDrawPushButton(Display * , GC , GC , GC , Window , short , short , sh
#ifdef _GFUN_C
static void filecopy(FILE * , FILE * );
static int TrivEqual(Window , Window );
static int TrivHashCode(Window , int );
#endif
```

7.2.8   global variables

⟨*view3d*⟩+≡

```
  float A[4][4];
  int ack=1;
  GC anotherGC;
  float array[4][4];
  int axesXY[3][4];
  float axesZ[3][2];

  float B[4][4];
  unsigned long backgroundColor;
  float backLightIntensity = 1.0;
  boxSideStruct box[6];
  XFontStruct *buttonFont;

  float Camb = 0.3;
  colorBuffer cBuffer[ARRAY_WIDTH];
  float Cdiff = 0.4;
  int changedEyeDistance;
  int changedIntensity;
  boxSideStruct clipBox[6];
  viewTriple clipCorners[8];
  float clipValue;               /* mouse input */
  float coeff = 35.0;
  Colormap colorMap;
  GC componentGC;
  controlPanelStruct *control;
  GC controlMessageGC;
  viewTriple corners[8];
  float cosPhi;
  float cosTheta;
  float Cspec = 0.30;

  float D[4];
  float dcolor;
  char deltaXReport[5];
  char deltaYReport[5];
  triple dnorm;
  int doingPanel=CONTROLpanel; /* rewrite titles in proper panel */
  int doingVolume;
  triple dpt;
  int drawMore;
  Display *dsply;
  float dzdx;
```

```
float E[4][4];
char *envAXIOM; /* used for ps file paths */
char errorStr[80];
static char *eyeMess2 = "Reference";
float eyePoint[3];

float F[4];
char filename[256]; /* used for writing viewport info out to a file */
int finishedList=no;
int firstTime=yes;
int flatClipBoxX[8];
int flatClipBoxY[8];
int followMouse=no;
unsigned long foregroundColor;
int frustrumVertex;

GCptr GChead=NULL; /* ptr to head of ps GC linked list */
XGCValues gcVals;
XFontStruct *globalFont;
GC globGC;
GC globalGC1;
GC globalGC2;
int gotToggle = no;
XFontStruct *graphFont;
GC graphGC;
XWindowAttributes graphWindowAttrib;

XFontStruct *headerFont;

float I[4][4];
XImage *imageX;
int inNextEvent=no; /* true just before a call to XNextEvent */
float intersectColor[2];

jmp_buf jumpFlag;

int last_tip_lat_x;
int last_tip_lat_y;
int last_tip_long_x;
int last_tip_long_y;
Window lightingAxes;
XFontStruct *lightingFont;
GC lightingGC;
Window lightingWindow;
float lightIntensity=1.0;
float lightPointer[3];
```

```
float lum;

int maxGreyShade=0;
int mono;
int movingLight = no;
int multiColorFlag = no;

int noTrans = yes;

GC opaqueGC;

int pixelSetFlag = no;
float point_norm[3];
int polyCount;
XPoint polygonMesh[20];
GC processGC;
char propertyBuffer[256];
char propertyName[14];
char *PSfilename; /* output file name used in user directory */
int psInit=no; /* need to call globaInitPs() each run */
pointInfo ptIA;
pointInfo ptIB;
pointInfo ptIC;
float pzMax;
float pzMin;

XPoint *quadMesh;
poly *quickList;
XFontStruct *quitFont;
GC quitGC;
Window quitWindow;

float R[4][4];
float R1[4][4];
XrmDatabase rDB;
int redoColor = no;
int redoDither = no;
int redoSmooth = no;
int redrawView = no;    /* set to yes when returning from subpanels */
GC renderGC;
int resMax=0;  /* number of points in the split point resevoir */
float reScale;
static int rotateX;
int rotated=yes;
static int rotateY;
static int rotateR;
```

```
Window rtWindow;

char *s;
float S[4][4];
float saturation = 0.8;
int saveFlag=no;
XFontStruct *saveFont;
GC saveGC;
Window saveWindow;
char scaleReport[5];
int scanline;
polyList *scanList[ARRAY_HEIGHT];
int screenX;  /* global point indicating mouse position on frustrum screen */
int scrn;
XFontStruct *serverFont;
float sinPhi;
float sinTheta;
int smoothConst = 50;
int smoothError = no;
int smoothHue;
int Socket=1;
int someInt;
unsigned long *spadColors;
int spadDraw=no; /* yes if drawing viewport for an Axiom command */
int spadMode=no; /* yes if receiving Axiom command and calling drawViewport */
/* yes if current state is a result of a signal from Axiom */
int spadSignalReceived=0;
viewTriple *splitPoints;
int startup = yes;
int switchedPerspective;

float T[4][4];
HashTable *table;
float tempLightIntensity;
float tempLightPointer[3];
XFontStruct *titleFont;
int totalColors;
int totalDithered;
int totalDitheredAndSolids;
int totalHues;
int totalSolid;
int totalSolidShades;
float transform[4][4];
float transform1[4][4];
int translated = yes;
GC trashGC;
```

totalShades is initially set to totalShadesConst. If X cannot allocate 8 shades for each hue, totalShades is decremented. there is currently only a check for this value to be positive. −¿ something to add: change over to monochrome if totalShades=0. just modify the spadcolors.c file. spadcolors.c has been modified so that it returns the value for totalShades. since the return value had previously been unused, a modification in this way ensures continued support of other routines calling this function (e.g. hypertex stuff).

⟨*view3d*⟩+≡

```
int totalShades;

int viewAloned; /** if not connected to Axiom **/
view3DStruct viewData;
Pixmap viewmap;
int viewmap_valid = 0;
viewPoints *viewport;
int viewportKeyNum=0;
float viewScale;
float viewScaleX;
float viewScaleY;
float viewScaleZ;
XSizeHints viewSizeHints;
XFontStruct *volumeFont;
GC volumeGC;
Window volumeWindow;
XWindowAttributes vwInfo;

Atom wm_delete_window;
int writeImage = no;

int xCenter;
float xClipMaxN; /* normalized values for clip volume */
float xClipMinN; /* normalized values for clip volume */
float xleft = (float)0;
int *xPts;  /* pointer to projected points (x followed by y) */
float xright = (float)ARRAY_WIDTH;

int yCenter;
float yClipMaxN; /* normalized values for clip volume */
float yClipMinN; /* normalized values for clip volume */

float zBuffer[ARRAY_WIDTH];
float zC;
float zClipMaxN; /* normalized values for clip volume */
float zClipMinN; /* normalized values for clip volume */
int zoomed=yes;
```

7.3 Code

7.3.1 initButtons

Creates the fields for each button window in the three dimensional control panel, and returns the number of buttons created.

⟨*view3d*⟩+≡
  ⟨*gfun.c*⟩

```
int initButtons(buttonStruct *buttons) {
  int PBY = 297; /* panel button Y coordinate at which buttons appear */
  int ii, num = 0;
  /* Rotate, Zoom, and Translate Potentiometer Buttons */
  /* Title:  "Rotate" */
  ii = rotate;
  buttons[ii].buttonX       = XEDGE;    buttons[ii].buttonY       = 85;
  buttons[ii].buttonWidth   = 110;      buttons[ii].buttonHeight   = PH;
  buttons[ii].buttonKey     = ii;
  buttons[ii].pot           = yes;  /* rotate is a potentiometer */
  buttons[ii].mask          = potMASK;
  buttons[ii].textColor     = 139;  /* line color of rotate dial */
  buttons[ii].xHalf         = buttons[ii].buttonWidth/2;
  buttons[ii].yHalf         = buttons[ii].buttonHeight/2;
  ++num;
  /* Title:  "Scale" */
  ii = zoom;
  buttons[ii].buttonX       = 121;   buttons[ii].buttonY       = 85;
  buttons[ii].buttonWidth   = 58;    buttons[ii].buttonHeight   = PH;
  buttons[ii].buttonKey     = ii;
  buttons[ii].pot           = yes;  /* zoom(scale) is a potentiometer */
  buttons[ii].mask          = potMASK;
  buttons[ii].textColor     = 165;  /* line color of scale arrow */
  buttons[ii].xHalf         = buttons[ii].buttonWidth/2;
  buttons[ii].yHalf         = buttons[ii].buttonHeight/2;
  ++num;
  /* Title:  "Translate" */
  ii = translate;
  buttons[ii].buttonX       = 185;   buttons[ii].buttonY      = 85;
  buttons[ii].buttonWidth   = 110;   buttons[ii].buttonHeight = PH;
  buttons[ii].buttonKey     = ii;
  buttons[ii].pot           = yes;  /* translate is a potentiometer */
  buttons[ii].mask          = potMASK;
  buttons[ii].textColor     = 21;   /* line color of translate arrows */
  buttons[ii].xHalf         = buttons[ii].buttonWidth/2;
  buttons[ii].yHalf         = buttons[ii].buttonHeight/2;
  ++num;
```

```
/* All the rest of the buttons are regular, toggle only buttons and
   have the potentiometer variable set to "no". */
/* First Row of Buttons */
/* The four rendering mode buttons:
   wireframe, hiddenline solid, hiddenline shaded and smooth shaded **/
/* Wiefram mesh */
ii = transparent;
buttons[ii].buttonX        = XEDGE;    buttons[ii].buttonY       = PBY;
buttons[ii].buttonWidth    = 45;       buttons[ii].buttonHeight  = BH;
buttons[ii].buttonKey      = ii;
buttons[ii].pot            = no;
buttons[ii].mask           = buttonMASK;
buttons[ii].text           = "Wire";
buttons[ii].textColor      = modeColor;
buttons[ii].xHalf          = buttons[ii].buttonWidth/2;
buttons[ii].yHalf          = buttons[ii].buttonHeight/2;
++num;
/* Hidden surface mesh */
ii = opaqueMesh;
buttons[ii].buttonX        = 55;   buttons[ii].buttonY       = PBY;
buttons[ii].buttonWidth    = 53;   buttons[ii].buttonHeight = BH;
buttons[ii].buttonKey      = ii;
buttons[ii].pot            = no;
buttons[ii].mask           = buttonMASK;
buttons[ii].text           = "Solid";
buttons[ii].textColor      = modeColor;
buttons[ii].xHalf          = buttons[ii].buttonWidth/2;
buttons[ii].yHalf          = buttons[ii].buttonHeight/2;
++num;
/* Lambertian polygon fill with phong illumination model */
ii = render;
buttons[ii].buttonX        = 113;  buttons[ii].buttonY        = PBY;
buttons[ii].buttonWidth    = 53;   buttons[ii].buttonHeight   = BH;
buttons[ii].buttonKey      = ii;
buttons[ii].pot            = no;
buttons[ii].mask           = buttonMASK;
buttons[ii].text           = "Shade";
buttons[ii].textColor      = modeColor;
buttons[ii].xHalf          = buttons[ii].buttonWidth/2;
buttons[ii].yHalf          = buttons[ii].buttonHeight/2;
++num;
/* Phong smooth shading and illumination */
ii = smooth;
buttons[ii].buttonX        = 171;  buttons[ii].buttonY        = PBY;
buttons[ii].buttonWidth    = 59;   buttons[ii].buttonHeight   = BH;
buttons[ii].buttonKey      = ii;
```

```
buttons[ii].pot            = no;
buttons[ii].mask           = buttonMASK;
buttons[ii].text           = "Smooth";
buttons[ii].textColor      = modeColor;
buttons[ii].xHalf          = buttons[ii].buttonWidth/2;
buttons[ii].yHalf          = buttons[ii].buttonHeight/2;
++num;
/* Reset View Position Button */
ii = resetView;
buttons[ii].buttonX     = 240;    buttons[ii].buttonY      = PBY;
buttons[ii].buttonWidth = 53;   buttons[ii].buttonHeight   = BH;
buttons[ii].buttonKey      = ii;
buttons[ii].pot            = no;
buttons[ii].mask           = buttonMASK;
buttons[ii].text           = "Reset";
buttons[ii].textColor      = 149;
buttons[ii].xHalf          = buttons[ii].buttonWidth/2;
buttons[ii].yHalf          = buttons[ii].buttonHeight/2;
++num;
/* Second Row of Buttons */
/* update y displacement of buttons row */
PBY=PBY+BH+3;
/* Bounding Region On/Off */
ii = region3D;
buttons[ii].buttonX     = XEDGE;    buttons[ii].buttonY      = PBY;
buttons[ii].buttonWidth = 58;        buttons[ii].buttonHeight  = BH;
buttons[ii].buttonKey   = ii;
buttons[ii].pot         = no;
buttons[ii].mask        = buttonMASK;
buttons[ii].text        = "Bounds";
buttons[ii].textColor   = 6;
buttons[ii].xHalf       = buttons[ii].buttonWidth/2;
buttons[ii].yHalf       = buttons[ii].buttonHeight/2;
++num;
/* Axes Turned On/Off */
ii = axesOnOff;
buttons[ii].buttonX     = 68;    buttons[ii].buttonY      = PBY;
buttons[ii].buttonWidth = 49;    buttons[ii].buttonHeight = BH;
buttons[ii].buttonKey   = ii;
buttons[ii].pot         = no;
buttons[ii].mask        = buttonMASK;
buttons[ii].text        = "Axes";
buttons[ii].textColor   = offColor;
buttons[ii].xHalf       = buttons[ii].buttonWidth/2;
buttons[ii].yHalf       = buttons[ii].buttonHeight/2;
++num;
```

```
/* Outline polygons with black lines in render mode */
ii = outlineOnOff;
buttons[ii].buttonX     = 122;    buttons[ii].buttonY      = PBY;
buttons[ii].buttonWidth = 70;     buttons[ii].buttonHeight = BH;
buttons[ii].buttonKey   = ii;
buttons[ii].pot         = no;
buttons[ii].mask        = buttonMASK;
buttons[ii].text        = "Outline";
buttons[ii].textColor   = offColor;
buttons[ii].xHalf       = buttons[ii].buttonWidth/2;
buttons[ii].yHalf       = buttons[ii].buttonHeight/2;
++num;
/* Display as if a 1-bit plane image */
ii = bwColor;
buttons[ii].buttonX     = 197; buttons[ii].buttonY      = PBY;
buttons[ii].buttonWidth = 33;  buttons[ii].buttonHeight = BH;
buttons[ii].buttonKey   = ii;
buttons[ii].pot         = no;
buttons[ii].mask        = buttonMASK;
buttons[ii].text        = "BW";
buttons[ii].textColor   = offColor;
buttons[ii].xHalf       = buttons[ii].buttonWidth/2;
buttons[ii].yHalf       = buttons[ii].buttonHeight/2;
++num;
/* Hide Control Panel */
ii = hideControl;
buttons[ii].buttonX     = 240;    buttons[ii].buttonY      = PBY;
buttons[ii].buttonWidth = 53;     buttons[ii].buttonHeight = BH;
buttons[ii].buttonKey   = ii;
buttons[ii].pot         = no;
buttons[ii].mask        = buttonMASK;
buttons[ii].text        = "Hide";
buttons[ii].textColor   = 149;
buttons[ii].xHalf       = buttons[ii].buttonWidth/2;
buttons[ii].yHalf       = buttons[ii].buttonHeight/2;
++num;
/* Third Row of Buttons */
/* update y displacement of buttons row */
PBY=PBY+BH+3;
/* Shows Lighting Control Panel */
ii = lighting;
buttons[ii].buttonX     = XEDGE;    buttons[ii].buttonY      = PBY;
buttons[ii].buttonWidth = 65;       buttons[ii].buttonHeight = BH;
buttons[ii].buttonKey   = ii;
buttons[ii].pot         = no;
buttons[ii].mask        = buttonMASK;
```

```
buttons[ii].text        = "Light";
buttons[ii].textColor   = 149;
buttons[ii].xHalf       = buttons[ii].buttonWidth/2;
buttons[ii].yHalf       = buttons[ii].buttonHeight/2;
++num;
/* Shows View Volume Control Panel */
ii = viewVolume;
buttons[ii].buttonX     = 75;    buttons[ii].buttonY      = PBY;
buttons[ii].buttonWidth = 100;   buttons[ii].buttonHeight = BH;
buttons[ii].buttonKey   = ii;
buttons[ii].pot         = no;
buttons[ii].mask        = buttonMASK;
buttons[ii].text        = "View Volume";
buttons[ii].textColor   = 149;
buttons[ii].xHalf       = buttons[ii].buttonWidth/2;
buttons[ii].yHalf       = buttons[ii].buttonHeight/2;
++num;
/* Shows Save Panel */
ii = saveit;
buttons[ii].buttonX     = 180;   buttons[ii].buttonY      = PBY;
buttons[ii].buttonWidth = 50;    buttons[ii].buttonHeight = BH;
buttons[ii].buttonKey   = ii;
buttons[ii].pot         = no;
buttons[ii].mask        = buttonMASK;
buttons[ii].text        = "Save";
buttons[ii].textColor   = 149;
buttons[ii].xHalf       = buttons[ii].buttonWidth/2;
buttons[ii].yHalf       = buttons[ii].buttonHeight/2;
++num;
/* Exits from the viewport running */
ii = closeAll;
buttons[ii].buttonX     = 240;   buttons[ii].buttonY      = PBY;
buttons[ii].buttonWidth = 53;    buttons[ii].buttonHeight = BH;
buttons[ii].buttonKey   = ii;
buttons[ii].pot         = no;
buttons[ii].mask        = buttonMASK;
buttons[ii].text        = "Quit";
buttons[ii].textColor   = offColor;
buttons[ii].xHalf       = buttons[ii].buttonWidth/2;
buttons[ii].yHalf       = buttons[ii].buttonHeight/2;
++num;
/* Buttons to control potentiometers */
/* These buttons appear above the potentiometer windows which they affect. */
/* Rotate potentiometer buttons */
/* Rotate about the origin indicated by the axes */
/* Red is off, Green is on */
```

```
ii = originr;
buttons[ii].buttonX          = XEDGE;   buttons[ii].buttonY         = 55;
buttons[ii].buttonWidth      = 53;      buttons[ii].buttonHeight    = 25;
buttons[ii].buttonKey        = ii;
buttons[ii].pot              = no;
buttons[ii].mask             = buttonMASK;
buttons[ii].text             = "origin";
buttons[ii].textColor        = onColor;
buttons[ii].xHalf            = buttons[ii].buttonWidth/2;
buttons[ii].yHalf            = buttons[ii].buttonHeight/2;
++num;
/* Rotate about the objects center of volume */
/* Red is off, Green is on */
ii = objectr;
buttons[ii].buttonX          = 62;   buttons[ii].buttonY         = 55;
buttons[ii].buttonWidth      = 53;     buttons[ii].buttonHeight    = 25;
buttons[ii].buttonKey        = ii;
buttons[ii].pot              = no;
buttons[ii].mask             = buttonMASK;
buttons[ii].text             = "object";
buttons[ii].textColor        = offColor;
buttons[ii].xHalf            = buttons[ii].buttonWidth/2;
buttons[ii].yHalf            = buttons[ii].buttonHeight/2;
++num;
/* Scale potentiometer buttons */
/* Scale along X axis:  Red is off, Green is on */
ii = zoomx;
buttons[ii].buttonX          = 121;   buttons[ii].buttonY         = 55;
buttons[ii].buttonWidth      = 17;    buttons[ii].buttonHeight    = 25;
buttons[ii].buttonKey        = ii;
buttons[ii].pot              = no;
buttons[ii].mask             = buttonMASK;
buttons[ii].text             = "x";
buttons[ii].textColor        = onColor;
buttons[ii].xHalf            = buttons[ii].buttonWidth/2;
buttons[ii].yHalf            = buttons[ii].buttonHeight/2;
++num;
/* Scale along Y axis:  Red is off, Green is on */
ii = zoomy;
buttons[ii].buttonX          = 141;   buttons[ii].buttonY         = 55;
buttons[ii].buttonWidth      = 17;    buttons[ii].buttonHeight    = 25;
buttons[ii].buttonKey        = ii;
buttons[ii].pot              = no;
buttons[ii].mask             = buttonMASK;
buttons[ii].text             = "y";
buttons[ii].textColor        = onColor;
```

```
buttons[ii].xHalf            = buttons[ii].buttonWidth/2;
buttons[ii].yHalf            = buttons[ii].buttonHeight/2;
++num;
/* Zoom along Z axis:  Red is off, Green is on */
ii = zoomz;
buttons[ii].buttonX          = 161;  buttons[ii].buttonY          = 55;
buttons[ii].buttonWidth      = 17;   buttons[ii].buttonHeight     = 25;
buttons[ii].buttonKey        = ii;
buttons[ii].pot              = no;
buttons[ii].mask             = buttonMASK;
buttons[ii].text             = "z";
buttons[ii].textColor        = onColor;
buttons[ii].xHalf            = buttons[ii].buttonWidth/2;
buttons[ii].yHalf            = buttons[ii].buttonHeight/2;
++num;
/* Translate potentiometer buttons */
/* Indicates an orthographic projection of the xy-plane,
   translation is in x and y coordinates */
ii = xy;
buttons[ii].buttonX   = 185;  buttons[ii].buttonY      = 55;
buttons[ii].buttonWidth= 34;   buttons[ii].buttonHeight = 25;
buttons[ii].buttonKey  = ii;
buttons[ii].pot        = no;
buttons[ii].mask       = buttonMASK;
buttons[ii].text       = "xy";
buttons[ii].textColor  = 35;
buttons[ii].xHalf      = buttons[ii].buttonWidth/2;
buttons[ii].yHalf      = buttons[ii].buttonHeight/2;
++num;
/* Indicates an orthographic projection of the xz-plane,
   translation is in x and z coordinates */
ii = xz;
buttons[ii].buttonX    = 223;  buttons[ii].buttonY      = 55;
buttons[ii].buttonWidth= 34;   buttons[ii].buttonHeight = 25;
buttons[ii].buttonKey  = ii;
buttons[ii].pot        = no;
buttons[ii].mask       = buttonMASK;
buttons[ii].text       = "xz";
buttons[ii].textColor  = 35;
buttons[ii].xHalf      = buttons[ii].buttonWidth/2;
buttons[ii].yHalf      = buttons[ii].buttonHeight/2;
++num;
/* Indicates an orthographic projection of the yz-plane,
   translation is in y and z coordinates */
ii = yz;
buttons[ii].buttonX    = 261;  buttons[ii].buttonY      = 55;
```

```
        buttons[ii].buttonWidth= 34;    buttons[ii].buttonHeight = 25;
        buttons[ii].buttonKey  = ii;
        buttons[ii].pot        = no;
        buttons[ii].mask       = buttonMASK;
        buttons[ii].text       = "yz";
        buttons[ii].textColor  = 35;
        buttons[ii].xHalf      = buttons[ii].buttonWidth/2;
        buttons[ii].yHalf      = buttons[ii].buttonHeight/2;
        ++num;
        return(num);
    } /* initButtons() */
```

7.3.2   closeViewport

This closes all of the windows created for the control panel window and the viewport window of the current graph being displayed. It does not currently return a specified value.

⟨*view3d*⟩+≡

```
  void closeViewport(void) {
    int i;
    /* First, unlink viewport from global list of viewports */
    if (viewport->prevViewport) {    /* if there is a viewport before it */
      (viewport->prevViewport)->nextViewport = viewport->nextViewport;
    } else {                         /* this is the first viewport */
      viewport = viewport->nextViewport;
    }
    if (viewport->nextViewport) {    /* if there is a viewport following it */
      (viewport->nextViewport)->prevViewport = viewport->prevViewport;
    }
    /* Free up the control panel button windows */
    for (i=0; i<maxButtons3D; i++) {
      XDeleteAssoc(dsply,table,(control->buttonQueue[i]).self);
    }
    /* Free up the control panel window */
    XDestroyWindow(dsply,control->controlWindow);
    free(control);
    /* Free up the viewport window */
    XDestroyWindow(dsply,viewport->viewWindow);
    XDestroyWindow(dsply,viewport->titleWindow);
    free(viewport);
    XFlush(dsply);
  } /* closeViewport() */
```

7.3.3 scaleComponents

⟨*view3d*⟩+≡

```
void scaleComponents(void) {
  double xRange,yRange,zRange;
  int i;
  viewTriple *aPoint;
  /* Temporary range limits until the three dimensional clipping
     package is fully functional */
  if (viewData.xmin < MIN_POINT) viewData.xmin = MIN_POINT;
  if (viewData.xmax > MAX_POINT) viewData.xmax = MAX_POINT;
  if (viewData.ymin < MIN_POINT) viewData.ymin = MIN_POINT;
  if (viewData.ymax > MAX_POINT) viewData.ymax = MAX_POINT;
  if (viewData.zmin < MIN_POINT) viewData.zmin = MIN_POINT;
  if (viewData.zmax > MAX_POINT) viewData.zmax = MAX_POINT;
  xRange = viewData.xmax - viewData.xmin;
  yRange = viewData.ymax - viewData.ymin;
  zRange = viewData.zmax - viewData.zmin;
  /* We scale down, normalize the data, if it is coming from Axiom
     (handled by viewman).  If the data is coming from a file (handled by
     viewalone) then it should already been scaled down.
     */
  /* Find the coordinate axis with the larges range of data and scale
     the others relative to it.
     */
  /* compare x and y ranges */
  if (xRange > yRange) {
    if (xRange > zRange) {
      if (absolute(viewData.xmax) >= absolute(viewData.xmin))
        viewData.scaleToView = axisLength/(absolute(viewData.xmax));
      else
        viewData.scaleToView = axisLength/(absolute(viewData.xmin));
    } else {
      if (absolute(viewData.zmax) >= absolute(viewData.zmin))
        viewData.scaleToView = axisLength/(absolute(viewData.zmax));
      else
        viewData.scaleToView = axisLength/(absolute(viewData.zmin));
    }
  } else {
    if (yRange > zRange) {
      if (absolute(viewData.ymax) >= absolute(viewData.ymin))
        viewData.scaleToView = axisLength/(absolute(viewData.ymax));
      else
        viewData.scaleToView = axisLength/(absolute(viewData.ymin));
    } else {
      if (absolute(viewData.zmax) >= absolute(viewData.zmin))
```

```
          viewData.scaleToView = axisLength/(absolute(viewData.zmax));
        else
          viewData.scaleToView = axisLength/(absolute(viewData.zmin));
    }
  }
  /* We now normalize all the points in this program. The information
     needed to link the normalized set of points back to the real object
     space scale created in Axiom is held in viewData.scaleToView. */
  viewData.xmin *= viewData.scaleToView;
  viewData.xmax *= viewData.scaleToView;
  viewData.ymin *= viewData.scaleToView;
  viewData.ymax *= viewData.scaleToView;
  viewData.zmin *= viewData.scaleToView;
  viewData.zmax *= viewData.scaleToView;
  viewData.clipXmin = viewData.xmin;
  viewData.clipXmax = viewData.xmax;
  viewData.clipYmin = viewData.ymin;
  viewData.clipYmax = viewData.ymax;
  viewData.clipZmin = viewData.zmin;
  viewData.clipZmax = viewData.zmax;
  for (i=0, aPoint=viewData.points; i<viewData.numOfPoints; i++,aPoint++) {
    aPoint->x *= viewData.scaleToView;
    aPoint->y *= viewData.scaleToView;
    aPoint->z *= viewData.scaleToView;
  }
} /* scaleComponents() */
```

7.3.4 makeTriangle

Given three indices to three points, a triangular polygon is created and inserted into the
polygon list of viewData. If two or more of the points are coincidental, no polygon is created
since that would be a degenerate (collapsed) polygon.

⟨*view3d*⟩+≡

```
void makeTriangle(int a, int b, int c) {
  poly *aPoly;
  if (!(samePoint(a,b) || samePoint(b,c) || samePoint(c,a))) {
    /* create triangle only if the three vertex points are distinct */
    aPoly = (poly *)saymem("component.c",1,sizeof(poly));
    aPoly->num = aPoly->sortNum = viewData.numPolygons++;
    aPoly->split = aPoly->moved = no;
    aPoly->numpts = 3;
    aPoly->primitiveType = polygonComponent;
    aPoly->indexPtr = (int *)saymem("component.c",3,sizeof(int));
    *(aPoly->indexPtr)     = a;
    *(aPoly->indexPtr + 1) = b;
    *(aPoly->indexPtr + 2) = c;
    aPoly->doNotStopDraw = yes;
    aPoly->next = viewData.polygons;
    viewData.polygons = aPoly;
  } /* if all points are unique */
} /* makeTriangle() */
```

7.3.5   triangulate

Only if there is more than one list do we triangulate; a single list is used for either a space curve or simply a point. Actually, in that case, we now make "flat" *polygons, flagged by the primitiveType field (pointComponent, etc. in tube.h). We need to examine two lists at a time (and if the structure is closed, the last and first as well). For every three points in the two lists, alternating between one in one and two in the other, we construct triangles. If one list is shorter, then its last point becomes the vertex for the remaining pairs of points from the other list. It turns out that any distribution of points in the two lists (preserving cyclic order) will produce the same desired polygon.

⟨*view3d*⟩+≡

```
void triangulate(void) {
  int u,l;
  int uBound,lBound;
  int i,j,k;
  LLPoint *anLLPoint;
  LPoint *list1,*list2;
  poly *aPoly;
  anLLPoint = viewData.lllp.llp;
  for (i=0; i<viewData.lllp.numOfComponents; i++,anLLPoint++) {
    if (anLLPoint->numOfLists > 1) {
      list2 = anLLPoint->lp;
      for (j=1; j<anLLPoint->numOfLists; j++) {
        list1 = list2;
        list2 = list1 + 1;
        u = l = 0;
        uBound = u+1 < list1->numOfPoints;
        lBound = l+1 < list2->numOfPoints;
        while (uBound || lBound) {
          if (uBound) {
            makeTriangle(*(list1->indices + u + 1),
                         *(list1->indices + u), *(list2->indices + l));
            u++;
            uBound = u+1 < list1->numOfPoints;
          }
          if (lBound) {
            makeTriangle(*(list2->indices + l),
                         *(list2->indices + l + 1), *(list1->indices + u));
            l++;
            lBound = l+1 < list2->numOfPoints;
          }
        } /* while (uBound || lBound) */
      } /* for j<anLLPoint->numOfLists */
    } /* if anLLPoint->numOfLists > 1 */
    else {
```

```
        /* if anLLPoint->numOfLists <= 1...assume this means =1 */
        /* Flat polygons are to be drawn when hidden
           surface algorithm is used.*/
        if (anLLPoint->numOfLists == 1) {
          if (anLLPoint->lp->numOfPoints == 1) {
            /* this graph is a single point */
            aPoly = (poly *)saymem("component.c",1,sizeof(poly));
            aPoly->num = aPoly->sortNum = viewData.numPolygons++;
            aPoly->split = aPoly->moved = no;
            aPoly->primitiveType = pointComponent;
            aPoly->numpts = 1;
            aPoly->indexPtr = (int *)saymem("component.c",1,intSize);
            *(aPoly->indexPtr) = *(anLLPoint->lp->indices);
            aPoly->doNotStopDraw = yes;
            aPoly->next = viewData.polygons;
            viewData.polygons = aPoly;
          } else {
            /* this graph is a curve */
            for (k=0; k<anLLPoint->lp->numOfPoints-1; k++) {
              aPoly = (poly *)saymem("component.c",1,sizeof(poly));
              aPoly->num = aPoly->sortNum = viewData.numPolygons++;
              aPoly->split = aPoly->moved = no;
              aPoly->primitiveType = lineComponent; /* curveComponent */
              aPoly->numpts = 2;
              aPoly->indexPtr =
                (int *)saymem("component.c",2,sizeof(int));
              *(aPoly->indexPtr)   = *(anLLPoint->lp->indices + k);
              *(aPoly->indexPtr+1) = *(anLLPoint->lp->indices + k + 1);
              aPoly->doNotStopDraw = yes;
              aPoly->next = viewData.polygons;
              viewData.polygons = aPoly;
            }  /* for k */
            if (anLLPoint->lp->prop.closed) {
              aPoly = (poly *)saymem("component.c",1,sizeof(poly));
              aPoly->num = aPoly->sortNum = viewData.numPolygons++;
              aPoly->split = aPoly->moved = no;
              aPoly->primitiveType = lineComponent; /* curveComponent */
              aPoly->numpts = 2;
              aPoly->indexPtr =
                (int *)saymem("component.c",2,sizeof(int));
              *(aPoly->indexPtr)   = *(anLLPoint->lp->indices + k);
              *(aPoly->indexPtr+1) = *(anLLPoint->lp->indices);
              aPoly->doNotStopDraw = yes;
              aPoly->next = viewData.polygons;
              viewData.polygons = aPoly;
            } /* if list of points is closed */
```

```
        }  /* else */
      } /* point, line, polygon, surface components are taken care of above */
     } /* else anLLPoint->numOfLists <= 1 */
   } /* for LLPoints in LLLPoints (i) */
 } /* triangulate */
```

7.3.6   readComponentsFromViewman

⟨*view3d*⟩+≡

```c
void readComponentsFromViewman(void) {
  int i,j,k;
  LLPoint *anLLPoint;
  LPoint *anLPoint;
  viewTriple *aPoint;
  /* maxLength holds the max(llp,lp) figure regarding how large to
     make the array of XPoints, i.e. quadMesh, for use in calling XDraw(). */
  int maxLength=0;
  int *anIndex;
  readViewman(&(viewData.numOfPoints),intSize);
  aPoint = viewData.points =
    (viewTriple *)saymem("component.c",viewData.numOfPoints,
                         sizeof(viewTriple));
  for (i=0; i<viewData.numOfPoints; i++, aPoint++) {
    readViewman(&(aPoint->x),floatSize);
    readViewman(&(aPoint->y),floatSize);
    readViewman(&(aPoint->z),floatSize);
    readViewman(&(aPoint->c),floatSize);
#ifdef NANQ_DEBUG
    if (!(aPoint->z < 0) && !(aPoint->z > 0) && !(aPoint->z == 0))
      fprintf(stderr,"%g\n", aPoint->z);
#endif
  }
  readViewman(&(viewData.lllp.numOfComponents),intSize);
  anLLPoint = viewData.lllp.llp =
    (LLPoint *)saymem("component.c, i",viewData.lllp.numOfComponents,
                      sizeof(LLPoint));
  for (i=0; i<viewData.lllp.numOfComponents; i++,anLLPoint++) {
    readViewman(&(anLLPoint->prop.closed),intSize);
    readViewman(&(anLLPoint->prop.solid),intSize);
    readViewman(&(anLLPoint->numOfLists),intSize);
    anLPoint = anLLPoint->lp =
      (LPoint *)saymem("component.c, ii",anLLPoint->numOfLists,
                       sizeof(LPoint));
    for (j=0; j<anLLPoint->numOfLists; j++,anLPoint++) {
      if (anLLPoint->numOfLists > maxLength)
        maxLength = anLLPoint->numOfLists;
      readViewman(&(anLPoint->prop.closed),intSize);
      readViewman(&(anLPoint->prop.solid),intSize);
      readViewman(&(anLPoint->numOfPoints),intSize);
      anIndex = anLPoint->indices =
        (int *)saymem("component.c, index",anLPoint->numOfPoints,intSize);
      if (anLPoint->numOfPoints > maxLength)
```

```
        maxLength = anLPoint->numOfPoints;
      for (k=0; k<anLPoint->numOfPoints; k++,anIndex++) {
        readViewman(anIndex,intSize);
        /* Axiom arrays are one based, C arrays are zero based */
        if (!viewAloned) (*anIndex)--;
      }
    } /* for LPoints in LLPoints (j) */
  } /* for LLPoints in LLLPoints (i) */
  quadMesh = (XPoint *)saymem("component.c",maxLength+2,sizeof(XPoint));
} /* readComponentsFromViewman() */
```

7.3.7 calcNormData

Calculates the surface normals for the polygons that make up the tube. Also finds the fourth coefficient to the plane equation:

$$Ax + By + Cz + D = 0$$

A, B, and C are in the normal $N[3]$ and D is the planeConst. Figures out the color as well (from the average of the points) and resets the moved flag

$\langle view3d \rangle +\equiv$

```
void calcNormData(void) {
  poly *aPoly;
  int *index;
  for (aPoly = viewData.polygons; aPoly != NIL(poly); aPoly = aPoly->next) {
    index = aPoly->indexPtr;
    switch (aPoly->primitiveType) {
    case pointComponent:
    case lineComponent:
      aPoly->moved = 0;
      aPoly->color = refPt3D(viewData,*index)->c;
      break;
    default:
      /*
         The following line takes 3 consecutive points and asks
         for the normal vector defined by them. This assumes that
         these do not contain co-linear points. For some reason,
         co-linear points are allowed, this needs to be changed.
         */
      getMeshNormal(refPt3D(viewData,*index)->x,
                    refPt3D(viewData,*index)->y,
                    refPt3D(viewData,*index)->z,
                    refPt3D(viewData,*(index+1))->x,
                    refPt3D(viewData,*(index+1))->y,
                    refPt3D(viewData,*(index+1))->z,
                    refPt3D(viewData,*(index+2))->x,
                    refPt3D(viewData,*(index+2))->y,
                    refPt3D(viewData,*(index+2))->z, 0.0, 1.0, aPoly->N);
      /* calculate the constant term, D,  for the plane equation */
      aPoly->planeConst =
        -(aPoly->N[0] * refPt3D(viewData,*index)->x +
          aPoly->N[1] * refPt3D(viewData,*index)->y +
          aPoly->N[2] * refPt3D(viewData,*index)->z);
      aPoly->moved = 0;
      aPoly->color = (refPt3D(viewData,*index)->c +
                      (refPt3D(viewData,*(index+1)))->c +
                      (refPt3D(viewData,*(index+2)))->c) / 3.0;
```

```
        break;
      } /* switch */
    }
  }  /* calcNormData() */
```

7.3.8 make3DComponents

Read in all the 3D data from the viewport manager and construct the model of it. The model is based upon a list of lists of lists of points. Each top level list makes a component in 3-space. The interpretation really begins at the level below that, where the list of lists of points is. For 3D explicit equations of two variables, the closed boolean for this level is False and the closed boolean for each sublist is False as well. For 3D parameterized curves of one variable, the closed boolean for this level is defined by the user from Axiom , (which defaults to False) and the closed boolean for each sublist is True.

⟨*view3d*⟩+≡

```
viewPoints *make3DComponents(void) {
  viewPoints *graphData;
  readComponentsFromViewman();
  /* The initial boundaries for the clipping region are set to those
     of the boundaries of the data region. */
  viewData.clipXmin = viewData.xmin;  viewData.clipXmax = viewData.xmax;
  viewData.clipYmin = viewData.ymin;  viewData.clipYmax = viewData.ymax;
  viewData.clipZmin = viewData.zmin;  viewData.clipZmax = viewData.zmax;
  /* normalize the data coordinates */
  if (viewData.scaleDown) scaleComponents();
  viewData.numPolygons = 0;
  /* initially the list of polygons is empty */
  viewData.polygons = NIL(poly);
  /* create the polygons; (sets viewData.polygons and viewData.numPolygons) */
  triangulate();
  /* calculate the plane equations for all the polygons */
  calcNormData();
  graphData = makeViewport();
  imageX = XCreateImage(/* display */       dsply,
                        /* visual */        DefaultVisual(dsply,scrn),
                        /* depth */         DefaultDepth(dsply,scrn),
                        /* format */        ZPixmap,
                        /* offset */        0,
                        /* data */          NULL,
                        /* width */         vwInfo.width,
                        /* height */        1,
                        /* bitmap_pad */    32,
                        /* bytes_per_line */ 0);
  imageX->data = NIL(char);
  /* windowing displaying */
  writeTitle();
  postMakeViewport();
  drawViewport(Xoption);
  firstTime = yes;
  XMapWindow(dsply, graphData->viewWindow);
```

```
    XMapWindow(dsply, graphData->titleWindow);
    XFlush(dsply);
    return(graphData);
  } /* make3DComponents */
```

7.3.9   draw3DComponents

⟨*view3d*⟩+≡

```
  void draw3DComponents(int dFlag) {
    int        i, j, k, hue, x1, y1, x2, y2;
    LLPoint    *anLLPoint;
    LPoint     *anLPoint;
    int        *anIndex;
    int        componentType;  /* what the component is to be interpreted as */
    int        clip_a,clip_i;  /* for use in wire mesh mode clipping */
    XEvent     peekEvent;
    viewTriple *aLPt;
    XPoint     line[2];
    RGB        col_rgb;
    calcEyePoint();
    while ((XPending(dsply) > 0) && (scanline > 0))
      XNextEvent(dsply,&peekEvent);
    switch (viewData.style) {
    case transparent:
      GSetLineAttributes(componentGC,0,LineSolid,CapButt,JoinMiter,dFlag);
      if (dFlag==Xoption) {
        if (mono || viewport->monoOn)
          GSetForeground(componentGC, (float)foregroundColor, dFlag);
        else
          GSetForeground(componentGC, (float) meshOutline, dFlag);
      } else {
        GSetForeground(componentGC, psBlack, dFlag);
      }
      /* no need to check "keep drawing" for ps */
      if (dFlag == Xoption) drawMore = keepDrawingViewport();
```

This is where we interpret the list of lists of lists of points struct. We want to extract the following forms of data:

- individual points (drawn as filled points)

- lines (space curves)

- defined polygon primitives

- surfaces

the last one is the one that will replace the function of 2 variables, tubes as well as 3D parameterized functions of 2 variables. Since there could be many other ways of constructing `List List List Points` - much more than could be usefully interpreted - any other formats are currently not allowed. When they are, this comment should be updated appropriately.

Traverse each component. We decide here, before we begin traversing the component what we want to interpret it as. Here's the convention used to figure that out:

- points: #anLLPoint→numOfLists was 1
  #anLPoint→numOfPoints is 1

- lines: #anLLPoint→ numOfLists was 1
  #anLPoint→numOfPoints ¿ 1

- polygons: #anLLPoint→numOfLists was 2
  #anLPoint→numOfPoints is 1

- surface: #anLLPoint→numOfLists was some $m > 1$
  #anLPoint→numOfPoints all point lists are the same.

⟨*view3d*⟩+≡

```
anLLPoint = viewData.lllp.llp;
for (i=0; i<viewData.lllp.numOfComponents; i++,anLLPoint++) {
  /* initially, component type is unknown */
  componentType = stillDontKnow;
  if (anLLPoint->numOfLists == 1) {
    if (anLLPoint->lp->numOfPoints == 1) componentType = pointComponent;
    else componentType = lineComponent;
  } else if (anLLPoint->numOfLists == 2) {
    if ((anLLPoint->lp->numOfPoints == 1) &&
        ((anLLPoint->lp+1)->numOfPoints > 2))
      componentType = polygonComponent;
  }
  /* Check for corrupt data and NaN data is made in Axiom . */
  if (componentType == stillDontKnow)
    componentType = surfaceComponent;
  anLPoint = anLLPoint->lp;
  switch (componentType) {
```

```
case pointComponent:
  /* anLLPoint->numOfLists == anLLPoint->lp->numOfPoints == 1 here */
  aLPt = refPt3D(viewData,*(anLPoint->indices));
  project(aLPt,quadMesh,0);
  if (dFlag==Xoption) {
    if (mono || viewport->monoOn)
      GSetForeground(componentGC, (float)foregroundColor, dFlag);
    else {
      hue = hueValue(aLPt->c);
      GSetForeground(componentGC, (float)XSolidColor(hue,2), dFlag);
    }
  } else GSetForeground(componentGC, psBlack, dFlag);
  GFillArc(componentGC,viewport->viewWindow,quadMesh->x,quadMesh->y,
           viewData.pointSize,viewData.pointSize,0,360*64,dFlag);
  break;
case lineComponent:
  /* anLLPoint->numOfLists == 1 here */
  anIndex = anLPoint->indices;
  aLPt = refPt3D(viewData,*anIndex);
  project(aLPt,quadMesh,0);
  x1 = quadMesh[0].x;  y1 = quadMesh[0].y;  anIndex++;
  for (k=1; k<anLPoint->numOfPoints; k++,anIndex++) {
    aLPt = refPt3D(viewData,*anIndex);
    project(aLPt,quadMesh,k);
    x2 = quadMesh[k].x; y2 = quadMesh[k].y;
    if (dFlag==Xoption) {
      if (mono || viewport->monoOn)
        GSetForeground(componentGC, (float)foregroundColor, dFlag);
      else {
        hue = hueValue(aLPt->c);
        GSetForeground(componentGC, (float)XSolidColor(hue,2), dFlag);
      }
      if (!eqNANQ(x1) && !eqNANQ(y1) && !eqNANQ(x2) && !eqNANQ(y2))
        GDrawLine(componentGC,viewport->viewWindow,x1,y1,x2,y2,dFlag);
    } else {
      if (dFlag==PSoption && !mono && !viewport->monoOn) {
        hue = getHue(aLPt->c);
        col_rgb = hlsTOrgb((float)hue,0.5,0.8);
        line[0].x = x1;  line[0].y = y1;
        line[1].x = x2;  line[1].y = y2;
        PSDrawColor(col_rgb.r,col_rgb.g,col_rgb.b,line,2);
      } else {
        if (foregroundColor == white)
          GSetForeground(componentGC, 0.0, dFlag);
        else
          GSetForeground(componentGC, psBlack, dFlag);
```

```
          if (!eqNANQ(x1) && !eqNANQ(y1) && !eqNANQ(x2) && !eqNANQ(y2))
            GDrawLine(componentGC,viewport->viewWindow,x1,y1,x2,y2,dFlag);
        }
      }
      x1 = x2; y1 = y2;
    } /* for points in LPoints (k) */
    if (anLPoint->prop.closed) {
      project(refPt3D(viewData,*(anLPoint->indices)),quadMesh,
              anLPoint->numOfPoints);
      x2 = quadMesh[anLPoint->numOfPoints].x;
      y2 = quadMesh[anLPoint->numOfPoints].y;
      if (dFlag==Xoption) {
        if (mono || viewport->monoOn)
          GSetForeground(componentGC, (float)foregroudColor, dFlag);
        else {
          hue = hueValue(aLPt->c);
          GSetForeground(componentGC, (float)XSolidColor(hue,2), dFlag);
        }
        if (!eqNANQ(x1) && !eqNANQ(y1) && !eqNANQ(x2) && !eqNANQ(y2))
          GDrawLine(componentGC,viewport->viewWindow,x1,y1,x2,y2,dFlag);
      }
      else {
        if (dFlag==PSoption && !mono && !viewport->monoOn) {
          hue = getHue(aLPt->c);
          col_rgb = hlsTOrgb((float)hue,0.5,0.8);
          line[0].x = x1;  line[0].y = y1;
          line[1].x = x2;  line[1].y = y2;
          PSDrawColor(col_rgb.r,col_rgb.g,col_rgb.b,line,2);
        }
        else {
          if (foregroundColor == white)
            GSetForeground(componentGC, 0.0, dFlag);
          else
            GSetForeground(componentGC, psBlack, dFlag);
          if (!eqNANQ(x1) && !eqNANQ(y1) && !eqNANQ(x2) && !eqNANQ(y2))
            GDrawLine(componentGC,viewport->viewWindow,x1,y1,x2,y2,dFlag);
        }
      }
    }
    break;
  case polygonComponent:
    /* first pt of polygon is a single list */
    project(refPt3D(viewData,*(anLPoint->indices)),quadMesh,0);
    /* remaining points in the 2nd list (always of size 2 or greater) */
    x1 = quadMesh[0].x;  y1 = quadMesh[0].y;
    anLPoint = anLLPoint->lp + 1;
```

```
                anIndex = anLPoint->indices;
                for (k=1; k<=anLPoint->numOfPoints; k++,anIndex++) {
                  aLPt = refPt3D(viewData,*anIndex);
                  project(aLPt,quadMesh,k);
                  x2 = quadMesh[k].x;  y2 = quadMesh[k].y;
                  if (dFlag==Xoption) {
                    if (mono || viewport->monoOn)
                      GSetForeground(componentGC, (float)foregroundColor, dFlag);
                    else {
                      hue = hueValue(aLPt->c);
                      GSetForeground(componentGC, (float)XSolidColor(hue,2), dFlag);
                    }
                    if (!eqNANQ(x1) && !eqNANQ(y1) && !eqNANQ(x2) && !eqNANQ(y2))
                      GDrawLine(componentGC,viewport->viewWindow,x1,y1,x2,y2,dFlag);
                  }
                  else {
                    if (dFlag==PSoption && !mono && !viewport->monoOn) {
                      hue = getHue(aLPt->c);
                      col_rgb = hlsTOrgb((float)hue,0.5,0.8);
                      line[0].x = x1;  line[0].y = y1;
                      line[1].x = x2;  line[1].y = y2;
                      PSDrawColor(col_rgb.r,col_rgb.g,col_rgb.b,line,2);
                    }
                    else {
                      if (foregroundColor == white)
                        GSetForeground(componentGC, 0.0, dFlag);
                      else
                        GSetForeground(componentGC, psBlack, dFlag);
                      if (!eqNANQ(x1) && !eqNANQ(y1) && !eqNANQ(x2) && !eqNANQ(y2))
                        GDrawLine(componentGC,viewport->viewWindow,x1,y1,x2,y2,dFlag);
                    }
                  }
                  x1 = x2; y1 = y2;
                } /* for points in LPoints (k) */
                project(refPt3D(viewData,*(anLLPoint->lp->indices)),quadMesh,k);
                x2 = quadMesh[k].x;  y2 = quadMesh[k].y;
                if (dFlag==Xoption) {
                  if (mono || viewport->monoOn)
                    GSetForeground(componentGC, (float)foregroundColor, dFlag);
                  else {
                    hue = hueValue(refPt3D(viewData,*anIndex)->c);
                    GSetForeground(componentGC, (float)XSolidColor(hue,2), dFlag);
                  }
                  if (!eqNANQ(x1) && !eqNANQ(y1) && !eqNANQ(x2) && !eqNANQ(y2))
                    GDrawLine(componentGC,viewport->viewWindow,x1,y1,x2,y2,dFlag);
                } else {
```

```
          if (dFlag==PSoption && !mono && !viewport->monoOn) {
            hue = getHue(refPt3D(viewData,*anIndex)->c);
            col_rgb = hlsTOrgb((float)hue,0.5,0.8);
            line[0].x = x1;  line[0].y = y1;
            line[1].x = x2;  line[1].y = y2;
            PSDrawColor(col_rgb.r,col_rgb.g,col_rgb.b,line,2);
          }
          else {
            if (foregroundColor == white)
              GSetForeground(componentGC, 0.0, dFlag);
            else
              GSetForeground(componentGC, psBlack, dFlag);
            if (!eqNANQ(x1) && !eqNANQ(y1) && !eqNANQ(x2) && !eqNANQ(y2))
              GDrawLine(componentGC,viewport->viewWindow,x1,y1,x2,y2,dFlag);
          }
        }
        /* close a polygon */
        break;
      case surfaceComponent:
        if (dFlag==Xoption) {
          if (mono || viewport->monoOn)
            GSetForeground(componentGC, (float)foregroundColor, dFlag);
          else
            GSetForeground(componentGC, (float) meshOutline, dFlag);
        }
        else {
          GSetForeground(componentGC, psBlack, dFlag);
        }
        /* traverse down one direction first (all points
           in a list at a time) */
        for (j=0; drawMore && j<anLLPoint->numOfLists; j++,anLPoint++) {
          anIndex = anLPoint->indices;
          clip_a = 0;
          for (k=0, clip_i=0;
               drawMore && k<anLPoint->numOfPoints;
               k++, anIndex++, clip_i++) {
            aLPt = refPt3D(viewData,*anIndex);
            project(aLPt,quadMesh,k);

            if (behindClipPlane(aLPt->pz) ||
                (viewData.clipStuff &&
                 outsideClippedBoundary(aLPt->x, aLPt->y, aLPt->z))) {
              if (clip_i - clip_a > 1) {
                GDrawLines(componentGC,viewport->viewWindow,(quadMesh+clip_a),
                          clip_i-clip_a, CoordModeOrigin, dFlag );
              }
```

```
            clip_a = clip_i + 1;
          }
          drawMore = keepDrawingViewport();
        } /* for points in LPoints (k) */
        if (drawMore) {
          /* if drawMore is true, then the above loop terminated with
             clip_i incremented properly */
          if (anLPoint->prop.closed) {
            /* If closed, then do the first point again - no need to project
               just copy over from the first one */
            aLPt = refPt3D(viewData,*(anLPoint->indices));
            project(aLPt,quadMesh, anLPoint->numOfPoints);
            if (behindClipPlane(aLPt->pz) ||
                (viewData.clipStuff &&
                 outsideClippedBoundary(aLPt->x, aLPt->y, aLPt->z))) {
              if (clip_i - clip_a > 1) {
                GDrawLines(componentGC, viewport->viewWindow,
                           (quadMesh+clip_a), clip_i-clip_a,
                           CoordModeOrigin, dFlag);
              }
              clip_a = clip_i + 1;
            }
            clip_i++;
          } /* closed */
          if (clip_i - clip_a > 1) {
            GDrawLines(componentGC, viewport->viewWindow, (quadMesh+clip_a),
                       clip_i-clip_a, CoordModeOrigin, dFlag);
          }
        } /* drawMore */
      } /* for LPoints in LLPoints (j) */
      /* now traverse down the list in the other direction
         (one point from each list at a time) */
      for (j=0; drawMore && j<anLLPoint->lp->numOfPoints; j++) {
        clip_a = 0;
        for (k=0, clip_i=0;
             drawMore && k<anLLPoint->numOfLists;
             k++, clip_i++) {
          aLPt = refPt3D(viewData,*((anLLPoint->lp + k)->indices + j));
          project(aLPt, quadMesh,k);

          if (behindClipPlane(aLPt->pz) ||
              (viewData.clipStuff &&
               outsideClippedBoundary(aLPt->x, aLPt->y, aLPt->z))) {
            if (clip_i - clip_a > 1) {
              GDrawLines(componentGC,viewport->viewWindow,quadMesh+clip_a,
                         clip_i-clip_a, CoordModeOrigin, dFlag );
```

```
                }
                clip_a = clip_i + 1;
              }
              drawMore = keepDrawingViewport();
          } /* for points in LPoints (k) */
          if (drawMore) {
            /* if drawMore is true, then the above loop terminated with
               clip_i incremented properly */
            if (anLLPoint->prop.closed) {
              /* if closed, do the first point again - no need to project
                 just copy over from the first one */
              aLPt = refPt3D(viewData,*((anLLPoint->lp + 0)->indices + j));
              project(aLPt, quadMesh, anLLPoint->numOfLists);
              if (behindClipPlane(aLPt->pz) ||
                  (viewData.clipStuff &&
                   outsideClippedBoundary(aLPt->x, aLPt->y, aLPt->z))) {
                if (clip_i - clip_a > 1) {
                  GDrawLines(componentGC, viewport->viewWindow,
                             quadMesh + clip_a, clip_i - clip_a,
                             CoordModeOrigin, dFlag);
                }
                clip_a = clip_i + 1;
              }
              clip_i++;
            } /* closed */
            if (clip_i - clip_a > 1) {
              GDrawLines(componentGC, viewport->viewWindow, quadMesh+clip_a,
                         clip_i-clip_a, CoordModeOrigin, dFlag);
            }
          } /* drawMore */
        } /* for a point in each LPoint (j) */
        break;
      } /* switch componentType */
    } /* for LLPoints in LLLPoints (i) */
    break;
  case opaqueMesh:
    if (dFlag==Xoption) {
      GSetForeground(globGC, (float)opaqueForeground, dFlag);
      GSetForeground(opaqueGC, (float)opaqueOutline, dFlag);
    }
    else {
      GSetForeground(globGC, psBlack, dFlag);
      GSetForeground(opaqueGC, psBlack, dFlag);
    }
    GSetLineAttributes(opaqueGC,0,LineSolid,CapButt,JoinRound,dFlag);
    drawPolygons(dFlag);
```

```
      break;
    case render:
      if (viewData.outlineRenderOn) {
        GSetLineAttributes(renderGC,0,LineSolid,CapButt,JoinRound,dFlag);
        if (dFlag==Xoption) GSetForeground(renderGC,(float)black, dFlag);
        else GSetForeground(renderGC,psBlack, dFlag );
      }
      drawPolygons(dFlag);
      break;
    case smooth:
      drawPhong(dFlag);
      break;
    } /* switch on style */
  } /* draw3DComponents() */
```

7.3.10 drawColorMap

⟨*view3d*⟩+≡

```
void drawColorMap (void) {
  controlPanelStruct *cp;
  int i,shadeWidth;
  /* Draw the color map window */
  cp = viewport->controlPanel;
  XClearArea(dsply,cp->controlWindow,5,colormapY,colormapW,colormapH,False);
  /* if window is grayscale, show the grayscale colormap */
  if (mono || (viewport->monoOn)) {
    shadeWidth = 230/maxGreyShade;
    for (i=0; i<maxGreyShade; i++) {
      XChangeShade(dsply, i);
      XShadeRectangle(dsply,cp->controlWindow,
                      colormapX + colorOffsetX + i*shadeWidth,
                      colormapY + colorOffsetY - 10, shadeWidth, 40);
    }
  } else {
    GDrawString(globalGC2,cp->controlWindow,colorWidth,
                colormapY + 13,"-",1,Xoption);
    GDrawString(globalGC2,cp->controlWindow,30*colorWidth + 40,
                colormapY + 13,"+",1,Xoption);
    GDrawString(globalGC2,cp->controlWindow,colorWidth,
                colormapY + 46,"-",1,Xoption);
    GDrawString(globalGC2,cp->controlWindow,30*colorWidth + 40,
                colormapY + 46,"+",1,Xoption);
    for (i=0; i<totalHues; i++) {
      GSetForeground(anotherGC, (float)XSolidColor(i,2), Xoption);
      GDrawLine(anotherGC,cp->controlWindow,
                colormapX + i*colorWidth + colorOffsetX,
                colormapY + colorOffsetY,
                colormapX + i*colorWidth + colorOffsetX,
                colormapY + colorOffsetY + colorHeight,Xoption);
    }
    if (viewport->hueTop > totalHues-1) viewport->hueTop = totalHues-1;
    if (viewport->hueOffset > totalHues-1) viewport->hueOffset = totalHues-1;
    GSetForeground(globGC, (float)monoColor(7), Xoption);
    /* Bottom (zmin) color indicator */
    GDrawLine(globGC,cp->controlWindow,
              colormapX + viewport->hueOffset * colorWidth + colorOffsetX,
              colormapY + colorOffsetY+colorHeight,
              colormapX + viewport->hueOffset * colorWidth + colorOffsetX,
              colormapY + colorOffsetY+colorHeight+colorPointer,Xoption);
    /* Top (zmax) color indicator */
    GDrawLine(globGC,cp->controlWindow,
```

```
                colormapX + viewport->hueTop * colorWidth+colorOffsetX,
                colormapY + colorOffsetY,
                colormapX + viewport->hueTop * colorWidth+colorOffsetX,
                colormapY + colorOffsetY-colorPointer,Xoption);
      /* Connect the bottom and top color indicator bars */
      GSetForeground(globGC, (float)monoColor(0), Xoption);
      GDrawLine(globGC,cp->controlWindow,
                colormapX + viewport->hueOffset * colorWidth + colorOffsetX,
                colormapY + colorOffsetY+colorHeight,
                colormapX + viewport->hueTop * colorWidth+colorOffsetX,
                colormapY + colorOffsetY,Xoption);
    }
    XSync(dsply,0);
  } /* drawColorMap() */
```

7.3.11   writeControlTitle

We need the window argument here because there are multiple control panels in 3D.

⟨view3d⟩+≡
```
  void writeControlTitle(Window w) {
    int strlength;
    s = viewport->title;
    strlength = strlen(s);
    XClearArea(dsply,w,0,0,controlWidth,potA,False);
    GSetForeground(anotherGC,(float)controlTitleColor,Xoption);
    GDrawString(anotherGC,w,centerX(anotherGC,s,strlength,controlWidth),
                15,s,strlength,Xoption);
  } /* writeControlTitle() */
```

7.3.12 clearControlMessage

⟨*view3d*⟩+≡

```
void clearControlMessage(void) {
  int strlength;
  strcpy(viewport->controlPanel->message,"                         ");
  strlength = strlen(viewport->controlPanel->message);
  GDrawImageString(globalGC1,viewport->controlPanel->controlWindow,
                   centerX(globalGC1,viewport->controlPanel->message,
                           strlength,controlWidth),
                   controlMessageY + globalFont->max_bounds.ascent + 8,
                   viewport->controlPanel->message,strlength,Xoption);
}
```

7.3.13 writeControlMessage

⟨*view3d*⟩+≡

```
void writeControlMessage(void) {
  int strlength;
  controlPanelStruct *cp;
  cp = viewport->controlPanel;
  strlength = strlen(cp->message);
  XClearArea(dsply,cp->controlWindow,
             0,controlMessageY+ globalFont->max_bounds.ascent + 8,
             0,controlMessageHeight,False);
  GSetForeground(globalGC1, (float)controlMessageColor, Xoption);
  GDrawImageString(globalGC1,cp->controlWindow,
                   centerX(globalGC1,cp->message,strlength,controlWidth),
                   controlMessageY + globalFont->max_bounds.ascent + 8,
                   cp->message,strlength,Xoption);
  XFlush(dsply);
}
```

7.3.14   drawControlPanel

⟨*view3d*⟩+≡

```
void drawControlPanel(void) {
  int                 offShade=14;
  controlPanelStruct  *cp;
  int                 i, strlength;
  char                *s;
  cp = viewport->controlPanel;
  GSetForeground(trashGC, (float)foregroundColor, Xoption);
  /* Draw border lines to separate the potentiometer, message, colormap and
     button regions of the control panel. */
  GSetLineAttributes(trashGC, 2, LineSolid, CapButt, JoinMiter, Xoption);
  /* Draw a horizontal white line below the potentiometer area. */
  GDrawLine(trashGC,cp->controlWindow,0,potB-1,controlWidth,potB-1,Xoption);
  /* Draw a horizontal white line above the rendering mode buttons. */
  GDrawLine(trashGC, cp->controlWindow, 0, butA, controlWidth, butA, Xoption);
  /* Draw a horizontal white line above the color mapping area. */
  GDrawLine(trashGC,cp->controlWindow,0, cmapA, controlWidth, cmapA, Xoption);
  GSetLineAttributes(trashGC, 3, LineSolid, CapButt, JoinMiter, Xoption);
  /* Draw a horizontal white line above the potentiometer area. */
  GDrawLine(trashGC, cp->controlWindow, 0, potA, controlWidth, potA, Xoption);
  /* Set the line width as 1 here because it is used below as well. */
  GSetLineAttributes(trashGC, 1, LineSolid, CapButt, JoinMiter, Xoption);
  /* Draw inner white lines around quit, hide panel, and reset buttons. */
  GDrawLine(trashGC,cp->controlWindow,closeL,butA, closeL, butA+110, Xoption);
  /* Write potentiometer titles on the control panel. */
  writeControlTitle(cp->controlWindow);
  GSetForeground(globGC, (float)controlPotHeaderColor, Xoption);
  s    = "Rotate";
  GDrawString(globGC,cp->controlWindow,35,31+headerHeight,s,strlen(s),Xoption);
  s    = "Translate";
  GDrawString(globGC,cp->controlWindow,202,31+headerHeight,s,
              strlen(s),Xoption);
  s    = "Scale";
  GDrawString(globGC,cp->controlWindow,126,31+headerHeight,s,
              strlen(s),Xoption);
  GSetForeground(globGC, (float)controlColorColor, Xoption);
  /* Write labels on regular buttons, draw pixmaps on the potentiometers. */
  GSetForeground(globalGC1, (float)monoColor(buttonColor), Xoption);
  for (i=controlButtonsStart3D; i<(controlButtonsEnd3D); i++) {
    /* special cases depending on initial conditions */
    /* check if axes are set on or off */
    if (((cp->buttonQueue[i]).buttonKey == axesOnOff) &&
          (viewport->axesOn)) {
      (cp->buttonQueue[i]).textColor = onColor;
```

```
        if (mono) {
          GSetForeground(globalGC1, (float)backgroundColor, Xoption);
          XFillRectangle(dsply, control->controlWindow, globalGC1,
                    (control->buttonQueue[axesOnOff]).buttonX,
                    (control->buttonQueue[axesOnOff]).buttonY,
                    (control->buttonQueue[axesOnOff]).buttonWidth,
                    (control->buttonQueue[axesOnOff]).buttonHeight);
          GSetForeground(globalGC1, (float)foregroundColor, Xoption);
          GDrawRectangle(globalGC1,control->controlWindow,
                      (control->buttonQueue[axesOnOff]).buttonX,
                      (control->buttonQueue[axesOnOff]).buttonY,
                      (control->buttonQueue[axesOnOff]).buttonWidth,
                      (control->buttonQueue[axesOnOff]).buttonHeight,Xoption);
        }
      } else {
        if (((cp->buttonQueue[i]).buttonKey == axesOnOff) &&
            (!viewport->axesOn)) {
          (cp->buttonQueue[i]).textColor = offColor;
          if (mono) {
            XChangeShade(dsply,offShade);
            XShadeRectangle(dsply,cp->controlWindow,
                      (cp->buttonQueue[i]).buttonX,
                      (cp->buttonQueue[i]).buttonY,
                      (cp->buttonQueue[i]).buttonWidth,
                      (cp->buttonQueue[i]).buttonHeight);
            s = (control->buttonQueue[axesOnOff]).text;
            strlength = strlen(s);
            GSetForeground(processGC,
              (float)monoColor((control->buttonQueue[axesOnOff]).textColor),
              Xoption);
            GDrawImageString(processGC,control->controlWindow,
                            (control->buttonQueue[axesOnOff]).buttonX +
                            centerX(processGC,s,strlength,
                            (control->buttonQueue[axesOnOff]).buttonWidth),
                            (control->buttonQueue[axesOnOff]).buttonY +
                            centerY(processGC,
                            (control->buttonQueue[axesOnOff]).buttonHeight),
                            s,strlength,Xoption);
          } /* if mono */
        }
      } /* if axes */
      /* check if bounding region is set on or off */
      if (((cp->buttonQueue[i]).buttonKey == region3D) &&
          (viewport->regionOn)) {
        (cp->buttonQueue[i]).textColor = onColor;
        if (mono) {
```

```
            GSetForeground(globalGC1, (float)backgroundColor, Xoption);
            XFillRectangle(dsply, control->controlWindow, globalGC1,
                        (control->buttonQueue[region3D]).buttonX,
                        (control->buttonQueue[region3D]).buttonY,
                        (control->buttonQueue[region3D]).buttonWidth,
                        (control->buttonQueue[region3D]).buttonHeight);
            GSetForeground(globalGC1, (float)foregroundColor, Xoption);
            GDrawRectangle(globalGC1,control->controlWindow,
                            (control->buttonQueue[region3D]).buttonX,
                            (control->buttonQueue[region3D]).buttonY,
                            (control->buttonQueue[region3D]).buttonWidth,
                            (control->buttonQueue[region3D]).buttonHeight,Xoption);
    }
} else {
    if (((cp->buttonQueue[i]).buttonKey == region3D) &&
        (!viewport->regionOn)) {
        (cp->buttonQueue[i]).textColor = offColor;
        if (mono) {
            XChangeShade(dsply,offShade);
            XShadeRectangle(dsply,cp->controlWindow,
                        (cp->buttonQueue[i]).buttonX,
                        (cp->buttonQueue[i]).buttonY,
                        (cp->buttonQueue[i]).buttonWidth,
                        (cp->buttonQueue[i]).buttonHeight);
            s = (control->buttonQueue[region3D]).text;
            strlength = strlen(s);
            GSetForeground(processGC,
                (float)monoColor((control->buttonQueue[region3D]).textColor),
                Xoption);
            GDrawImageString(processGC,control->controlWindow,
                            (control->buttonQueue[region3D]).buttonX +
                            centerX(processGC,s,strlength,
                            (control->buttonQueue[region3D]).buttonWidth),
                            (control->buttonQueue[region3D]).buttonY +
                            centerY(processGC,
                            (control->buttonQueue[region3D]).buttonHeight),
                            s,strlength,Xoption);
        } /* if mono */
    }
} /* if bounding region */
/* check if black and white is set on or off */
if (((cp->buttonQueue[i]).buttonKey == bwColor) && (mono)) {
    (cp->buttonQueue[i]).text = " ";
    XChangeShade(dsply,offShade);
    XShadeRectangle(dsply,cp->controlWindow,
                (cp->buttonQueue[i]).buttonX,
```

```
                        (cp->buttonQueue[i]).buttonY,
                        (cp->buttonQueue[i]).buttonWidth,
                        (cp->buttonQueue[i]).buttonHeight);
      } else {
        if (((cp->buttonQueue[i]).buttonKey == bwColor) && viewport->monoOn) {
          (cp->buttonQueue[i]).textColor = onColor;
          s = (control->buttonQueue[bwColor]).text;
          strlength = strlen(s);
          GSetForeground(processGC,
            (float)monoColor((control->buttonQueue[bwColor]).textColor),Xoption);
          GDrawImageString(processGC,control->controlWindow,
                          (control->buttonQueue[bwColor]).buttonX +
                          centerX(processGC,s,strlength,
                                  (control->buttonQueue[bwColor]).buttonWidth),
                          (control->buttonQueue[bwColor]).buttonY +
                          centerY(processGC,
                                  (control->buttonQueue[bwColor]).buttonHeight),
                          s,strlength,Xoption);
        } else {
          if (((cp->buttonQueue[i]).buttonKey == bwColor) &&
              (!viewport->monoOn)) {
            (cp->buttonQueue[i]).textColor = offColor;
            s = (control->buttonQueue[bwColor]).text;
            strlength = strlen(s);
            GSetForeground(processGC,
              (float)monoColor((control->buttonQueue[bwColor]).textColor),
              Xoption);
            GDrawImageString(processGC,control->controlWindow,
                            (control->buttonQueue[bwColor]).buttonX +
                            centerX(processGC,s,strlength,
                            (control->buttonQueue[bwColor]).buttonWidth),
                            (control->buttonQueue[bwColor]).buttonY +
                            centerY(processGC,
                            (control->buttonQueue[bwColor]).buttonHeight),
                            s,strlength,Xoption);
          }
        }
      } /* if black and white */
      /* check if object rotation is set on or off */
      if (((cp->buttonQueue[i]).buttonKey == objectr) &&
              (viewport->objectrOn)) {
          (control->buttonQueue[objectr]).textColor = onColor;
          if (mono) {
            GSetForeground(globalGC1, (float)backgroundColor, Xoption);
            XFillRectangle(dsply, control->controlWindow, globalGC1,
                    (control->buttonQueue[objectr]).buttonX,
```

```
                       (control->buttonQueue[objectr]).buttonY,
                       (control->buttonQueue[objectr]).buttonWidth,
                       (control->buttonQueue[objectr]).buttonHeight);
              GSetForeground(globalGC1, (float)foregroundColor, Xoption);
              GDrawRectangle(globalGC1,control->controlWindow,
                             (control->buttonQueue[objectr]).buttonX,
                             (control->buttonQueue[objectr]).buttonY,
                             (control->buttonQueue[objectr]).buttonWidth,
                             (control->buttonQueue[objectr]).buttonHeight,Xoption);
       }
    } else {
      if (((cp->buttonQueue[i]).buttonKey == objectr) &&
              (!viewport->objectrOn)) {
        (control->buttonQueue[objectr]).textColor = offColor;
        if (mono) {
          XChangeShade(dsply,offShade);
          XShadeRectangle(dsply,control->controlWindow,
                  (control->buttonQueue[objectr]).buttonX,
                  (control->buttonQueue[objectr]).buttonY,
                  (control->buttonQueue[objectr]).buttonWidth,
                  (control->buttonQueue[objectr]).buttonHeight);
          GSetForeground(globalGC1, (float)foregroundColor, Xoption);
          GDrawRectangle(globalGC1,control->controlWindow,
                         (control->buttonQueue[objectr]).buttonX,
                         (control->buttonQueue[objectr]).buttonY,
                         (control->buttonQueue[objectr]).buttonWidth,
                         (control->buttonQueue[objectr]).buttonHeight,Xoption);
          GSetForeground(processGC,
            (float)monoColor((control->buttonQueue[objectr]).textColor),
            Xoption);
          GDrawImageString(processGC,control->controlWindow,
                    (control->buttonQueue[objectr]).buttonX +
                    centerX(processGC,(control->buttonQueue[objectr]).text,
                           strlen((control->buttonQueue[objectr]).text),
                           (control->buttonQueue[objectr]).buttonWidth),
                           (control->buttonQueue[objectr]).buttonY +
                           centerY(processGC,
                           (control->buttonQueue[objectr]).buttonHeight),
                           (control->buttonQueue[objectr]).text,
                           strlen((control->buttonQueue[objectr]).text),
                           Xoption);
        }
    } /* else not object rotation */
  } /* if object rotation */
  /* check if origin rotation is set on or off */
  if (((cp->buttonQueue[i]).buttonKey == originr) &&
```

```
            (viewport->originrOn)) {
        (control->buttonQueue[originr]).textColor = onColor;
        if (mono) {
          GSetForeground(globalGC1, (float)backgroundColor, Xoption);
          XFillRectangle(dsply, control->controlWindow, globalGC1,
                 (control->buttonQueue[originr]).buttonX,
                 (control->buttonQueue[originr]).buttonY,
                 (control->buttonQueue[originr]).buttonWidth,
                 (control->buttonQueue[originr]).buttonHeight);
          GSetForeground(globalGC1, (float)foregroundColor, Xoption);
          GDrawRectangle(globalGC1,control->controlWindow,
                     (control->buttonQueue[originr]).buttonX,
                     (control->buttonQueue[originr]).buttonY,
                     (control->buttonQueue[originr]).buttonWidth,
                     (control->buttonQueue[originr]).buttonHeight,Xoption);
        }
    } else {
      if (((cp->buttonQueue[i]).buttonKey == originr) &&
             (!viewport->originrOn)) {
        (control->buttonQueue[originr]).textColor = offColor;
        if (mono) {
          XChangeShade(dsply,offShade);
          XShadeRectangle(dsply,control->controlWindow,
                 (control->buttonQueue[originr]).buttonX,
                 (control->buttonQueue[originr]).buttonY,
                 (control->buttonQueue[originr]).buttonWidth,
                 (control->buttonQueue[originr]).buttonHeight);
          GSetForeground(globalGC1, (float)foregroundColor, Xoption);
          GDrawRectangle(globalGC1,control->controlWindow,
                     (control->buttonQueue[originr]).buttonX,
                     (control->buttonQueue[originr]).buttonY,
                     (control->buttonQueue[originr]).buttonWidth,
                     (control->buttonQueue[originr]).buttonHeight,Xoption);

          GSetForeground(processGC,
                 (float)monoColor((control->buttonQueue[originr]).textColor),
                 Xoption);
          GDrawImageString(processGC,control->controlWindow,
                   (control->buttonQueue[originr]).buttonX +
                   centerX(processGC,(control->buttonQueue[originr]).text,
                         strlen((control->buttonQueue[originr]).text),
                         (control->buttonQueue[originr]).buttonWidth),
                         (control->buttonQueue[originr]).buttonY +
                         centerY(processGC,
                         (control->buttonQueue[originr]).buttonHeight),
                         (control->buttonQueue[originr]).text,
```

```
                          strlen((control->buttonQueue[originr]).text),
                          Xoption);
          }
      } /* else not origin rotation */
  } /* if origin rotation */
  /* check if zoom X is set on or off */
  if (((cp->buttonQueue[i]).buttonKey == zoomx) &&
          (viewport->zoomXOn)) {
      (control->buttonQueue[zoomx]).textColor = onColor;
      if (mono) {
          GSetForeground(globalGC1, (float)backgroundColor, Xoption);
          XFillRectangle(dsply, control->controlWindow, globalGC1,
                  (control->buttonQueue[zoomx]).buttonX,
                  (control->buttonQueue[zoomx]).buttonY,
                  (control->buttonQueue[zoomx]).buttonWidth,
                  (control->buttonQueue[zoomx]).buttonHeight);
          GSetForeground(globalGC1, (float)foregroundColor, Xoption);
          GDrawRectangle(globalGC1,control->controlWindow,
                      (control->buttonQueue[zoomx]).buttonX,
                      (control->buttonQueue[zoomx]).buttonY,
                      (control->buttonQueue[zoomx]).buttonWidth,
                      (control->buttonQueue[zoomx]).buttonHeight,Xoption);
      }
  } else {
    if (((cp->buttonQueue[i]).buttonKey == zoomx) &&
            (!viewport->zoomXOn)) {
      (control->buttonQueue[zoomx]).textColor = offColor;
      if (mono) {
        XChangeShade(dsply,offShade);
        XShadeRectangle(dsply,control->controlWindow,
                (control->buttonQueue[zoomx]).buttonX,
                (control->buttonQueue[zoomx]).buttonY,
                (control->buttonQueue[zoomx]).buttonWidth,
                (control->buttonQueue[zoomx]).buttonHeight);
        GSetForeground(globalGC1, (float)foregroundColor, Xoption);
        GDrawRectangle(globalGC1,control->controlWindow,
                    (control->buttonQueue[zoomx]).buttonX,
                    (control->buttonQueue[zoomx]).buttonY,
                    (control->buttonQueue[zoomx]).buttonWidth,
                    (control->buttonQueue[zoomx]).buttonHeight,Xoption);

        GSetForeground(processGC,
                (float)monoColor((control->buttonQueue[zoomx]).textColor),
                Xoption);
        GDrawImageString(processGC,control->controlWindow,
                    (control->buttonQueue[zoomx]).buttonX +
```

```
                     centerX(processGC,(control->buttonQueue[zoomx]).text,
                              strlen((control->buttonQueue[zoomx]).text),
                              (control->buttonQueue[zoomx]).buttonWidth),
                              (control->buttonQueue[zoomx]).buttonY +
                              centerY(processGC,
                                      (control->buttonQueue[zoomx]).buttonHeight),
                              (control->buttonQueue[zoomx]).text,
                              strlen((control->buttonQueue[zoomx]).text),Xoption);
        }
      } /* else not zoom X */
    } /* if zoom X */
    /* check if zoom Y is set on or off */
    if (((cp->buttonQueue[i]).buttonKey == zoomy) &&
            (viewport->zoomYOn)) {
        (control->buttonQueue[zoomy]).textColor = onColor;
        if (mono) {
          GSetForeground(globalGC1, (float)backgroundColor, Xoption);
          XFillRectangle(dsply, control->controlWindow, globalGC1,
                  (control->buttonQueue[zoomy]).buttonX,
                  (control->buttonQueue[zoomy]).buttonY,
                  (control->buttonQueue[zoomy]).buttonWidth,
                  (control->buttonQueue[zoomy]).buttonHeight);
          GSetForeground(globalGC1, (float)foregroundColor, Xoption);
          GDrawRectangle(globalGC1, control->controlWindow,
                          (control->buttonQueue[zoomy]).buttonX,
                          (control->buttonQueue[zoomy]).buttonY,
                          (control->buttonQueue[zoomy]).buttonWidth,
                          (control->buttonQueue[zoomy]).buttonHeight,Xoption);
        }
    } else {
      if (((cp->buttonQueue[i]).buttonKey == zoomy) &&
              (!viewport->zoomYOn)) {
        (control->buttonQueue[zoomy]).textColor = offColor;
        if (mono) {
          XChangeShade(dsply,offShade);
          XShadeRectangle(dsply,control->controlWindow,
                  (control->buttonQueue[zoomy]).buttonX,
                  (control->buttonQueue[zoomy]).buttonY,
                  (control->buttonQueue[zoomy]).buttonWidth,
                  (control->buttonQueue[zoomy]).buttonHeight);
          GSetForeground(globalGC1, (float)foregroundColor, Xoption);
          GDrawRectangle(globalGC1,control->controlWindow,
                          (control->buttonQueue[zoomy]).buttonX,
                          (control->buttonQueue[zoomy]).buttonY,
                          (control->buttonQueue[zoomy]).buttonWidth,
                          (control->buttonQueue[zoomy]).buttonHeight,Xoption);
```

```
                    GSetForeground(processGC,
                            (float)monoColor((control->buttonQueue[zoomy]).textColor),
                            Xoption);
                    GDrawImageString(processGC,control->controlWindow,
                            (control->buttonQueue[zoomy]).buttonX +
                            centerX(processGC,(control->buttonQueue[zoomy]).text,
                                    strlen((control->buttonQueue[zoomy]).text),
                                    (control->buttonQueue[zoomy]).buttonWidth),
                                    (control->buttonQueue[zoomy]).buttonY +
                                    centerY(processGC,
                                            (control->buttonQueue[zoomy]).buttonHeight),
                                    (control->buttonQueue[zoomy]).text,
                                    strlen((control->buttonQueue[zoomy]).text),Xoption);
                }
            } /* else not zoom Y */
        } /* if zoom Y */
        /* check if zoom Z is set on or off */
        if (((cp->buttonQueue[i]).buttonKey == zoomz) &&
                (viewport->zoomZOn)) {
            (control->buttonQueue[zoomz]).textColor = onColor;
            if (mono) {
                GSetForeground(globalGC1, (float)backgroundColor, Xoption);
                XFillRectangle(dsply, control->controlWindow, globalGC1,
                        (control->buttonQueue[zoomz]).buttonX,
                        (control->buttonQueue[zoomz]).buttonY,
                        (control->buttonQueue[zoomz]).buttonWidth,
                        (control->buttonQueue[zoomz]).buttonHeight);
                GSetForeground(globalGC1, (float)foregroundColor, Xoption);
                GDrawRectangle(globalGC1,control->controlWindow,
                                (control->buttonQueue[zoomz]).buttonX,
                                (control->buttonQueue[zoomz]).buttonY,
                                (control->buttonQueue[zoomz]).buttonWidth,
                                (control->buttonQueue[zoomz]).buttonHeight,Xoption);
            }
        } else {
            if (((cp->buttonQueue[i]).buttonKey == zoomz) &&
                    (!viewport->zoomZOn)) {
                (control->buttonQueue[zoomz]).textColor = offColor;
                if (mono) {
                    XChangeShade(dsply,offShade);
                    XShadeRectangle(dsply,control->controlWindow,
                            (control->buttonQueue[zoomz]).buttonX,
                            (control->buttonQueue[zoomz]).buttonY,
                            (control->buttonQueue[zoomz]).buttonWidth,
                            (control->buttonQueue[zoomz]).buttonHeight);
```

```
                        GSetForeground(globalGC1, (float)foregroundColor, Xoption);
                        GDrawRectangle(globalGC1,control->controlWindow,
                                         (control->buttonQueue[zoomz]).buttonX,
                                         (control->buttonQueue[zoomz]).buttonY,
                                         (control->buttonQueue[zoomz]).buttonWidth,
                                         (control->buttonQueue[zoomz]).buttonHeight,Xoption);

                        GSetForeground(processGC,
                                (float)monoColor((control->buttonQueue[zoomz]).textColor),
                                Xoption);
                        GDrawImageString(processGC,control->controlWindow,
                                  (control->buttonQueue[zoomz]).buttonX +
                                  centerX(processGC,(control->buttonQueue[zoomz]).text,
                                          strlen((control->buttonQueue[zoomz]).text),
                                          (control->buttonQueue[zoomz]).buttonWidth),
                                          (control->buttonQueue[zoomz]).buttonY +
                                          centerY(processGC,
                                                  (control->buttonQueue[zoomz]).buttonHeight),
                                          (control->buttonQueue[zoomz]).text,
                                          strlen((control->buttonQueue[zoomz]).text),Xoption);
            }
        } /* else not zoom Y */
    } /* if zoom Y */
    /* check if outline is set on or off */
    if ((((cp->buttonQueue[i]).buttonKey == outlineOnOff) &&
          (viewData.outlineRenderOn)) {
        (cp->buttonQueue[i]).textColor = onColor;
    } else {
        if ((((cp->buttonQueue[i]).buttonKey == outlineOnOff) &&
              !(viewData.outlineRenderOn)) {
            (cp->buttonQueue[i]).textColor = offColor;
            if (mono) {
                XChangeShade(dsply,offShade);
                XShadeRectangle(dsply,cp->controlWindow,
                          (cp->buttonQueue[i]).buttonX,
                          (cp->buttonQueue[i]).buttonY,
                          (cp->buttonQueue[i]).buttonWidth,
                          (cp->buttonQueue[i]).buttonHeight);
                s = (control->buttonQueue[outlineOnOff]).text;
                strlength = strlen(s);

                GSetForeground(processGC,
                  (float)monoColor((control->buttonQueue[outlineOnOff]).textColor),
                  Xoption);
                GDrawImageString(processGC,control->controlWindow,
                                  (control->buttonQueue[outlineOnOff]).buttonX +
```

```
                              centerX(processGC,s,strlength,
                              (control->buttonQueue[outlineOnOff]).buttonWidth),
                              (control->buttonQueue[outlineOnOff]).buttonY +
                              centerY(processGC,
                              (control->buttonQueue[outlineOnOff]).buttonHeight),
                              s,strlength,Xoption);
        } /* if mono */
      } /* outline off */
    } /* outline on */
    /* Draw the button window border */
    GDraw3DButtonOut(globalGC1,cp->controlWindow,
                    (cp->buttonQueue[i]).buttonX, (cp->buttonQueue[i]).buttonY,
                    (cp->buttonQueue[i]).buttonWidth,
                    (cp->buttonQueue[i]).buttonHeight,Xoption);
    GSetForeground(trashGC,
                    (float)monoColor((cp->buttonQueue[i]).textColor), Xoption);
    switch (i) {
    case rotate:
      GDrawArc(trashGC, cp->controlWindow,
              rotateX, rotateY, rotateR, rotateR, 0, 360*64, Xoption);
      break;
    case zoom:
      GDrawLines(trashGC, cp->controlWindow, zoomArrow, zoomArrowN,
                CoordModeOrigin, Xoption);
      break;
    case translate:
      GDrawLines(trashGC, cp->controlWindow, translateArrow,
                translateArrowN, CoordModeOrigin, Xoption);
      break;
    default:
      s = (cp->buttonQueue[i]).text;
      strlength = strlen(s);
      GDrawString(trashGC, cp->controlWindow,
                  (cp->buttonQueue[i]).buttonX +
                  centerX(processGC,s,strlength,
                        (cp->buttonQueue[i]).buttonWidth),
                  (cp->buttonQueue[i]).buttonY +
                  centerY(processGC,
                        (cp->buttonQueue[i]).buttonHeight),s,strlen(s),
                        Xoption);
      break;
    };
    if ((cp->buttonQueue[i]).pot) {
      /* draw horizontal and vertical centerlines */
      GDrawLine(globalGC1,cp->controlWindow,
                (cp->buttonQueue[i]).buttonX + (cp->buttonQueue[i]).xHalf,
```

```
                    (cp->buttonQueue[i]).buttonY,
                    (cp->buttonQueue[i]).buttonX + (cp->buttonQueue[i]).xHalf,
                    (cp->buttonQueue[i]).buttonY + 2*(cp->buttonQueue[i]).yHalf,
                    Xoption);
         GDrawLine(globalGC1,cp->controlWindow,
                    (cp->buttonQueue[i]).buttonX,
                    (cp->buttonQueue[i]).buttonY + (cp->buttonQueue[i]).yHalf,
                    (cp->buttonQueue[i]).buttonX + 2*(cp->buttonQueue[i]).xHalf,
                    (cp->buttonQueue[i]).buttonY + (cp->buttonQueue[i]).yHalf,
                    Xoption);
      }
   }
   /* refresh the latest message */
   clearControlMessage();
   strcpy(control->message,viewport->title);
   writeControlMessage();
  /* Draw the color map window */
   cp = viewport->controlPanel;
   drawColorMap();
   XFlush(dsply);
 }  /* drawControlPanel() */
```

7.3.15   getControlXY

Determines the x and y coordinate where the control panel is to be placed, based upon where the mouse button was pressed within the graph viewport window.

⟨*view3d*⟩+≡

```
controlXY getControlXY(int whereDoYouWantPanel) {
  XWindowAttributes wAttrib;
  controlXY         cXY = {0,0};
  int               viewX, viewY, viewW, viewH, tmp=1;
  Window            rootW, parentW, *childrenWs, tmpW;
  unsigned int      nChildren;
  tmpW = viewport->titleWindow;
  while(tmp) {
    XQueryTree(dsply,tmpW,&rootW,&parentW,&childrenWs,&nChildren);
    XFree(childrenWs);
    if (parentW == rtWindow) {
      tmp = 0;
    } else {
      tmpW = parentW;
    }
  }
  XGetWindowAttributes(dsply,tmpW,&wAttrib);
  viewX = wAttrib.x;
  viewY = wAttrib.y;
  viewW = wAttrib.width;
  viewH = wAttrib.height;
  if (whereDoYouWantPanel) {
    switch (whereDoYouWantPanel) {
    case 1: /* right */
      cXY.putX = viewX + viewW;
      cXY.putY = viewY;
      break;
    case 2: /* bottom */
      cXY.putX = viewX + (viewW - controlWidth)/2;   /* center it */
      cXY.putY = viewY + viewH;
      break;
    case 3: /* left */
      cXY.putX = viewX - controlWidth  - borderWidth;
      cXY.putY = viewY;
      break;
    case 4: /* top */
      cXY.putX = viewX + (viewW - controlWidth)/2;   /* center it */
      cXY.putY = viewY - controlHeight - borderHeight;
    }
  } else {
```

```
      if ((physicalWidth - (viewX + viewW)) >= controlWidth) {
        cXY.putX = viewX + viewW;
        cXY.putY = viewY;
      } else if ((physicalHeight - (viewY + viewH)) >= controlHeight) {
        cXY.putX = viewX + (viewW - controlWidth)/2;   /* center it */
        cXY.putY = viewY + viewH;
      } else if (viewX >= controlWidth) {
        cXY.putX = viewX - controlWidth - borderWidth;
        cXY.putY = viewY;
      } else if (viewY >= controlHeight) {
        cXY.putX = viewX + (viewW - controlWidth)/2;   /* center it */
        cXY.putY = viewY - controlHeight - borderHeight;
      } else {                          /* put inside of viewport */
        cXY.putX = viewX + viewW - controlWidth;
        cXY.putY = viewY + viewH - controlHeight;
      }
    }
    if (cXY.putX < 0) cXY.putX = 0;
    if (cXY.putY < 0) cXY.putY = 0;
    return(cXY);
  }
```

7.3.16   makeControlPanel

⟨*view3d*⟩+≡

```
controlPanelStruct *makeControlPanel(void) {
  Window               cw;
  int                  i, num;
  controlPanelStruct   *control;
  buttonStruct         *buttons;
  controlXY            cXY = {0,0};
  XSetWindowAttributes cwAttrib, controlAttrib;
  XSizeHints           sizehint;
  Pixmap               mousebits, mousemask;
  XColor               foreColor, backColor;
  if (!(control = (controlPanelStruct *)saymem("control.c",1,
                        sizeof(controlPanelStruct)))) {
    fprintf(stderr,"Ran out of memory trying to create control panel.\n");
    exitWithAck(RootWindow(dsply,scrn),Window,-1);
  }
  cXY = getControlXY(0);
  mousebits = XCreateBitmapFromData(dsply,rtWindow, mouseBitmap_bits,
                                    mouseBitmap_width, mouseBitmap_height);
  mousemask = XCreateBitmapFromData(dsply,rtWindow, mouseMask_bits,
                                    mouseMask_width, mouseMask_height);
  cwAttrib.background_pixel    = backgroundColor;
  cwAttrib.border_pixel        = foregroundColor;
  cwAttrib.event_mask          = controlMASK;
  cwAttrib.colormap            = colorMap;
  cwAttrib.override_redirect   = overrideManager;
  foreColor.pixel              = controlCursorForeground;
  XQueryColor(dsply,colorMap,&foreColor);
  backColor.pixel              = controlCursorBackground;
  XQueryColor(dsply,colorMap,&backColor);
  cwAttrib.cursor              = XCreatePixmapCursor(dsply,mousebits,
                                      mousemask, &foreColor,&backColor,
                                      mouseBitmap_x_hot,mouseBitmap_y_hot);
  cw = XCreateWindow(dsply,rtWindow,
                     cXY.putX,cXY.putY,controlWidth,controlHeight,3,
                     CopyFromParent,InputOutput,CopyFromParent,
                     controlCreateMASK,&cwAttrib);
  sizehint.flags = PPosition | PSize;
  sizehint.x     = cXY.putX;
  sizehint.y     = cXY.putY;
  sizehint.width = controlWidth;
  sizehint.height = controlHeight;
  /*** the None stands for icon pixmap  ***/
  XSetNormalHints(dsply,cw,&sizehint);
```

```
      XSetStandardProperties(dsply,cw,"3D Control Panel","3D Control Panel",
                             None,NULL,0,&sizehint);
      /* Define and assign a mouse cursor */
      control->controlWindow = cw;
      num = initButtons(control->buttonQueue);
      buttons = control->buttonQueue;
      for (i=controlButtonsStart3D; i<(controlButtonsEnd3D); i++) {
        controlAttrib.event_mask = (control->buttonQueue[i]).mask;
        (control->buttonQueue[i]).self = XCreateWindow(dsply,cw,
                                            (control->buttonQueue[i]).buttonX,
                                            (control->buttonQueue[i]).buttonY,
                                            (control->buttonQueue[i]).buttonWidth,
                                            (control->buttonQueue[i]).buttonHeight,
                                            0,0,InputOnly,CopyFromParent,
                                            buttonCreateMASK,&controlAttrib);
        XMakeAssoc(dsply,table,(control->buttonQueue[i]).self,
                   &((control->buttonQueue[i]).buttonKey));
        /* use buttonKey and not i because buttonKey has a permanent address */
        XMapWindow(dsply,(control->buttonQueue[i]).self);
      } /* for each button */
      /* Set up the potentiometer pixmaps. */
      for (i=0; i<zoomArrowN; i++) {
        zoomArrow[i].x += buttons[zoom].buttonX;
        zoomArrow[i].y += buttons[zoom].buttonY;
      }
      for (i=0; i<translateArrowN; i++) {
        translateArrow[i].x += buttons[translate].buttonX;
        translateArrow[i].y += buttons[translate].buttonY;
      }
      rotateX = control->buttonQueue[rotate].buttonX+17;
      rotateY = control->buttonQueue[rotate].buttonY+2;
      rotateR = control->buttonQueue[rotate].buttonHeight-4;
      strcpy(control->message,"                                  ");
      /* Create the color mapping window */
      controlAttrib.event_mask   = colorMASK;
      control->colormapWindow = XCreateWindow(dsply,cw, colorWidth,colormapY,
                                 colormapW,colormapH,0, 0,InputOnly,
                                 CopyFromParent, colormapCreateMASK,
                                 &controlAttrib);
      XMapWindow(dsply,control->colormapWindow);
      viewport->justMadeControl = yes;
      return(control);
   } /* makeControlPanel() */
```

7.3.17   putControlPanelSomewhere

This routine puts up the control panel associated with the viewport passed in. It first tries
to put it to the right of the viewport. If there isn't enough room, it tries the bottom and so
on going clockwise. If the viewport is too big and there is no room to put the control panel
outside of it, the control panel is placed on the bottom right hand corner of the viewport.

⟨*view3d*⟩+≡

```
  void putControlPanelSomewhere(int whereDoesPanelGo) {
    controlPanelStruct *control;
    controlXY          whereControl = {0,0};
    control      = viewport->controlPanel;
    whereControl = getControlXY(whereDoesPanelGo);
    viewport->haveControl = yes;
    XRaiseWindow(dsply,control->controlWindow);
    XMoveWindow(dsply, control->controlWindow,
                whereControl.putX, whereControl.putY);
    drawControlPanel();
    XSync(dsply,0);
    if (viewport->justMadeControl) {
      XMapWindow(dsply,control->controlWindow);
      viewport->justMadeControl = no;
    }
    XMapWindow(dsply,control->controlWindow);
    XFlush(dsply);
  }
```

7.3.18   phong

A general routine for determining the intensity values at a particular point using the Phong illumination model with Phong shading

⟨*view3d*⟩+≡

```
float phong(triple pt,float N[3]) {
  float     dotLN, dotHN, H[3], E[3], P[3], NM[3], L[3];
  float     color, diffuse, specular;
  diffuse = 0.0;  specular = 0.0;
  /* renormalize average normal vector for the point */
  normalizeVector(N);
  /* temporary norm, in case the sign is switched */
  NM[0] = N[0];  NM[1] = N[1];  NM[2] = N[2];
  P[0] = pt.x;  P[1] = pt.y;  P[2] = pt.z;
  normalizeVector(P);
  /* vector from infinite light source */
  L[0] = viewport->lightVector[0];
  L[1] = viewport->lightVector[1];
  L[2] = viewport->lightVector[2];
  normalizeVector(L);
  /* vector from point to observers eye */
  normalizeVector(eyePoint);
  E[0] = 8.0*eyePoint[0] - P[0];
  E[1] = 8.0*eyePoint[1] - P[1];
  E[2] = 8.0*eyePoint[2] - P[2];
  normalizeVector(E);
  /*  light returned even if normal faces away from light source */
  dotLN = L[0]*NM[0] + L[1]*NM[1] + L[2]*NM[2];
  if (dotLN < 0.0) dotLN = -dotLN;
  diffuse = dotLN*lightIntensity;
  /* calculate specular highlight if surface faces light source */
  if (dotLN > 0.0) {
    H[0] = E[0] + L[0];
    H[1] = E[1] + L[1];
    H[2] = E[2] + L[2];
    normalizeVector(H);
    dotHN = NM[0]*H[0]+NM[1]*H[1]+NM[2]*H[2];
    if (dotHN < 0.0) dotHN = -dotHN;
    specular = pow((double)dotHN,coeff)*lightIntensity;
  }
  /* return intensity value from 0.0 to 1.0 */
  color = Camb + diffuse*Cdiff + specular*Cspec;
  if (color > 1.0) color = 1.0;
  if (color < 0.0) color = 0.0;
  return(color);
```

```
  }
```

7.3.19 hueValue

⟨*view3d*⟩+≡
```
  int hueValue(float val) {
    int hue;
    hue = floor(absolute(val) * viewport->numberOfHues) + viewport->hueOffset;
    if (hue > 26) hue = 26;
    return hue;
  }
```

7.3.20 getHue

⟨*view3d*⟩+≡
```
  int getHue(float val) {
    int hue;
    hue = hueValue(val);
    if (hue < 11)
      hue *= 6;
    else
      if (hue > 10 && hue < 16)
        hue = hue*20 - 140;
      else
        hue = hue*12 - 12;
    return hue;
  }
```

7.3.21   Value

⟨*view3d*⟩+≡
```
float Value(float n1, float n2, float hue) {
  float v;
  if (hue > 360.0) hue -= 360.0;
  if (hue < 0.0) hue += 360.0;
  if (hue < 60.0) {
    v = n1 + (n2-n1)*hue/60.0;
  } else {
    if (hue < 180.0) {
      v = n2;
    } else {
      if (hue < 240.0) {
        v = n1 + (n2-n1)*(240.0-hue)/60.0;
      } else {
        v = n1;
      }
    }
  }
  return(v);
}
```

7.3.22   hlsTOrgb

⟨*view3d*⟩+≡
```
RGB hlsTOrgb(float h,float l,float s) {
  RGB rgb;
  float m1, m2;
  if (l <= 0.5) {
    m2 = l*(1.0+s);
  }
  else {
    m2 = l+s-l*s;
  }
  m1 = 2.0*l-m2;
  rgb.r = Value(m1,m2,h+120.0);
  rgb.g = Value(m1,m2,h);
  rgb.b = Value(m1,m2,h-120.0);
  return(rgb);
}
```

7.3.23   initLightButtons

Creates the fields for each button window in the three dimensional lighting subpanel, and
returns the number of buttons created.

⟨*view3d*⟩+≡
```
  int initLightButtons(buttonStruct *lightButtons) {
    int ii;
    int num = 0;
    /* Not functional -- can be used to allow light window as a potentiometer */
    ii = lightMove;
    lightButtons[ii].buttonX      = 63;
    lightButtons[ii].buttonY      = 30;
    lightButtons[ii].buttonWidth  = 171;
    lightButtons[ii].buttonHeight = 171;
    lightButtons[ii].buttonKey    = ii;
    lightButtons[ii].pot          = yes;            /* a potentiometer */
    lightButtons[ii].mask         = potMASK;
    lightButtons[ii].textColor    = 163;
    lightButtons[ii].xHalf        = lightButtons[ii].buttonWidth/2;
    lightButtons[ii].yHalf        = lightButtons[ii].buttonHeight/2;
    ++num;
    /* Change x and y coordinate of light source */
    ii = lightMoveXY;
    lightButtons[ii].buttonX      = 20;
    lightButtons[ii].buttonY      = 247;
    lightButtons[ii].buttonWidth  = 110;
    lightButtons[ii].buttonHeight = 110;
    lightButtons[ii].buttonKey    = ii;
    lightButtons[ii].pot          = yes;            /* a potentiometer */
    lightButtons[ii].mask         = potMASK;
    lightButtons[ii].textColor    = 133;
    lightButtons[ii].xHalf        = lightButtons[ii].buttonWidth/2;
    lightButtons[ii].yHalf        = lightButtons[ii].buttonHeight/2;
    ++num;
    /* Change z coordinate of light source */
    ii = lightMoveZ;
    lightButtons[ii].buttonX      = 149;
    lightButtons[ii].buttonY      = 247;
    lightButtons[ii].buttonWidth  = 58;
    lightButtons[ii].buttonHeight = 110;
    lightButtons[ii].buttonKey    = ii;
    lightButtons[ii].pot          = yes;            /* a potentiometer */
    lightButtons[ii].mask         = potMASK;
    lightButtons[ii].textColor    = 165;
    lightButtons[ii].xHalf        = lightButtons[ii].buttonWidth/2;
```

```
    lightButtons[ii].yHalf        = lightButtons[ii].buttonHeight/2;
    ++num;
    /* Change intensity of light source */
    ii = lightTranslucent;
    lightButtons[ii].buttonX       = 250;
    lightButtons[ii].buttonY       = 247;
    lightButtons[ii].buttonWidth  = 34;
    lightButtons[ii].buttonHeight = 110;
    lightButtons[ii].buttonKey     = ii;
    lightButtons[ii].pot           = yes;          /* a potentiometer */
    lightButtons[ii].mask          = potMASK;
    lightButtons[ii].textColor     = 37;
    lightButtons[ii].xHalf         = lightButtons[ii].buttonWidth/2;
    lightButtons[ii].yHalf         = lightButtons[ii].buttonHeight/2;
    ++num;
    /* Leave lighting window without updating changes made. */
    ii = lightAbort;
    lightButtons[ii].buttonX       = 36;
    lightButtons[ii].buttonY       = 370;
    lightButtons[ii].buttonWidth  = 110;
    lightButtons[ii].buttonHeight = 24;
    lightButtons[ii].buttonKey     = ii;
    lightButtons[ii].pot           = no;
    lightButtons[ii].mask          = buttonMASK;
    lightButtons[ii].text          = "Abort";
    lightButtons[ii].textColor     = 52;
    lightButtons[ii].xHalf         = lightButtons[ii].buttonWidth/2;
    lightButtons[ii].yHalf         = lightButtons[ii].buttonHeight/2;
    ++num;
    /* Leave lighting window and update changes made. */
    ii = lightReturn;
    lightButtons[ii].buttonX       = 154;
    lightButtons[ii].buttonY       = 370;
    lightButtons[ii].buttonWidth  = 110;
    lightButtons[ii].buttonHeight = 24;
    lightButtons[ii].buttonKey     = ii;
    lightButtons[ii].pot           = no;
    lightButtons[ii].mask          = buttonMASK;
    lightButtons[ii].text          = "Return";
    lightButtons[ii].textColor     = 28;
    lightButtons[ii].xHalf         = lightButtons[ii].buttonWidth/2;
    lightButtons[ii].yHalf         = lightButtons[ii].buttonHeight/2;
    ++num;
    return(num);
}
```

7.3.24   makeLightingPanel

⟨*view3d*⟩+≡

```
int makeLightingPanel(void) {
  int i;
  XSetWindowAttributes cwAttrib, controlAttrib;
  XSizeHints sizehint;
  Pixmap lightbits,lightmask;
  XColor foreColor, backColor;
  lightbits = XCreateBitmapFromData(dsply,rtWindow, lightBitmap_bits,
                                    lightBitmap_width,lightBitmap_height);
  lightmask = XCreateBitmapFromData(dsply,rtWindow, lightMask_bits,
                                    lightMask_width,lightMask_height);
  cwAttrib.background_pixel = backgroundColor;
  cwAttrib.border_pixel = foregroundColor;
  cwAttrib.event_mask = lightMASK;
  cwAttrib.colormap = colorMap;
  cwAttrib.override_redirect = overrideManager;
  foreColor.pixel = lightCursorForeground;
  XQueryColor(dsply,colorMap,&foreColor);
  backColor.pixel = lightCursorBackground;
  XQueryColor(dsply,colorMap,&backColor);
  cwAttrib.cursor = XCreatePixmapCursor(dsply,lightbits,lightmask,
                                        &foreColor,&backColor,
                                        lightBitmap_x_hot,lightBitmap_y_hot);
  lightingWindow = XCreateWindow(dsply,control->controlWindow,
                                 -3,-3,controlWidth,controlHeight,3,
                                 CopyFromParent,InputOutput,CopyFromParent,
                                 controlCreateMASK,&cwAttrib);
  sizehint.flags  = USPosition | USSize;
  sizehint.x      = 0;
  sizehint.y      = 0;
  sizehint.width  = controlWidth;
  sizehint.height = controlHeight;
          /*** the None stands for icon pixmap. ***/
  XSetNormalHints(dsply,lightingWindow,&sizehint);
  XSetStandardProperties(dsply,lightingWindow,"Lighting Panel 3D",
                         "Lighting Panel",None,NULL,0,&sizehint);
      /*** lighting axes window ***/
  cwAttrib.event_mask = 0;
  lightingAxes = XCreateWindow(dsply,lightingWindow,
                               lightingAxesX,lightingAxesY,
                               lightingAxesSize,lightingAxesSize,
                               0,CopyFromParent,InputOutput,CopyFromParent,
                               controlCreateMASK,&cwAttrib);
```

```
    sizehint.flags  = USPosition | USSize;
    sizehint.x      = lightingAxesX;
    sizehint.y      = lightingAxesY;
    sizehint.width  = lightingAxesSize;
    sizehint.height = lightingAxesSize;
            /*** the None stands for icon pixmap ***/
    XSetNormalHints(dsply,lightingAxes,&sizehint);
    XSetStandardProperties(dsply,lightingAxes,"Lighting Axes","Lighting Axes",
                          None,NULL,0,&sizehint);
    XMapWindow(dsply,lightingAxes);
      /*** draw lighting buttons ***/
    initLightButtons(control->buttonQueue);
    for (i=(lightingButtonsStart + 1); i<(lightingButtonsEnd); i++) {
      controlAttrib.event_mask = (control->buttonQueue[i]).mask;
      (control->buttonQueue[i]).self =
                      XCreateWindow(dsply,lightingWindow,
                        (control->buttonQueue[i]).buttonX,
                        (control->buttonQueue[i]).buttonY,
                        (control->buttonQueue[i]).buttonWidth,
                        (control->buttonQueue[i]).buttonHeight,
                        0,0,InputOnly,CopyFromParent,
                        buttonCreateMASK,&controlAttrib);
      XMakeAssoc(dsply,table,(control->buttonQueue[i]).self,
                &((control->buttonQueue[i]).buttonKey));
      XMapWindow(dsply,(control->buttonQueue[i]).self);
    }
      /* assign global direction variables for light projections */
    sinTheta  = sin(-viewport->theta);
    cosTheta  = cos(-viewport->theta);
    sinPhi    = sin(viewport->phi);
    cosPhi    = cos(viewport->phi);
    return(0);
  } /* makeLightingPanel() */
```

7.3.25   drawLightingAxes

⟨*view3d*⟩+≡

```
void drawLightingAxes(void) {
  XWindowAttributes laInfo;
  int i,xCenter,yCenter;
  float Px0,Py0;
  int vPx0,vPy0,vPx1,vPy1;
  viewTriple pointX,pointY,pointXY,pointXYZ;
  XGetWindowAttributes(dsply,lightingAxes,&laInfo);
  XClearWindow(dsply,lightingAxes);
  xCenter = laInfo.width / 2;
  yCenter = laInfo.height / 2;
  sinTheta = sin(-viewport->theta);
  cosTheta = cos(-viewport->theta);
  sinPhi   = sin(viewport->phi);
  cosPhi   = cos(viewport->phi);
  GSetForeground(lightingGC,(float)monoColor(buttonColor),Xoption);
  for (i=0; i < 3; i++) {
    Px0 = proj2PX(axes[i][0],axes[i][1]);
    Py0 = proj2PY(axes[i][0],axes[i][1],axes[i][2]);
    vPx0 =                    Px0 * lightScale + xCenter;
    vPy0 = laInfo.height - (Py0 * lightScale + yCenter);
    Px0 = proj2PX(axes[i][3],axes[i][4]);
    Py0 = proj2PY(axes[i][3],axes[i][4],axes[i][5]);
    vPx1 =                    Px0 * lightScale + xCenter;
    vPy1 = laInfo.height - (Py0 * lightScale + yCenter);
    GDrawLine(lightingGC,lightingAxes,vPx0,vPy0,vPx1,vPy1,Xoption);
  }
  GSetForeground(lightingGC,(float)lightingLabelColor,Xoption);
  for (i=0; i < basicScreen; i++) {
    Px0 = proj2PX(labels[i][0],labels[i][1]);
    Py0 = proj2PY(labels[i][0],labels[i][1],labels[i][2]);
    vPx0 =                    Px0 * lightScale + xCenter;
    vPy0 = laInfo.height - (Py0 * lightScale + yCenter);
    Px0 = proj2PX(labels[i][3],labels[i][4]);
    Py0 = proj2PY(labels[i][3],labels[i][4],labels[i][5]);
    vPx1 =                    Px0 * lightScale + xCenter;
    vPy1 = laInfo.height - (Py0 * lightScale + yCenter);
    GDrawLine(lightingGC,lightingAxes,vPx0,vPy0,vPx1,vPy1,Xoption);
  }
  GSetForeground(lightingGC,(float)lightingBoxColor,Xoption);
  pointX.x = tempLightPointer[0] * lightAxesScale;
  pointX.y = 0;
  pointX.z = 0;
  pointY.x = 0;
```

```
      pointY.y = tempLightPointer[1] * lightAxesScale;
      pointY.z = 0;
      pointXY.x = tempLightPointer[0] * lightAxesScale;
      pointXY.y = tempLightPointer[1] * lightAxesScale;
      pointXY.z = 0;
      pointXYZ.x = tempLightPointer[0] * lightAxesScale;
      pointXYZ.y = tempLightPointer[1] * lightAxesScale;
      pointXYZ.z = tempLightPointer[2] * lightAxesScale;
      Px0 = proj2PX(pointXY.x,pointXY.y);
      Py0 = proj2PY(pointXY.x,pointXY.y,pointXY.z);
      vPx0 =                  Px0 * lightScale + xCenter;
      vPy0 = laInfo.height - (Py0 * lightScale + yCenter);
      Px0 = proj2PX(pointX.x,pointX.y);
      Py0 = proj2PY(pointX.x,pointX.y,pointX.z);
      vPx1 =                  Px0 * lightScale + xCenter;
      vPy1 = laInfo.height - (Py0 * lightScale + yCenter);
      GDrawLine(lightingGC,lightingAxes,vPx0,vPy0,vPx1,vPy1,Xoption);
      Px0 = proj2PX(pointY.x,pointY.y);
      Py0 = proj2PY(pointY.x,pointY.y,pointY.z);
      vPx1 =                  Px0 * lightScale + xCenter;
      vPy1 = laInfo.height - (Py0 * lightScale + yCenter);
      GDrawLine(lightingGC,lightingAxes,vPx0,vPy0,vPx1,vPy1,Xoption);
      Px0 = proj2PX(pointXYZ.x,pointXYZ.y);
      Py0 = proj2PY(pointXYZ.x,pointXYZ.y,pointXYZ.z);
      vPx1 =                  Px0 * lightScale + xCenter;
      vPy1 = laInfo.height - (Py0 * lightScale + yCenter);
      GDrawLine(lightingGC,lightingAxes,vPx0,vPy0,vPx1,vPy1,Xoption);
      GSetForeground(lightingGC,(float)lightingLightColor,Xoption);
      Px0 = proj2PX(point0.x,point0.y);
      Py0 = proj2PY(point0.x,point0.y,point0.z);
      vPx0 =                  Px0 * lightScale + xCenter;
      vPy0 = laInfo.height - (Py0 * lightScale + yCenter);
      GDrawLine(lightingGC,lightingAxes,vPx0,vPy0,vPx1,vPy1,Xoption);
  }   /* drawLightingAxes */
```

7.3.26   drawLightTransArrow

⟨*view3d*⟩+≡

```
void drawLightTransArrow(void) {
  int i;
  float f;
  /*** Draw the intensity potentiometer window. ***/
  XClearArea(dsply,lightingWindow,
            (control->buttonQueue[lightTranslucent]).buttonX,
            (control->buttonQueue[lightTranslucent]).buttonY-5,
            (control->buttonQueue[lightTranslucent]).buttonWidth,
            (control->buttonQueue[lightTranslucent]).buttonHeight+10,
            False);
  GDrawLine(controlMessageGC,lightingWindow,
            (control->buttonQueue[lightTranslucent]).buttonX,
            (control->buttonQueue[lightTranslucent]).buttonY,
            (control->buttonQueue[lightTranslucent]).buttonX,
            (control->buttonQueue[lightTranslucent]).buttonY +
            (control->buttonQueue[lightTranslucent]).buttonHeight,Xoption);
  GDrawLine(controlMessageGC,lightingWindow,
            (control->buttonQueue[lightTranslucent]).buttonX + 1,
            (control->buttonQueue[lightTranslucent]).buttonY,
            (control->buttonQueue[lightTranslucent]).buttonX +
            (control->buttonQueue[lightTranslucent]).buttonWidth * 3/10,
            (control->buttonQueue[lightTranslucent]).buttonY,Xoption);
  GDrawLine(controlMessageGC,lightingWindow,
            (control->buttonQueue[lightTranslucent]).buttonX + 1,
            (control->buttonQueue[lightTranslucent]).buttonY +
            (control->buttonQueue[lightTranslucent]).yHalf/2,
            (control->buttonQueue[lightTranslucent]).buttonX +
            (control->buttonQueue[lightTranslucent]).buttonWidth * 2/10,
            (control->buttonQueue[lightTranslucent]).buttonY +
            (control->buttonQueue[lightTranslucent]).yHalf/2,Xoption);
  GDrawLine(controlMessageGC,lightingWindow,
            (control->buttonQueue[lightTranslucent]).buttonX + 1,
            (control->buttonQueue[lightTranslucent]).buttonY +
            (control->buttonQueue[lightTranslucent]).yHalf,
            (control->buttonQueue[lightTranslucent]).buttonX +
            (control->buttonQueue[lightTranslucent]).buttonWidth * 3/10,
            (control->buttonQueue[lightTranslucent]).buttonY +
            (control->buttonQueue[lightTranslucent]).yHalf,Xoption);
  GDrawLine(controlMessageGC,lightingWindow,
            (control->buttonQueue[lightTranslucent]).buttonX + 1,
            (control->buttonQueue[lightTranslucent]).buttonY +
            (control->buttonQueue[lightTranslucent]).buttonHeight*3/4,
            (control->buttonQueue[lightTranslucent]).buttonX +
```

```
                (control->buttonQueue[lightTranslucent]).buttonWidth * 2/10,
                (control->buttonQueue[lightTranslucent]).buttonY +
                (control->buttonQueue[lightTranslucent]).buttonHeight*3/4,Xoption);
    GDrawLine(controlMessageGC,lightingWindow,
                (control->buttonQueue[lightTranslucent]).buttonX + 1,
                (control->buttonQueue[lightTranslucent]).buttonY +
                (control->buttonQueue[lightTranslucent]).buttonHeight,
                (control->buttonQueue[lightTranslucent]).buttonX +
                (control->buttonQueue[lightTranslucent]).buttonWidth * 3/10,
                (control->buttonQueue[lightTranslucent]).buttonY +
                (control->buttonQueue[lightTranslucent]).buttonHeight,Xoption);
    /*** Draw the intensity selection arrow ***/
    GSetForeground(lightingGC,(float)lightingTransArrowColor,Xoption);
    f = (control->buttonQueue[lightTranslucent].buttonY +
          control->buttonQueue[lightTranslucent].buttonHeight) -
          (tempLightIntensity *
           control->buttonQueue[lightTranslucent].buttonHeight);
    i = f;
    GDrawLine(lightingGC, lightingWindow, arrowHead + 10, i,
                arrowHead + 22, i + 2, Xoption);
    GDrawLine(lightingGC, lightingWindow, arrowHead + 22, i + 2,
                arrowHead + 22, i - 2, Xoption);
    GDrawLine(lightingGC, lightingWindow, arrowHead + 22, i - 2,
                arrowHead + 10, i, Xoption);
}  /* drawLightTransArrow() */
```

7.3.27   drawLightingPanel

⟨*view3d*⟩+≡

```
void drawLightingPanel(void) {
  char *s;
  int i,strlength;
  /* Draw border lines to separate the lighting window, potentiometers,
     and button regions of the lightng subpanel. */
  GSetForeground(trashGC,(float)foregroundColor,Xoption);
  GSetLineAttributes(trashGC,3,LineSolid,CapButt,JoinMiter,Xoption);
  GDrawLine(trashGC, lightingWindow, 0,  potA, controlWidth, potA, Xoption);
  GSetLineAttributes(trashGC,2,LineSolid,CapButt,JoinMiter,Xoption);
  GDrawLine(trashGC, lightingWindow, 0, lightB, controlWidth, lightB, Xoption);
  GDrawLine(trashGC, lightingWindow, 0, lightPotA, controlWidth,
            lightPotA, Xoption);
  GDrawLine(trashGC, lightingWindow, 0, lightPotB, controlWidth,
            lightPotB, Xoption);
  GDrawLine(trashGC, lightingWindow, lightTransL, lightPotA,
            lightTransL, lightPotB, Xoption);
  writeControlTitle(lightingWindow);
  s = "Lighting Control Panel";
  strlength = strlen(s);
  GSetForeground(anotherGC,(float)lightingTitleColor,Xoption);
  GDrawString(anotherGC, lightingWindow,
              centerX(anotherGC, s, strlength, controlWidth),
              lightB+18, s, strlength, Xoption);
  for (i=lightingButtonsStart; i<(lightingButtonsEnd); i++) {
    switch (i) {
    case lightMove:
      GSetForeground(lightingGC,(float)lightingButtonColor,Xoption);
      GDraw3DButtonOut(lightingGC,lightingWindow,
                    (control->buttonQueue[i]).buttonX,
                    (control->buttonQueue[i]).buttonY,
                    (control->buttonQueue[i]).buttonWidth,
                    (control->buttonQueue[i]).buttonHeight,Xoption);
      GSetForeground(lightingGC,(float)monoColor(buttonColor),Xoption);
      GDrawLine(lightingGC,lightingWindow,
                (control->buttonQueue[i]).buttonX +
                (control->buttonQueue[i]).xHalf,
                (control->buttonQueue[i]).buttonY,
                (control->buttonQueue[i]).buttonX +
                (control->buttonQueue[i]).xHalf,
                (control->buttonQueue[i]).buttonY +
                2*(control->buttonQueue[i]).yHalf,Xoption);
      GDrawLine(lightingGC,lightingWindow,
                (control->buttonQueue[i]).buttonX,
```

```
                        (control->buttonQueue[i]).buttonY +
                        (control->buttonQueue[i]).yHalf,
                        (control->buttonQueue[i]).buttonX +
                        2*(control->buttonQueue[i]).xHalf,
                        (control->buttonQueue[i]).buttonY +
                        (control->buttonQueue[i]).yHalf,Xoption);
      break;
    case lightMoveXY:
      GSetForeground(lightingGC,(float)lightingButtonColor,Xoption);
      GDraw3DButtonOut(lightingGC,lightingWindow,
                        (control->buttonQueue[i]).buttonX,
                        (control->buttonQueue[i]).buttonY,
                        (control->buttonQueue[i]).buttonWidth,
                        (control->buttonQueue[i]).buttonHeight,Xoption);
      GSetForeground(lightingGC,(float)monoColor(buttonColor),Xoption);
      GDrawLine(lightingGC,lightingWindow,
                  (control->buttonQueue[i]).buttonX +
                  (control->buttonQueue[i]).xHalf,
                  (control->buttonQueue[i]).buttonY,
                  (control->buttonQueue[i]).buttonX +
                  (control->buttonQueue[i]).xHalf,
                  (control->buttonQueue[i]).buttonY +
                  2*(control->buttonQueue[i]).yHalf,Xoption);
      GDrawLine(lightingGC,lightingWindow,
                  (control->buttonQueue[i]).buttonX,
                  (control->buttonQueue[i]).buttonY +
                  (control->buttonQueue[i]).yHalf,
                  (control->buttonQueue[i]).buttonX +
                  2*(control->buttonQueue[i]).xHalf,
                  (control->buttonQueue[i]).buttonY +
                  (control->buttonQueue[i]).yHalf,Xoption);
      break;
    case lightMoveZ:
      GSetForeground(lightingGC,(float)lightingButtonColor,Xoption);
      GDraw3DButtonOut(lightingGC,lightingWindow,
                        (control->buttonQueue[i]).buttonX,
                        (control->buttonQueue[i]).buttonY,
                        (control->buttonQueue[i]).buttonWidth,
                        (control->buttonQueue[i]).buttonHeight,Xoption);
      GSetForeground(lightingGC,(float)monoColor(buttonColor),Xoption);
      GDrawLine(lightingGC,lightingWindow,
                  (control->buttonQueue[i]).buttonX +
                  (control->buttonQueue[i]).xHalf,
                  (control->buttonQueue[i]).buttonY,
                  (control->buttonQueue[i]).buttonX +
                  (control->buttonQueue[i]).xHalf,
```

```
                     (control->buttonQueue[i]).buttonY +
                     2*(control->buttonQueue[i]).yHalf,Xoption);
        GDrawLine(lightingGC,lightingWindow,
                     (control->buttonQueue[i]).buttonX +
                     (control->buttonQueue[i]).xHalf -
                     (control->buttonQueue[i]).xHalf/2,
                     (control->buttonQueue[i]).buttonY +
                     (control->buttonQueue[i]).yHalf,
                     (control->buttonQueue[i]).buttonX +
                     (control->buttonQueue[i]).xHalf +
                     (control->buttonQueue[i]).xHalf/2,
                     (control->buttonQueue[i]).buttonY +
                     (control->buttonQueue[i]).yHalf,Xoption);
        break;
      case lightTranslucent:
        drawLightTransArrow();
        break;
      default:
        GDraw3DButtonOut(lightingGC,lightingWindow,
                        (control->buttonQueue[i]).buttonX,
                        (control->buttonQueue[i]).buttonY,
                        (control->buttonQueue[i]).buttonWidth,
                        (control->buttonQueue[i]).buttonHeight,Xoption);
        s = (control->buttonQueue[i]).text;
        strlength = strlen(s);
        GSetForeground(trashGC,
                        (float)monoColor((control->buttonQueue[i]).textColor),
                        Xoption);
        GDrawString(trashGC, lightingWindow,
                     (control->buttonQueue[i]).buttonX +
                     centerX(processGC,s,strlength,
                             (control->buttonQueue[i]).buttonWidth),
                     (control->buttonQueue[i]).buttonY +
                     centerY(processGC,(control->buttonQueue[i]).buttonHeight),
                     s,strlen(s),Xoption);
    }  /* switch */
  }  /* for i in control->buttonQueue */
  GSetForeground(lightingGC,(float)monoColor(labelColor),Xoption);
  GDrawString(lightingGC,lightingWindow,
               control->buttonQueue[lightMoveXY].buttonX +
               control->buttonQueue[lightMoveXY].buttonWidth + 3,
               control->buttonQueue[lightMoveXY].buttonY +
               control->buttonQueue[lightMoveXY].yHalf,
               "x",1,Xoption);
  GDrawString(lightingGC,lightingWindow,
               control->buttonQueue[lightMoveXY].buttonX +
```

```
                control->buttonQueue[lightMoveXY].xHalf - 2,
                control->buttonQueue[lightMoveXY].buttonY - 4,
                "y",1,Xoption);
    GDrawString(lightingGC,lightingWindow,
                control->buttonQueue[lightMoveZ].buttonX +
                control->buttonQueue[lightMoveZ].xHalf - 2,
                control->buttonQueue[lightMoveZ].buttonY - 4,
                "z",1,Xoption);
    /** Draw the title for the intensity potentiometer. */
    GSetForeground(lightingGC,(float)lightingTransColor,Xoption);

    GDrawString(lightingGC,lightingWindow,
                control->buttonQueue[lightTranslucent].buttonX +
                control->buttonQueue[lightTranslucent].buttonWidth + 3,
                control->buttonQueue[lightTranslucent].buttonY,
                "I",1,Xoption);
    GDrawString(lightingGC,lightingWindow,
                control->buttonQueue[lightTranslucent].buttonX +
                control->buttonQueue[lightTranslucent].buttonWidth + 3,
                control->buttonQueue[lightTranslucent].buttonY +
                lightFontHeight,
                "n",1,Xoption);
    GDrawString(lightingGC,lightingWindow,
                control->buttonQueue[lightTranslucent].buttonX +
                control->buttonQueue[lightTranslucent].buttonWidth + 3,
                control->buttonQueue[lightTranslucent].buttonY +
                lightFontHeight*2,
                "t",1,Xoption);
    GDrawString(lightingGC,lightingWindow,
                control->buttonQueue[lightTranslucent].buttonX +
                control->buttonQueue[lightTranslucent].buttonWidth + 3,
                control->buttonQueue[lightTranslucent].buttonY +
                lightFontHeight*3,
                "e",1,Xoption);
    GDrawString(lightingGC,lightingWindow,
                control->buttonQueue[lightTranslucent].buttonX +
                control->buttonQueue[lightTranslucent].buttonWidth + 3,
                control->buttonQueue[lightTranslucent].buttonY +
                lightFontHeight*4,
                "n",1,Xoption);
    GDrawString(lightingGC,lightingWindow,
                control->buttonQueue[lightTranslucent].buttonX +
                control->buttonQueue[lightTranslucent].buttonWidth + 3,
                control->buttonQueue[lightTranslucent].buttonY +
                lightFontHeight*5,
                "s",1,Xoption);
```

```
GDrawString(lightingGC,lightingWindow,
            control->buttonQueue[lightTranslucent].buttonX +
            control->buttonQueue[lightTranslucent].buttonWidth + 3,
            control->buttonQueue[lightTranslucent].buttonY +
            lightFontHeight*6,
            "i",1,Xoption);
GDrawString(lightingGC,lightingWindow,
            control->buttonQueue[lightTranslucent].buttonX +
            control->buttonQueue[lightTranslucent].buttonWidth + 3,
            control->buttonQueue[lightTranslucent].buttonY +
            lightFontHeight*7,
            "t",1,Xoption);
GDrawString(lightingGC,lightingWindow,
            control->buttonQueue[lightTranslucent].buttonX +
            control->buttonQueue[lightTranslucent].buttonWidth + 3,
            control->buttonQueue[lightTranslucent].buttonY +
            lightFontHeight*8,
            "y",1,Xoption);
drawLightingAxes();
drawLightTransArrow();
} /* drawLightingPanel */
```

7.3.28   theHandler

⟨*view3d*⟩+≡
```
int theHandler(Display *display,XErrorEvent *event) {
  char buffer[512];
  XGetErrorText(display,event->error_code,buffer,511);
  fprintf(stderr,"%s\n",buffer);
  return(0);
}
```

7.3.29 mergeDatabases

⟨*view3d*⟩+≡

```
void mergeDatabases(void) {
  XrmDatabase homeDB,serverDB,applicationDB;
  char filenamebuf[1024];
  char *filename = &filenamebuf[0];
  char *classname = "Axiom";
  char name[255];
  (void) XrmInitialize();
  (void) strcpy(name, "/usr/lib/X11/app-defaults/");
  (void) strcat(name, classname);
  applicationDB = XrmGetFileDatabase(name);
  (void) XrmMergeDatabases(applicationDB, &rDB);
  if (XResourceManagerString(dsply) != NULL){
    serverDB = XrmGetStringDatabase(XResourceManagerString(dsply));
  }
  else {
    (void) strcpy(filename,getenv("HOME"));
    (void) strcat(filename,"/.Xdefaults");
    serverDB = XrmGetFileDatabase(filename);
  }
  XrmMergeDatabases(serverDB,&rDB);
  if ( getenv ("XENVIRONMENT") == NULL) {
    int len;
    (void) strcpy(filename,getenv("HOME"));
    (void) strcat(filename,"/.Xdefaults-");
    len = strlen(filename);
    (void) gethostname(filename+len,1024-len);
  }
  else {
    (void) strcpy (filename,getenv ("XENVIRONMENT"));
  }
  homeDB = XrmGetFileDatabase(filename);
  XrmMergeDatabases(homeDB,&rDB);
}
```

7.3.30   getMeshNormal

⟨*view3d*⟩+≡
```
  void getMeshNormal(float x0,float y0,float z0,float x1,float y1,float z1,
        float x2,float y2,float z2,float zMin,float zRange,float Normal[3]) {
    float Ax,Ay,Az,Bx,By,Bz, UnitFactor;
    Ax = x0-x1;   Ay = y0-y1;   Az = z0-z1;
    Bx = x2-x1;   By = y2-y1;   Bz = z2-z1;
    /* compute cross product */
    Normal[0] = (Ay*Bz - Az*By);
    Normal[1] = (Az*Bx - Ax*Bz);
    Normal[2] = (Ax*By - Ay*Bx);
    /* normalize normal vector */
    UnitFactor = sqrt(Normal[0]*Normal[0] +
                      Normal[1]*Normal[1] +
                      Normal[2]*Normal[2]);
    if (UnitFactor > 0.0) {
      Normal[0] /= UnitFactor;
      Normal[1] /= UnitFactor;
      Normal[2] /= UnitFactor;
    } else {
      Normal[0] = 0.0;
      Normal[1] = 0.0;
      Normal[2] = 0.0;
    }
  } /* getMeshNormal() */
```

7.3.31   normalizeVector

⟨*view3d*⟩+≡
```
  void normalizeVector(float *v) {
     /* v should be a triple (ignoring the rest of the array if necessary) */
    float UnitFactor;
    UnitFactor = sqrt(v[0]*v[0] + v[1]*v[1] + v[2]*v[2]);
    if (UnitFactor != 0.0) {
      v[0] /= UnitFactor;
      v[1] /= UnitFactor;
      v[2] /= UnitFactor;
    } else {
      v[0] = v[1] = v[2] = 0.0;
    }
  } /* normalizeVector() */
```

7.3.32 dotProduct

⟨*view3d*⟩+≡
```
float dotProduct(float * a,float *b,int size) {
  int i;
  float f=0;
  for (i=0; i<size; i++)
    f += (a[i]*b[i]);
  return(f);
} /* dotProduct() */
```

This file depends on the file msort.h. There, a data type called linkElement is defined. It is used here and is the main structure being sorted here. You can sort any linked structure, under any name - so long as it has a next field (see below). The define statement, below, renames linkElement to linkThing. All you need to do is change the define statement to rename your structure to linkThing. The first argument you pass to the sort routine is a pointer to the unsorted list. The function returns with that same pointer pointing to a sorted list.

```
Usage:
  linkElement *msort(p,min,max,compare)
    linkElement *L;
    int min,max;
    int (*compare)();

e.g.
  msort(L,0,N,compare);

where
  L is the list of things to be sorted,
    it is expected to be a linked list
    where the following element is pointed
    to by a field called "next"
  0 is the index of the first element
    (since this routine is called recursively,
    this field is kept for clarity; it will
    always be zero at top level)
  N the number of elements in the list         *
    minus one
  compare(X,Y) is a comparison function that
    returns a -1 if X is less than Y
               0 if X is the same as Y
           and 1 if X is greater than Y
```

7.3.33   merge

⟨*view3d*⟩+≡
```
linkThing *merge(linkThing *p, linkThing *q,
                 int (*compare)(linkThing *, linkThing *)) {
  linkThing *returnVal,*current,*pN,*qN;
  /* return if only one item - take out when insert sort implemented */
  if (!p) return(q); else if (!q) return(p);
    /* set up the head of the list (first element) */
  if (compare(p,q) <= 0) {
    returnVal = current = p;
    pN = p->next;
    qN = q;
  } else {
    returnVal = current = q;
```

```
    pN = p;
    qN = q->next;
  }
    /* merge the two lists */
  while ((pN != NULL) && (qN != NULL)) {
    if (compare(pN,qN) <= 0) {   /* pN <= qN */
      current->next = pN;
      current = pN;
      pN = pN->next;
    } else {
      current->next = qN;
      current = qN;
      qN = qN->next;
    }
  }
    /* tag on the tail end */
  if (pN == NULL) current->next = qN;
  else current->next = pN;
  return(returnVal);
} /* merge() */
```

7.3.34   msort

⟨*view3d*⟩+≡

```
  linkThing *msort(linkThing *p,int min,int max,
                   int (*compare)(linkThing *, linkThing *)) {
    int mid;
    int i;
    linkThing *q,*temp,*xxx;
    if (min == max) return p;
    else {
      mid = (min + max - 1)/2;
          /* e.g. [min,max] = [1,6] => mid=3 => q points to 4th */
      for (i=min,q=p; i<mid; i++,q=q->next);
      temp = q->next;
      q->next = 0;
      xxx = merge(msort(p,min,mid,compare),
                  msort(temp,mid+1,max,compare), compare);
      return(xxx);
    }
  } /* msort() */
```

7.3.35   getPotValue

⟨*view3d*⟩+≡

```
mouseCoord getPotValue(short eX,short eY,short xH,short yH) {
  mouseCoord whereMouse;
  float x,y;
  x = (float)eX/xH - 1;
  y = -((float)eY/yH -1);
  /* make non-linear potentiometer */
  whereMouse.x = x*x*x;
  whereMouse.y = y*y*y;
  if (whereMouse.x > 1.0)  whereMouse.x = 1.0;
  if (whereMouse.y > 1.0)  whereMouse.y = 1.0;
  if (whereMouse.x < -1.0) whereMouse.x = -1.0;
  if (whereMouse.y < -1.0) whereMouse.y = -1.0;
  return(whereMouse);
} /* getPotValue() */
```

7.3.36   getLinearPotValue

⟨*view3d*⟩+≡

```
mouseCoord getLinearPotValue(short eX,short eY,short xH,short yH) {
  mouseCoord whereMouse;
  whereMouse.x = (float)eX/xH - 1;
  whereMouse.y = -((float)eY/yH -1);
  return(whereMouse);
} /* getLinearPotValue() */
```

7.3.37 buttonAction

⟨*view3d*⟩+≡

```
void buttonAction(int bKey) {
  char *s1, *s2;
  int strL, strL1, strL2, offShade=14;
  /* Button colors which are offColor, RED, are turned off, and those which
     are onColor, GREEN, indicate the mode is in effect. */
  switch (bKey) {
  case hideControl:
    if (viewport->haveControl) {
      viewport->haveControl = no;
      XUnmapWindow(dsply,control->controlWindow);
    }
    break;
  case region3D:
    clearControlMessage();
    strcpy(control->message,viewport->title);
    writeControlMessage();
    if (viewport->regionOn) {
      viewport->regionOn = no;
      (control->buttonQueue[region3D]).textColor = offColor;
      viewData.box = 0;
      if (mono) {
        XChangeShade(dsply,offShade);
        XShadeRectangle(dsply,control->controlWindow,
                   (control->buttonQueue[region3D]).buttonX,
                   (control->buttonQueue[region3D]).buttonY,
                   (control->buttonQueue[region3D]).buttonWidth,
                   (control->buttonQueue[region3D]).buttonHeight);
        GSetForeground(globalGC1,(float)foregroundColor,Xoption);
        GDrawRectangle(globalGC1, control->controlWindow,
                       (control->buttonQueue[region3D]).buttonX,
                       (control->buttonQueue[region3D]).buttonY,
                       (control->buttonQueue[region3D]).buttonWidth,
                       (control->buttonQueue[region3D]).buttonHeight,Xoption);
      }
    } else {  /* inverted color for region off */
      viewport->regionOn = yes;
      viewData.box = 1;
      (control->buttonQueue[region3D]).textColor = onColor;
      if (mono) {
        GSetForeground(globalGC1,(float)backgroundColor,Xoption);
        XFillRectangle(dsply, control->controlWindow, globalGC1,
                   (control->buttonQueue[region3D]).buttonX,
                   (control->buttonQueue[region3D]).buttonY,
```

```
                       (control->buttonQueue[region3D]).buttonWidth,
                       (control->buttonQueue[region3D]).buttonHeight);
         GSetForeground(globalGC1,(float)foregroundColor,Xoption);
         GDrawRectangle(globalGC1, control->controlWindow,
                        (control->buttonQueue[region3D]).buttonX,
                        (control->buttonQueue[region3D]).buttonY,
                        (control->buttonQueue[region3D]).buttonWidth,
                        (control->buttonQueue[region3D]).buttonHeight,Xoption);
     }
   }
   s = (control->buttonQueue[region3D]).text;
   strL = strlen(s);
   GSetForeground(processGC,
           (float)monoColor((control->buttonQueue[region3D]).textColor),
           Xoption);
   GDrawImageString(processGC,control->controlWindow,
                       (control->buttonQueue[region3D]).buttonX +
                       centerX(processGC,s,strL,
                               (control->buttonQueue[region3D]).buttonWidth),
                       (control->buttonQueue[region3D]).buttonY +
                       centerY(processGC,
                               (control->buttonQueue[region3D]).buttonHeight),
                       s,strL,Xoption);
   redoSmooth = yes;
   drawViewport(Xoption);
   break;
case bwColor:
   clearControlMessage();
   strcpy(control->message,viewport->title);
   writeControlMessage();
   if (!mono) {
     if (viewport->monoOn) {
       viewport->monoOn = no;
       if (viewport->hueTop == viewport->hueOffset) redoColor = yes;
       else redoDither = yes;
       (control->buttonQueue[bwColor]).textColor = offColor;
       (control->buttonQueue[bwColor]).text = "BW";
     } else {
       viewport->monoOn = yes;
       maxGreyShade = XInitShades(dsply,scrn);
       if (viewport->hueTop == viewport->hueOffset) redoColor = yes;
       else redoDither = yes;
       (control->buttonQueue[bwColor]).textColor = onColor;
       (control->buttonQueue[bwColor]).text = "BW";
       GSetForeground(globalGC1,(float)backgroundColor,Xoption);
       XFillRectangle(dsply, control->controlWindow, globalGC1,
```

```
                        (control->buttonQueue[bwColor]).buttonX,
                        (control->buttonQueue[bwColor]).buttonY,
                        (control->buttonQueue[bwColor]).buttonWidth,
                        (control->buttonQueue[bwColor]).buttonHeight);
          GSetForeground(globalGC1,(float)monoColor(buttonColor),Xoption);
          GDrawRectangle(globalGC1, control->controlWindow,
                        (control->buttonQueue[bwColor]).buttonX,
                        (control->buttonQueue[bwColor]).buttonY,
                        (control->buttonQueue[bwColor]).buttonWidth,
                        (control->buttonQueue[bwColor]).buttonHeight,Xoption);
        }
        s = (control->buttonQueue[bwColor]).text;
        strL = strlen(s);
        GSetForeground(processGC,
                (float)monoColor((control->buttonQueue[bwColor]).textColor),
                Xoption);
        GDrawImageString(processGC,control->controlWindow,
                        (control->buttonQueue[bwColor]).buttonX +
                        centerX(processGC,s,strL,
                                (control->buttonQueue[bwColor]).buttonWidth),
                        (control->buttonQueue[bwColor]).buttonY +
                        centerY(processGC,
                                (control->buttonQueue[bwColor]).buttonHeight),
                        s,strL,Xoption);
        drawColorMap();
        redoSmooth = yes;
        writeTitle();
        drawViewport(Xoption);
      }
      break;
    case outlineOnOff:
      clearControlMessage();
      strcpy(control->message,viewport->title);
      writeControlMessage();
      if (viewData.outlineRenderOn) {
        viewData.outlineRenderOn = 0;
        (control->buttonQueue[outlineOnOff]).textColor = offColor;
        if (mono) {
          XChangeShade(dsply,offShade);
          XShadeRectangle(dsply,control->controlWindow,
                    (control->buttonQueue[outlineOnOff]).buttonX,
                    (control->buttonQueue[outlineOnOff]).buttonY,
                    (control->buttonQueue[outlineOnOff]).buttonWidth,
                    (control->buttonQueue[outlineOnOff]).buttonHeight);
          GSetForeground(globalGC1,(float)foregroundColor,Xoption);
          GDrawRectangle(globalGC1, control->controlWindow,
```

```
                                  (control->buttonQueue[outlineOnOff]).buttonX,
                                  (control->buttonQueue[outlineOnOff]).buttonY,
                                  (control->buttonQueue[outlineOnOff]).buttonWidth,
                                  (control->buttonQueue[outlineOnOff]).buttonHeight,
                                  Xoption);
        }
      } else {
        viewData.outlineRenderOn = 1;
        (control->buttonQueue[outlineOnOff]).textColor = onColor;
        if (mono) {
          GSetForeground(globalGC1,(float)backgroundColor,Xoption);
          XFillRectangle(dsply, control->controlWindow, globalGC1,
                    (control->buttonQueue[outlineOnOff]).buttonX,
                    (control->buttonQueue[outlineOnOff]).buttonY,
                    (control->buttonQueue[outlineOnOff]).buttonWidth,
                    (control->buttonQueue[outlineOnOff]).buttonHeight);
          GSetForeground(globalGC1,(float)foregroundColor,Xoption);
          GDrawRectangle(globalGC1, control->controlWindow,
                        (control->buttonQueue[outlineOnOff]).buttonX,
                        (control->buttonQueue[outlineOnOff]).buttonY,
                        (control->buttonQueue[outlineOnOff]).buttonWidth,
                        (control->buttonQueue[outlineOnOff]).buttonHeight,
                        Xoption);
        }
      }
      s = (control->buttonQueue[outlineOnOff]).text;
      strL = strlen(s);
      GSetForeground(processGC,
              (float)monoColor((control->buttonQueue[outlineOnOff]).textColor),
              Xoption);
      GDrawImageString(processGC,control->controlWindow,
                        (control->buttonQueue[outlineOnOff]).buttonX +
                        centerX(processGC,s,strL,
                                (control->buttonQueue[outlineOnOff]).buttonWidth),
                        (control->buttonQueue[outlineOnOff]).buttonY +
                        centerY(processGC,
                        (control->buttonQueue[outlineOnOff]).buttonHeight),
                        s,strL,Xoption);
      if (viewData.style == render) {
        drawViewport(Xoption);
      }
      break;
    case lighting:
      if (saveFlag) {
        doingPanel = CONTROLpanel;
        XUnmapWindow(dsply,saveWindow);
```

```
      }
      doingPanel = LIGHTpanel;
      tempLightPointer[0] = viewport->lightVector[0];
      tempLightPointer[1] = viewport->lightVector[1];
      tempLightPointer[2] = viewport->lightVector[2];
      tempLightIntensity = lightIntensity;
      XMapWindow(dsply,lightingWindow);
      break;
    case viewVolume:
      if (saveFlag) {
        doingPanel = CONTROLpanel;
        XUnmapWindow(dsply,saveWindow);
      }
      doingPanel = VOLUMEpanel;
      XMapWindow(dsply,volumeWindow);
      redrawView = yes;
      drawViewport(Xoption);      /* draw it with doingVolume set to yes */
      break;
    case volumeReturn:
      doingPanel = CONTROLpanel;
      redoSmooth = yes;
      redrawView = yes;
      XUnmapWindow(dsply,volumeWindow);
      break;
    case volumeAbort:
      doingPanel = CONTROLpanel;
      redrawView = yes;
      XUnmapWindow(dsply,volumeWindow);
      break;
    case lightReturn:
      doingPanel = CONTROLpanel;
      viewport->lightVector[0] = lightPointer[0] = tempLightPointer[0];
      viewport->lightVector[1] = lightPointer[1] = tempLightPointer[1];
      viewport->lightVector[2] = lightPointer[2] = tempLightPointer[2];
      lightIntensity = tempLightIntensity;
      normalizeVector(viewport->lightVector);
      redrawView = ((viewData.style == render) || (viewData.style == smooth));
      if (movingLight || changedIntensity) redoSmooth = yes;
      XUnmapWindow(dsply,lightingWindow);
      break;
    case lightAbort:
      movingLight = no;  changedIntensity = no;
      doingPanel = CONTROLpanel;
      XUnmapWindow(dsply,lightingWindow);
      break;
    case resetView:
```

```
    clearControlMessage();
    strcpy(control->message,viewport->title);
    writeControlMessage();
    viewport->axesOn       = yes;
    viewport->regionOn     = no;  viewData.box = 0;
    viewData.outlineRenderOn = 0;
    viewport->monoOn       = no;
    viewport->zoomXOn      = viewport->zoomYOn = viewport->zoomZOn = yes;
    viewport->originrOn    = yes;  viewport->objectrOn    = no;
    viewport->originFlag   = no;
    viewport->xyOn         = viewport->xzOn = viewport->yzOn = no;
    viewport->lightVector[0] = -0.5;
    viewport->lightVector[1] = 0.5;
    viewport->lightVector[2] = 0.5;
    viewport->translucency  = viewData.translucency;
    viewport->deltaX       = viewport->deltaX0;
    viewport->deltaY       = viewport->deltaY0;
    viewport->deltaY       = viewport->deltaZ0;
    viewport->scale        = viewport->scale0;
    viewport->scaleX       = viewport->scaleY = viewport->scaleZ = 1.0;
    if (!equal(viewport->theta,viewport->theta0) ||
        !equal(viewport->phi,viewport->phi0))
      rotated = yes;
    viewport->theta = viewport->axestheta = viewport->theta0 = viewData.theta;
    viewport->phi   = viewport->axesphi   = viewport->phi0   = viewData.phi;
    viewport->thetaObj = 0.0;
    viewport->phiObj   = 0.0;
    redoSmooth = yes;
    drawViewport(Xoption);
    if (viewport->haveControl) drawControlPanel();
    writeTitle();
    break;
  case axesOnOff:
    clearControlMessage();
    strcpy(control->message,viewport->title);
    writeControlMessage();
    if (viewport->axesOn) {
      viewport->axesOn = no;
      (control->buttonQueue[axesOnOff]).textColor = offColor;
      if (mono) {
        XChangeShade(dsply,offShade);
        XShadeRectangle(dsply,control->controlWindow,
                    (control->buttonQueue[axesOnOff]).buttonX,
                    (control->buttonQueue[axesOnOff]).buttonY,
                    (control->buttonQueue[axesOnOff]).buttonWidth,
                    (control->buttonQueue[axesOnOff]).buttonHeight);
```

```
              GSetForeground(globalGC1,(float)foregroundColor,Xoption);
              GDrawRectangle(globalGC1, control->controlWindow,
                            (control->buttonQueue[axesOnOff]).buttonX,
                            (control->buttonQueue[axesOnOff]).buttonY,
                            (control->buttonQueue[axesOnOff]).buttonWidth,
                            (control->buttonQueue[axesOnOff]).buttonHeight,Xoption);
          }
      } else {   /* draw invert-color button */
        viewport->axesOn = yes;
        (control->buttonQueue[axesOnOff]).textColor = onColor;
        if (mono) {
          GSetForeground(globalGC1,(float)backgroundColor,Xoption);
          XFillRectangle(dsply, control->controlWindow, globalGC1,
                        (control->buttonQueue[axesOnOff]).buttonX,
                        (control->buttonQueue[axesOnOff]).buttonY,
                        (control->buttonQueue[axesOnOff]).buttonWidth,
                        (control->buttonQueue[axesOnOff]).buttonHeight);
          GSetForeground(globalGC1,(float)foregroundColor,Xoption);
          GDrawRectangle(globalGC1, control->controlWindow,
                        (control->buttonQueue[axesOnOff]).buttonX,
                        (control->buttonQueue[axesOnOff]).buttonY,
                        (control->buttonQueue[axesOnOff]).buttonWidth,
                        (control->buttonQueue[axesOnOff]).buttonHeight,Xoption);
          }
        }
      s = (control->buttonQueue[axesOnOff]).text;
      strL = strlen(s);
      GSetForeground(processGC,
              (float)monoColor((control->buttonQueue[axesOnOff]).textColor),
              Xoption);
      GDrawImageString(processGC,control->controlWindow,
                        (control->buttonQueue[axesOnOff]).buttonX +
                        centerX(processGC,s,strL,
                                (control->buttonQueue[axesOnOff]).buttonWidth),
                        (control->buttonQueue[axesOnOff]).buttonY +
                        centerY(processGC,
                                (control->buttonQueue[axesOnOff]).buttonHeight),
                        s,strL,Xoption);
      if (viewData.style == smooth) {
        if (multiColorFlag) redoDither = yes;
        else redoColor = yes;
      }
      drawViewport(Xoption);
      break;
    case zoomx:
      if (viewport->zoomXOn) {
```

```
      viewport->zoomXOn = no;
      (control->buttonQueue[zoomx]).textColor = offColor;
      if (mono) {
        XChangeShade(dsply,offShade);
        XShadeRectangle(dsply,control->controlWindow,
                    (control->buttonQueue[zoomx]).buttonX,
                    (control->buttonQueue[zoomx]).buttonY,
                    (control->buttonQueue[zoomx]).buttonWidth,
                    (control->buttonQueue[zoomx]).buttonHeight);
        GSetForeground(globalGC1,(float)foregroundColor,Xoption);
        GDrawRectangle(globalGC1, control->controlWindow,
                        (control->buttonQueue[zoomx]).buttonX,
                        (control->buttonQueue[zoomx]).buttonY,
                        (control->buttonQueue[zoomx]).buttonWidth,
                        (control->buttonQueue[zoomx]).buttonHeight,Xoption);
      }
    } else {
      viewport->zoomXOn = yes;
      (control->buttonQueue[zoomx]).textColor = onColor;
      if (mono) {
        GSetForeground(globalGC1,(float)backgroundColor,Xoption);
        XFillRectangle(dsply, control->controlWindow, globalGC1,
                    (control->buttonQueue[zoomx]).buttonX,
                    (control->buttonQueue[zoomx]).buttonY,
                    (control->buttonQueue[zoomx]).buttonWidth,
                    (control->buttonQueue[zoomx]).buttonHeight);
        GSetForeground(globalGC1,(float)foregroundColor,Xoption);
        GDrawRectangle(globalGC1, control->controlWindow,
                        (control->buttonQueue[zoomx]).buttonX,
                        (control->buttonQueue[zoomx]).buttonY,
                        (control->buttonQueue[zoomx]).buttonWidth,
                        (control->buttonQueue[zoomx]).buttonHeight,Xoption);
      }
    }
    s = (control->buttonQueue[zoomx]).text;
    strL = strlen(s);
    GSetForeground(processGC,
          (float)monoColor((control->buttonQueue[zoomx]).textColor),Xoption);
    GDrawImageString(processGC,control->controlWindow,
                    (control->buttonQueue[zoomx]).buttonX +
                    centerX(processGC,s,strL,
                            (control->buttonQueue[zoomx]).buttonWidth),
                    (control->buttonQueue[zoomx]).buttonY +
                    centerY(processGC,
                            (control->buttonQueue[zoomx]).buttonHeight),
                    s,strL,Xoption);
```

```
        clearControlMessage();
        strcpy(control->message,viewport->title);
        writeControlMessage();
        break;
    case zoomy:
      if (viewport->zoomYOn) {
        viewport->zoomYOn = no;
        (control->buttonQueue[zoomy]).textColor = offColor;
        if (mono) {
          XChangeShade(dsply,offShade);
          XShadeRectangle(dsply,control->controlWindow,
                     (control->buttonQueue[zoomy]).buttonX,
                     (control->buttonQueue[zoomy]).buttonY,
                     (control->buttonQueue[zoomy]).buttonWidth,
                     (control->buttonQueue[zoomy]).buttonHeight);
          GSetForeground(globalGC1,(float)foregroundColor,Xoption);
          GDrawRectangle(globalGC1, control->controlWindow,
                         (control->buttonQueue[zoomy]).buttonX,
                         (control->buttonQueue[zoomy]).buttonY,
                         (control->buttonQueue[zoomy]).buttonWidth,
                         (control->buttonQueue[zoomy]).buttonHeight,Xoption);
        }
      } else {
        viewport->zoomYOn = yes;
        (control->buttonQueue[zoomy]).textColor = onColor;
        if (mono) {
          GSetForeground(globalGC1,(float)backgroundColor,Xoption);
          XFillRectangle(dsply, control->controlWindow, globalGC1,
                     (control->buttonQueue[zoomy]).buttonX,
                     (control->buttonQueue[zoomy]).buttonY,
                     (control->buttonQueue[zoomy]).buttonWidth,
                     (control->buttonQueue[zoomy]).buttonHeight);
          GSetForeground(globalGC1,(float)foregroundColor,Xoption);
          GDrawRectangle(globalGC1, control->controlWindow,
                         (control->buttonQueue[zoomy]).buttonX,
                         (control->buttonQueue[zoomy]).buttonY,
                         (control->buttonQueue[zoomy]).buttonWidth,
                         (control->buttonQueue[zoomy]).buttonHeight,Xoption);
        }
      }
      s = (control->buttonQueue[zoomy]).text;
      strL = strlen(s);
      GSetForeground(processGC,
              (float)monoColor((control->buttonQueue[zoomy]).textColor),Xoption);
      GDrawImageString(processGC,control->controlWindow,
                       (control->buttonQueue[zoomy]).buttonX +
```

```
                            centerX(processGC,s,strL,
                                    (control->buttonQueue[zoomy]).buttonWidth),
                            (control->buttonQueue[zoomy]).buttonY +
                            centerY(processGC,
                                    (control->buttonQueue[zoomy]).buttonHeight),
                            s,strL,Xoption);
        clearControlMessage();
        strcpy(control->message,viewport->title);
        writeControlMessage();
        break;
    case zoomz:
        if (viewport->zoomZOn) {
          viewport->zoomZOn = no;
          (control->buttonQueue[zoomz]).textColor = offColor;
          if (mono) {
            XChangeShade(dsply,offShade);
            XShadeRectangle(dsply,control->controlWindow,
                        (control->buttonQueue[zoomz]).buttonX,
                        (control->buttonQueue[zoomz]).buttonY,
                        (control->buttonQueue[zoomz]).buttonWidth,
                        (control->buttonQueue[zoomz]).buttonHeight);
            GSetForeground(globalGC1,(float)foregroundColor,Xoption);
            GDrawRectangle(globalGC1, control->controlWindow,
                            (control->buttonQueue[zoomz]).buttonX,
                            (control->buttonQueue[zoomz]).buttonY,
                            (control->buttonQueue[zoomz]).buttonWidth,
                            (control->buttonQueue[zoomz]).buttonHeight,Xoption);
          }
        } else {
          viewport->zoomZOn = yes;
          (control->buttonQueue[zoomz]).textColor = onColor;
          if (mono) {
            GSetForeground(globalGC1,(float)backgroundColor,Xoption);
            XFillRectangle(dsply, control->controlWindow, globalGC1,
                        (control->buttonQueue[zoomz]).buttonX,
                        (control->buttonQueue[zoomz]).buttonY,
                        (control->buttonQueue[zoomz]).buttonWidth,
                        (control->buttonQueue[zoomz]).buttonHeight);
            GSetForeground(globalGC1,(float)foregroundColor,Xoption);
            GDrawRectangle(globalGC1, control->controlWindow,
                            (control->buttonQueue[zoomz]).buttonX,
                            (control->buttonQueue[zoomz]).buttonY,
                            (control->buttonQueue[zoomz]).buttonWidth,
                            (control->buttonQueue[zoomz]).buttonHeight,Xoption);
          }
        }
```

```
      s = (control->buttonQueue[zoomz]).text;
      strL = strlen(s);
      GSetForeground(processGC,
              (float)monoColor((control->buttonQueue[zoomz]).textColor),Xoption);
      GDrawImageString(processGC,control->controlWindow,
                      (control->buttonQueue[zoomz]).buttonX +
                      centerX(processGC,s,strL,
                             (control->buttonQueue[zoomz]).buttonWidth),
                      (control->buttonQueue[zoomz]).buttonY +
                      centerY(processGC,
                             (control->buttonQueue[zoomz]).buttonHeight),
                      s,strL,Xoption);
      clearControlMessage();
      strcpy(control->message,viewport->title);
      writeControlMessage();
      break;
    case originr:
      viewport->originrOn = yes;
      (control->buttonQueue[originr]).textColor = onColor;
      viewport->objectrOn = no;
      (control->buttonQueue[objectr]).textColor = offColor;
      viewport->originFlag = yes;
      if (mono) {
        XChangeShade(dsply,offShade);
        XShadeRectangle(dsply,control->controlWindow,
                    (control->buttonQueue[objectr]).buttonX,
                    (control->buttonQueue[objectr]).buttonY,
                    (control->buttonQueue[objectr]).buttonWidth,
                    (control->buttonQueue[objectr]).buttonHeight);
        GSetForeground(globalGC1,(float)foregroundColor,Xoption);
        GDrawRectangle(globalGC1, control->controlWindow,
                      (control->buttonQueue[objectr]).buttonX,
                      (control->buttonQueue[objectr]).buttonY,
                      (control->buttonQueue[objectr]).buttonWidth,
                      (control->buttonQueue[objectr]).buttonHeight,Xoption);
        GSetForeground(globalGC1,(float)backgroundColor,Xoption);
        XFillRectangle(dsply, control->controlWindow, globalGC1,
                    (control->buttonQueue[originr]).buttonX,
                    (control->buttonQueue[originr]).buttonY,
                    (control->buttonQueue[originr]).buttonWidth,
                    (control->buttonQueue[originr]).buttonHeight);
        GSetForeground(globalGC1,(float)foregroundColor,Xoption);
        GDrawRectangle(globalGC1, control->controlWindow,
                      (control->buttonQueue[originr]).buttonX,
                      (control->buttonQueue[originr]).buttonY,
                      (control->buttonQueue[originr]).buttonWidth,
```

```
                              (control->buttonQueue[originr]).buttonHeight,Xoption);
          }
          s1 = (control->buttonQueue[objectr]).text;
          strL1 = strlen(s1);
          s2 = (control->buttonQueue[originr]).text;
          strL2 = strlen(s2);
          GSetForeground(processGC,
                  (float)monoColor((control->buttonQueue[objectr]).textColor),
                  Xoption);
          GDrawImageString(processGC,control->controlWindow,
                            (control->buttonQueue[objectr]).buttonX +
                            centerX(processGC,s1,strL1,
                                    (control->buttonQueue[objectr]).buttonWidth),
                            (control->buttonQueue[objectr]).buttonY +
                            centerY(processGC,
                                    (control->buttonQueue[objectr]).buttonHeight),
                            s1,strL1,Xoption);
          GSetForeground(processGC,
                  (float)monoColor((control->buttonQueue[originr]).textColor),
                  Xoption);
          GDrawImageString(processGC,control->controlWindow,
                            (control->buttonQueue[originr]).buttonX +
                            centerX(processGC,s2,strL2,
                                    (control->buttonQueue[originr]).buttonWidth),
                            (control->buttonQueue[originr]).buttonY +
                            centerY(processGC,
                                    (control->buttonQueue[originr]).buttonHeight),
                            s2,strL2,Xoption);
          clearControlMessage();
          strcpy(control->message,viewport->title);
          writeControlMessage();
          break;
        case objectr:
          viewport->objectrOn = yes;
          (control->buttonQueue[objectr]).textColor = onColor;
          viewport->originrOn = no;
          (control->buttonQueue[originr]).textColor = offColor;
          viewport->originFlag = no;
          if (mono) {
            XChangeShade(dsply,offShade);
            XShadeRectangle(dsply,control->controlWindow,
                        (control->buttonQueue[originr]).buttonX,
                        (control->buttonQueue[originr]).buttonY,
                        (control->buttonQueue[originr]).buttonWidth,
                        (control->buttonQueue[originr]).buttonHeight);
            GSetForeground(globalGC1,(float)foregroundColor,Xoption);
```

```
              GDrawRectangle(globalGC1, control->controlWindow,
                          (control->buttonQueue[originr]).buttonX,
                          (control->buttonQueue[originr]).buttonY,
                          (control->buttonQueue[originr]).buttonWidth,
                          (control->buttonQueue[originr]).buttonHeight,Xoption);
          GSetForeground(globalGC1,(float)backgroundColor,Xoption);
          XFillRectangle(dsply, control->controlWindow, globalGC1,
                      (control->buttonQueue[objectr]).buttonX,
                      (control->buttonQueue[objectr]).buttonY,
                      (control->buttonQueue[objectr]).buttonWidth,
                      (control->buttonQueue[objectr]).buttonHeight);
          GSetForeground(globalGC1,(float)foregroundColor,Xoption);
          GDrawRectangle(globalGC1, control->controlWindow,
                          (control->buttonQueue[objectr]).buttonX,
                          (control->buttonQueue[objectr]).buttonY,
                          (control->buttonQueue[objectr]).buttonWidth,
                          (control->buttonQueue[objectr]).buttonHeight,Xoption);
      }
      s1 = (control->buttonQueue[objectr]).text;
      strL1 = strlen(s1);
      s2 = (control->buttonQueue[originr]).text;
      strL2 = strlen(s2);

      GSetForeground(processGC,
              (float)monoColor((control->buttonQueue[objectr]).textColor),
              Xoption);
      GDrawImageString(processGC,control->controlWindow,
                          (control->buttonQueue[objectr]).buttonX +
                          centerX(processGC,s1,strL1,
                                  (control->buttonQueue[objectr]).buttonWidth),
                          (control->buttonQueue[objectr]).buttonY +
                          centerY(processGC,
                                  (control->buttonQueue[objectr]).buttonHeight),
                          s1,strL1,Xoption);
      GSetForeground(processGC,
              (float)monoColor((control->buttonQueue[originr]).textColor),
              Xoption);
      GDrawImageString(processGC,control->controlWindow,
                          (control->buttonQueue[originr]).buttonX +
                          centerX(processGC,s2,strL2,
                                  (control->buttonQueue[originr]).buttonWidth),
                          (control->buttonQueue[originr]).buttonY +
                          centerY(processGC,
                                  (control->buttonQueue[originr]).buttonHeight),
                          s2,strL2,Xoption);
      clearControlMessage();
```

```
        strcpy(control->message,viewport->title);
        writeControlMessage();
        break;
    case ps:
        strcpy(control->message,"   Creating postscript file ...    ");
        writeControlMessage();
        if (PSInit(viewport->viewWindow, viewport->titleWindow) == psError) {
            strcpy(control->message,"    Aborted: PSInit error.    ");
            writeControlMessage();
            return;   /* make new tmpnam for new file */
        }
        redoSmooth = yes;
        drawViewport(PSoption);      /* draw picture in PS; create ps script file */

        if (PSCreateFile(viewBorderWidth, viewport->viewWindow,
             viewport->titleWindow, viewport->title) == psError) {
            strcpy(control->message,"    Aborted: PSCreateFile error.    ");
            writeControlMessage();
            return;
        }
        clearControlMessage();
        strcpy(control->message,PSfilename);
        strcat(control->message," in working dir ");
        writeControlMessage();
        break;
    case pixmap:
        strcpy(control->message,"   Creating axiom3D.xpm now ...    ");
        writeControlMessage();
            XGetWindowAttributes(dsply,viewport->viewWindow,&vwInfo);
            write_pixmap_file(dsply,scrn,"axiom3D.xpm",
                              viewport->titleWindow,0,0,vwInfo.width,
                              vwInfo.height+titleHeight);
        clearControlMessage();
        strcpy(control->message,"   axiom3D.xpm in working dir     ");
        writeControlMessage();
        break;
    case transparent:
    case opaqueMesh:
    case render:
    case smooth:
        clearControlMessage();
        strcpy(control->message,viewport->title);
        writeControlMessage();
        viewData.style = bKey;
        drawViewport(Xoption);  /* draw picture in viewWindow with X routines */
        break;
```

```
      case closeAll:
        clearControlMessage();
        strcpy(control->message,viewport->title);
        writeControlMessage();
        doingPanel = QUITpanel;
        viewport->closing = yes;
        XMapWindow(dsply,quitWindow);
        break;
      case quitReturn:
        XUnmapWindow(dsply,quitWindow);
        break;
      case quitAbort:
        doingPanel = CONTROLpanel;
        XUnmapWindow(dsply,quitWindow);
        break;
      case saveit:
        clearControlMessage();
        strcpy(control->message,viewport->title);
        writeControlMessage();
        saveFlag = yes;
        doingPanel = SAVEpanel;
        XMapWindow(dsply,saveWindow);
        break;
      case saveExit:
        saveFlag = no;
        doingPanel = CONTROLpanel;
        XUnmapWindow(dsply,saveWindow);
        break;
      case xy:
        viewport->theta = pi;
        viewport->phi   = 0.0;
        viewport->axestheta = pi;
        viewport->axesphi   = 0.0;
        rotated = yes;
        viewport->yzOn = viewport->xzOn = no;
        viewport->xyOn = yes;
        drawViewport(Xoption);
        break;
      case xz:
        viewport->theta = pi;
        viewport->phi   = -pi_half;
        viewport->axestheta = pi;
        viewport->axesphi   = -pi_half;
        rotated = yes;
        viewport->yzOn = viewport->xyOn = no;
        viewport->xzOn = yes;
```

```
        drawViewport(Xoption);
        break;
      case yz:
        viewport->theta = pi_half;
        viewport->phi   = -pi_half;
        viewport->axestheta = pi_half;
        viewport->axesphi   = -pi_half;
        rotated = yes;
        viewport->xzOn = viewport->xyOn = no;
        viewport->yzOn = yes;
        drawViewport(Xoption);
        break;
      default:
        fprintf(stderr,"Received a non-functioning button request: %d \n",bKey);
        break;
    } /* switch (action) */
  } /* processEvents() */
```

7.3.38   processEvents

X Event Processing

⟨*view3d*⟩+≡

```
void processEvents(void) {
  XEvent                *event, tempEvent;
  Window                whichWindow;
  buttonStruct          *controlButton;
  mouseCoord            mouseXY = {0.0,0.0};
  mouseCoord            linearMouseXY= {0.0,0.0};
  int                   someInt, mouseW4, mouseH4;
  int                   toggleReady =yes;
  int                   checkButton = no;
  int                   first_time = yes;
  int                   changingColor = yes;
  int                   gotEvent = 0, exposeView = no;
  int                   tempTW, tempTH, tempVW, tempVH;
  int                   buttonTablePtr;
  float                 f1, f2;
  int                   px, py, lx, ly;
  unsigned int          lbuttons;
  Window                dummy;
  int                   Xcon,externalControl,len;
  fd_set                rd;
  externalControl = 0;
  Xcon = ConnectionNumber(dsply);
  /** assign lightPointer for light panel **/
  lightPointer[0] = tempLightPointer[0] = viewport->lightVector[0];
  lightPointer[1] = tempLightPointer[1] = viewport->lightVector[1];
  lightPointer[2] = tempLightPointer[2] = viewport->lightVector[2];
  if (!(event = (XEvent *)saymem("process.c",1,sizeof(XEvent)))) {
    fprintf(stderr,"Ran out of memory initializing event processing.\n");
    exitWithAck(RootWindow(dsply,scrn),Window,-1);
  }
  controlButton = 0;
  while(1) {
    /* Store old viewport window size attributes for resizing comparison. */
    XGetWindowAttributes(dsply,viewport->titleWindow,&graphWindowAttrib);
    tempTW = graphWindowAttrib.width;
    tempTH = graphWindowAttrib.height;
    XGetWindowAttributes(dsply,viewport->viewWindow,&graphWindowAttrib);
    tempVW = graphWindowAttrib.width;
    tempVH = graphWindowAttrib.height;
    /* Get the next X event. The check for pending events is so that
       a held down mouse button is interpreted as an event
```

```
    even if nothing is pending. */
len=0;
while(len<=0) {
  FD_ZERO(&rd);
  if (externalControl==0) FD_SET(0, &rd);
  FD_SET(Xcon,&rd);

  if (XEventsQueued(dsply, QueuedAlready)) {
    len=1;
    break;
  }
  if (!followMouse)
    len=select(FD_SETSIZE,(void *)&rd,0,0,0);
  else
    len=1;
}
if (FD_ISSET(Xcon,&rd)||
    XEventsQueued(dsply, QueuedAfterFlush) ||
    followMouse) {

  if (followMouse) {
    if (XPending(dsply))
      XNextEvent(dsply,event);
    gotEvent++;
  } else {
    XNextEvent(dsply,event);
    gotEvent++;
  }
  if (gotToggle || !followMouse)
    checkButton = no;
  if (gotEvent) {
    whichWindow = ((XButtonEvent *)event)->window;
    first_time = no;

    switch(((XEvent *)event)->type) {
    case ClientMessage:
      if (event->xclient.data.l[0] == wm_delete_window) {
        goodbye(-1);
      }
      else {
        fprintf(stderr,"Unknown Client Message ...\n");
      }
      break;
    case Expose:
      if (whichWindow == viewport->titleWindow) {
        exposeView = yes;
```

```
            followMouse = no;
            XSync(dsply,0);
            /* get rid of redundant exposure events */
            XCheckWindowEvent(dsply,viewport->titleWindow,
                              ExposureMask,&tempEvent);
            writeTitle();
            XGetWindowAttributes(dsply,viewport->titleWindow,
                                 &graphWindowAttrib);
            if ((graphWindowAttrib.width!=tempTW) ||
                ((graphWindowAttrib.height)!=tempTH)) {
              XResizeWindow(dsply,viewport->viewWindow,
                            graphWindowAttrib.width, graphWindowAttrib.height);
              redoSmooth = yes;  /* recompute smooth image pixmap if resized */
            }
        } else if (whichWindow == viewport->viewWindow) {
          exposeView = yes;
          followMouse = no;
          XSync(dsply,0);
          XCheckWindowEvent(dsply,viewport->viewWindow,ExposureMask,
                            &tempEvent);
          XGetWindowAttributes(dsply,viewport->viewWindow,&graphWindowAttrib);
          if ((graphWindowAttrib.width!=tempVW) ||
              ((graphWindowAttrib.height)!=tempVH)) {
            XResizeWindow(dsply,viewport->viewWindow,graphWindowAttrib.width,
                          graphWindowAttrib.height);
            redoSmooth = yes;  /* recompute smooth image pixmap if resized */
          }
          drawViewport(Xoption);
          XMapWindow(dsply,whichWindow);
        } else if (whichWindow == lightingWindow) {
          XGetWindowAttributes(dsply,control->controlWindow,
                               &graphWindowAttrib);
          /* do not allow resizing of control panel */
          if ((graphWindowAttrib.width!=controlWidth) ||
              (graphWindowAttrib.height!=controlHeight)) {
            XResizeWindow(dsply,control->controlWindow,controlWidth,
                          controlHeight);
          }
          drawLightingPanel();
        } else if (whichWindow == volumeWindow) {
          XGetWindowAttributes(dsply,control->controlWindow,
                               &graphWindowAttrib);
          /* do not allow resizing of control panel */
          if ((graphWindowAttrib.width!=controlWidth) ||
              (graphWindowAttrib.height!=controlHeight)) {
            XResizeWindow(dsply,control->controlWindow,controlWidth,
```

```
                               controlHeight);
      }
      drawVolumePanel();
      if (redrawView) {
        redrawView = no;
        drawViewport(Xoption);
      }
    } else if (whichWindow == quitWindow) {
      XGetWindowAttributes(dsply,control->controlWindow,
                           &graphWindowAttrib);
      /* do not allow resizing of control panel */
      if ((graphWindowAttrib.width!=controlWidth) ||
          (graphWindowAttrib.height!=controlHeight)) {
        XResizeWindow(dsply,control->controlWindow,controlWidth,
                      controlHeight);
      }
      drawQuitPanel();
    } else if (whichWindow == saveWindow) {
      XGetWindowAttributes(dsply,control->controlWindow,
                           &graphWindowAttrib);
      /* do not allow resizing of control panel */
      if ((graphWindowAttrib.width!=controlWidth) ||
          (graphWindowAttrib.height!=controlHeight)) {
        XResizeWindow(dsply,control->controlWindow,controlWidth,
                      controlHeight);
      }
      drawSavePanel();
    } else if (whichWindow == control->controlWindow) {
      XGetWindowAttributes(dsply,control->controlWindow,
                           &graphWindowAttrib);
      /* do not allow resizing of control panel */
      if ((graphWindowAttrib.width != controlWidth) ||
          (graphWindowAttrib.height != controlHeight)) {
        XResizeWindow(dsply,control->controlWindow,
                      controlWidth,controlHeight);
      }
      if (viewport->haveControl) drawControlPanel();
      followMouse = no;
      if (redrawView || exposeView) {
        redrawView = no;
        drawViewport(Xoption);
      }
      exposeView = no;
    } else {
      fprintf(stderr,"Not a valid window.\n");
    }
```

```
                     XFlush(dsply);
                     while(XCheckTypedWindowEvent(dsply, whichWindow, Expose, &tempEvent));
                     break;
                 case MotionNotify:
                     exposeView = no;
                     if (followMouse) {
                       if (whichWindow == control->colormapWindow) {
                         while (XCheckMaskEvent(dsply,ButtonMotionMask,event));
                         first_time = checkButton = followMouse = changingColor = yes;
                         gotToggle = no;
                       } else if (whichWindow != control->controlWindow) {
                         if (controlButton->pot) {
                           while (XCheckMaskEvent(dsply,ButtonMotionMask,event));
                           mouseXY = getPotValue(((XButtonEvent *)event)->x,
                                                 ((XButtonEvent *)event)->y,
                                                 controlButton->xHalf,
                                                 controlButton->yHalf);
                           linearMouseXY = getLinearPotValue(((XButtonEvent *)event)->x,
                                                             ((XButtonEvent *)event)->y,
                                                             controlButton->xHalf,
                                                             controlButton->yHalf);
                           first_time = checkButton = followMouse = yes;
                           gotToggle = no;
                         }
                       }
                     }
                     break;
                 case ButtonRelease:
                     exposeView = followMouse = no;
                     toggleReady = yes;  gotToggle = yes;
                     break;
                 case LeaveNotify:
                     XQueryPointer(dsply,rtWindow,&dummy,&dummy,&px,&py,&lx,&ly,&lbuttons);
                     if ( (controlButton) &&
                          ((whichWindow == control->colormapWindow) ||
                           (controlButton->pot)) &&
                          (lbuttons & Button1Mask ||
                           lbuttons & Button2Mask ||
                           lbuttons & Button3Mask)) {
                       followMouse = yes;
                       if (whichWindow == control->colormapWindow)
                         changingColor = yes;
                     }
                     else {
                       followMouse = no;
                       changingColor = no;
```

```
          }
          toggleReady = yes;
          checkButton = exposeView = no;
          break;
        case ButtonPress:
          exposeView = no;  changingColor = no;
          if (whichWindow == viewport->viewWindow) {
            /* find out where the mouse button is pressed on the viewport,
               this determines where to put the control panel */
            XGetWindowAttributes(dsply,whichWindow,&graphWindowAttrib);
            mouseW4 = graphWindowAttrib.width/4;
            if (((XButtonEvent *)event)->x > (graphWindowAttrib.width-mouseW4))
              someInt = 1;
            else {
              mouseH4 = graphWindowAttrib.height/4;
              if (((XButtonEvent *)event)->y >
                  (graphWindowAttrib.height - mouseH4)) someInt = 2;
              else if (((XButtonEvent *)event)->x < mouseW4) someInt = 3;
              else if (((XButtonEvent *)event)->y < mouseH4) someInt = 4;
              else someInt = 0;
            }
            if (viewport->haveControl) {
              XUnmapWindow(dsply,control->controlWindow);
            }
            putControlPanelSomewhere(someInt);
            writeControlMessage();
            XSync(dsply,0);
          } else if (whichWindow == control->colormapWindow) {
            gotToggle = no;
            first_time = checkButton = followMouse = changingColor = yes;
          } else if (whichWindow != control->controlWindow) {
            /* mouse clicked on one of the buttons */
            if (!controlButton || (controlButton->self != whichWindow)) {
              buttonTablePtr = *((int *)XLookUpAssoc(dsply,table,whichWindow));
              /** lighting buttons have indices greater than 100 **/
              /** all buttons share the same array now **/
              controlButton = &(control->buttonQueue[buttonTablePtr]);
            }
            if (controlButton->pot) {
              /* figure out [x,y] for this button in the range [-1..1,-1..1] */
              mouseXY = getPotValue(((XButtonEvent *)event)->x,
                                    ((XButtonEvent *)event)->y,
                                    controlButton->xHalf,controlButton->yHalf);
              linearMouseXY = getLinearPotValue(((XButtonEvent *)event)->x,
                                                ((XButtonEvent *)event)->y,
                                                controlButton->xHalf,
```

```
                                                controlButton->yHalf);
          followMouse = yes;
          gotToggle = no;
        } else {
          followMouse = no;
          gotToggle = yes;    /*auto-repeat of toggle buttons not allowed*/
          if (toggleReady) toggleReady = no;
        }
        checkButton = yes;
        first_time  = yes;
      }
      break;
    default:
      toggleReady = gotToggle = yes;
      exposeView = changingColor = checkButton = followMouse = no;
      break;
    } /* switch */
    gotEvent--;
  }  /* if gotEvent */
  /* Allow a pressed mouse button on a potentiometer to poll repeatedly. */
  if (followMouse && !first_time && (followMouse++ > mouseWait)) {
    /* reset for next timing loop */
    followMouse = yes;
    checkButton = yes;
  }
  if (checkButton) {
    if (viewport->closing && (controlButton->buttonKey == quitReturn)) {
      goodbye(-1);
    } else if (changingColor) {
      viewport->closing = no;
      /* moving top color map pointer */
      if (((XButtonEvent *)event)->y < colorOffsetY) {
        if (((XButtonEvent *)event)->x < (colorOffset+colorWidth)) {
          /* decreasing top hue number */
          if (viewport->hueTop > 0) viewport->hueTop--;
        } else if (((XButtonEvent *)event)->x >=
                    (colorOffsetX + totalHues*colorWidth + colorWidth)) {
          if (viewport->hueTop < totalHues) viewport->hueTop++;
        } else {
          viewport->hueTop =
            (((XButtonEvent *)event)->x -
            colorOffsetX + colorWidth/2 - 13) / colorWidth;
        }
      } else if (((XButtonEvent *)event)->y >
                  (colorOffsetY + colorHeight)) {
        /* moving bottom color map pointer */
```

```
          if ((((XButtonEvent *)event)->x < (colorOffset+colorWidth)) {
            /* decreasing offset number */
            if (viewport->hueOffset > 0) viewport->hueOffset--;
          } else if ((((XButtonEvent *)event)->x >=
                        (colorOffsetX + totalHues*colorWidth + colorWidth)) {
            if (viewport->hueOffset < totalHues) viewport->hueOffset++;
          } else {
            viewport->hueOffset =
              (((XButtonEvent *)event)->x -
               colorOffsetX + colorWidth/2 - 13) / colorWidth;
          }
        }
        /* color map pointer does not wrap around */
        if (viewport->hueOffset < 0) viewport->hueOffset = 0;
        if (viewport->hueTop < 0) viewport->hueTop = 0;
        if (viewport->hueOffset >= totalHues)
          viewport->hueOffset = totalHues-1;
        if (viewport->hueTop >= totalHues) viewport->hueTop = totalHues-1;
        viewport->numberOfHues = viewport->hueTop - viewport->hueOffset;
        if ((viewport->hueTop == viewport->hueOffset) && !viewport->monoOn) {
          redoColor = yes;
          redoDither = no;
        } else {
          redoColor = no;
          redoDither = yes;
        }
        /* update color map changes on control panel */
        drawColorMap();
      } else {
        viewport->closing = no;
        clearControlMessage();
        /* reset all the things that might affect a recalculation for
           redrawing removing hidden surfaces */
        /* determine what type of button has been pressed */
        switch(controlButton->buttonKey) {
          /*** Potentiometers ***/
          case rotate:
            if (!((viewport->originrOn) && (viewport->objectrOn))) {
              /* update the amount of rotation around the object center
                 of volume */
              if (viewport->objectrOn) {
                viewport->thetaObj += mouseXY.x * rotateFactor;
                viewport->phiObj  -= mouseXY.y * rotateFactor;
                while (viewport->thetaObj >= two_pi) {
                  viewport->thetaObj -= two_pi;
                }
```

```
                        while (viewport->thetaObj < 0.0) {
                          viewport->thetaObj += two_pi;
                        }
                        while (viewport->phiObj > pi) {
                          viewport->phiObj -= two_pi;
                        }
                        while (viewport->phiObj <= -pi) {
                          viewport->phiObj += two_pi;
                        }
                      }
                      /* update amount of rotation around the world space origin */
                      if (viewport->originrOn) {
                        viewport->theta += mouseXY.x * rotateFactor;
                        viewport->phi   -= mouseXY.y * rotateFactor;
                        while (viewport->theta >= two_pi) {
                          viewport->theta -= two_pi;
                        }
                        while (viewport->theta < 0.0) {
                          viewport->theta += two_pi;
                        }
                        while (viewport->phi > pi) {
                          viewport->phi -= two_pi;
                        }
                        while (viewport->phi <= -pi) {
                          viewport->phi += two_pi;
                        }
                        viewport->axestheta += mouseXY.x * rotateFactor;
                        viewport->axesphi   -= mouseXY.y * rotateFactor;
                        while (viewport->axestheta >= two_pi) {
                          viewport->axestheta -= two_pi;
                        }
                        while (viewport->axestheta < 0.0) {
                          viewport->axestheta += two_pi;
                        }
                        while (viewport->axesphi > pi) {
                          viewport->axesphi -= two_pi;
                        }
                        while (viewport->axesphi <= -pi) {
                          viewport->axesphi += two_pi;
                        }
                      }
                      rotated = yes;
                      viewport->yzOn = viewport->xzOn = viewport->xyOn = no;
                      clearControlMessage();
                      strcpy(control->message,viewport->title);
                      writeControlMessage();
```

```
              drawViewport(Xoption);
            }
            break;
          case zoom:
            /* if uniform scaling */
            if ((viewport->zoomXOn) &&
                (viewport->zoomYOn) &&
                (viewport->zoomZOn)) {
              viewport->scale *= 1 - mouseXY.y * scaleFactor;
            } else { /* else scale axes independently */
              if (viewport->zoomXOn) viewport->scaleX *= (1 - mouseXY.y);
              if (viewport->zoomYOn) viewport->scaleY *= (1 - mouseXY.y);
              if (viewport->zoomZOn) viewport->scaleZ *= (1 - mouseXY.y);
            }
            if (viewport->scale > maxScale) viewport->scale = maxScale;
            else if (viewport->scale < minScale) viewport->scale = minScale;
            if (viewport->scaleX > maxScale) viewport->scaleX = maxScale;
            else if (viewport->scaleX < minScale) viewport->scaleX = minScale;
            if (viewport->scaleY > maxScale) viewport->scaleY = maxScale;
            else if (viewport->scaleY < minScale) viewport->scaleY = minScale;
            if (viewport->scaleZ > maxScale) viewport->scaleZ = maxScale;
            else if (viewport->scaleZ < minScale) viewport->scaleZ = minScale;
            zoomed = yes;
            clearControlMessage();
            strcpy(control->message,viewport->title);
            writeControlMessage();
            if ((viewport->zoomXOn) ||
                (viewport->zoomYOn) ||
                (viewport->zoomZOn))
              drawViewport(Xoption);
            break;
          case translate:
            viewport->deltaX += mouseXY.x * translateFactor;
            viewport->deltaY += mouseXY.y * translateFactor;
            if (viewport->deltaX > maxDeltaX)
                    viewport->deltaX = maxDeltaX;
            else if (viewport->deltaX < -maxDeltaX)
                    viewport->deltaX = -maxDeltaX;
            if (viewport->deltaY > maxDeltaY)
                    viewport->deltaY = maxDeltaY;
            else if (viewport->deltaY < -maxDeltaY)
                    viewport->deltaY = -maxDeltaY;
            translated = yes;
            clearControlMessage();
            strcpy(control->message,viewport->title);
            writeControlMessage();
```

```
                      drawViewport(Xoption);
                      break;
                      /*** Lighting panel ***/
                  case lightMoveXY:
                      tempLightPointer[0] = linearMouseXY.x;
                      tempLightPointer[1] = linearMouseXY.y;
                      if (tempLightPointer[0] > 1) tempLightPointer[0] = 1;
                      else if (tempLightPointer[0] < -1) tempLightPointer[0] = -1;
                      if (tempLightPointer[1] > 1) tempLightPointer[1] = 1;
                      else if (tempLightPointer[1] < -1) tempLightPointer[1] = -1;
                      movingLight = yes;
                      drawLightingAxes();
                      break;
                  case lightMoveZ:
                      tempLightPointer[2] = linearMouseXY.y;
                      /* linearMouse => no checking necessary */
                      if (tempLightPointer[2] > 1) tempLightPointer[2] = 1;
                      else if (tempLightPointer[2] < -1) tempLightPointer[2] = -1;
                      movingLight = yes;
                      drawLightingAxes();
                      break;
                      /* changes the light intensity */
                  case lightTranslucent:
                      tempLightIntensity = (linearMouseXY.y+1)/2;
                      if (tempLightIntensity > 1) tempLightIntensity = 1;
                      else if (tempLightIntensity < 0) tempLightIntensity = 0;
                      changedIntensity = yes;
                      drawLightTransArrow();
                      break;
                      /*** volume panel ***/
                  case frustrumBut:
                      screenX = ((XButtonEvent *)event)->x;
                      if inside(eyeMinX,eyeMaxX) {
                        /* object coordinate */
                        f2 = mouseXY.x * (maxEyeDistance - minEyeDistance) +
                          minEyeDistance;
                        if (f2 != viewData.eyeDistance) {
                          doingVolume = 2;   /* flag for using screenX */
                          changedEyeDistance = yes;
                          viewData.eyeDistance = f2;
                          drawFrustrum();
                          drawViewport(Xoption);
                        }
                      }
                      else if inside(hitherMinX,hitherMaxX) {
                        f1 = ((float)hitherMaxX - ((XButtonEvent *)event)->x) /
```

```
                    (hitherMaxX - hitherMinX);
                  /* object coordinate */
                  f2 = f1 * (clipPlaneMax - clipPlaneMin) + clipPlaneMin;
                  if (f2 != viewData.clipPlane) {
                    doingVolume = 3;   /* flag for using screenX */
                    viewData.clipPlane = f2;
                    drawFrustrum();
                    drawViewport(Xoption);
                  }
                }
                else {
                  doingVolume = 1;  /* check out doingVolume */
                  doingPanel = VOLUMEpanel;
                }
                break;
              case clipXBut:  /* this is a horizontal button */
                clipValue = linearMouseXY.x * 0.5 + 0.5;   /* normalize to 0..1 */
                if (lessThan(clipValue,0.0)) clipValue = 0.0;
                if (greaterThan(clipValue,1.0)) clipValue = 1.0;
                if (lessThan(linearMouseXY.y,0.0)) {
                  if (!equal(xClipMinN,clipValue)) {
                    if (greaterThan(xClipMaxN-clipValue,minDistXY))
                      xClipMinN = clipValue;
                    else
                      xClipMinN = xClipMaxN - minDistXY;
                    viewData.clipXmin = xClipMinN *
                      (viewData.xmax - viewData.xmin) +
                      viewData.xmin;
                    drawClipXBut();
                    drawClipVolume();
                    if (viewData.clipbox)
                      drawViewport(Xoption);
                  }
                } else {
                  if (!equal(xClipMaxN,clipValue)) {
                    if (greaterThan(clipValue-xClipMinN,minDistXY))
                      xClipMaxN = clipValue;
                    else
                      xClipMaxN = xClipMinN + minDistXY;
                    viewData.clipXmax = xClipMaxN *
                      (viewData.xmax - viewData.xmin) +
                      viewData.xmin;
                    drawClipXBut();
                    drawClipVolume();
                    if (viewData.clipbox)
                      drawViewport(Xoption);
```

```
          }
        }
      break;
    case clipYBut:  /* this is a vertical button */
      /* normalize to 0..1, bottom up */
      clipValue = 1 - (linearMouseXY.y * 0.5 + 0.5);
      if (lessThan(clipValue,0.0)) clipValue = 0.0;
      if (greaterThan(clipValue,1.0)) clipValue = 1.0;
      if (lessThan(linearMouseXY.x,0.0)) {
        if (!equal(yClipMinN,clipValue)) {
          if (greaterThan(yClipMaxN-clipValue,minDistXY))
            yClipMinN = clipValue;
          else
            yClipMinN = yClipMaxN - minDistXY;
          viewData.clipYmin = yClipMinN *
            (viewData.ymax - viewData.ymin) +
            viewData.ymin;
          drawClipYBut();
          drawClipVolume();
          if (viewData.clipbox)
            drawViewport(Xoption);
        }
      } else {
        if (!equal(yClipMaxN,clipValue)) {
          if (greaterThan(clipValue-yClipMinN,minDistXY))
            yClipMaxN = clipValue;
          else
            yClipMaxN = yClipMinN + minDistXY;
          viewData.clipYmax = yClipMaxN *
            (viewData.ymax - viewData.ymin) +
            viewData.ymin;
          drawClipYBut();
          drawClipVolume();
          if (viewData.clipbox)
            drawViewport(Xoption);
        }
      }
      break;
    case clipZBut:  /* this is a diagonally aligned button! */
      /* f1 is the distance from the center of the button along
         the diagonal line with a slope of -1. If f1 is negative,
         the direction is downward from the center, if f1 is
         positive, the direction is upward from the center.
         Note that there ought to be a constant factor, namely
         cos(45), multiplied by f1 for the correct normalized value;
         however, we exploit this by foreshortening the length of the
```

```
      diagonal by that same factor (so instead of normalizing the
      numbers to, the line we normalize the line to the numbers)
      since we need to shorten the line at some point anyway
      (both to match the length of the diagonal side of the box
      and to allow more area for mouse input. */

  /* cos(45), etc => 0.4 */
  f1 = (linearMouseXY.x - linearMouseXY.y) * 0.4 + 0.5;
  if (lessThan(f1,0.0)) f1 = 0.0;
  if (greaterThan(f1,1.0)) f1 = 1.0;
  /* note that x<y => moving upward */
  if (lessThan(-linearMouseXY.x,linearMouseXY.y)) {
    if (!equal(zClipMaxN,f1)) {
      if (greaterThan(f1-zClipMinN,minDistZ))
        zClipMaxN = f1;
      else
        zClipMaxN = zClipMinN + minDistZ;
      viewData.clipZmax = zClipMaxN *
        (viewData.zmax - viewData.zmin) +
        viewData.zmin;
      drawClipZBut();
      drawClipVolume();
      if (viewData.clipbox)
        drawViewport(Xoption);
    }
  } else {
    if (!equal(zClipMinN,f1)) {
      if (greaterThan(zClipMaxN-f1,minDistZ))
        zClipMinN = f1;
      else
        zClipMinN = zClipMaxN - minDistZ;
      viewData.clipZmin = zClipMinN *
        (viewData.zmax - viewData.zmin) +
        viewData.zmin;
      drawClipZBut();
      drawClipVolume();
      if (viewData.clipbox)
        drawViewport(Xoption);
    }
  }   /* if lessThan(x,y) */
  break;
case perspectiveBut:
  if ((viewData.perspective = !viewData.perspective)) {
    switchedPerspective = yes;
    GSetForeground(volumeGC,
  (float)monoColor((control->buttonQueue[perspectiveBut]).textColor),
```

```
              Xoption);
          GDrawString(volumeGC,volumeWindow,
                      controlButton->buttonX +
                      centerX(volumeGC,"x",1,controlButton->buttonWidth),
                      controlButton->buttonY +
                      centerY(volumeGC,controlButton->buttonHeight),
                      "x",1,Xoption);
    }
    else
      XClearArea(dsply,volumeWindow,
                  controlButton->buttonX+1,
                  controlButton->buttonY+1,
                  controlButton->buttonHeight-2,
                  controlButton->buttonWidth-2,
                  False);
    drawViewport(Xoption);
    break;

  case clipRegionBut:
    if ((viewData.clipbox = !viewData.clipbox)) {
      GSetForeground(volumeGC,
      (float)monoColor((control->buttonQueue[clipRegionBut]).textColor),
                      Xoption);
      GDrawString(volumeGC,volumeWindow,
                      controlButton->buttonX +
                      centerX(volumeGC,"x",1,controlButton->buttonWidth),
                      controlButton->buttonY +
                      centerY(volumeGC,controlButton->buttonHeight),
                      "x",1,Xoption);
    }
    else
      XClearArea(dsply,volumeWindow,
                  controlButton->buttonX+1,
                  controlButton->buttonY+1,
                  controlButton->buttonWidth-2,
                  controlButton->buttonHeight-2,
                  False);

    drawViewport(Xoption);
    break;
  case clipSurfaceBut:
    if ((viewData.clipStuff = !viewData.clipStuff)) {
      GSetForeground(volumeGC,
      (float)monoColor((control->buttonQueue[clipSurfaceBut]).textColor),
                      Xoption);
      GDrawString(volumeGC,volumeWindow,
```

```
                              controlButton->buttonX +
                              centerX(volumeGC,"x",1,controlButton->buttonWidth),
                              controlButton->buttonY +
                              centerY(volumeGC,controlButton->buttonHeight),
                              "x",1,Xoption);
              }
              else
                XClearArea(dsply,volumeWindow,
                              controlButton->buttonX+1,
                              controlButton->buttonY+1,
                              controlButton->buttonWidth-2,
                              controlButton->buttonHeight-2,
                              False);
              break;

            default:
              buttonAction(controlButton->buttonKey);
            } /* switch on buttonKey */
          }  /* else - not closing */
        }   /* if checkButton */
      } /* if FD_ISSET(Xcon,.. */
      else if FD_ISSET(0,&rd) {
        externalControl = spadAction();
        if (spadDraw && (externalControl==0)) drawViewport(Xoption);
      }
    }        /* for (until closed) */
}            /* processEvents() */
```

7.3.39 project

Orthogonal projection for a point setting the ith Xpoint as well.

⟨*view3d*⟩+≡

```
void project(viewTriple * aViewTriple,XPoint *someXpoints,int i) {
  float Vtmp[4], V[4], V1[4];
  V[0] = aViewTriple->x;  V[1] = aViewTriple->y;
  V[2] = aViewTriple->z;  V[3] = 1.0;
  if (isNaNPoint(V[0], V[1], V[2])) {
    (someXpoints+i)->x = aViewTriple->px = NotPoint;
    (someXpoints+i)->y = aViewTriple->py = NotPoint;
    return;
  }
  V[0] -= viewport->transX; V[1] -= viewport->transY;
  V[2] -= viewport->transZ;
  vectorMatrix4(V,R1,Vtmp);
  matrixMultiply4x4(S,R,transform);
  vectorMatrix4(Vtmp,transform,V1);
  aViewTriple->wx = V1[0]; aViewTriple->wy = V1[1];
  aViewTriple->wz = V1[2];
  V1[0] *= reScale;  V1[1] *= reScale;  V1[2] *= reScale;
  aViewTriple->pz = V1[2];
  if (viewData.perspective) {
    V1[0] *= projPersp(aViewTriple->pz);
    V1[1] *= projPersp(aViewTriple->pz);
  }
  matrixMultiply4x4(I,T,transform);
  vectorMatrix4(V1,transform,V);
  V[0] = V[0]*viewScale + xCenter;
  V[1] = vwInfo.height - (V[1]*viewScale + yCenter);
  (someXpoints+i)->x = aViewTriple->px = V[0];
  (someXpoints+i)->y = aViewTriple->py = V[1];
}
```

7.3.40   projectAPoint

Orthogonal projection for a point. sort of like the above, but no Xpoint assignment.

⟨*view3d*⟩+≡

```
void projectAPoint(viewTriple *aViewTriple) {
  float Vtmp[4], V[4], V1[4];
  V[0] = aViewTriple->x;  V[1] = aViewTriple->y;
  V[2] = aViewTriple->z;  V[3] = 1.0;
  if (isNaNPoint(V[0], V[1], V[2])) {
    aViewTriple->px = NotPoint;
    aViewTriple->py = NotPoint;
    return;
  }
  V[0] -= viewport->transX; V[1] -= viewport->transY;
  V[2] -= viewport->transZ;
  vectorMatrix4(V,R1,Vtmp);
  matrixMultiply4x4(S,R,transform);
  vectorMatrix4(Vtmp,transform,V1);
  aViewTriple->wx = V1[0]; aViewTriple->wy = V1[1];
  aViewTriple->wz = V1[2];
  V1[0] *= reScale;  V1[1] *= reScale;  V1[2] *= reScale;
  aViewTriple->pz = V1[2];
  if (viewData.perspective) {
    V1[0] *= projPersp(aViewTriple->pz);
    V1[1] *= projPersp(aViewTriple->pz);
  }
  matrixMultiply4x4(I,T,transform);
  vectorMatrix4(V1,transform,V);
  V[0] = V[0]*viewScale + xCenter;
  V[1] = vwInfo.height - (V[1]*viewScale + yCenter);
  aViewTriple->px = V[0];
  aViewTriple->py = V[1];
}
```

7.3.41   projectAllPoints

⟨*view3d*⟩+≡

```
void projectAllPoints(void) {
  int i,j,k;
  LLPoint *anLLPoint;
  LPoint *anLPoint;
  int *anIndex;
  anLLPoint = viewData.lllp.llp;
  for (i=0; i<viewData.lllp.numOfComponents; i++,anLLPoint++) {
    anLPoint = anLLPoint->lp;
    for (j=0; j<anLLPoint->numOfLists; j++,anLPoint++) {
      anIndex = anLPoint->indices;
      for (k=0; k<anLPoint->numOfPoints; k++,anIndex++) {
        projectAPoint(refPt3D(viewData,*anIndex));
      } /* for points in LPoints (k) */
    } /* for LPoints in LLPoints (j) */
  } /* for LLPoints in LLLPoints (i) */
} /* projectAllPoints() */
```

7.3.42   projectAllPolys

Orthogonal projection of all the polygons in a given list in one go. pz holds the projected depth info for hidden surface removal. Polygons totally outside of the window dimensions after projection are discarded from the list.

⟨*view3d*⟩+≡

```
void projectAllPolys(poly *pList) {
  int i,clipped,clippedPz;
  float x0=0.0;
  float y0=0.0;
  float xA=0.0;
  float yA=0.0;
  float xB=0.0;
  float yB=0.0;
  int *anIndex;
  viewTriple *aPt;
  strcpy(control->message,"        Projecting Polygons        ");
  writeControlMessage();
  projectAllPoints();
  for (;pList != NIL(poly);pList=pList->next) {
      /* totalClip==yes => partialClip==yes (of course) */
    pList->totalClipPz = yes;  /* start with 1, AND all points with Pz<0 */
    pList->partialClipPz = no; /* start with 0, OR any points with Pz<0 */
    pList->totalClip = yes;  /* same idea, only wrt clip volume */
    pList->partialClip = no;
    for (i=0,anIndex=pList->indexPtr; i<pList->numpts; i++,anIndex++) {
      aPt = refPt3D(viewData,*anIndex);
      clipped = outsideClippedBoundary(aPt->x, aPt->y, aPt->z);
      pList->totalClip = pList->totalClip && clipped;
      pList->partialClip = pList->partialClip || clipped;
      clippedPz = behindClipPlane(aPt->pz);
      pList->totalClipPz = pList->totalClipPz && clippedPz;
      pList->partialClipPz = pList->partialClipPz || clippedPz;
        /* stuff for figuring out normalFacingOut, after the loop */
      if (!i) {
        x0 = aPt->px; y0 = aPt->py;
      } else if (i==1) {
        xA = x0 - aPt->px; yA = y0 - aPt->py;
        x0 = aPt->px;      y0 = aPt->py;
      } else if (i==2) {
        xB = aPt->px - x0; yB = aPt->py - y0;
      }
    } /* for i */
    /* store face facing info */
    /* For now, we shall give faces facing the user a factor of -1,
```

```
        and faces facing away from the user a factor of +1. this is
        to mimic the eye vector (pointing away from the user) dotted
        into the surface normal.
        This routine is being done because the surface normal in object
        space does not transform over to image space linearly and so
        has to be recalculated. but the triple product is zero in the
        X and Y directions so we just take the Z component, of which,
        we just examine the sign. */
    if ((x0 = xA*yB - yA*xB) > machine0) pList->normalFacingOut = 1;
    else if (x0 < machine0) pList->normalFacingOut = -1;
    else pList->normalFacingOut = 0;
  }
  strcpy(control->message,viewport->title);
  writeControlMessage();
} /* projectAllPolys */
```

7.3.43   projectAPoly

Orthogonal projection of all a polygon. pz holds the projected depth info for hidden surface removal.

⟨*view3d*⟩+≡

```
  void projectAPoly(poly *p) {
    int i,clipped,clippedPz;
    float Vtmp[4],V[4],V1[4];
    float x0=0.0;
    float y0=0.0;
    float xA=0.0;
    float yA=0.0;
    float xB=0.0;
    float yB=0.0;
    int *anIndex;
    viewTriple *aPt;
 /*  totalClip==yes => partialClip==yes */
    p->totalClipPz = yes; /*  start with 1, AND all points with Pz<0 */
    p->partialClipPz = no;  /* start with 0, OR any points with Pz<0 */
    p->totalClip = yes;  /* same idea, only with respect to clip volume */
    p->partialClip = no;
    for (i=0,anIndex=p->indexPtr; i<p->numpts; i++,anIndex++) {
      aPt  = refPt3D(viewData,*anIndex);
      V[0] = aPt->x;  V[1] = aPt->y;  V[2] = aPt->z;  V[3] = 1.0;
      V[0] -= viewport->transX; V[1] -= viewport->transY;
      V[2] -= viewport->transZ;
      vectorMatrix4(V,R1,Vtmp);
      matrixMultiply4x4(S,R,transform);
      vectorMatrix4(Vtmp,transform,V1);
      aPt->wx = V1[0];  aPt->wy = V1[1];  aPt->wz = V1[2];
      V1[0] *= reScale;  V1[1] *= reScale;  V1[2] *= reScale;
      aPt->pz = V1[2];
      if (viewData.perspective) {
        V1[0] *= projPersp(V1[2]);
        V1[1] *= projPersp(V1[2]);
      }
      matrixMultiply4x4(I,T,transform);
      vectorMatrix4(V1,transform,V);
      V[0] = V[0]*viewScale + xCenter;
      V[1] = vwInfo.height - (V[1]*viewScale + yCenter);
      aPt->px = V[0];  aPt->py = V[1];
      clipped = outsideClippedBoundary(aPt->x, aPt->y, aPt->z);
      p->totalClip = p->totalClip && clipped;
      p->partialClip = p->partialClip || clipped;
      clippedPz = behindClipPlane(aPt->pz);
```

```
      p->totalClipPz = p->totalClipPz && clippedPz;
      p->partialClipPz = p->partialClipPz || clippedPz;
      /* stuff for figuring out normalFacingOut, after the loop */
      if (!i) {
        x0 = aPt->px; y0 = aPt->py;
      } else if (i==1) {
        xA = x0 - aPt->px; yA = y0 - aPt->py;
        x0 = aPt->px;      y0 = aPt->py;
      } else if (i==2) {
        xB = aPt->px - x0; yB = aPt->py - y0;
      }
    }
    if ((x0 = xA*yB - yA*xB) > machine0) p->normalFacingOut = 1;
    else if (x0 < machine0) p->normalFacingOut = -1;
    else p->normalFacingOut = 0;
  }  /*  projectAPoly */
```

7.3.44   projectStuff

Sort of like the project stuff in tube.c but used exclusively for the functions of two variables. probably will need to be changed later to be more general (i.e. have everybody use the viewTriple point structure).

⟨*view3d*⟩+≡

```
void projectStuff(float x,float y,float z,int *px,int *py,float *Pz) {
  float tempx,tempy,tempz,temps,V[4],V1[4],stuffScale=100.0;
  tempx = viewport->scaleX;
  tempy = viewport->scaleY;
  tempz = viewport->scaleZ;
  temps = viewScale;
  if (viewport->scaleX > 5.0) viewport->scaleX = 5.0;
  if (viewport->scaleY > 5.0) viewport->scaleY = 5.0;
  if (viewport->scaleZ > 3.0) viewport->scaleZ = 3.0;
  if (viewScale > 5.0) viewScale = 5.0;
  V[0] = x;   V[1] = y;
  V[2] = z;   V[3] = 1.0;
  V[0] -= viewport->transX*stuffScale;
  V[1] -= viewport->transY*stuffScale;
  V[2] -= viewport->transZ*stuffScale;
  matrixMultiply4x4(S,R,transform);
  vectorMatrix4(V,transform,V1);
  *Pz = V1[2];
  if (viewData.perspective) {
    V1[0] *= projPersp(V1[2]);
    V1[1] *= projPersp(V1[2]);
  }
  matrixMultiply4x4(I,T,transform);
  vectorMatrix4(V1,transform,V);
  V[0] = V[0]*viewScale + xCenter;
  V[1] = vwInfo.height - (V[1]*viewScale + yCenter);
  *px = V[0];
  *py = V[1];
  viewport->scaleX = tempx;
  viewport->scaleY = tempy;
  viewport->scaleZ = tempz;
  viewScale = temps;
}
```

7.3.45 makeQuitPanel

⟨view3d⟩+≡

```
int makeQuitPanel(void) {
  int i;
  XSetWindowAttributes quitter,QuitterAttrib;
  XSizeHints sizeh;
  Pixmap quitbits, quitmask;
  XColor quitterColor,qColor;
  quitbits = XCreateBitmapFromData(dsply,rtWindow, volumeBitmap_bits,
                                   volumeBitmap_width,volumeBitmap_height);
  quitmask = XCreateBitmapFromData(dsply,rtWindow,volumeMask_bits,
                                   volumeMask_width,volumeMask_height);
  quitter.background_pixel = backgroundColor;
  quitter.border_pixel = foregroundColor;
  quitter.event_mask = quitMASK;
  quitter.colormap = colorMap;
  quitter.override_redirect = overrideManager;
  quitterColor.pixel = quitCursorForeground;
  XQueryColor(dsply,colorMap,&quitterColor);
  qColor.pixel = quitCursorBackground;
  XQueryColor(dsply,colorMap,&qColor);
  quitter.cursor = XCreatePixmapCursor(dsply,quitbits,quitmask,
                                       &quitterColor,&qColor,
                                       volumeBitmap_x_hot,volumeBitmap_y_hot);
  quitWindow = XCreateWindow(dsply,control->controlWindow,
                     controlWidth-quitWidth-2,controlHeight-quitHeight-2,
                     quitWidth-2,quitHeight-2,2,
                     CopyFromParent,InputOutput,CopyFromParent,
                     controlCreateMASK,&quitter);
  sizeh.flags  = USPosition | USSize;
  sizeh.x      = 0;
  sizeh.y      = 0;
  sizeh.width  = quitWidth-2;
  sizeh.height = quitHeight-2;
  XSetNormalHints(dsply,quitWindow,&sizeh);
  XSetStandardProperties(dsply,quitWindow,"Quit Panel","Quit Panel",
                         None,NULL,0,&sizeh);
  /*** do quit buttons ***/
  initQuitButtons(control->buttonQueue);
  for (i=quitButtonsStart; i<(quitButtonsEnd); i++) {
    QuitterAttrib.event_mask = (control->buttonQueue[i]).mask;
    (control->buttonQueue[i]).self =
                XCreateWindow(dsply,quitWindow,
                              (control->buttonQueue[i]).buttonX,
                              (control->buttonQueue[i]).buttonY,
```

```
                                (control->buttonQueue[i]).buttonWidth,
                                (control->buttonQueue[i]).buttonHeight,
                                0,0,InputOnly,CopyFromParent,
                                buttonCreateMASK,&QuitterAttrib);
        XMakeAssoc(dsply,table,(control->buttonQueue[i]).self,
                  &((control->buttonQueue[i]).buttonKey));
        XMapWindow(dsply,(control->buttonQueue[i]).self);
    }
    return(0);
}  /* makeQuitPanel() */
```

7.3.46   drawQuitPanel

⟨*view3d*⟩+≡

```
void drawQuitPanel(void) {
  char *s;
  int i,strlength;
  s = "Really?";
  strlength = strlen(s);
  GSetForeground(anotherGC,(float)quitTitleColor,Xoption);
  GDrawString(anotherGC,quitWindow,
              centerX(anotherGC,s,strlength,quitWidth),
              centerY(anotherGC,39),s,strlength,Xoption);
  GSetForeground(anotherGC,(float)quitButtonColor,Xoption);
  for (i=quitButtonsStart; i<(quitButtonsEnd); i++) {
      GDraw3DButtonOut(quitGC,quitWindow,
                       (control->buttonQueue[i]).buttonX,
                       (control->buttonQueue[i]).buttonY,
                       (control->buttonQueue[i]).buttonWidth,
                       (control->buttonQueue[i]).buttonHeight,Xoption);
      s = (control->buttonQueue[i]).text;
      strlength = strlen(s);
      GSetForeground(trashGC,
                     (float)monoColor((control->buttonQueue[i]).textColor),
                     Xoption);
      GDrawString(trashGC,quitWindow,
                  (control->buttonQueue[i]).buttonX +
                  centerX(processGC,s,strlength,
                          (control->buttonQueue[i]).buttonWidth),
                  (control->buttonQueue[i]).buttonY +
                  centerY(processGC,(control->buttonQueue[i]).buttonHeight),
                  s,strlen(s),Xoption);
  } /* for i in control->buttonQueue */
} /* drawQuitPanel */
```

7.3.47 initQuitButtons

⟨*view3d*⟩+≡

```
int initQuitButtons(buttonStruct *quitButtons) {
  int ii;
  int num = 0;
  ii = quitAbort;
  quitButtons[ii].buttonX      = 5;
  quitButtons[ii].buttonY      = 41;
  quitButtons[ii].buttonWidth  = 53;
  quitButtons[ii].buttonHeight = 25;
  quitButtons[ii].buttonKey    = ii;
  quitButtons[ii].pot          = no;
  quitButtons[ii].mask         = buttonMASK;
  quitButtons[ii].text         = "No";
  quitButtons[ii].textColor    = 6;
  quitButtons[ii].xHalf        = quitButtons[ii].buttonWidth/2;
  quitButtons[ii].yHalf        = quitButtons[ii].buttonHeight/2;
  ++num;
  ii = quitReturn;
  quitButtons[ii].buttonX      = 5;
  quitButtons[ii].buttonY      = 75;
  quitButtons[ii].buttonWidth  = 53;
  quitButtons[ii].buttonHeight = 25;
  quitButtons[ii].buttonKey    = ii;
  quitButtons[ii].pot          = no;
  quitButtons[ii].mask         = buttonMASK;
  quitButtons[ii].text         = "Yes";
  quitButtons[ii].textColor    = onColor;
  quitButtons[ii].xHalf        = quitButtons[ii].buttonWidth/2;
  quitButtons[ii].yHalf        = quitButtons[ii].buttonHeight/2;
  ++num;
  return(num);
}
```

7.3.48 makeSavePanel

⟨*view3d*⟩+≡

```
int makeSavePanel(void) {
  int i;
  XSetWindowAttributes saver,SaverAttrib;
  XSizeHints sizeh;
  Pixmap savebits, savemask;
  XColor saveColor,sColor;
  savebits = XCreateBitmapFromData(dsply,rtWindow, volumeBitmap_bits,
                                   volumeBitmap_width,volumeBitmap_height);
  savemask = XCreateBitmapFromData(dsply,rtWindow,volumeMask_bits,
                                   volumeMask_width,volumeMask_height);
  saver.background_pixel = backgroundColor;
  saver.border_pixel = foregroundColor;
  saver.event_mask = saveMASK;
  saver.colormap = colorMap;
  saver.override_redirect = overrideManager;
  saveColor.pixel = saveCursorForeground;
  XQueryColor(dsply,colorMap,&saveColor);
  sColor.pixel = saveCursorBackground;
  XQueryColor(dsply,colorMap,&sColor);
  saver.cursor =
    XCreatePixmapCursor(dsply,savebits,savemask,&saveColor,&sColor,
                        volumeBitmap_x_hot,volumeBitmap_y_hot);
  saveWindow =
    XCreateWindow(dsply,control->controlWindow,controlWidth-saveWidth-2,
                  controlHeight-saveHeight-2,saveWidth-2,saveHeight-2,2,
                  CopyFromParent,InputOutput,CopyFromParent,
                  controlCreateMASK,&saver);
  sizeh.flags  = USPosition | USSize;
  sizeh.x      = 0;
  sizeh.y      = 0;
  sizeh.width  = saveWidth-2;
  sizeh.height = saveHeight-2;
  XSetNormalHints(dsply,saveWindow,&sizeh);
  XSetStandardProperties(dsply,saveWindow,"Save Panel","Save Panel",
                         None,NULL,0,&sizeh);
  /*** Create save buttons ***/
  initSaveButtons(control->buttonQueue);
  for (i=saveButtonsStart; i<(saveButtonsEnd); i++) {
    SaverAttrib.event_mask = (control->buttonQueue[i]).mask;
    (control->buttonQueue[i]).self =
                XCreateWindow(dsply,saveWindow,
                              (control->buttonQueue[i]).buttonX,
                              (control->buttonQueue[i]).buttonY,
```

```
                                 (control->buttonQueue[i]).buttonWidth,
                                 (control->buttonQueue[i]).buttonHeight,
                                 0,0,InputOnly,CopyFromParent,
                                 buttonCreateMASK,&SaverAttrib);
       XMakeAssoc(dsply,table,(control->buttonQueue[i]).self,
                   &((control->buttonQueue[i]).buttonKey));
       XMapWindow(dsply,(control->buttonQueue[i]).self);
     }
     return(0);
   } /* makeSavePanel() */
```

7.3.49   drawSavePanel

⟨*view3d*⟩+≡

```
  void drawSavePanel(void) {
    char *s;
    int i,strlength;
    GSetForeground(saveGC,(float)saveButtonColor,Xoption);
    for (i=saveButtonsStart; i<(saveButtonsEnd); i++) {
        GDraw3DButtonOut(saveGC,saveWindow,
                         (control->buttonQueue[i]).buttonX,
                         (control->buttonQueue[i]).buttonY,
                         (control->buttonQueue[i]).buttonWidth,
                         (control->buttonQueue[i]).buttonHeight,Xoption);
        s = (control->buttonQueue[i]).text;
        strlength = strlen(s);
        GSetForeground(trashGC,
                       (float)monoColor((control->buttonQueue[i]).textColor),
                       Xoption);
        GDrawString(trashGC,saveWindow,
                    (control->buttonQueue[i]).buttonX +
                    centerX(processGC,s,strlength,
                            (control->buttonQueue[i]).buttonWidth),
                    (control->buttonQueue[i]).buttonY +
                    centerY(processGC,(control->buttonQueue[i]).buttonHeight),
                    s,strlen(s),Xoption);
    } /* for i in control->buttonQueue */
  } /* drawSavePanel */
```

7.3.50 initSaveButtons

⟨*view3d*⟩+≡

```
int initSaveButtons(buttonStruct *saveButtons) {
  int ii;
  int num = 0;
  ii = saveExit;
  saveButtons[ii].buttonX      = 5;
  saveButtons[ii].buttonY      = 7;
  saveButtons[ii].buttonWidth  = 53;
  saveButtons[ii].buttonHeight = 25;
  saveButtons[ii].buttonKey    = ii;
  saveButtons[ii].pot          = no;
  saveButtons[ii].mask         = buttonMASK;
  saveButtons[ii].text         = "Cancel";
  saveButtons[ii].textColor    = 6;
  saveButtons[ii].xHalf        = saveButtons[ii].buttonWidth/2;
  saveButtons[ii].yHalf        = saveButtons[ii].buttonHeight/2;
  ++num;
  ii = pixmap;
  saveButtons[ii].buttonX      = 5;
  saveButtons[ii].buttonY      = 41;
  saveButtons[ii].buttonWidth  = 53;
  saveButtons[ii].buttonHeight = 25;
  saveButtons[ii].buttonKey    = ii;
  saveButtons[ii].pot          = no;
  saveButtons[ii].mask         = buttonMASK;
  saveButtons[ii].text         = "Pixmap";
  saveButtons[ii].textColor    = 28;
  saveButtons[ii].xHalf        = saveButtons[ii].buttonWidth/2;
  saveButtons[ii].yHalf        = saveButtons[ii].buttonHeight/2;
  ++num;
  ii = ps;
  saveButtons[ii].buttonX         = 5;
  saveButtons[ii].buttonY         = 75;
  saveButtons[ii].buttonWidth     = 53;
  saveButtons[ii].buttonHeight    = 25;
  saveButtons[ii].buttonKey       = ii;
  saveButtons[ii].pot             = no;
  saveButtons[ii].mask            = buttonMASK;
  saveButtons[ii].text            = "PS";
  saveButtons[ii].textColor       = 149;
  saveButtons[ii].xHalf           = saveButtons[ii].buttonWidth/2;
  saveButtons[ii].yHalf           = saveButtons[ii].buttonHeight/2;
  return(num);
}
```

7.3.51   getCBufferAxes

⟨*view3d*⟩+≡
```
  char getCBufferAxes(int ix) {
    if( ix >=0 && ix <ARRAY_WIDTH) return (cBuffer[ix].axes);
    return ('0');
  }
```

7.3.52   putCBufferAxes

⟨*view3d*⟩+≡
```
  void putCBufferAxes(int ix,char val) {
    if( ix >=0 && ix <ARRAY_WIDTH) cBuffer[ix].axes = val;
  }
```

7.3.53   getCBufferIndx

⟨*view3d*⟩+≡
```
  int getCBufferIndx(int ix) {
    if( ix >=0 && ix <ARRAY_WIDTH) return (cBuffer[ix].indx);
    return (-1);
  }
```

7.3.54   putCBufferIndx

⟨*view3d*⟩+≡
```
  void putCBufferIndx(int ix,int val) {
    if( ix >=0 && ix <ARRAY_WIDTH) cBuffer[ix].indx = val;
  }
```

7.3.55 putZBuffer

⟨*view3d*⟩+≡

```
void putZBuffer(int ix,float val) {
  if (ix >=0 && ix <ARRAY_WIDTH) zBuffer[ix] = val;
}
```

7.3.56 getZBuffer

⟨*view3d*⟩+≡

```
float getZBuffer(int ix) {
  return (zBuffer[ix]);
}
```

7.3.57 putImageX

⟨*view3d*⟩+≡

```
void putImageX(int ix,char val) {
  if (ix <=0 && ix <vwInfo.width) imageX->data[ix] = val;
}
```

7.3.58   drawPhongSpan

This routine sets the buffer values for each span of pixels which intersect the current scanline.

⟨*view3d*⟩+≡

```
void drawPhongSpan(triple pt,float N[3],int dFlag) {
  int              xpixel,hue,shade;
  float            colorindx, col;
  triple           hs;
  /* negative values of xleft and xright have been pushed to machine0 */
  xpixel = (int)xleft;
  while (xpixel <= (int)xright) {
    /* if z is closer to viewer than value in zBuffer continue */
    if ( (zC < getZBuffer(xpixel)) ) {
      /* get the intensity for current point */
      col = phong(pt,N);
      putCBufferAxes(xpixel,'0');
      putZBuffer(xpixel,zC);
      /* if mono (bw dsply) do black and white semi-random dithering */
      if (mono || (dFlag == PSoption) || viewport->monoOn) {
        if (getRandom() < 100.0*exp((double)-1.3*(pi_sq*(col-.2)*(col-.2))))
        {
          putCBufferIndx(xpixel,black);
        } else {
          putCBufferIndx(xpixel,white);
        }
      } else {
        /* glossy shading for one hue else dithered for many hues */
        if (viewport->hueOffset == viewport->hueTop && !smoothError) {
          colorindx = (float)(smoothConst+1) * col;
          if (colorindx > (smoothConst+1)) colorindx = smoothConst+1;
          putCBufferIndx(xpixel,XPixelColor((int)colorindx-1));
        } else { /* probabalistic multi-hued dithering */
          hs = normDist();
          hue = (int)(intersectColor[0]+hs.x/20.0);
          /* cannot dither out of color map range */
          if (viewport->hueOffset < viewport->hueTop) {
            if (hue < viewport->hueOffset)
              hue = viewport->hueOffset;
            else {
              if (hue > viewport->hueTop)
                hue = viewport->hueTop;
            }
          } else {
            if (hue < viewport->hueTop)
```

```
                hue = viewport->hueTop;
              else {
                if (hue > viewport->hueOffset)
                  hue = viewport->hueOffset;
              }
            }
            col += hs.y/6.0;  /* perturb intensity */
            if (col > 1.0) putCBufferIndx(xpixel,white);
            else {
              if (col < 0.0) putCBufferIndx(xpixel,black);
              else {
                shade = (int)(col * 4.0);
                putCBufferIndx(xpixel,XSolidColor(hue,shade));
              }
            }
          }
        }
      } /* zC < zBuffer */
      zC += dzdx;
      if (viewport->hueOffset != viewport->hueTop || smoothError ||
          viewport->monoOn)
        intersectColor[0] += dcolor;
      N[0] += dnorm.x;  N[1] += dnorm.y;  N[2] += dnorm.z;
      pt.x += dpt.x;  pt.y += dpt.y;  pt.z += dpt.z;
      xpixel++;
    } /* while each pixel */
}
```

7.3.59 scanPhong

This routine takes all polygons that intersect with the current scanline and calculates the
intersecting x and z points as well as the color at each point. Interpolation is done according
to Phong.

⟨*view3d*⟩+≡
```
  void scanPhong(int dFlag) {
    viewTriple *p1, *p2;
    polyList    *polygon;
    poly        *p;
    int         i,num,xtemp,numttt;
    int         *anIndex, *start, *end;
    float       x1,x2,y1,y2,z2,zright,wx1,wx2,wy1,wy2,wz1,wz2;
    float       intersectionx[2], intersectionz[2];
    float       c1,c2,colortemp,ztemp,dY,diffy,diffx,n1[3],n2[3],NV[3];
    triple      ntemp, intersectPt[2], ptemp, pt, intersectN[2];
    /* polygon list intersecting the current scanline, will be modified to
       edge list structure */
    polygon = scanList[scanline];
    while (polygon != NIL(polyList) && polygon->polyIndx != NIL(poly) ) {
      /* for each polygon in the list */
      p = polygon->polyIndx;
      /* don't include clipped polygons */
      if ( ! ( p->partialClipPz ||
               p->totalClipPz   ||
               (viewData.clipStuff && (p->partialClip  || p->totalClip ) ) ) ) {
        num = 0; /* 0 & 1, for the 2 edges of polygon that intersect scanline */
        numttt =0;
        if ((scanline >= (int)p->pymin) && (scanline <= (int)p->pymax)) {
          /* which edges of the polygon intersect the scanline */
          for (i=0, anIndex=p->indexPtr; i<p->numpts; i++) {
            start = anIndex + i;
            p1 = refPt3D(viewData,*(start));
            x1 = p1->px;  y1 = p1->py;  zC = p1->pz;  c1 = p1->sc;
/*          if (x1 < machine0){ x1 = machine0; } */
            wx1 = p1->wx; wy1 = p1->wy; wz1 = p1->wz;
            n1[0] = p1->norm[0]; n1[1] = p1->norm[1]; n1[2] = p1->norm[2];
            end = (i != (p->numpts - 1)) ? anIndex + (i + 1) : anIndex;
            p2 = refPt3D(viewData,*(end));
            x2 = p2->px; y2 = p2->py; z2 = p2->pz; c2 = p2->sc;
/*           if (x2 < machine0){ x2 = machine0; } */
            wx2 = p2->wx; wy2 = p2->wy; wz2 = p2->wz;
            n2[0] = p2->norm[0]; n2[1] = p2->norm[1]; n2[2] = p2->norm[2];
            /* find beginning and end for intersecting edge */
            if ((scanline < y1 && scanline >= y2) ||
```

```
        (scanline >= y1 && scanline < y2)) {
      dY = (float)scanline - y1;
      diffy = y2 - y1;
      if (absolute(diffy) < 0.01) diffy = 1.0;
      intersectionx[num] = x1 + ((x2-x1)/diffy) * dY;
      intersectionz[num] = zC + ((z2-zC)/diffy) * dY;
      if (viewport->hueOffset != viewport->hueTop || smoothError ||
          viewport->monoOn)
        intersectColor[num] = c1 + ((c2 - c1)/diffy) * dY;
      intersectN[num].x = n1[0] + ((n2[0] - n1[0])/diffy)*dY;
      intersectN[num].y = n1[1] + ((n2[1] - n1[1])/diffy)*dY;
      intersectN[num].z = n1[2] + ((n2[2] - n1[2])/diffy)*dY;
      intersectPt[num].x = wx1 + ((wx2 - wx1)/diffy)*dY;
      intersectPt[num].y = wy1 + ((wy2 - wy1)/diffy)*dY;
      intersectPt[num].z = wz1 + ((wz2 - wz1)/diffy)*dY;
      num = 1-num;
      numttt++;
    } /* if edge intersects scanline */
  } /* for each edge */
if (numttt>=2) { /* if numttt 0 or 1 something has gone wrong */
   xleft = intersectionx[0];  xright = intersectionx[1];
   zC  = intersectionz[0];     zright = intersectionz[1];
   /* edges are drawn from left to right, so switch if necessary */
   if (xright < xleft) {
     xtemp = xright;  xright = xleft;  xleft = xtemp;
     ztemp = zright;  zright = zC;  zC = ztemp;
     if (viewport->hueOffset != viewport->hueTop || smoothError ||
         viewport->monoOn) {
       colortemp = intersectColor[1];
       intersectColor[1] = intersectColor[0];
       intersectColor[0] = colortemp;
     }
     ntemp = intersectN[1];  intersectN[1] = intersectN[0];
     intersectN[0] = ntemp;
     ptemp = intersectPt[1];
     intersectPt[1] = intersectPt[0];
     intersectPt[0] = ptemp;
   }
   diffx = xright - xleft;
   if (absolute(diffx) > .01) {
     if (viewport->hueOffset != viewport->hueTop || smoothError ||
         viewport->monoOn)
       dcolor = (intersectColor[1] - intersectColor[0]) / diffx;
     dnorm.x = (intersectN[1].x - intersectN[0].x) / diffx;
     dnorm.y = (intersectN[1].y - intersectN[0].y) / diffx;
     dnorm.z = (intersectN[1].z - intersectN[0].z) / diffx;
```

```
            dpt.x = (intersectPt[1].x - intersectPt[0].x) / diffx;
            dpt.y = (intersectPt[1].y - intersectPt[0].y) / diffx;
            dpt.z = (intersectPt[1].z - intersectPt[0].z) / diffx;
            dzdx = (zright - zC) / diffx;
          } else {
            if (viewport->hueOffset != viewport->hueTop || smoothError ||
                viewport->monoOn)
              dcolor = intersectColor[1];
            dnorm.x = 0.0;  dnorm.y = 0.0;  dnorm.z = 0.0;
            dpt.x = 0.0;  dpt.y = 0.0;  dpt.z = 0.0;
            dzdx = 0.0;
          }
          NV[0] = intersectN[0].x;
          NV[1] = intersectN[0].y;
          NV[2] = intersectN[0].z;
          pt.x = intersectPt[0].x;
          pt.y = intersectPt[0].y;
          pt.z = intersectPt[0].z;
          drawPhongSpan(pt,NV,dFlag);
        } /* numttt guard */
         } /* if scanline intersect */
      } /* clipped */
    polygon = polygon->next;
  } /* while still polygons */
}
```

7.3.60   boxTObuffer

boxTObuffer() writes the projection of the x,y bounding box to the z-buffer.

⟨*view3d*⟩+≡

```
void boxTObuffer(void) {
  int    xpix,i,j,k,count,decision;
  int    xA,xB,yA,yB;
  float  x,xend,y,yend,diffy,dX,dY,dXY,intersectionx;
  for (i=0;i<6;i++) {
    if (box[i].inside) {
      for (j=0; j<3; j++) {
        quadMesh[j].x = box[i].pointsPtr[j]->px;
        quadMesh[j].y = box[i].pointsPtr[j]->py;
      }
      intersectionx = 0.0;
      for (k=0; k<2; k++) {
        xA = quadMesh[k].x; yA = quadMesh[k].y;
        xB = quadMesh[k+1].x; yB = quadMesh[k+1].y;
/*
        if (xA > graphWindowAttrib.width+1) xA = graphWindowAttrib.width+1;
        if (xB > graphWindowAttrib.width+1) xB = graphWindowAttrib.width+1;
        if (yA > graphWindowAttrib.height) yA = graphWindowAttrib.height;
        if (yB > graphWindowAttrib.height) yB = graphWindowAttrib.height;
        if (xA < 0) xA = 0;  if (xB < 0) xB = 0;
        if (yA < 0) yA = 0;  if (yB < 0) yB = 0;
*/
        x = xA;  xend = xB;  y = yA;  yend = yB;
        diffy = (float)scanline - y;
        dX = xend - x;  dY = yend - y;
        if (absolute(dY) > machine0) {
          dXY = dX/dY;
        } else {
          dXY = dX;
        }
        if (dXY < 0.0) dXY = -dXY;
        if ((scanline == (int)y) && (absolute(dY) <= 1.0)) {
          if (x <= xend) {
            for (xpix = (int)x; xpix <= (int)xend; xpix++) {
              putCBufferAxes(xpix,'b');
            }
          } else {
            for (xpix = (int)x; xpix >= (int)xend; xpix--) {
              putCBufferAxes(xpix,'b');
            }
          }
```

```
        } else {
          if (xend < x)
            decision = (scanline < y && scanline >= yend) ||
                       (scanline > y && scanline <= yend);
          else
            decision = (scanline <= y && scanline > yend) ||
                       (scanline >= y && scanline < yend);
          if (decision) {
            intersectionx = x + dX/dY * diffy;
            for (count = (int)intersectionx;
                 count <= (int)intersectionx + (int)dXY; count++) {
              putCBufferAxes(count,'b');
            }
          }
        }
      }
    }
  }
}
```

7.3.61 clipboxTObuffer

clipboxTObuffer() writes the projection of the x,y,z clipping region box to the z-buffer.

⟨*view3d*⟩+≡

```
void clipboxTObuffer(void) {
  int    xpix,i,j,k,count,decision;
  int    xA,xB,yA,yB;
  float  x,xend,y,yend,diffy,dX,dY,dXY,intersectionx;
  for (i=0;i<6;i++) {
    if (clipBox[i].inside) {
      for (j=0; j<3; j++) {
        quadMesh[j].x = clipBox[i].pointsPtr[j]->px;
        quadMesh[j].y = clipBox[i].pointsPtr[j]->py;
      }
      intersectionx = 0.0;
      for (k=0; k<2; k++) {
        xA = quadMesh[k].x; yA = quadMesh[k].y;
        xB = quadMesh[k+1].x; yB = quadMesh[k+1].y;
/*
        if (xA > graphWindowAttrib.width+1) xA = graphWindowAttrib.width+1;
        if (xB > graphWindowAttrib.width+1) xB = graphWindowAttrib.width+1;
        if (yA > graphWindowAttrib.height) yA = graphWindowAttrib.height;
        if (yB > graphWindowAttrib.height) yB = graphWindowAttrib.height;
        if (xA < 0) xA = 0;  if (xB < 0) xB = 0;
        if (yA < 0) yA = 0;  if (yB < 0) yB = 0;
*/
        x = xA;  xend = xB;  y = yA;  yend = yB;
        diffy = (float)scanline - y;
        dX = xend - x;  dY = yend - y;
        if (absolute(dY) > machine0) {
          dXY = dX/dY;
        } else {
          dXY = dX;
        }
        if (dXY < 0.0) dXY = -dXY;

        if ((scanline == (int)y) && (absolute(dY) <= 1.0)) {
          if (x <= xend) {
            for (xpix = (int)x; xpix <= (int)xend; xpix++) {
              putCBufferAxes(xpix,'c');
            }
          } else {
            for (xpix = (int)x; xpix >= (int)xend; xpix--) {
              putCBufferAxes(xpix,'c');
            }
```

```
          }
        } else {
          if (xend < x)
            decision = (scanline < y && scanline >= yend) ||
                       (scanline > y && scanline <= yend);
          else
            decision = (scanline <= y && scanline > yend) ||
                       (scanline >= y && scanline < yend);
          if (decision) {
            intersectionx = x + dX/dY * diffy;
            for (count = (int)intersectionx;
                 count <= (int)intersectionx + (int)dXY; count++) {
              putCBufferAxes(count,'c');
            }
          }
        }
      }
    }
  }
}
```

7.3.62 axesTObuffer

axesTObuffer() writes the projection of the x,y,z axes to the z-buffer.

⟨*view3d*⟩+≡

```
void axesTObuffer(void) {
  int    xpix,i,count,decision;
  int    xA,xB,yA,yB;
  float  x,xend,y,yend,diffy,dX,dY,dXY,intersectionx;
  float  zA,zB,z,zend;
  float  dZ,dZX,dZY,intersectionz;
  intersectionz = 0.0;   intersectionx = 0.0;
  for (i=0; i<3; i++) {
    xA = axesXY[i][0]; yA = axesXY[i][1]; zA = axesZ[i][0];
    xB = axesXY[i][2]; yB = axesXY[i][3]; zB = axesZ[i][1];
/*
    if (xA > graphWindowAttrib.width+1) xA = graphWindowAttrib.width+1;
    if (xB > graphWindowAttrib.width+1) xB = graphWindowAttrib.width+1;
    if (yA > graphWindowAttrib.height) yA = graphWindowAttrib.height;
    if (yB > graphWindowAttrib.height) yB = graphWindowAttrib.height;
    if (xA < 0) xA = 0;  if (xB < 0) xB = 0;
    if (yA < 0) yA = 0;  if (yB < 0) yB = 0;
*/
    x = xA;  xend = xB;  y = yA;  yend = yB;  z = zA;  zend = zB;
    diffy = (float)scanline - y;
    dX = xend - x;  dY = yend - y;  dZ = zend - z;
      dZY = dZ/dY;
      dXY = dX/dY;
    if (dXY < 0.0) dXY = -dXY;
    dZX = dZ/dX;
    if ((scanline == (int)y) && (absolute(dY) <= 1.0)) {
      if (x <= xend) {
        for (xpix = (int)x; xpix <= (int)xend; xpix++) {
          putCBufferAxes(xpix,'a');
          putZBuffer(xpix,z + dZY * diffy);
        } /* for x */
      } else {
        for (xpix = (int)x; xpix >= (int)xend; xpix--) {
          putCBufferAxes(xpix,'a');
          putZBuffer(xpix,z + dZY * diffy);
        } /* for x */
      }
    } else {
      if (xend < x)
        decision = (scanline < y && scanline >= yend) ||
                   (scanline > y && scanline <= yend);
```

```
      else
        decision = (scanline <= y && scanline > yend) ||
                   (scanline >= y && scanline < yend);
      if (decision) {
        intersectionx = x + dX/dY * diffy;
        intersectionz = z + dZY * diffy;
        for (count = (int)intersectionx;
             count <= (int)intersectionx + (int)dXY; count++) {
          putCBufferAxes(count,'a');
          putZBuffer(count,intersectionz);
          intersectionz += dZX;
        }
      } /* if edge intersects scanline */
    }
  } /* for each axes */
}
```

7.3.63 scanLines

scanLines() scanline z-buffer algorithm initialize z-buffer and color buffer for all scanlines.

⟨*view3d*⟩+≡

```
void scanLines(int dFlag) {
  unsigned long pixColor;
  int           i;
  char          tempA;
  if (dFlag == Xoption) {
    if (viewmap_valid) {
      XFreePixmap(dsply,viewmap);
      viewmap_valid=0;
    }
    viewmap = XCreatePixmap(/* display */    dsply,
                            /* drawable */   viewport->viewWindow,
                            /* width */      vwInfo.width,
                            /* height */     vwInfo.height,
                            /* depth */      DefaultDepth(dsply,scrn));
    viewmap_valid =1;
    GSetForeground(trashGC,(float)backgroundColor,dFlag);
    XFillRectangle(dsply,viewmap,trashGC,0,0,vwInfo.width,vwInfo.height);
    XFillRectangle(dsply,viewport->viewWindow,trashGC,0,0,
                   vwInfo.width,vwInfo.height);
  } else {
    GSetForeground(GC9991,
                   1.0-(float)((int)(psShadeMax-0.3*psShadeMax)-1)*psShadeMul,dFlag);
    quadMesh[0].x = 0;  quadMesh[0].y = 0;
    quadMesh[1].x = graphWindowAttrib.width+2;
    quadMesh[1].y = 0;
    quadMesh[2].x = graphWindowAttrib.width+2;
    quadMesh[2].y = graphWindowAttrib.height;
    quadMesh[3].x = 0;
    quadMesh[3].y = graphWindowAttrib.height;
    quadMesh[4].x = 0;  quadMesh[4].y = 0;
    PSFillPolygon(GC9991, quadMesh, 5);
  }
  if (graphWindowAttrib.height >= physicalHeight)
    graphWindowAttrib.height = physicalHeight - 1;
  if (graphWindowAttrib.width >= physicalWidth)
    graphWindowAttrib.width = physicalWidth - 1;
  if (dFlag == Xoption)
    strcpy(control->message,"        Display Scanlines         ");
  else
    strcpy(control->message,"          Writing Output          ");
  writeControlMessage();
```

```
scanline = graphWindowAttrib.height-1;
imageX = XCreateImage(/* display */          dsply,
                      /* visual */           DefaultVisual(dsply,scrn),
                      /* depth */            DefaultDepth(dsply,scrn),
                      /* format */           ZPixmap,
                      /* offset */           0,
                      /* data */             0,
                      /* width */            vwInfo.width,
                      /* height */           1,
                      /* bitmap_pad */       32,
                      /* bytes_per_line */ 0);
imageX->data = (char *)malloc(imageX->bytes_per_line);
while (scanline >= 0 && keepDrawingViewport()) {
  /* initialize buffer values for scanline */
  pixColor = backgroundColor;
  for (i=0; i < (int)graphWindowAttrib.width; i++) {
    putZBuffer(i,10000.0);
    putCBufferIndx(i,-1);
    putCBufferAxes(i,'0');
    if (mono || viewport->monoOn)
      if ((scanline % 2) == 0)
        if ((i % 2) == 0) {
          if (i>=0 && i<vwInfo.width) XPutPixel(imageX,i,0,backgroundColor);
        }
        else {
          if (i>=0 && i<vwInfo.width) XPutPixel(imageX,i,0,foregroundColor);
        }
      else
        if ((i % 2) == 0) {
          if (i>=0 && i<vwInfo.width) XPutPixel(imageX,i,0,foregroundColor);
        }
        else {
          if (i>=0 && i<vwInfo.width) XPutPixel(imageX,i,0,backgroundColor);
        }
    else {
      if (i>=0 && i<vwInfo.width) XPutPixel(imageX,i,0,backgroundColor);
    }
  }
  /* writes the axes info to the buffers */
  if (viewData.box) boxTObuffer();
  if (viewData.clipbox) clipboxTObuffer();
  if (viewport->axesOn) axesTObuffer();
  /* fill buffers for current scanline */
  scanPhong(dFlag);
  for (i=0; i < (int)graphWindowAttrib.width; i++) {
    /* include bounding region info */
```

```
            if (viewData.box) {
              if (getCBufferAxes(i) == 'b') {
                if (dFlag==Xoption) {
                  if (mono || (viewport->monoOn)) pixColor = foregroundColor;
                  else pixColor = boxInline;
                  if (i >=0 && i<vwInfo.width) XPutPixel(imageX,i,0,pixColor);
                } else {
                  GSetForeground(GC9991, psBlack, dFlag );
                  GDrawPoint(viewport->viewWindow, GC9991, i,scanline,dFlag);
                }
              }
            }
            /* include clipping box info */
            if (viewData.clipbox) {
              if (getCBufferAxes(i)== 'c') {
                if (dFlag==Xoption) {
                  if (mono || (viewport->monoOn)) pixColor = foregroundColor;
                  else pixColor = clipBoxInline;
                  if (i >=0 && i<vwInfo.width) XPutPixel(imageX,i,0,pixColor);
                } else {
                  GSetForeground(GC9991, psBlack, dFlag );
                  GDrawPoint(viewport->viewWindow, GC9991, i,scanline,dFlag);
                }
              }
            }
            /* include axes info */
            if (viewport->axesOn) {
              if (getCBufferAxes(i) == 'a') {
                if (dFlag == Xoption) {
                  if (mono || (viewport->monoOn)) pixColor = foregroundColor;
                  else pixColor = monoColor(axesColor);
                  if (i >=0 && i<vwInfo.width)  XPutPixel(imageX,i,0,pixColor);
                } else {
                  GSetForeground(GC9991,psBlack,dFlag);
                  GDrawPoint(viewport->viewWindow, GC9991, i,scanline,dFlag);
                }
              } /* if buffer slot is an axes point */
              tempA = getCBufferAxes(i);
            } else tempA = '0';   /* else axes not on */

            if (getCBufferIndx(i) >= 0 && (tempA == '0')) {
              if (dFlag == Xoption) {
                GSetForeground(trashGC,(float)getCBufferIndx(i),dFlag);
                pixColor = getCBufferIndx(i);
                if (i >=0 && i<vwInfo.width) XPutPixel(imageX,i,0,pixColor);
              }
```

```
        else {
          GSetForeground(GC9991,(float)getCBufferIndx(i),dFlag);
          GDrawPoint(viewport->viewWindow, GC9991, i,scanline,dFlag);
        }
      }
    } /* for each pixel in scanline */
    if (dFlag == Xoption) {
      XPutImage(dsply,viewport->viewWindow,trashGC,imageX,0,0,0,
                scanline,vwInfo.width,1);
      XPutImage(dsply,viewmap,trashGC,imageX,0,0,0,
                scanline,vwInfo.width,1);
    }
    scanline--;
  } /* while each scanline */
  XDestroyImage(imageX);
}
```

7.3.64   freePolyList

frees up the global scanList l-l

⟨*view3d*⟩+≡

```
void freePolyList(void) {
  polyList *P, *nextP;
  int  i;
  for (i = 0; (i < ARRAY_HEIGHT); i++) {
    P = scanList[i];
    while((P != NIL(polyList))) {
      nextP = P->next;
      free(P);
      P = nextP;
    }
  }
} /* freePolyList() */
```

7.3.65 showAxesLabels

showAxesLabels() writes the axes labels onto the viewmap of a graph.

⟨*view3d*⟩+≡

```
void showAxesLabels(int dFlag) {
  int  xcoord2,ycoord2;
  if (dFlag == Xoption)
    if (mono || (viewport->monoOn))
      GSetForeground(globGC,(float)foregroundColor,dFlag);
    else
      GSetForeground(globGC,(float)monoColor(labelColor),dFlag);
  else GSetForeground(GC9991,psBlack,dFlag);
  /* axes label for X */
  if ((int)axesZ[0][0] >= (int)axesZ[0][2]) {
    if (axesXY[0][2] < axesXY[0][0]) xcoord2 = axesXY[0][2]-5;
    else xcoord2 = axesXY[0][2] + 5;
    if (axesXY[0][3] < axesXY[0][1]) ycoord2 = axesXY[0][3]-5;
    else ycoord2 = axesXY[0][3] + 5;
    if (!viewport->yzOn) {
      if (dFlag == Xoption)
        GDrawString(globGC,viewmap,xcoord2,ycoord2,"X",1,dFlag);
      else
        GDrawString(GC9991,viewport->viewWindow,xcoord2,ycoord2,"X",1,dFlag);
    }
  }
  /* axes label for Y */
  if ((int)axesZ[1][0] >= (int)axesZ[1][1]) {
    if (axesXY[1][2] < axesXY[1][0]) xcoord2 = axesXY[1][2]-5;
    else xcoord2 = axesXY[1][2] + 5;
    if (axesXY[1][3] < axesXY[1][1]) ycoord2 = axesXY[1][3]-5;
    else ycoord2 = axesXY[1][3] + 5;
    if (!viewport->xzOn) {
      if (dFlag == Xoption)
        GDrawString(globGC,viewmap,xcoord2,ycoord2,"Y",1,dFlag);
      else
       GDrawString(GC9991,viewport->viewWindow,xcoord2,ycoord2,"Y",1,dFlag);
    }
  }
  /* axes label for Z */
  if ((int)axesZ[2][0] >= (int)axesZ[2][1]) {
    if (axesXY[2][2] < axesXY[2][0]) xcoord2 = axesXY[2][2]-5;
    else xcoord2 = axesXY[2][2] + 5;
    if (axesXY[2][3] < axesXY[2][1]) ycoord2 = axesXY[2][3]-5;
    else ycoord2 = axesXY[2][3] + 5;
    if (!viewport->xyOn) {
```

```
      if (dFlag == Xoption)
        GDrawString(globGC,viewmap,xcoord2,ycoord2,"Z",1,dFlag);
      else
        GDrawString(GC9991,viewport->viewWindow,xcoord2,ycoord2,"Z",1,dFlag);
    }
  }
}
```

7.3.66   makeTriangle

changeColorMap() modifies the color map for moving in and out of smooth shading.

⟨*view3d*⟩+≡

```
void changeColorMap(void) {
  int        okay, i, hue, *index;
  poly       *cp;
  viewTriple *pt;
  strcpy(control->message,"        Make New Color Map          ");
  writeControlMessage();
  if ((viewport->hueOffset == viewport->hueTop) &&
      !mono && !viewport->monoOn) {
    /* colormap is not an even distribution across spectrum */
    /* see spadcolors.c code to understand why this is done */
    if (viewport->hueTop < 11) smoothHue = viewport->hueTop * 6;
    else
      if (viewport->hueTop > 10 && viewport->hueTop < 16) {
        smoothHue = viewport->hueTop*20 - 140;
      }
      else {
        smoothHue = viewport->hueTop*12 - 12;
      }
    if (redoColor) {
       /* reallocate colormap for new hue */
      redoColor = no;
      if (pixelSetFlag) {
        FreePixels(dsply,colorMap,smoothConst+1);
      }
      okay = makeNewColorMap(dsply,colorMap,smoothHue);
      if (okay) {
        pixelSetFlag = yes;
        smoothError = no; }
      else {
        pixelSetFlag = no;
        smoothError = yes; }
    } /* if redoColor */
  } else {
    redoDither = no;
    if (pixelSetFlag && !mono) {
      FreePixels(dsply,colorMap,smoothConst);
      pixelSetFlag = no;
      redoColor = no;
      multiColorFlag = yes;
    }
    if (!mono && !viewport->monoOn) {
```

```
        cp = quickList;
        while (cp != NIL(poly) && keepDrawingViewport()) {
          for (i = 0, index = cp->indexPtr;
               i < cp->numpts; i++, index++) {
            pt = refPt3D(viewData,*(index));
            /* get hue for each point if multi-dithering is used */
            if (absolute(cp->color) > 1.0)
              hue = floor(absolute(cp->color));
            else
              hue = floor(absolute(cp->color) * viewport->numberOfHues) +
                    viewport->hueOffset;
            pt->sc = (float)hue;
          } /* for each point in polygon */
          cp = cp->next;
        }
      } /* multi-color dither */
    } /* else hueOffset != hueTop */
  }
```

7.3.67 drawPhong

A general routine for displaying a list of polygons using a simple scanline z-buffer algorithm
with phong shading.

⟨*view3d*⟩+≡

```
void drawPhong(int dFlag) {
    poly          *p, *head;
    polyList      *s;
    int           i,j,hue;
    int           *anIndex, redo;
    viewTriple    *aPoint, *polyPt;
    redo = (recalc || redoSmooth);
    if (redo || redoColor || redoDither) {
      rotated = no;  zoomed = no;  translated = no;
      switchedPerspective = no;  changedEyeDistance = no;
      redoSmooth = no;  movingLight = no;
      /* If only a color change don't recalculate polygon info. */
      if (!redo) {
        /* glossy shading if a single hue is indicated */
        changeColorMap();
        scanLines(dFlag);
        /* if axes are on then show axes labels */
        if (viewport->axesOn) showAxesLabels(dFlag);
        /* show pixmap of image */
        XCopyArea(dsply,viewmap,viewport->viewWindow,trashGC,0,0,
                  vwInfo.width,vwInfo.height,0,0);
      } else {
        if (keepDrawingViewport()) {
          if (!firstTime && !(scanline > 0)) {
            strcpy(control->message,"         Freeing Polygons         ");
            writeControlMessage();
            freeListOfPolygons(quickList);
            freePointResevoir();
          }
          if (keepDrawingViewport()) {
            strcpy(control->message,"         Collecting Polygons         ");
            writeControlMessage();
            quickList = copyPolygons(viewData.polygons);
            if (keepDrawingViewport()) {
              strcpy(control->message,"         Projecting Polygons         ");
              writeControlMessage();
              projectAllPolys(quickList);
              if (keepDrawingViewport()) {
                strcpy(control->message,
                       "         Setting Polygon Extremes         ");
```

```
                      writeControlMessage();
                      minMaxPolygons(quickList);
                      if (keepDrawingViewport()) {
                        strcpy(control->message,
                               "          Sorting Polygons          ");
                        writeControlMessage();
                        quickList = msort(quickList,0,viewData.numPolygons,
                                          polyCompare);
                        calcEyePoint();
                        head = p = quickList;

                        /* glossy shading if a single hue is indicated */
                        changeColorMap();

                        for (i=0, aPoint=viewData.points;
                             i<viewData.numOfPoints; i++,aPoint++) {
                          aPoint->norm[0]= 0.0;
                          aPoint->norm[1]= 0.0;
                          aPoint->norm[2]= 0.0;
                        }
                        freePolyList();
                        for (i = 0; i < ARRAY_HEIGHT; i++)
                          scanList[i] = NIL(polyList);
                        /* for each polygon  */
                        /* calculate average normal for each vertex  */
                        strcpy(control->message,
                               "          Build Polygon Lists          ");
                        writeControlMessage();
                        p = head;
                        while ((p != NIL(poly)) && keepDrawingViewport()) {
                          for (j = 0, anIndex = p->indexPtr;
                               j < p->numpts; j++, anIndex++) {
                            polyPt = refPt3D(viewData,*(anIndex));
                            polyPt->norm[0] += p->N[0];
                            polyPt->norm[1] += p->N[1];
                            polyPt->norm[2] += p->N[2];
                            normalizeVector(polyPt->norm);
                            /* get hue for each point if multi-dithering is used */
                            if ((viewport->hueOffset != viewport->hueTop ||
                                 smoothError) && !mono) {
                              if (absolute(p->color) > 1.0) {
                                hue = floor(absolute(p->color));
                              } else {
                                hue = floor(absolute(p->color) *
                                            viewport->numberOfHues) +
                                            viewport->hueOffset;
```

```
                                  }
                                  polyPt->sc = (float)hue;
                                } /* multi-color dither */
                              } /* for each point in polygon */
                              if ( ! ( p->partialClipPz ||
                                       p->totalClipPz   ||
                                       (viewData.clipStuff &&
                                        (p->partialClip  || p->totalClip ) ) ) ) {
                                /* put polygon in each scanline list it intersects */
                                for (i=(int)p->pymin; i<= (int)p->pymax; i++) {
                                  if ( (i>=0) && (i<ARRAY_HEIGHT ) ){
                                    s = (polyList *)
                                            saymem("smoothShade.c",1,sizeof(polyList));
                                    s->polyIndx = p;
                                    s->next = scanList[i];
                                    scanList[i] = s;
                                  }
                                } /* put polygon in each scanline it intersects */
                              } /* if polygon not clipped */
                              p = p->next;
                            } /* while still polygons */
                            scanLines(dFlag);
                            /* if axes are on then show axes labels */
                            if (viewport->axesOn) showAxesLabels(dFlag);
                            /* show pixmap of image */
                            XCopyArea(dsply,viewmap,viewport->viewWindow,trashGC,0,0,
                                    vwInfo.width,vwInfo.height,0,0);
                            /* freePolyList(scanList);    */
                          } /* keepDrawingViewport() after setting extreme values */
                        } /* keepDrawingViewport() after projecting all polygons */
                      } /* keepDrawingViewport() after collecting polygons */
                    } /* keepDrawingViewport() after freeing polygons */
                  } /* keepDrawingViewport() after recalc */
                  finishedList = !(scanline>0);
                  if (firstTime) firstTime = no;
                } /* not only a color change */
              } else { /* else just redisplay current pixmap of image */
                XCopyArea(dsply,viewmap,viewport->viewWindow,trashGC,0,0,
                          vwInfo.width,vwInfo.height,0,0);
              }
              clearControlMessage();
              strcpy(control->message,viewport->title);
              writeControlMessage();
            } /* drawPhong */
```

7.3.68   readViewman

⟨*view3d*⟩+≡
```
  int readViewman(void *info,int size) {
    int m = 0;
    sprintf(errorStr,"%s","read from viewport manager\n");
    m = check(read( 0, info, size));
    return(m);
  }
```

7.3.69   scalePoint

⟨*view3d*⟩+≡
```
  void scalePoint(viewTriple *p) {
    p->x *= viewData.scaleToView;
    p->y *= viewData.scaleToView;
    p->z *= viewData.scaleToView;
    if (viewData.cmin != viewData.cmax)
      p->c = (p->c - viewData.cmin)/(viewData.cmax-viewData.cmin);
    if (p->c > 1.0) p->c = 1.0;
    else if (p->c < 0) p->c = 0.0;
  } /* scalePoint */
```

7.3.70 spadAction

⟨*view3d*⟩+≡

```
int spadAction(void) {
  int code, viewCommand;
  float f1, f2, f3;
  int i1, i2, i3;
  if (viewAloned==yes) {
      close(0);
      return(-1);
      }
  readViewman(&viewCommand, intSize);
  switch (viewCommand) {
  case rotate:
    readViewman(&f1, floatSize);
    readViewman(&f2, floatSize);
    viewport->theta = f1;
    viewport->phi   = f2;
    while (viewport->theta >= two_pi) viewport->theta -= two_pi;
    while (viewport->theta < 0.0)     viewport->theta += two_pi;
    while (viewport->phi > pi)        viewport->phi   -= two_pi;
    while (viewport->phi <= -pi)      viewport->phi   += two_pi;
    viewport->axestheta = viewport->theta;
    viewport->axesphi = viewport->phi;
    spadDraw=yes;
    rotated=yes;
    viewport->yzOn = viewport->xzOn = viewport->xyOn = no;
    break;
  case zoom:
    readViewman(&f1, floatSize);
    viewport->scale = f1;
    if (viewport->scale > maxScale) viewport->scale = maxScale;
    else if (viewport->scale < minScale) viewport->scale = minScale;
    spadDraw=yes;
    zoomed = yes;
    break;
  case zoomx:
    readViewman(&f1, floatSize);
    readViewman(&f2, floatSize);
    readViewman(&f3, floatSize);
    viewport->scaleX = f1;  viewport->scaleY = f2;  viewport->scaleZ = f3;
    if ((viewport->scaleX == 1.0) &&
        (viewport->scaleY == 1.0) &&
        (viewport->scaleZ == 1.0)) {
      viewport->zoomXOn = viewport->zoomYOn = viewport->zoomZOn = yes;
    } else {
```

```
      if (viewport->scaleX == 1.0) viewport->zoomXOn = no;
      else {
        if (viewport->scaleX > maxScale) viewport->scaleX = maxScale;
        else if (viewport->scaleX < minScale) viewport->scaleX = minScale;
      }
      if (viewport->scaleY == 1.0) viewport->zoomYOn = no;
      else {
        if (viewport->scaleY > maxScale) viewport->scaleY = maxScale;
        else if (viewport->scaleY < minScale) viewport->scaleY = minScale;
      }
      if (viewport->scaleZ == 1.0) viewport->zoomZOn = no;
      else {
        if (viewport->scaleZ > maxScale) viewport->scaleZ = maxScale;
        else if (viewport->scaleZ < minScale) viewport->scaleZ = minScale;
      }
    }
    spadDraw=yes;
    zoomed = yes;
    break;
  case translate:
    readViewman(&(viewport->deltaX),floatSize);
    readViewman(&(viewport->deltaY),floatSize);
    if (viewport->deltaX > maxDeltaX) viewport->deltaX = maxDeltaX;
    else if (viewport->deltaX < -maxDeltaX) viewport->deltaX = -maxDeltaX;
    if (viewport->deltaY > maxDeltaY) viewport->deltaY = maxDeltaY;
    else if (viewport->deltaY < -maxDeltaY) viewport->deltaY = -maxDeltaY;
    spadDraw=yes;
    translated = yes;
    break;
  case modifyPOINT:
    readViewman(&i1,intSize);
    i1--;
    readViewman(&(refPt3D(viewData,i1)->x),floatSize);
    readViewman(&(refPt3D(viewData,i1)->y),floatSize);
    readViewman(&(refPt3D(viewData,i1)->z),floatSize);
    readViewman(&(refPt3D(viewData,i1)->c),floatSize);
    scalePoint(refPt3D(viewData,i1));
    spadDraw=yes;
    break;
  case hideControl:
    readViewman(&i1,intSize);
    if (i1) {                             /* show control panel */
      if (viewport->haveControl)
        XUnmapWindow(dsply,control->controlWindow);
      putControlPanelSomewhere(someInt);
    } else {    /* turn off control panel */
```

```
      if (viewport->haveControl) {
        viewport->haveControl = no;
        XUnmapWindow(dsply,control->controlWindow);
      }
    }
    break;
  case axesOnOff:
    readViewman(&i1,intSize);
    viewport->axesOn = i1;
    spadDraw=yes;
    if (viewData.style == smooth) {
      if (multiColorFlag) redoDither = yes;
      else redoColor = yes;
    }
    if (viewport->haveControl) drawControlPanel();
    break;
/* Non-uniform scaling is not in Axiom yet. */
/* Neither is object or origin rotation. */
  case perspectiveOnOff:
    readViewman(&i1,intSize);
    viewData.perspective = i1;
    switchedPerspective = yes;
    spadDraw=yes;
    break;
  case region3D:
    readViewman(&i1,intSize);
    viewport->regionOn = i1;
    viewData.box = i1;
    spadDraw=yes;
    if (viewport->haveControl) drawControlPanel();
    redoSmooth = yes;
    break;
  case clipRegionOnOff:
    readViewman(&i1,intSize);
    viewData.clipbox = i1;
    spadDraw=yes;
    break;
  case clipSurfaceOnOff:
    readViewman(&i1,intSize);
    viewData.clipStuff = i1;
    spadDraw=yes;
    break;
  case eyeDistanceData:
    readViewman(&f1,floatSize);
    viewData.eyeDistance = f1;
    if (viewData.eyeDistance > maxEyeDistance)
```

```
       viewData.eyeDistance = maxEyeDistance;
     else if (viewData.eyeDistance < minEyeDistance)
           viewData.eyeDistance = minEyeDistance;
     spadDraw=yes;
     changedEyeDistance = yes;
     break;
  case hitherPlaneData:
     readViewman(&f1,floatSize);
     viewData.clipPlane = f1;
     spadDraw=yes;
     changedEyeDistance = yes;
     break;
  case colorDef:
     readViewman(&(viewport->hueOffset),intSize);
     readViewman(&(viewport->numberOfHues),intSize);
     /* spadcolors is indexed by 0 */
     viewport->hueOffset --;
     viewport->numberOfHues --;
     viewport->hueTop = viewport->numberOfHues;
     if (viewport->hueOffset < 0) viewport->hueOffset = 0;
     if (viewport->hueTop < 0) viewport->hueTop = 0;
     if (viewport->hueOffset >= totalHues)
       viewport->hueOffset = totalHues-1;
     if (viewport->hueTop >= totalHues) viewport->hueTop = totalHues-1;
     viewport->numberOfHues = viewport->hueTop - viewport->hueOffset;
     if ((viewport->hueTop == viewport->hueOffset) && (!viewport->monoOn))
       redoColor = yes;
     else {
       redoColor = no;
       redoDither = yes;
     }
     if (viewport->haveControl) drawColorMap();
     break;
  case closeAll:
     code = check(write(Socket,&ack,intSize));
     goodbye(-1);
  case moveViewport:
     readViewman(&i1,intSize);
     readViewman(&i2,intSize);
     XMoveWindow(dsply,viewport->titleWindow,i1,i2);
     XSync(dsply,0);
     break;
  case resizeViewport:
     readViewman(&i1,intSize);
     readViewman(&i2,intSize);
     XResizeWindow(dsply,viewport->titleWindow,i1,i2+titleHeight);
```

```
      XResizeWindow(dsply,viewport->viewWindow,i1,i2);
      spadDraw=yes;
      redoSmooth =yes;
      break;
    case transparent:
    case opaqueMesh:
    case render:
    case smooth:
      viewData.style = viewCommand;
      spadDraw=yes;
      redoSmooth =yes;
      break;
    case lightDef:
      readViewman(&(viewport->lightVector[0]),floatSize);
      readViewman(&(viewport->lightVector[1]),floatSize);
      readViewman(&(viewport->lightVector[2]),floatSize);
      normalizeVector(viewport->lightVector);
      movingLight = yes;
      drawLightingAxes();
      XSync(dsply,0);
      break;
    case translucenceDef:
      readViewman(&backLightIntensity,floatSize);
      tempLightIntensity = backLightIntensity;
      lightIntensity = tempLightIntensity;
      changedIntensity = yes;
      drawLightTransArrow();
      XSync(dsply,0);
      break;
    case changeTitle:
      readViewman(&i1,intSize);
      readViewman(viewport->title,i1);
      viewport->title[i1] = '\0';
      writeTitle();
      switch (doingPanel) {
      case CONTROLpanel:
      case CONTOURpanel:
        writeControlTitle(control->controlWindow);
        break;
      case VOLUMEpanel:
        writeControlTitle(volumeWindow);
        break;
      case LIGHTpanel:
        writeControlTitle(lightingWindow);
        break;
      } /* switch */
```

```
    XFlush(dsply);
    break;
  case writeView:
    readViewman(&i1,intSize);
    readViewman(filename,i1);
    filename[i1] = '\0';
    sprintf(errorStr,"writing of viewport data");
    i3 = 0;
    readViewman(&i2,intSize);
    while (i2) {
      i3 = i3 | (1<<i2);
      readViewman(&i2,intSize);
    }
    if (writeViewport(i3) < 0)
      fprintf(stderr,"          Nothing was written\n");
    break;
  case diagOnOff:
    readViewman(&i1,intSize);
    if (viewData.outlineRenderOn) {
      viewport->diagonals = i1;
      spadDraw=yes;
    } else {
      strcpy(control->message,"  Use this option with Outline  ");
      writeControlMessage();
    }
    break;
  case outlineOnOff:
    readViewman(&i1,intSize);
    if (viewData.style == render) {
      viewData.outlineRenderOn = i1;
      spadDraw=yes;
      if (viewport->haveControl) drawControlPanel();
    } else {
      strcpy(control->message," Use this option in Shaded mode ");
      writeControlMessage();
    }
    break;
  case spadPressedAButton:
    readViewman(&i1,intSize);
    buttonAction(i1);
    break;
  default:
    return(-1);
  } /* switch */
  ack++;
  code = check(write(Socket,&ack,intSize));
```

```
    return(0);
  }
```

7.3.71   traverse

Returns the nth point in a point resevoir.

⟨*view3d*⟩+≡
```
  viewTriple *traverse(int n) {
    int i;
    viewTriple *v;
    v = splitPoints;
    for (i=0; i<n; i++) v = v->next;
    return(v);
  } /* traverse */
```

7.3.72   absolute

⟨*view3d*⟩+≡
```
  float absolute(float x) {
    if (x<0.0) return(-x);
    else return(x);
  }
```

7.3.73   getRandom

⟨*view3d*⟩+≡
```
  float getRandom(void) {
    float x;
    x = (float)(rand() % 100);
    return(x);
  }
```

7.3.74   normDist

⟨*view3d*⟩+≡

```
triple normDist(void) {
  float   u1, u2, v1, v2, ss, rad;
  triple  pert  = {0,0,0};
  ss = 2.0;
  while (ss >= 1.0) {
    u1 = getRandom()/100.0;
    u2 = getRandom()/100.0;
    v1 = 2.0*u1 - 1.0;  v2 = 2.0*u2 - 1.0;
    ss = v1*v1 + v2*v2;
  }
  if (ss == 0.0) ss += .1;
  rad = -2.0*log(ss)/ss;
  pert.x = v1 * sqrt(rad);
  pert.y = v2 * sqrt(rad);
  return(pert);
}
```

7.3.75   goodbye

⟨*view3d*⟩+≡

```
void goodbye(int sig) {
  int Command;
  PSClose(); /* free PS file and data structure space */
  if (pixelSetFlag) FreePixels(dsply,colorMap,smoothConst);
  if (!viewAloned) {
    Command = viewportClosing;
    check(write(Socket,&Command,intSize));
    }
  XCloseDisplay(dsply);
  exit(0);
}            /* goodbye */
```

7.3.76   drawLineComponent

⟨*view3d*⟩+≡

```
void drawLineComponent(poly * p, int dFlag) {
  int i, hue;
  int *anIndex;
  RGB col_rgb;
  /* If the polygon is clipped against the hither plane (clipPz) then
     it is not drawn...or...if the clipStuff is set to true, and
     the polygon is clipped against the user defined clip volume, it
     is also not drawn. */
  if (!((p->partialClipPz) || (viewData.clipStuff && (p->partialClip)))) {
    /* This routine will eventually only be skipped if
       p->totalClip is true and another routine would
       handle the partialClip. this routine would handle
       only those polygons without any clipped points */
    for (i=0, anIndex=p->indexPtr; i<p->numpts; i++,anIndex++) {
      quadMesh[i].x = refPt3D(viewData,*anIndex)->px;
      quadMesh[i].y = refPt3D(viewData,*anIndex)->py;
    }
    if (dFlag==Xoption) {
      if (mono || viewport->monoOn)
        GSetForeground(opaqueGC, (float)foregroundColor, dFlag);
      else {
        hue = hueValue(p->color);
        GSetForeground(opaqueGC, (float)XSolidColor(hue,2), dFlag);
      }
    } else
      GSetForeground(opaqueGC, psBlack, dFlag);
    if (dFlag==PSoption && !mono && !viewport->monoOn) {
      hue = getHue(p->color);
      col_rgb = hlsTOrgb((float)hue,0.5,0.8);
      PSDrawColor(col_rgb.r,col_rgb.g,col_rgb.b,quadMesh,p->numpts);
    } else {
      GDrawLines(opaqueGC, viewport->viewWindow, quadMesh, p->numpts,
                 CoordModeOrigin, dFlag);
    }
    if (dFlag == Xoption)
      XMapWindow(dsply, viewport->viewWindow);
  }
}
```

7.3.77   drawOpaquePolygon

⟨*view3d*⟩+≡

```
void drawOpaquePolygon(poly *p,GC aGC,GC anotherGC,int dFlag) {
  int *anIndex, i, hue, isNaN = 0;
  RGB col_rgb;
  if (mono || viewport->monoOn) {
    GSetForeground(anotherGC, (float)foregroundColor, dFlag);
  } else {
    hue = hueValue(p->color);
    GSetForeground(anotherGC, (float)XSolidColor(hue,2), dFlag);
  }
  /* If the polygon is clipped against the hither plane (clipPz) then
     it is not drawn, or if the clipStuff is set to true, and
     the polygon is clipped against the user defined clip volume, it
     is also not drawn. */
  if (!((p->partialClipPz) || (viewData.clipStuff && (p->partialClip)))) {
    /* This routine should eventually only be skipped if
       p->totalClip is true and another routine would
       handle the partialClip. This routine would handle
       only those polygons without any clipped points. */
    for (i=0, anIndex=p->indexPtr; i<p->numpts; i++,anIndex++) {
      quadMesh[i].x = refPt3D(viewData,*anIndex)->px;
      quadMesh[i].y = refPt3D(viewData,*anIndex)->py;
      if (eqNANQ(quadMesh[i].x) || eqNANQ(quadMesh[i].y)) isNaN = 1;
    }
    quadMesh[i].x =refPt3D(viewData,*(p->indexPtr))->px;
    quadMesh[i].y =refPt3D(viewData,*(p->indexPtr))->py;
    if (eqNANQ(quadMesh[i].x) || eqNANQ(quadMesh[i].y)) isNaN = 1;
    if (dFlag==PSoption && !mono && !viewport->monoOn && !isNaN) {
      GSetForeground(GC9991, (float)backgroundColor, PSoption);
      PSFillPolygon(GC9991, quadMesh, p->numpts+1);
      hue = getHue(p->color);
      col_rgb = hlsTOrgb((float)hue,0.5,0.8);
      if (viewport->diagonals)
        PSDrawColor(col_rgb.r,col_rgb.g,col_rgb.b,quadMesh,p->numpts+1);
      else
        PSDrawColor(col_rgb.r,col_rgb.g,col_rgb.b,quadMesh,p->numpts);
    } else {
      if (mono || viewport->monoOn) {
        GSetForeground(anotherGC, (float)foregroundColor, dFlag);
      } else {
        hue = hueValue(p->color);
        GSetForeground(anotherGC, (float)XSolidColor(hue,2), dFlag);
      }
      GSetForeground(aGC,(float)backgroundColor,dFlag);
```

```
      if (!isNaN) {
        XFillPolygon(dsply, viewport->viewWindow, aGC, quadMesh, p->numpts,
                     Convex,CoordModeOrigin);
        if (viewport->diagonals)
          GDrawLines(anotherGC,viewport->viewWindow,quadMesh,p->numpts+1,
                     CoordModeOrigin, dFlag);
        else
          GDrawLines(anotherGC,viewport->viewWindow,quadMesh,p->numpts,
                     CoordModeOrigin, dFlag);
      }
    }
    if (dFlag == Xoption) XMapWindow(dsply,viewport->viewWindow);
  }   /* if not totally clipped */
}
```

7.3.78   copyPolygons

Copies the given list of polygons into a newly allocated list

⟨*view3d*⟩+≡
```
poly *copyPolygons(poly *polygonList) {
  int i;
  poly *aPoly,*retval,*prev;
  prev = retval = aPoly = (poly *)saymem("surface.c",1,sizeof(poly));
  aPoly->indexPtr = (int *)saymem("surface.c",
                                  polygonList->numpts,sizeof(int));
  aPoly->num = polygonList->num;
  aPoly->sortNum = polygonList->sortNum;
  aPoly->split = polygonList->split;
  aPoly->numpts = polygonList->numpts;
  for (i=0; i<aPoly->numpts; i++)
    *((aPoly->indexPtr) + i) = *((polygonList->indexPtr) + i);
  aPoly->N[0] = polygonList->N[0];
  aPoly->N[1] = polygonList->N[1];
  aPoly->N[2] = polygonList->N[2];
  aPoly->planeConst = polygonList->planeConst;
  aPoly->color = polygonList->color;
  aPoly->moved = no;
  aPoly->pxmin = polygonList->pxmin;
  aPoly->pxmax = polygonList->pxmax;
  aPoly->pymin = polygonList->pymin;
  aPoly->pymax = polygonList->pymax;
  aPoly->pzmin = polygonList->pzmin;
  aPoly->pzmax = polygonList->pzmax;
  aPoly->xmin = polygonList->xmin;
  aPoly->xmax = polygonList->xmax;
  aPoly->ymin = polygonList->ymin;
  aPoly->ymax = polygonList->ymax;
  aPoly->zmin = polygonList->zmin;
  aPoly->zmax = polygonList->zmax;
  aPoly->normalFacingOut = polygonList->normalFacingOut;
  aPoly->primitiveType = polygonList->primitiveType;
  for (polygonList = polygonList->next;
       polygonList != NIL(poly);
       polygonList = polygonList->next) {
    prev->next = aPoly = (poly *)saymem("surface.c",1,sizeof(poly));
    aPoly->indexPtr = (int *)saymem("surface.c",
                                    polygonList->numpts,sizeof(int));
    aPoly->num = polygonList->num;
    aPoly->sortNum = polygonList->sortNum;
    aPoly->numpts = polygonList->numpts;
```

```
      aPoly->split = polygonList->split;
      for (i=0; i<aPoly->numpts; i++)
        *((aPoly->indexPtr) + i) = *((polygonList->indexPtr) + i);
      aPoly->N[0] = polygonList->N[0];
      aPoly->N[1] = polygonList->N[1];
      aPoly->N[2] = polygonList->N[2];
      aPoly->planeConst = polygonList->planeConst;
      aPoly->color = polygonList->color;
      aPoly->moved = no;
      aPoly->pxmin = polygonList->pxmin;
      aPoly->pxmax = polygonList->pxmax;
      aPoly->pymin = polygonList->pymin;
      aPoly->pymax = polygonList->pymax;
      aPoly->pzmin = polygonList->pzmin;
      aPoly->pzmax = polygonList->pzmax;
      aPoly->xmin = polygonList->xmin;
      aPoly->xmax = polygonList->xmax;
      aPoly->ymin = polygonList->ymin;
      aPoly->ymax = polygonList->ymax;
      aPoly->zmin = polygonList->zmin;
      aPoly->zmax = polygonList->zmax;
      aPoly->normalFacingOut = polygonList->normalFacingOut;
      aPoly->primitiveType = polygonList->primitiveType;
      prev = aPoly;
    }
    aPoly->next = 0;
    return(retval);
  }
```

7.3.79   minMaxPolygons

Sets up the xmin, etc, for each polygon for sorting and extent checking.

⟨*view3d*⟩+≡

```
void minMaxPolygons(poly *aPoly) {
  int *anIndex;
  int i;
  for (; aPoly != NIL(poly); aPoly = aPoly->next) {
    anIndex = aPoly->indexPtr;
    aPoly->pxmin = aPoly->pxmax = refPt3D(viewData,*anIndex)->px;
    aPoly->pymin = aPoly->pymax = refPt3D(viewData,*anIndex)->py;
    aPoly->pzmin = aPoly->pzmax = refPt3D(viewData,*anIndex)->pz;
    aPoly->xmin = aPoly->xmax = refPt3D(viewData,*anIndex)->x;
    aPoly->ymin = aPoly->ymax = refPt3D(viewData,*anIndex)->y;
    aPoly->zmin = aPoly->zmax = refPt3D(viewData,*anIndex)->z;
    for (i=1,anIndex++; i<aPoly->numpts; i++,anIndex++) {
      if (refPt3D(viewData,*anIndex)->px < aPoly->pxmin)
        aPoly->pxmin = refPt3D(viewData,*anIndex)->px;
      else if (refPt3D(viewData,*anIndex)->px > aPoly->pxmax)
        aPoly->pxmax = refPt3D(viewData,*anIndex)->px;
      if (refPt3D(viewData,*anIndex)->py < aPoly->pymin)
        aPoly->pymin = refPt3D(viewData,*anIndex)->py;
      else if (refPt3D(viewData,*anIndex)->py > aPoly->pymax)
        aPoly->pymax = refPt3D(viewData,*anIndex)->py;
      if (refPt3D(viewData,*anIndex)->pz < aPoly->pzmin)
        aPoly->pzmin = refPt3D(viewData,*anIndex)->pz;
      else if (refPt3D(viewData,*anIndex)->pz > aPoly->pzmax)
        aPoly->pzmax = refPt3D(viewData,*anIndex)->pz;
      if (refPt3D(viewData,*anIndex)->x < aPoly->xmin)
        aPoly->xmin = refPt3D(viewData,*anIndex)->x;
      else if (refPt3D(viewData,*anIndex)->x > aPoly->xmax)
        aPoly->xmax = refPt3D(viewData,*anIndex)->x;
      if (refPt3D(viewData,*anIndex)->y < aPoly->ymin)
        aPoly->ymin = refPt3D(viewData,*anIndex)->y;
      else if (refPt3D(viewData,*anIndex)->y > aPoly->ymax)
        aPoly->ymax = refPt3D(viewData,*anIndex)->y;
      if (refPt3D(viewData,*anIndex)->z < aPoly->zmin)
        aPoly->zmin = refPt3D(viewData,*anIndex)->z;
      else if (refPt3D(viewData,*anIndex)->z > aPoly->zmax)
        aPoly->zmax = refPt3D(viewData,*anIndex)->z;
    }
  }
} /* minMaxPolygons */
```

7.3.80   polyCompare

```
returns -1 if p1 < p2
         0 if p1 = p2
         1 if p1 > p2
```

Note that this is the reverse of what the msort requested. This is so that the list will be sorted from max to min.

⟨*view3d*⟩+≡
```
int polyCompare(poly *p1,poly *p2) {
  if (p1->pzmax > p2->pzmax) return(-1);
  else if (p1->pzmax < p2->pzmax) return(1);
  else return(0);
}
```

7.3.81   makeTriangle

Sets the global variable eyePoint[] to where the viewer is pointed towards.

⟨*view3d*⟩+≡
```
void calcEyePoint(void) {
  eyePoint[0] = sinPhi * (sinTheta);
  eyePoint[1] = sinPhi * (-cosTheta);
  eyePoint[2] = cosPhi;
}
```

7.3.82   makeTriangle

A general routine for displaying a list of polygons with the proper hidden surfaces removed.
Assumes the list of polygons is in viewData.polygons. Needs a routine to split intersecting
polygons in object space. Calculate the color for the polygon p and draw it.

⟨*view3d*⟩+≡

```
void drawRenderedPolygon(poly *p,int dFlag) {
  int        i,hue,shade, isNaN = 0;
  float      whichSide,H[3],P[3],LN,HN,diff,spec,tempLight,lumens,E[3],N[3];
  int        *anIndex, *indx;
  RGB        col_rgb = {0.0,0.0,0.0};
  if (!((p->partialClipPz) || (viewData.clipStuff && (p->partialClip)))) {
      /* This routine should eventually only be skipped if
         p->totalClip is true and another routine would
         handle the partialClip. This routine would handle
         only those polygons without any clipped points. */
    for (i=0, anIndex=p->indexPtr; i<p->numpts; i++,anIndex++) {
      quadMesh[i].x = refPt3D(viewData,*anIndex)->px;
      quadMesh[i].y = refPt3D(viewData,*anIndex)->py;
      if (eqNANQ(quadMesh[i].x) || eqNANQ(quadMesh[i].y)) isNaN = 1;
    }
    quadMesh[i].x = refPt3D(viewData,*(p->indexPtr))->px;
    quadMesh[i].y = refPt3D(viewData,*(p->indexPtr))->py;
    if (eqNANQ(quadMesh[i].x) || eqNANQ(quadMesh[i].y)) isNaN = 1;

    if (!isNaN) {
      /* calculate polygon illumination */
      indx = p->indexPtr;
      P[0] = (refPt3D(viewData,*(indx))->wx +
              refPt3D(viewData,*(indx+1))->wx +
              refPt3D(viewData,*(indx+2))->wx);
      P[1] = (refPt3D(viewData,*(indx))->wy +
              refPt3D(viewData,*(indx+1))->wy +
              refPt3D(viewData,*(indx+2))->wy);
      P[2] = (refPt3D(viewData,*(indx))->wz +
              refPt3D(viewData,*(indx+1))->wz +
              refPt3D(viewData,*(indx+2))->wz);
      normalizeVector(P);
      N[0] = p->N[0];  N[1] = p->N[1];  N[2] = p->N[2];
      normalizeVector(eyePoint);
      E[0] = 4.0*eyePoint[0] - P[0];
      E[1] = 4.0*eyePoint[1] - P[1];
      E[2] = 4.0*eyePoint[2] - P[2];
      normalizeVector(E);
      diff = 0.0;        spec = 0.0;
```

```
       LN = N[0]*viewport->lightVector[0] +
            N[1]*viewport->lightVector[1] +
            N[2]*viewport->lightVector[2];
       if (LN < 0.0) LN = -LN;
       diff = LN*Cdiff;
       if (LN > 0.0) {
         H[0] = E[0] + viewport->lightVector[0];
         H[1] = E[1] + viewport->lightVector[1];
         H[2] = E[2] + viewport->lightVector[2];
         normalizeVector(H);
         HN = dotProduct(N,H,3);
         if (HN < 0.0) HN = -HN;
         spec = pow((double)absolute(HN),coeff);
         if (spec > 1.0) spec = 1.0;
       }
       lumens = ((Camb + 0.15) + diff + spec*Cspec);
       if (lumens > 1.0) lumens = 1.0;
       if (lumens < 0.0) lumens = 0.0;
       if (dFlag==PSoption && !mono && !viewport->monoOn) {
         hue = getHue(p->color);
         col_rgb = hlsTOrgb((float)hue,lumens,0.8);
         /* NTSC color to grey =  .299 red + .587 green + .114 blue */
         maxGreyShade = (int) psShadeMax;
         whichSide = (.299*col_rgb.r + .587*col_rgb.g + .114*col_rgb.b) *
                       (maxGreyShade-1);
       }
       else {
         if (mono || viewport->monoOn) {
           hue = getHue(p->color);
           col_rgb = hlsTOrgb((float)hue,lumens,0.8);
           whichSide = (.299*col_rgb.r + .587*col_rgb.g + .114*col_rgb.b) *
                         (maxGreyShade-1);
         } else
           whichSide = lumens*(totalShades-1);
       }
       tempLight = lightIntensity;
       if (lightIntensity < Camb) lightIntensity = Camb;
       shade = floor(lightIntensity * absolute(whichSide));
       lightIntensity = tempLight;
       if (shade < totalShades) {
         /* shade < totalShades is (temporarily) necessary here
            because, currently, parameterizations for things like
            the sphere would produce triangular shaped polygons
            close to the poles which get triangularized leaving a
            triangle with coincidental points. the normal for this
            would be undefined (since coincidental points would create
```

```
          a zero vector) and the shade would be large, hence,
          the conditional.  */
      if (mono || viewport->monoOn) {
        if (dFlag == Xoption) {
          XChangeShade(dsply,maxGreyShade-shade-1);
          XShadePolygon(dsply,viewport->viewWindow,quadMesh,p->numpts+1,
                        Convex,CoordModeOrigin);
        }
        else if (dFlag == PSoption) { /* renderGC has number 9991
                                         (see main.c, header.h) */
          GSetForeground(GC9991,
                         1.0-(float)(maxGreyShade-shade-1)*psShadeMul,
                         PSoption);
          PSFillPolygon(GC9991, quadMesh, p->numpts+1);
        }
      } else { /* not mono */
        if (dFlag == Xoption) {
          hue = hueValue(p->color);
          XSpadFillPolygon(dsply, viewport->viewWindow, quadMesh,
                           p->numpts+1, Convex,CoordModeOrigin, hue, shade);
        }
        else if (dFlag == PSoption)  /* draws it out in monochrome  */
          PSColorPolygon(col_rgb.r,col_rgb.g,col_rgb.b,quadMesh,p->numpts+1);
      }        /* if mono-else */
      if (viewData.outlineRenderOn) {
        if (viewport->diagonals) {
          if (dFlag == PSoption) {
            GSetForeground(renderGC,psBlack, dFlag);
            GDrawLines(renderGC,viewport->viewWindow,quadMesh,p->numpts+1,
                       CoordModeOrigin,dFlag);
          } else
            GDrawLines(renderGC,viewport->viewWindow,quadMesh,p->numpts+1,
                       CoordModeOrigin,dFlag);
        } else {
          if (dFlag == PSoption) {
            GSetForeground(renderGC,psBlack,PSoption);
            GDrawLines(renderGC,viewport->viewWindow,quadMesh,p->numpts,
                       CoordModeOrigin,PSoption);
          } else
            GDrawLines(renderGC,viewport->viewWindow,quadMesh,p->numpts,
                       CoordModeOrigin,dFlag);
        }
      }
    }
  } /* if not NaN */
  if (dFlag == Xoption) XMapWindow(dsply,viewport->viewWindow);
```

```
    }  /* if not clipped */
  } /* drawRenderedPolygon */
```

7.3.83 freePointResevoir

⟨*view3d*⟩+≡
```
  void freePointResevoir(void) {
    viewTriple *v;
    while (splitPoints != NIL(viewTriple)) {
      v = splitPoints;
      splitPoints = splitPoints->next;
      free(v);
    }
  } /* freePointResevoir */
```

7.3.84 freeListOfPolygons

Frees up a list of polygons.

⟨*view3d*⟩+≡
```
  void freeListOfPolygons(poly *pList) {
    poly *nextP;
    for (; pList != NIL(poly); pList=nextP) {
      nextP=pList->next;
      free(pList->indexPtr);
      free(pList);
    }
  } /* freeListOfPolygons() */
```

7.3.85   drawPolygons

⟨*view3d*⟩+≡
```
void drawPolygons(int dFlag) {
  poly *p,*head;
  poly *tempQuick=NULL;
  int quickFirst=yes;
  if (recalc) {
    /* To get around multiple X Expose events the server tends
       to send upon startup, leave negation of firstTime to the end. */
    rotated = no;
    zoomed = no;
    translated = no;
    switchedPerspective = no;
    changedEyeDistance = no;
    redoSmooth = yes;
    if (keepDrawingViewport()) {
      if (!firstTime) {
        strcpy(control->message,"         Creating Polygons          ");
        writeControlMessage();
        freeListOfPolygons(quickList);
        freePointResevoir();
      }
      strcpy(control->message,"         Collecting Polygons         ");
      writeControlMessage();
      quickList = copyPolygons(viewData.polygons);
      if (keepDrawingViewport()) {
        /* to get normal facing outside info     */
        strcpy(control->message,"         Projecting Polygons         ");
        writeControlMessage();
        projectAllPolys(quickList);
        if (keepDrawingViewport()) {
          strcpy(control->message,"       Setting Extreme Values        ");
          writeControlMessage();
          minMaxPolygons(quickList);
          if (keepDrawingViewport()) {
            strcpy(control->message,"          Sorting Polygons           ");
            writeControlMessage();
            quickList = msort(quickList,0,viewData.numPolygons,polyCompare);
            if (keepDrawingViewport()) {
              calcEyePoint();
              head = p = quickList;
              clearControlMessage();
              strcpy(control->message,viewport->title);
              writeControlMessage();
              if (viewData.scaleDown) {
```

```
                    if (keepDrawingViewport()) {
                      for (p=quickList;
                           keepDrawingViewport() && (p != NIL(poly));
                           p=p->next) {
                        switch (p->primitiveType) {
                        case pointComponent:
                          if (dFlag==Xoption) {
                            if (mono || viewport->monoOn)
                              GSetForeground(componentGC,
                                            (float)foregroundColor, dFlag);
                            else
                              GSetForeground(componentGC,
                                            (float)meshOutline, dFlag);
                          } else {
                            GSetForeground(componentGC, psBlack, dFlag);
                            GFillArc(componentGC, viewport->viewWindow,
                                    (int)refPt3D(viewData,*(p->indexPtr))->px,
                                    (int)refPt3D(viewData,*(p->indexPtr))->py,
                                    viewData.pointSize,viewData.pointSize,0,
                                    360*64, dFlag);
                          }
                          break;
                        case lineComponent:
                          drawLineComponent(p,dFlag);
                          break;
                        default:
                          if (viewData.style == opaqueMesh) {
                            GSetForeground(globGC,(float)backgroundColor,dFlag);
                            drawOpaquePolygon(p,globGC,opaqueGC,dFlag);
                          } else {
                            drawRenderedPolygon(p,dFlag);
                          }
                        } /* switch */
                      }
                    }
                  }
                  if (!quickFirst) {
                    /* append the rest of the polygons onto the list */
                    tempQuick->next = head;
                    /* but do not continue the drawing... */
                    if (head != NIL(poly)) head->doNotStopDraw = no;
                  }  /* if !quickFirst */
                  finishedList = (p==NIL(poly));
              }       /*      for various      */
            }         /*         steps         */
          }           /*          of           */
```

```
    }     /* keepDrawingViewport() */
    }               /*        ***            */
  /* May want to have a flag somewhere to stop the drawing yet
     continue the freeing */
  if (firstTime) firstTime = no;
} else {  /* if recalc else if not recalc just draw stuff in list */
  if (keepDrawingViewport()) {
    for (p=quickList;
         keepDrawingViewport() && p != NIL(poly) &&
           (viewData.scaleDown || p->doNotStopDraw);  p=p->next) {
      projectAPoly(p);
      switch (p->primitiveType) {
      case pointComponent:
        if (dFlag==Xoption) {
          if (mono || viewport->monoOn)
             GSetForeground(componentGC,(float)foregroundColor, dFlag);
          else
             GSetForeground(componentGC,(float)meshOutline, dFlag);
        } else
          GSetForeground(componentGC,psBlack, dFlag);
        GFillArc(componentGC, viewport->viewWindow,
                 (int)refPt3D(viewData,*(p->indexPtr))->px,
                 (int)refPt3D(viewData,*(p->indexPtr))->py,
                 viewData.pointSize,viewData.pointSize,0,360*64,dFlag);
        break;
      case lineComponent:
        drawLineComponent(p,dFlag);
        break;
      default:
        if (viewData.style == opaqueMesh) {
          GSetForeground(globGC,(float)backgroundColor,dFlag);
          drawOpaquePolygon(p,globGC,opaqueGC,dFlag);
        } else
          drawRenderedPolygon(p,dFlag);
      } /* switch */
    }
  }
}
} /* drawPolygons */
```

7.3.86 lessThan

Compares two floating point numbers for precision of up to one place in a thousand. Returns 1 if true 0 otherwise.

⟨*view3d*⟩+≡
```
int lessThan(float x,float y) {
  int xI,yI;
  xI = x*precisionFactor;
  yI = y*precisionFactor;
  return(xI<yI);
}
```

7.3.87 greaterThan

Compares two floating point numbers for precision of up to one place in a thousand. Returns 1 if true 0 otherwise.

⟨*view3d*⟩+≡
```
int greaterThan(float x,float y) {
  int xI,yI;
  xI = x*precisionFactor;
  yI = y*precisionFactor;
  return(xI>yI);
}
```

7.3.88 isNaN

⟨*view3d*⟩+≡
```
int isNaN(float v) {
  return (v != v);
}
```

7.3.89 isNaNPoint

⟨*view3d*⟩+≡
```
int isNaNPoint(float x,float y,float z) {
  return (isNaN(x) || isNaN(y) || isNaN(z));
}
```

7.3.90   equal

$\langle view3d \rangle + \equiv$

```
int equal(float x,float y) {
  int xI,yI;
  xI = x*precisionFactor;
  yI = y*precisionFactor;
  return(xI==yI);
}
```

7.3.91 matrixMultiply4x4

⟨*view3d*⟩+≡
```
void matrixMultiply4x4(float xxA[4][4],float xxB[4][4],float array[4][4]) {
  array[0][0] = xxA[0][0]*xxB[0][0] + xxA[0][1]*xxB[1][0] +
                xxA[0][2]*xxB[2][0] + xxA[0][3]*xxB[3][0];
  array[1][0] = xxA[1][0]*xxB[0][0] + xxA[1][1]*xxB[1][0] +
                xxA[1][2]*xxB[2][0] + xxA[1][3]*xxB[3][0];
  array[2][0] = xxA[2][0]*xxB[0][0] + xxA[2][1]*xxB[1][0] +
                xxA[2][2]*xxB[2][0] + xxA[2][3]*xxB[3][0];
  array[3][0] = xxA[3][0]*xxB[0][0] + xxA[3][1]*xxB[1][0] +
                xxA[3][2]*xxB[2][0] + xxA[3][3]*xxB[3][0];
  array[0][1] = xxA[0][0]*xxB[0][1] + xxA[0][1]*xxB[1][1] +
                xxA[0][2]*xxB[2][1] + xxA[0][3]*xxB[3][1];
  array[1][1] = xxA[1][0]*xxB[0][1] + xxA[1][1]*xxB[1][1] +
                xxA[1][2]*xxB[2][1] + xxA[1][3]*xxB[3][1];
  array[2][1] = xxA[2][0]*xxB[0][1] + xxA[2][1]*xxB[1][1] +
                xxA[2][2]*xxB[2][1] + xxA[2][3]*xxB[3][1];
  array[3][1] = xxA[3][0]*xxB[0][1] + xxA[3][1]*xxB[1][1] +
                xxA[3][2]*xxB[2][1] + xxA[3][3]*xxB[3][1];
  array[0][2] = xxA[0][0]*xxB[0][2] + xxA[0][1]*xxB[1][2] +
                xxA[0][2]*xxB[2][2] + xxA[0][3]*xxB[3][2];
  array[1][2] = xxA[1][0]*xxB[0][2] + xxA[1][1]*xxB[1][2] +
                xxA[1][2]*xxB[2][2] + xxA[1][3]*xxB[3][2];
  array[2][2] = xxA[2][0]*xxB[0][2] + xxA[2][1]*xxB[1][2] +
                xxA[2][2]*xxB[2][2] + xxA[2][3]*xxB[3][2];
  array[3][2] = xxA[3][0]*xxB[0][2] + xxA[3][1]*xxB[1][2] +
                xxA[3][2]*xxB[2][2] + xxA[3][3]*xxB[3][2];
  array[0][3] = xxA[0][0]*xxB[0][3] + xxA[0][1]*xxB[1][3] +
                xxA[0][2]*xxB[2][3] + xxA[0][3]*xxB[3][3];
  array[1][3] = xxA[1][0]*xxB[0][3] + xxA[1][1]*xxB[1][3] +
                xxA[1][2]*xxB[2][3] + xxA[1][3]*xxB[3][3];
  array[2][3] = xxA[2][0]*xxB[0][3] + xxA[2][1]*xxB[1][3] +
                xxA[2][2]*xxB[2][3] + xxA[2][3]*xxB[3][3];
  array[3][3] = xxA[3][0]*xxB[0][3] + xxA[3][1]*xxB[1][3] +
                xxA[3][2]*xxB[2][3] + xxA[3][3]*xxB[3][3];
}
```

7.3.92   vectorMatrix4

⟨*view3d*⟩+≡

```
void vectorMatrix4(float xxD[4],float xxE[4][4],float xxF[4]) {
  xxF[0]=xxD[0]*xxE[0][0]+xxD[1]*xxE[1][0]+xxD[2]*xxE[2][0]+xxD[3]*xxE[3][0];
  xxF[1]=xxD[0]*xxE[0][1]+xxD[1]*xxE[1][1]+xxD[2]*xxE[2][1]+xxD[3]*xxE[3][1];
  xxF[2]=xxD[0]*xxE[0][2]+xxD[1]*xxE[1][2]+xxD[2]*xxE[2][2]+xxD[3]*xxE[3][2];
  xxF[3]=xxD[0]*xxE[0][3]+xxD[1]*xxE[1][3]+xxD[2]*xxE[2][3]+xxD[3]*xxE[3][3];
}
```

7.3.93   ROTATE

⟨*view3d*⟩+≡

```
void ROTATE(float xxR[4][4]) {
  xxR[0][0]= -(cosTheta);
  xxR[0][1]= -(-sinTheta*cosPhi);
  xxR[0][2]= -(sinTheta*sinPhi);
  xxR[0][3]= 0.0;
  xxR[1][0]= -(sinTheta);
  xxR[1][1]= -(cosTheta*cosPhi);
  xxR[1][2]= -(-cosTheta*sinPhi);
  xxR[1][3]= 0.0;
  xxR[2][0]= 0.0;
  xxR[2][1]= -(sinPhi);
  xxR[2][2]= -(cosPhi);
  xxR[2][3]= 0.0;
  xxR[3][0]= 0.0;
  xxR[3][1]= 0.0;
  xxR[3][2]= 0.0;
  xxR[3][3]= -(1.0);
}
```

7.3.94 ROTATE1

⟨*view3d*⟩+≡

```
void ROTATE1(float xxR[4][4]) {
  xxR[0][0]= (cosTheta);
  xxR[0][1]= (-sinTheta*cosPhi);
  xxR[0][2]= (sinTheta*sinPhi);
  xxR[0][3]= 0.0;
  xxR[1][0]= (sinTheta);
  xxR[1][1]= (cosTheta*cosPhi);
  xxR[1][2]= (-cosTheta*sinPhi);
  xxR[1][3]= 0.0;
  xxR[2][0]= 0.0;
  xxR[2][1]= (sinPhi);
  xxR[2][2]= (cosPhi);
  xxR[2][3]= 0.0;
  xxR[3][0]= 0.0;
  xxR[3][1]= 0.0;
  xxR[3][2]= 0.0;
  xxR[3][3]= (1.0);
}
```

7.3.95 SCALE

⟨*view3d*⟩+≡

```
void SCALE(float x,float y,float z,float xxS[4][4]) {
  xxS[0][0] = x;   xxS[0][1] = 0.0; xxS[0][2] = 0.0; xxS[0][3] = 0.0;
  xxS[1][0] = 0.0; xxS[1][1] = y;   xxS[1][2] = 0.0; xxS[1][3] = 0.0;
  xxS[2][0] = 0.0; xxS[2][1] = 0.0; xxS[2][2] = z;   xxS[2][3] = 0.0;
  xxS[3][0] = 0.0; xxS[3][1] = 0.0; xxS[3][2] = 0.0; xxS[3][3] = 1.0;
}
```

7.3.96 TRANSLATE

⟨*view3d*⟩+≡

```
void TRANSLATE(float x,float y,float z,float xxT[4][4]) {
  xxT[0][0] = 1.0; xxT[0][1] = 0.0; xxT[0][2] = 0.0;  xxT[0][3] = 0.0;
  xxT[1][0] = 0.0; xxT[1][1] = 1.0; xxT[1][2] = 0.0;  xxT[1][3] = 0.0;
  xxT[2][0] = 0.0; xxT[2][1] = 0.0; xxT[2][2] = -1.0; xxT[2][3] = 0.0;
  xxT[3][0] = x;   xxT[3][1] = y;   xxT[3][2] = z;    xxT[3][3] = 1.0;
}
```

7.3.97   writeTitle

Definition for the axes and labels - this is the minimun that will be drawn on the window -
thus allowing the user some idea of the orientation of the coordinate axes when rotating, etc.
The drawing of the mesh is aborted when an appropriate X event occurs. The mesh should
be scaled to the range of [-100..100] in all directions. axisRange defines the range...change
the stuff below if that has changed.

⟨*view3d*⟩+≡
```
  void writeTitle(void) {
    int                  strlength;
    XWindowAttributes    twInfo;
    XGetWindowAttributes(dsply, viewport->titleWindow, &twInfo);
    if (mono || viewport->monoOn)
      GSetForeground(anotherGC, (float)foregroundColor, Xoption);
    else
      GSetForeground(anotherGC, (float)titleColor, Xoption);
    XClearWindow(dsply, viewport->titleWindow);
    strlength = strlen(viewport->title);
    GDrawImageString(anotherGC, viewport->titleWindow,
                     centerX(anotherGC, viewport->title, strlength,
                             twInfo.width), 15,
                     viewport->title, strlength, Xoption);
  }
```

7.3.98 drawPreViewport

Draws the axes and boxes before the actual stuff. All incoming signals should be block and no check for pending X events are made.

⟨*view3d*⟩+≡

```
void drawPreViewport(int dFlag) {
  int i, j, vPx0, vPy0, vPx1, vPy1;
      /* for drawing the box */
  float vPz, absTransX, absTransY;
  XPoint blackbox[3], line[2];
  RGB   axes_rgb, clipbox_rgb, boundbox_rgb;
  axes_rgb.r = 0.8;      axes_rgb.g = 0.6;      axes_rgb.b = 0.2;
  clipbox_rgb.r = 0.4;  clipbox_rgb.g = 0.5;  clipbox_rgb.b = 0.9;
  boundbox_rgb.r = 0.4; boundbox_rgb.g = 0.7; boundbox_rgb.b = 0.9;
  XGetWindowAttributes(dsply, viewport->viewWindow, &vwInfo);
  graphWindowAttrib = vwInfo;
        /* Calculate various factors for use in projection */
        /* Scale so that plot the scaling between the axes remains constant
            and fits within the smaller of the two dimensions. */
  xCenter = vwInfo.width / 2;
  yCenter = vwInfo.height / 2;
  if (vwInfo.height <= vwInfo.width) {
    viewScale = viewport->scale * vwInfo.height / viewHeight;
  }
  else {
    viewScale = viewport->scale * vwInfo.width / viewWidth;
  }
              /* Draw the projected image */
    /** draw the axes without heeding to X interrupts, first **/
  if (dFlag == Xoption)   /* do this for X option only */
    XClearWindow(dsply, viewport->viewWindow);
  sinTheta  = sin(-viewport->axestheta);
  cosTheta  = cos(-viewport->axestheta);
  sinPhi    = sin(viewport->axesphi);
  cosPhi    = cos(viewport->axesphi);
  /* Create transformation matrices */
  ROTATE(R); /* angles theta and phi are global */
  SCALE(viewport->scaleX,viewport->scaleY,viewport->scaleZ,S);
  TRANSLATE(-viewport->deltaX,-viewport->deltaY,0.0,T);
  /**** Pre Draw Routine ****/
  if ((dFlag == PSoption) && (foregroundColor == white)) {
    GSetForeground(globGC,(float)backgroundColor,dFlag);
    blackbox[0].x = vwInfo.width;  blackbox[0].y = vwInfo.height;
    blackbox[1].x = 0;                blackbox[1].y = 0;
    blackbox[2].x = 0;                blackbox[2].y = vwInfo.height;
```

```
    if (viewport->monoOn || mono) {
      PSFillPolygon(globGC, blackbox, 3);
    } else {
      PSColorPolygon(0.0,0.0,0.0,blackbox,4);
    }
    blackbox[0].x = vwInfo.width;  blackbox[0].y = 0;
    blackbox[1].x = 0;             blackbox[1].y = 0;
    blackbox[2].x = vwInfo.width;  blackbox[2].y = vwInfo.height;
    if (viewport->monoOn || mono) {
      PSFillPolygon(globGC, blackbox, 3);
    } else {
      PSColorPolygon(0.0,0.0,0.0,blackbox,4);
    }
  }
}
/*  axes  */
for (i=0; i < 3; i++) {
  projectStuff(axes[i][0],axes[i][1],axes[i][2],&vPx0,&vPy0,&vPz);
  axesXY[i][0] = vPx0; axesXY[i][1] = vPy0; axesZ[i][0] = vPz;
  projectStuff(axes[i][3],axes[i][4],axes[i][5],&vPx1,&vPy1,&vPz);
  axesXY[i][2] = vPx1; axesXY[i][3] = vPy1; axesZ[i][1] = vPz;
  if (viewport->axesOn) {
    if (viewport->monoOn || mono) {
      GSetForeground(globalGC1,(float)foregroundColor,dFlag);
      GSetForeground(globGC,(float)foregroundColor,dFlag);
      GDrawLine(globalGC1,viewport->viewWindow,vPx0,vPy0,vPx1,vPy1,dFlag);
    } else {
      if (dFlag == PSoption) {
        GSetForeground(globGC,(float)foregroundColor,dFlag);
        line[0].x = vPx0;  line[0].y = vPy0;
        line[1].x = vPx1;  line[1].y = vPy1;
        PSDrawColor(axes_rgb.r,axes_rgb.g,axes_rgb.b,line,2);
      } else {
        GSetForeground(globalGC1,(float)monoColor(axesColor),dFlag);
        GSetForeground(globGC,(float)monoColor(labelColor),dFlag);
        GDrawLine(globalGC1,viewport->viewWindow,vPx0,vPy0,vPx1,vPy1,dFlag);
      }
    }
    if (i == 0) {
      if (axesXY[0][2] < axesXY[0][0]) vPx1 -= axesOffset;
      else vPx1 += axesOffset;
      if (axesXY[0][3] < axesXY[0][1]) vPy1 -= axesOffset;
      else vPy1 += axesOffset;
      if (!viewport->yzOn)
        GDrawString(globGC,viewport->viewWindow,vPx1,vPy1,"X",1,dFlag);
    } else {
      if (i == 1) {
```

```
          if (axesXY[1][2] < axesXY[1][0]) vPx1 -= axesOffset;
          else vPx1 += axesOffset;
          if (axesXY[1][3] < axesXY[1][1]) vPy1 -= axesOffset;
          else vPy1 += axesOffset;
          if (!viewport->xzOn)
            GDrawString(globGC,viewport->viewWindow,vPx1,vPy1,"Y",1,dFlag);
        } else {
          if (axesXY[2][2] < axesXY[2][0]) vPx1 -= axesOffset;
          else vPx1 += axesOffset;
          if (axesXY[2][3] < axesXY[2][1]) vPy1 -= axesOffset;
          else vPy1 += axesOffset;
          if (!viewport->xyOn)
            GDrawString(globGC,viewport->viewWindow,vPx1,vPy1,"Z",1,dFlag);
        }
      }
      GSetForeground(globalGC1,(float)monoColor(buttonColor),dFlag);
      GSetForeground(globGC,(float)monoColor(buttonColor),dFlag);
  }      /* if viewport->axesOn */
}
viewport->transX = (viewData.xmax + viewData.xmin)/2.0;
viewport->transY = (viewData.ymax + viewData.ymin)/2.0;
viewport->transZ = (viewData.zmax + viewData.zmin)/2.0;
absTransX = absolute(viewport->transX);
absTransY = absolute(viewport->transY);
if ((absTransX > 0.5) || (absTransY > 0.5)) {
  if (absTransX > absTransY)
    reScale = 50.0 * absTransX / viewData.scaleToView;
  else
    reScale = 50.0 * absTransY / viewData.scaleToView;
  if (reScale < 100.0) reScale = 100.0;
} else {
  reScale = 100.0;
}
sinTheta  = sin(-viewport->thetaObj);
cosTheta  = cos(-viewport->thetaObj);
sinPhi    = sin(viewport->phiObj);
cosPhi    = cos(viewport->phiObj);
ROTATE1(R1);
if (viewport->originFlag) viewport->originFlag = no;
sinTheta  = sin(-viewport->axestheta);
cosTheta  = cos(-viewport->axestheta);
sinPhi    = sin(viewport->axesphi);
cosPhi    = cos(viewport->axesphi);
ROTATE(R);
    /*  region box */
if (viewData.clipbox) {
```

```
clipCorners[0].x = viewData.clipXmin;
clipCorners[0].y = viewData.clipYmin;
clipCorners[0].z = viewData.clipZmin;
clipCorners[1].x = viewData.clipXmax;
clipCorners[1].y = viewData.clipYmin;
clipCorners[1].z = viewData.clipZmin;
clipCorners[2].x = viewData.clipXmax;
clipCorners[2].y = viewData.clipYmin;
clipCorners[2].z = viewData.clipZmax;
clipCorners[3].x = viewData.clipXmin;
clipCorners[3].y = viewData.clipYmin;
clipCorners[3].z = viewData.clipZmax;
clipCorners[4].x = viewData.clipXmin;
clipCorners[4].y = viewData.clipYmax;
clipCorners[4].z = viewData.clipZmin;
clipCorners[5].x = viewData.clipXmax;
clipCorners[5].y = viewData.clipYmax;
clipCorners[5].z = viewData.clipZmin;
clipCorners[6].x = viewData.clipXmax;
clipCorners[6].y = viewData.clipYmax;
clipCorners[6].z = viewData.clipZmax;
clipCorners[7].x = viewData.clipXmin;
clipCorners[7].y = viewData.clipYmax;
clipCorners[7].z = viewData.clipZmax;
GSetLineAttributes(trashGC,0,LineSolid,CapButt,JoinMiter,dFlag);
/* project the 8 corners of the box */
for (i=0;i<8;i++) projectAPoint(&(clipCorners[i]));
for (i=0;i<6;i++) {
   clipBox[i].inside = ((clipBox[i].pointsPtr[2]->px -
                         clipBox[i].pointsPtr[1]->px) *
                        (clipBox[i].pointsPtr[1]->py -
                         clipBox[i].pointsPtr[0]->py) -
                        (clipBox[i].pointsPtr[2]->py -
                         clipBox[i].pointsPtr[1]->py) *
                        (clipBox[i].pointsPtr[1]->px -
                         clipBox[i].pointsPtr[0]->px)) < 0;
  if (clipBox[i].inside) {
     for (j=0; j<3; j++) {
         quadMesh[j].x = clipBox[i].pointsPtr[j]->px;
         quadMesh[j].y = clipBox[i].pointsPtr[j]->py;
     }
     if (viewport->monoOn || mono) {
       GSetForeground(trashGC,(float)foregroundColor,dFlag);
       GDrawLines(trashGC, viewport->viewWindow, quadMesh, 3,
                  CoordModeOrigin, dFlag);
     } else {
```

```
            if (dFlag == PSoption) {
              GSetForeground(trashGC,(float)clipBoxInline, dFlag);
              line[0].x = quadMesh[0].x;   line[0].y = quadMesh[0].y;
              line[1].x = quadMesh[1].x;   line[1].y = quadMesh[1].y;
              PSDrawColor(clipbox_rgb.r,clipbox_rgb.g,clipbox_rgb.b,line,2);
              line[0].x = quadMesh[1].x;   line[0].y = quadMesh[1].y;
              line[1].x = quadMesh[2].x;   line[1].y = quadMesh[2].y;
              PSDrawColor(clipbox_rgb.r,clipbox_rgb.g,clipbox_rgb.b,line,2);
            } else {
              GSetForeground(trashGC,(float)clipBoxInline, dFlag);
              GDrawLines(trashGC, viewport->viewWindow, quadMesh, 3,
                      CoordModeOrigin, dFlag);
            }
          }
        }
      }
} /* if viewData.clipbox */
    /* VOLUME panel stuff */
if ((doingPanel == VOLUMEpanel) || viewData.box) {
  GSetLineAttributes(trashGC,0,LineSolid,CapButt,JoinMiter,dFlag);
  for (i=0;i<8;i++) {
      /* project the 8 corners of the box */
      projectAPoint(&(corners[i]));
      if (i) {
         if (corners[i].pz > pzMax) pzMax = corners[i].pz;
         else if (corners[i].pz < pzMin) pzMin = corners[i].pz;
      } else
         pzMax = pzMin = corners[i].pz;
  }
  for (i=0;i<6;i++) {
      /* Process the 6 sides of the boxes.
         Here, we calculate, for each side (defined by two segments)
         whether it is facing towards or away from the viewer. if
         facing, away, we draw them first. later we draw the ones
         facing the viewer. (this is a sort of backface removal
         scheme. */
      /* We define the normal vector for the box as vA X vB where
         vA=p2-p0 and vB=p1-p0. All we really care about, though,
         is what sign the normal is (whether it is towards or away
         from the viewer - so we just take the triple product of
         it against the eye vector, which is, conveniently enough,
         simply [0 0 1]. Hence, only the Z component of the
         cross product is calculated. (Actually, we are using the
         projected normal - that's how we are able to use the
         trick of just taking the Z component. */
    box[i].inside = ((box[i].pointsPtr[2]->px -
```

```
                            box[i].pointsPtr[0]->px) *                 /* Ax *  */
                           (box[i].pointsPtr[1]->py -
                            box[i].pointsPtr[0]->py) -                 /* By -  */
                           (box[i].pointsPtr[2]->py -
                            box[i].pointsPtr[0]->py) *                 /* Ay *  */
                           (box[i].pointsPtr[1]->px -
                            box[i].pointsPtr[0]->px))                  /* Bx    */
                           < 0;
        if (box[i].inside) {
           for (j=0; j<3; j++) {
               quadMesh[j].x = box[i].pointsPtr[j]->px;
               quadMesh[j].y = box[i].pointsPtr[j]->py;
           }
           if (viewport->monoOn || mono) {
             GSetForeground(trashGC,(float)foregroundColor,dFlag);
             GDrawLines(trashGC, viewport->viewWindow, quadMesh, 3,
                        CoordModeOrigin, dFlag);
           } else {
             if (dFlag == PSoption) {
               GSetForeground(trashGC,(float)boxInline, dFlag );
               line[0].x = quadMesh[0].x;  line[0].y = quadMesh[0].y;
               line[1].x = quadMesh[1].x;  line[1].y = quadMesh[1].y;
               PSDrawColor(boundbox_rgb.r,boundbox_rgb.g,boundbox_rgb.b,line,2);
               line[0].x = quadMesh[1].x;  line[0].y = quadMesh[1].y;
               line[1].x = quadMesh[2].x;  line[1].y = quadMesh[2].y;
               PSDrawColor(boundbox_rgb.r,boundbox_rgb.g,boundbox_rgb.b,line,2);
             } else {
               GSetForeground(trashGC,(float)boxInline, dFlag );
               GDrawLines(trashGC, viewport->viewWindow, quadMesh, 3,
                          CoordModeOrigin, dFlag);
             }
           }
        }
     }
   }
 } /* if viewData.box */
       /* Write out view data */
 if (dFlag == Xoption) {   /* do this only for X option */
   writeControlMessage();
   XFlush(dsply);
 }
}
```

7.3.99 drawTheViewport

⟨*view3d*⟩+≡

```
void drawTheViewport(int dFlag) {
  int    i,j;
  XPoint line[2];
  RGB    clipbox_rgb, boundbox_rgb;
  clipbox_rgb.r = 0.4;  clipbox_rgb.g = 0.5;  clipbox_rgb.b = 0.9;
  boundbox_rgb.r = 0.4; boundbox_rgb.g = 0.7; boundbox_rgb.b = 0.9;
    /**** Draw Routine ****/
  if (viewport->allowDraw && (doingPanel != VOLUMEpanel)) {
    /* Do not draw the mesh stuff if we're in the process of changing
       the viewing volume; we just want to see the volume */
    /* drawMore allows the drawing to continue if no relevant XEvent occurs */
    drawMore = yes;
    drawMore = keepDrawingViewport();
    draw3DComponents(dFlag);
  } /*if viewport->allowDraw */
    /**** Post Draw Routine ****/
  if (viewData.clipbox) {   /* draw the front 3 lines of region box */
    GSetLineAttributes(trashGC,0,LineSolid,CapButt,JoinMiter,dFlag);
    for (i=0; i<6; i++) {
      if (!(clipBox[i].inside)) {
        for (j=0; j<4; j++) {
          quadMesh[j].x = clipBox[i].pointsPtr[j]->px;
          quadMesh[j].y = clipBox[i].pointsPtr[j]->py;
        }
        if (viewport->monoOn || mono) {
          GSetForeground(trashGC,(float)foregroundColor,dFlag);
          GDrawLines(trashGC, viewport->viewWindow, quadMesh, 3,
                     CoordModeOrigin, dFlag);
        } else {
          if (dFlag == PSoption) {
            GSetForeground(trashGC,(float)boxInline, dFlag );
            line[0].x = quadMesh[0].x;  line[0].y = quadMesh[0].y;
            line[1].x = quadMesh[1].x;  line[1].y = quadMesh[1].y;
            PSDrawColor(clipbox_rgb.r,clipbox_rgb.g,clipbox_rgb.b,line,2);
            line[0].x = quadMesh[1].x;  line[0].y = quadMesh[1].y;
            line[1].x = quadMesh[2].x;  line[1].y = quadMesh[2].y;
            PSDrawColor(clipbox_rgb.r,clipbox_rgb.g,clipbox_rgb.b,line,2);
          } else {
            GSetForeground(trashGC,(float)boxInline, dFlag );
            GDrawLines(trashGC, viewport->viewWindow, quadMesh, 3,
                       CoordModeOrigin, dFlag);
          }
        }
```

```
      }
    }
  }
  if ((doingPanel==VOLUMEpanel) || viewData.box) {
    GSetLineAttributes(trashGC,0,LineSolid,CapButt,JoinMiter,dFlag);
    for (i=0; i<6; i++) {
      if (!(box[i].inside)) {
        for (j=0; j<4; j++) {
          quadMesh[j].x = box[i].pointsPtr[j]->px;
          quadMesh[j].y = box[i].pointsPtr[j]->py;
        }
        if (viewport->monoOn || mono) {
          GSetForeground(trashGC,(float)foregroundColor,dFlag);
          GDrawLines(trashGC, viewport->viewWindow, quadMesh, 3,
                     CoordModeOrigin, dFlag);
        } else {
          if (dFlag == PSoption) {
            GSetForeground(trashGC,(float)boxInline, dFlag );
            line[0].x = quadMesh[0].x;  line[0].y = quadMesh[0].y;
            line[1].x = quadMesh[1].x;  line[1].y = quadMesh[1].y;
            PSDrawColor(boundbox_rgb.r,boundbox_rgb.g,boundbox_rgb.b,line,2);
            line[0].x = quadMesh[1].x;  line[0].y = quadMesh[1].y;
            line[1].x = quadMesh[2].x;  line[1].y = quadMesh[2].y;
            PSDrawColor(boundbox_rgb.r,boundbox_rgb.g,boundbox_rgb.b,line,2);
          } else {
            GSetForeground(trashGC,(float)boxInline, dFlag );
            GDrawLines(trashGC, viewport->viewWindow, quadMesh, 3,
                       CoordModeOrigin, dFlag);
          }
        }
      }
    }
  }
  if (dFlag == Xoption)  /* do this for X option only */
    XFlush(dsply);
  if (smoothError) {
    strcpy(control->message,"Cannot alloc more smooth shades.");
    writeControlMessage();
    smoothError = no;
  }
} /* drawTheViewport */
```

7.3.100 makeViewport

⟨*view3d*⟩+≡

```
viewPoints *makeViewport(void) {
  Pixmap             spadbits,spadmask;
  XSetWindowAttributes viewAttrib;
  XSizeHints         titleSizeHints;
  Window             viewTitleWindow, viewGraphWindow;
  XColor             foreColor, backColor;
          /**** create a viewport ****/
  if (!(viewport = (viewPoints *)saymem("viewport3D.c",
                                   1,sizeof(viewPoints)))) {
    fprintf(stderr,"Ran out of memory trying to create a viewport.\n");
    exitWithAck(RootWindow(dsply,scrn),Window,-1);
  }
  /* Definition of the 4x4 identity matrix. */
  I[0][0] = 1.0; I[0][1] = 0.0; I[0][2] = 0.0; I[0][3] = 0.0;
  I[1][0] = 0.0; I[1][1] = 1.0; I[1][2] = 0.0; I[1][3] = 0.0;
  I[2][0] = 0.0; I[2][1] = 0.0; I[2][2] = 1.0; I[2][3] = 0.0;
  I[3][0] = 0.0; I[3][1] = 0.0; I[3][2] = 0.0; I[3][3] = 1.0;
  viewport->viewportKey  = viewportKeyNum++;
  viewport->nextViewport = 0;
  viewport->prevViewport = 0;
  viewport->deltaX       = viewport->deltaX0 = viewData.deltaX;
  viewport->deltaY       = viewport->deltaY0 = viewData.deltaY;
  viewport->deltaZ       = viewport->deltaZ0 = viewData.deltaZ;
  viewport->scale        = viewport->scale0  = viewData.scale;
  viewport->scaleX       = viewData.scaleX;
  viewport->scaleY       = viewData.scaleY;
  viewport->scaleZ       = viewData.scaleZ;
  viewport->transX       = (viewData.xmax + viewData.xmin)/2.0;
  viewport->transY       = (viewData.ymax + viewData.ymin)/2.0;
  viewport->transZ       = (viewData.zmax + viewData.zmin)/2.0;
  viewport->theta = viewport->axestheta = viewport->theta0 = viewData.theta;
  viewport->phi   = viewport->axesphi   = viewport->phi0   = viewData.phi;
  viewport->thetaObj = 0.0;
  viewport->phiObj   = 0.0;
  strcpy(viewport->title,viewData.title);
  viewport->axesOn       = yes;
  viewport->regionOn     = no;
  viewport->monoOn       = no;
  viewport->zoomXOn      = yes;
  viewport->zoomYOn      = yes;
  viewport->zoomZOn      = yes;
  viewport->originrOn    = yes;
  viewport->objectrOn    = no;
```

```
viewport->originFlag   = no;
viewport->xyOn         = no;
viewport->xzOn         = no;
viewport->yzOn         = no;
viewport->closing      = no;
viewport->allowDraw    = yes;    /*if no, just draw axes the first time */
viewport->needNorm     = yes;
viewport->lightVector[0] = -0.5;
viewport->lightVector[1] = 0.5;
viewport->lightVector[2] = 0.5;
viewport->translucency   = viewData.translucency;
viewport->hueOffset      = viewData.hueOff;
viewport->numberOfHues   = viewData.numOfHues;
viewport->hueTop         = viewData.hueOff + viewData.numOfHues;
if (viewport->hueTop > totalHues-1) viewport->hueTop = totalHues-1;
viewport->diagonals      = viewData.diagonals;
/* make theta in [0..2pi) and phi in (-pi..pi] */
while (viewport->theta >= two_pi) {
  viewport->theta -= two_pi;
}
while (viewport->theta < 0.0) {
  viewport->theta += two_pi;
}
while (viewport->phi > pi) {
  viewport->phi -= two_pi;
}
while (viewport->phi <= -pi) {
  viewport->phi += two_pi;
}
while (viewport->axestheta >= two_pi) {
  viewport->axestheta -= two_pi;
}
while (viewport->axestheta < 0.0) {
  viewport->axestheta += two_pi;
}
while (viewport->axesphi > pi) {
  viewport->axesphi -= two_pi;
}
while (viewport->axesphi <= -pi) {
  viewport->axesphi += two_pi;
}
/* Initialize the rotation matrix about the origin. */
sinTheta  = sin(-viewport->theta);
cosTheta  = cos(-viewport->theta);
sinPhi    = sin(viewport->phi);
cosPhi    = cos(viewport->phi);
```

```
    ROTATE(R);  /* angles theta and phi are global */
    /* Initialize the rotation matrix about the object's center of volume. */
    sinTheta  = sin(-viewport->thetaObj);
    cosTheta  = cos(-viewport->thetaObj);
    sinPhi    = sin(viewport->phiObj);
    cosPhi    = cos(viewport->phiObj);
    ROTATE1(R1);  /* angles theta and phi are global  */
    /* Initialize the non-uniform scaling matrix. */
    SCALE(viewport->scaleX,viewport->scaleY,viewport->scaleZ,S);
    /* Initialize the translation matrix. */
    TRANSLATE(-viewport->deltaX,-viewport->deltaY,0.0,T);
            /**** make the windows for the viewport ****/
    spadbits = XCreateBitmapFromData(dsply,rtWindow,
                                     spadBitmap_bits,
                                     spadBitmap_width,spadBitmap_height);
    spadmask = XCreateBitmapFromData(dsply,rtWindow,
                                     spadMask_bits,
                                     spadMask_width,spadMask_height);
    viewAttrib.background_pixel = backgroundColor;
    viewAttrib.border_pixel = foregroundColor;
    viewAttrib.override_redirect = overrideManager;
    viewAttrib.colormap = colorMap;
    foreColor.pixel = foregroundColor;
    backColor.pixel = backgroundColor;
 /*
    foreColor.pixel = viewCursorForeground;
    backColor.pixel = viewCursorBackground;
 */
    XQueryColor(dsply,colorMap,&foreColor);
    XQueryColor(dsply,colorMap,&backColor);
    viewAttrib.cursor =
      XCreatePixmapCursor(dsply,spadbits,spadmask,&foreColor,
                          &backColor,spadBitmap_x_hot,spadBitmap_y_hot);
    viewAttrib.event_mask = titleMASK;
    if (viewData.vW) {
      titleSizeHints.flags  = PPosition | PSize;
      titleSizeHints.x      = viewData.vX;
      titleSizeHints.y      = viewData.vY;
      titleSizeHints.width  = viewData.vW;
      titleSizeHints.height = viewData.vH;
    } else {    /* ain't gonna allow this for now... */
      titleSizeHints.flags  = PSize;
      titleSizeHints.width  = viewWidth;
      titleSizeHints.height = viewHeight;
    }
    viewTitleWindow = XCreateWindow(dsply                      /* display */,
```

```
                                    rtWindow,                /* parent */
                                    viewData.vX,             /* x */
                                    viewData.vY,             /* y */
                                    viewData.vW,             /* width */
                                    viewData.vH,             /* height */
                                    /* viewBorderWidth+3*/ 0, /* border width */
                                    CopyFromParent,          /* depth */
                                    InputOutput,             /* class */
                                    CopyFromParent,          /* visual */
                                    viewportTitleCreateMASK,/* valuemask */
                                    &viewAttrib              /* attributes */);
    wm_delete_window = XInternAtom(dsply, "WM_DELETE_WINDOW", False);
    (void) XSetWMProtocols(dsply, viewTitleWindow, &wm_delete_window, 1);
    XSetNormalHints(dsply,viewTitleWindow,&titleSizeHints);
    if (strlen(viewport->title) < 30)
      XSetStandardProperties(dsply,viewTitleWindow,"Axiom 3D",viewport->title,
                            None,NULL,0,&titleSizeHints);
    else
      XSetStandardProperties(dsply,viewTitleWindow,"Axiom 3D","3D Axiom Graph",
                            None,NULL,0,&titleSizeHints);
    viewport->titleWindow = viewTitleWindow;
    viewAttrib.event_mask = viewportMASK;
    viewSizeHints.flags   = PPosition | PSize;
    viewSizeHints.x       = -(viewBorderWidth+3);
    viewSizeHints.x       = 0; /* lose this */
    viewSizeHints.y       = titleHeight;
    viewSizeHints.width   = titleSizeHints.width;
    viewSizeHints.height  = titleSizeHints.height-(titleHeight+appendixHeight);
    viewGraphWindow = XCreateWindow(dsply,               /* display */
                                    viewTitleWindow,      /* parent */
                                    viewSizeHints.x,      /* x */
                                    viewSizeHints.y,      /* y */
                                    viewSizeHints.width,  /* width */
                                    viewSizeHints.height, /* height */
                                    /* viewBorderWidth+3*/0, /* border width */
                                    CopyFromParent,       /* depth */
                                    InputOutput,          /* class */
                                    CopyFromParent,       /* visual */
                                    viewportCreateMASK,   /* valuemask */
                                    &viewAttrib           /* attributes */);
    XSetNormalHints(dsply,viewGraphWindow,&viewSizeHints);
    XSetStandardProperties(dsply,viewGraphWindow,"","",None,NULL,0,
                            &viewSizeHints);
    viewport->viewWindow = viewGraphWindow;
    graphWindowAttrib.width = viewSizeHints.width;
    graphWindowAttrib.height = viewSizeHints.height;
```

```
  if (viewport->hueOffset != viewport->hueTop) {
    multiColorFlag = yes;
    redoColor = no;
  } else {
    if (viewport->hueTop < 11)
      smoothHue = viewport->hueTop*6;
    else {
      if (viewport->hueTop > 10 && viewport->hueTop < 16)
        smoothHue = viewport->hueTop*20 - 140;
      else smoothHue = viewport->hueTop*12 - 12;
    }
    redoColor = yes;
  }
            /**** Make the control panel for the viewport. ****/
  XSync(dsply,0);
  control = viewport->controlPanel = makeControlPanel();
  makeLightingPanel();
  makeVolumePanel();
  makeSavePanel();
  makeQuitPanel();
  if ((viewport->haveControl = viewData.showCP))
    putControlPanelSomewhere(anywhere);
  firstTime = yes;
  return(viewport);
} /* makeViewport() */
```

7.3.101   postMakeViewport

Post processing when creating a viewport. Assign min,max values for the box volume.

⟨*view3d*⟩+≡

```
void postMakeViewport(void) {
  corners[0].x = viewData.xmin; corners[0].y = viewData.ymin;
  corners[0].z = viewData.zmin;
  corners[1].x = viewData.xmax; corners[1].y = viewData.ymin;
  corners[1].z = viewData.zmin;
  corners[2].x = viewData.xmax; corners[2].y = viewData.ymin;
  corners[2].z = viewData.zmax;
  corners[3].x = viewData.xmin; corners[3].y = viewData.ymin;
  corners[3].z = viewData.zmax;
  corners[4].x = viewData.xmin; corners[4].y = viewData.ymax;
  corners[4].z = viewData.zmin;
  corners[5].x = viewData.xmax; corners[5].y = viewData.ymax;
  corners[5].z = viewData.zmin;
  corners[6].x = viewData.xmax; corners[6].y = viewData.ymax;
  corners[6].z = viewData.zmax;
  corners[7].x = viewData.xmin; corners[7].y = viewData.ymax;
  corners[7].z = viewData.zmax;
  box[2].pointsPtr[0] = &(corners[0]);
  box[2].pointsPtr[1] = &(corners[1]);
  box[2].pointsPtr[2] = &(corners[2]);
  box[2].pointsPtr[3] = &(corners[3]);
  box[3].pointsPtr[0] = &(corners[1]);
  box[3].pointsPtr[1] = &(corners[5]);
  box[3].pointsPtr[2] = &(corners[6]);
  box[3].pointsPtr[3] = &(corners[2]);
  box[0].pointsPtr[0] = &(corners[4]);
  box[0].pointsPtr[1] = &(corners[7]);
  box[0].pointsPtr[2] = &(corners[6]);
  box[0].pointsPtr[3] = &(corners[5]);
  box[1].pointsPtr[0] = &(corners[0]);
  box[1].pointsPtr[1] = &(corners[3]);
  box[1].pointsPtr[2] = &(corners[7]);
  box[1].pointsPtr[3] = &(corners[4]);
  box[5].pointsPtr[0] = &(corners[3]);
  box[5].pointsPtr[1] = &(corners[2]);
  box[5].pointsPtr[2] = &(corners[6]);
  box[5].pointsPtr[3] = &(corners[7]);
  box[4].pointsPtr[0] = &(corners[0]);
  box[4].pointsPtr[1] = &(corners[4]);
  box[4].pointsPtr[2] = &(corners[5]);
  box[4].pointsPtr[3] = &(corners[1]);
```

```
 /* clip box */
clipBox[0].pointsPtr[0] = &(clipCorners[0]);
clipBox[0].pointsPtr[1] = &(clipCorners[1]);
clipBox[0].pointsPtr[2] = &(clipCorners[2]);
clipBox[0].pointsPtr[3] = &(clipCorners[3]);
clipBox[1].pointsPtr[0] = &(clipCorners[1]);
clipBox[1].pointsPtr[1] = &(clipCorners[5]);
clipBox[1].pointsPtr[2] = &(clipCorners[6]);
clipBox[1].pointsPtr[3] = &(clipCorners[2]);
clipBox[2].pointsPtr[0] = &(clipCorners[4]);
clipBox[2].pointsPtr[1] = &(clipCorners[7]);
clipBox[2].pointsPtr[2] = &(clipCorners[6]);
clipBox[2].pointsPtr[3] = &(clipCorners[5]);
clipBox[3].pointsPtr[0] = &(clipCorners[0]);
clipBox[3].pointsPtr[1] = &(clipCorners[3]);
clipBox[3].pointsPtr[2] = &(clipCorners[7]);
clipBox[3].pointsPtr[3] = &(clipCorners[4]);
clipBox[4].pointsPtr[0] = &(clipCorners[3]);
clipBox[4].pointsPtr[1] = &(clipCorners[2]);
clipBox[4].pointsPtr[2] = &(clipCorners[6]);
clipBox[4].pointsPtr[3] = &(clipCorners[7]);
clipBox[5].pointsPtr[0] = &(clipCorners[0]);
clipBox[5].pointsPtr[1] = &(clipCorners[4]);
clipBox[5].pointsPtr[2] = &(clipCorners[5]);
clipBox[5].pointsPtr[3] = &(clipCorners[1]);
}
```

7.3.102   keepDrawingViewport

⟨*view3d*⟩+≡

```
int keepDrawingViewport(void) {
  XEvent peekEvent;
  int retVal;
  if (XPending(dsply)) {
    XPeekEvent(dsply,&peekEvent);
    if ((((peekEvent.type == Expose) &&
          ((peekEvent.xany).window == viewport->viewWindow)) ||
         ((peekEvent.type == Expose) &&
          ((peekEvent.xany).window == viewport->titleWindow)) ||
         ((peekEvent.type == Expose) &&
          ((peekEvent.xany).window == control->controlWindow)))) {
      retVal = firstTime;
    } else if ((peekEvent.xbutton.type == ButtonRelease) ||
                ((peekEvent.type == LeaveNotify) && !(followMouse)) ||
                ((peekEvent.type == MotionNotify) && !(followMouse)) ||
                (peekEvent.type == ResizeRequest)) {
      XNextEvent(dsply,&peekEvent);
      followMouse = no;
      retVal = yes;
    } else if ((peekEvent.xany).window == (control->buttonQueue[hideControl]).sel
      viewport->haveControl = no;
      XUnmapWindow(dsply,control->controlWindow);
      retVal = yes;
    } else {
      retVal = no;
    }
  } else {
    retVal = !followMouse;
  }
  if (writeImage) retVal = yes;
  drawMore = no;
  return(retVal);
}
```

7.3.103 initVolumeButtons

⟨*view3d*⟩+≡

```
int initVolumeButtons(buttonStruct *volumeButtons) {
  int ii, num = 0;
  ii = volumeReturn;
  volumeButtons[ii].buttonX      = 154;
  volumeButtons[ii].buttonY      = 370;
  volumeButtons[ii].buttonWidth  = 110;
  volumeButtons[ii].buttonHeight = 24;
  volumeButtons[ii].buttonKey    = ii;
  volumeButtons[ii].pot          = no;
  volumeButtons[ii].mask         = buttonMASK;
  volumeButtons[ii].text         = "Return";
  volumeButtons[ii].textColor    = 52;
  volumeButtons[ii].xHalf        = volumeButtons[ii].buttonWidth/2;
  volumeButtons[ii].yHalf        = volumeButtons[ii].buttonHeight/2;
  ++num;
  ii = volumeAbort;
  volumeButtons[ii].buttonX      = 36;
  volumeButtons[ii].buttonY      = 370;
  volumeButtons[ii].buttonWidth  = 110;
  volumeButtons[ii].buttonHeight = 24;
  volumeButtons[ii].buttonKey    = ii;
  volumeButtons[ii].pot          = no;
  volumeButtons[ii].mask         = buttonMASK;
  volumeButtons[ii].text         = "Abort";
  volumeButtons[ii].textColor    = 28;
  volumeButtons[ii].xHalf        = volumeButtons[ii].buttonWidth/2;
  volumeButtons[ii].yHalf        = volumeButtons[ii].buttonHeight/2;
  ++num;
  ii = frustrumBut;
  volumeButtons[ii].buttonX      = frustrumWindowX;
  volumeButtons[ii].buttonY      = frustrumWindowY;
  volumeButtons[ii].buttonWidth  = frustrumWindowWidth;
  volumeButtons[ii].buttonHeight = frustrumWindowHeight;
  volumeButtons[ii].buttonKey    = ii;
  volumeButtons[ii].pot          = yes;
  volumeButtons[ii].mask         = potMASK;
  volumeButtons[ii].text         = "Frustrum Window";
  volumeButtons[ii].textColor    = frustrumColor;
  volumeButtons[ii].xHalf        = volumeButtons[ii].buttonWidth/2;
  volumeButtons[ii].yHalf        = volumeButtons[ii].buttonHeight/2;
  ++num;
  ii = perspectiveBut;
  volumeButtons[ii].buttonX      = toggleX;
```

```
volumeButtons[ii].buttonY       = toggleY;
volumeButtons[ii].buttonWidth   = 10;
volumeButtons[ii].buttonHeight  = 10;
volumeButtons[ii].buttonKey     = ii;
volumeButtons[ii].pot           = no;
volumeButtons[ii].mask          = potMASK;
volumeButtons[ii].text          = "Perspective";
volumeButtons[ii].textColor     = arcColor;
volumeButtons[ii].xHalf         = volumeButtons[ii].buttonWidth/2;
volumeButtons[ii].yHalf         = volumeButtons[ii].buttonHeight/2;
++num;
ii = clipRegionBut;
volumeButtons[ii].buttonX       = toggleX;
volumeButtons[ii].buttonY       = toggleY+20;
volumeButtons[ii].buttonWidth   = 10;
volumeButtons[ii].buttonHeight  = 10;
volumeButtons[ii].buttonKey     = ii;
volumeButtons[ii].pot           = no;
volumeButtons[ii].mask          = potMASK;
volumeButtons[ii].text          = "Show Region";
volumeButtons[ii].textColor     = arcColor;
volumeButtons[ii].xHalf         = volumeButtons[ii].buttonWidth/2;
volumeButtons[ii].yHalf         = volumeButtons[ii].buttonHeight/2;
++num;
ii = clipSurfaceBut;
volumeButtons[ii].buttonX       = toggleX;
volumeButtons[ii].buttonY       = toggleY+40;
volumeButtons[ii].buttonWidth   = 10;
volumeButtons[ii].buttonHeight  = 10;
volumeButtons[ii].buttonKey     = ii;
volumeButtons[ii].pot           = no;
volumeButtons[ii].mask          = potMASK;
volumeButtons[ii].text          = "Clipping On";
volumeButtons[ii].textColor     = arcColor;
volumeButtons[ii].xHalf         = volumeButtons[ii].buttonWidth/2;
volumeButtons[ii].yHalf         = volumeButtons[ii].buttonHeight/2;
++num;
ii = clipXBut;
volumeButtons[ii].buttonX       = clipXButX;
volumeButtons[ii].buttonY       = clipXButY;
volumeButtons[ii].buttonWidth   = majorAxis;
volumeButtons[ii].buttonHeight  = minorAxis;
volumeButtons[ii].buttonKey     = ii;
volumeButtons[ii].pot           = yes;
volumeButtons[ii].mask          = potMASK;
volumeButtons[ii].text          = "Clip X";
```

```
      volumeButtons[ii].textColor    = clipButtonColor;
      volumeButtons[ii].xHalf        = volumeButtons[ii].buttonWidth/2;
      volumeButtons[ii].yHalf        = volumeButtons[ii].buttonHeight/2;
      ++num;
      ii = clipYBut;
      volumeButtons[ii].buttonX      = clipYButX;
      volumeButtons[ii].buttonY      = clipYButY;
      volumeButtons[ii].buttonWidth  = minorAxis;
      volumeButtons[ii].buttonHeight = majorAxis;
      volumeButtons[ii].buttonKey    = ii;
      volumeButtons[ii].pot          = yes;
      volumeButtons[ii].mask         = potMASK;
      volumeButtons[ii].text         = "Clip Y";
      volumeButtons[ii].textColor    = clipButtonColor;
      volumeButtons[ii].xHalf        = volumeButtons[ii].buttonWidth/2;
      volumeButtons[ii].yHalf        = volumeButtons[ii].buttonHeight/2;
      ++num;
      ii = clipZBut;
      volumeButtons[ii].buttonX      = clipZButX;
      volumeButtons[ii].buttonY      = clipZButY;
      volumeButtons[ii].buttonWidth  = midAxis;
      volumeButtons[ii].buttonHeight = midAxis;
      volumeButtons[ii].buttonKey    = ii;
      volumeButtons[ii].pot          = yes;
      volumeButtons[ii].mask         = potMASK;
      volumeButtons[ii].text         = "Clip Z";
      volumeButtons[ii].textColor    = clipButtonColor;
      volumeButtons[ii].xHalf        = volumeButtons[ii].buttonWidth/2;
      volumeButtons[ii].yHalf        = volumeButtons[ii].buttonHeight/2;
      ++num;
      return(num);
   }
```

7.3.104   makeVolumePanel

⟨*view3d*⟩+≡
```
  void makeVolumePanel(void) {
    int i;
    XSetWindowAttributes cwAttrib, controlAttrib;
    XSizeHints sizehint;
    Pixmap volumebits, volumemask;
    XColor foreColor, backColor;
    volumebits = XCreateBitmapFromData(dsply,rtWindow,volumeBitmap_bits,
                                       volumeBitmap_width,volumeBitmap_height);
    volumemask = XCreateBitmapFromData(dsply,rtWindow,volumeMask_bits,
                                       volumeMask_width,volumeMask_height);
    cwAttrib.background_pixel = backgroundColor;
    cwAttrib.border_pixel = foregroundColor;
    cwAttrib.event_mask = volumeMASK;
    cwAttrib.colormap = colorMap;
    cwAttrib.override_redirect = overrideManager;
    foreColor.pixel = volumeCursorForeground;
    XQueryColor(dsply,colorMap,&foreColor);
    backColor.pixel = volumeCursorBackground;
    XQueryColor(dsply,colorMap,&backColor);
    cwAttrib.cursor = XCreatePixmapCursor(dsply,volumebits,volumemask,
                                          &foreColor,&backColor,
                                          volumeBitmap_x_hot,
                                          volumeBitmap_y_hot);
    volumeWindow = XCreateWindow(dsply,control->controlWindow,
                                 -3,-3,controlWidth,controlHeight,3,
                                 CopyFromParent,InputOutput,CopyFromParent,
                                 controlCreateMASK,&cwAttrib);
    sizehint.flags  = USPosition | USSize;
    sizehint.x      = 0;
    sizehint.y      = 0;
    sizehint.width  = controlWidth;
    sizehint.height = controlHeight;
            /*** the None stands for icon pixmap ***/
    XSetNormalHints(dsply,volumeWindow,&sizehint);
    XSetStandardProperties(dsply,volumeWindow,"Volume Panel 3D",
                           "View Volume",None,NULL,0,&sizehint);
      /*** volume frustrum window ***/
     /*** do volume buttons ***/
    initVolumeButtons(control->buttonQueue);
    for (i=volumeButtonsStart; i<(volumeButtonsEnd); i++) {
      controlAttrib.event_mask = (control->buttonQueue[i]).mask;
      (control->buttonQueue[i]).self =
                  XCreateWindow(dsply,volumeWindow,
```

```
                              (control->buttonQueue[i]).buttonX,
                              (control->buttonQueue[i]).buttonY,
                              (control->buttonQueue[i]).buttonWidth,
                              (control->buttonQueue[i]).buttonHeight,
                              0,0,InputOnly,CopyFromParent,
                              buttonCreateMASK,&controlAttrib);
       XMakeAssoc(dsply,table,(control->buttonQueue[i]).self,
                 &((control->buttonQueue[i]).buttonKey));
       XMapWindow(dsply,(control->buttonQueue[i]).self);
      }
    }  /* makeVolumePanel() */
```

7.3.105   drawClipXBut

⟨view3d⟩+≡
```
void drawClipXBut(void) {
  XClearArea(dsply,volumeWindow,clipXButX,clipXButY,
             majorAxis+blank,minorAxis+blank,False);
  GSetForeground(trashGC,(float)monoColor(toggleColor),Xoption);
  GDrawLine(trashGC,volumeWindow,
            (control->buttonQueue[clipXBut]).buttonX,
            (control->buttonQueue[clipXBut]).buttonY +
            (control->buttonQueue[clipXBut]).yHalf,
            (control->buttonQueue[clipXBut]).buttonX +
            (control->buttonQueue[clipXBut]).buttonWidth,
            (control->buttonQueue[clipXBut]).buttonY +
            (control->buttonQueue[clipXBut]).yHalf,Xoption);
  GDrawLine(trashGC,volumeWindow,
            (control->buttonQueue[clipXBut]).buttonX-3,
            (control->buttonQueue[clipXBut]).buttonY +
            (control->buttonQueue[clipXBut]).yHalf-3,
            (control->buttonQueue[clipXBut]).buttonX,
            (control->buttonQueue[clipXBut]).buttonY +
            (control->buttonQueue[clipXBut]).yHalf,Xoption);
  GDrawLine(trashGC,volumeWindow,
            (control->buttonQueue[clipXBut]).buttonX-3,
            (control->buttonQueue[clipXBut]).buttonY +
            (control->buttonQueue[clipXBut]).yHalf+3,
            (control->buttonQueue[clipXBut]).buttonX,
            (control->buttonQueue[clipXBut]).buttonY +
            (control->buttonQueue[clipXBut]).yHalf,Xoption);
  GDrawLine(trashGC,volumeWindow,
            (control->buttonQueue[clipXBut]).buttonX +
            (control->buttonQueue[clipXBut]).buttonWidth+3,
            (control->buttonQueue[clipXBut]).buttonY +
            (control->buttonQueue[clipXBut]).yHalf-3,
            (control->buttonQueue[clipXBut]).buttonX +
            (control->buttonQueue[clipXBut]).buttonWidth,
            (control->buttonQueue[clipXBut]).buttonY +
            (control->buttonQueue[clipXBut]).yHalf,Xoption);
  GDrawLine(trashGC,volumeWindow,
            (control->buttonQueue[clipXBut]).buttonX +
            (control->buttonQueue[clipXBut]).buttonWidth+3,
            (control->buttonQueue[clipXBut]).buttonY +
            (control->buttonQueue[clipXBut]).yHalf+3,
            (control->buttonQueue[clipXBut]).buttonX +
            (control->buttonQueue[clipXBut]).buttonWidth,
            (control->buttonQueue[clipXBut]).buttonY +
```

```
                (control->buttonQueue[clipXBut]).yHalf,Xoption);
    GSetForeground(trashGC,(float)monoColor(arcColor),Xoption);
    GFillArc(trashGC,volumeWindow,
            (int)(xClipMinN * (majorAxis-tinyArc) + clipXButX),  /* x value */
            (int)(clipXButY + minorAxis/2 + 1),                  /* y value */
            arcSize,arcSize,0,360*64,Xoption);        /* 64 units per degree */
    GFillArc(trashGC,volumeWindow,
            (int)(xClipMaxN * (majorAxis-tinyArc) + clipXButX),  /* x value */
            (int)(clipXButY + minorAxis/2 - 7),                  /* y value */
            arcSize,arcSize,0,360*64,Xoption);        /* 64 units per degree */
    GSetForeground(volumeGC,(float)monoColor(toggleColor),Xoption);
    GDrawString(volumeGC,volumeWindow,clipXMessX,clipXMessY,"X",1,Xoption);
}
```

7.3.106   drawClipYBut

⟨*view3d*⟩+≡

```
void drawClipYBut(void) {
  XClearArea(dsply,volumeWindow,clipYButX,clipYButY,
             minorAxis+blank,majorAxis+blank,False);
  GSetForeground(trashGC,(float)monoColor(toggleColor),Xoption);
  GDrawLine(trashGC,volumeWindow,
            (control->buttonQueue[clipYBut]).buttonX +
            (control->buttonQueue[clipYBut]).xHalf,
            (control->buttonQueue[clipYBut]).buttonY,
            (control->buttonQueue[clipYBut]).buttonX +
            (control->buttonQueue[clipYBut]).xHalf,
            (control->buttonQueue[clipYBut]).buttonY +
            (control->buttonQueue[clipYBut]).buttonHeight,Xoption);
  GDrawLine(trashGC,volumeWindow,
            (control->buttonQueue[clipYBut]).buttonX +
            (control->buttonQueue[clipYBut]).xHalf-3,
            (control->buttonQueue[clipYBut]).buttonY-3,
            (control->buttonQueue[clipYBut]).buttonX +
            (control->buttonQueue[clipYBut]).xHalf,
            (control->buttonQueue[clipYBut]).buttonY,Xoption);
  GDrawLine(trashGC,volumeWindow,
            (control->buttonQueue[clipYBut]).buttonX +
            (control->buttonQueue[clipYBut]).xHalf+3,
            (control->buttonQueue[clipYBut]).buttonY-3,
            (control->buttonQueue[clipYBut]).buttonX +
            (control->buttonQueue[clipYBut]).xHalf,
            (control->buttonQueue[clipYBut]).buttonY,Xoption);
  GDrawLine(trashGC,volumeWindow,
            (control->buttonQueue[clipYBut]).buttonX +
            (control->buttonQueue[clipYBut]).xHalf-3,
            (control->buttonQueue[clipYBut]).buttonY +
            (control->buttonQueue[clipYBut]).buttonHeight+3,
            (control->buttonQueue[clipYBut]).buttonX +
            (control->buttonQueue[clipYBut]).xHalf,
            (control->buttonQueue[clipYBut]).buttonY +
            (control->buttonQueue[clipYBut]).buttonHeight,Xoption);
  GDrawLine(trashGC,volumeWindow,
            (control->buttonQueue[clipYBut]).buttonX +
            (control->buttonQueue[clipYBut]).xHalf+3,
            (control->buttonQueue[clipYBut]).buttonY +
            (control->buttonQueue[clipYBut]).buttonHeight+3,
            (control->buttonQueue[clipYBut]).buttonX +
            (control->buttonQueue[clipYBut]).xHalf,
            (control->buttonQueue[clipYBut]).buttonY +
```

```
                (control->buttonQueue[clipYBut]).buttonHeight,Xoption);
     GSetForeground(trashGC,(float)monoColor(arcColor),Xoption);
     /* note: minimum buttons closer to the box */
     GFillArc(trashGC,volumeWindow,
              (int)(clipYButX + minorAxis/2 - 8),
              (int)(yClipMinN * (majorAxis-tinyArc) + clipYButY),
              arcSize,arcSize,90*64,360*64,Xoption); /* 64 units per degree */
     GFillArc(trashGC,volumeWindow,
              (int)(clipYButX + minorAxis/2 + 3),
              (int)(yClipMaxN * (majorAxis-tinyArc) + clipYButY),
              arcSize,arcSize,90*64,360*64,Xoption); /* 64 units per degree */
     GSetForeground(volumeGC,(float)monoColor(toggleColor),Xoption);
     GDrawString(volumeGC,volumeWindow,clipYMessX,clipYMessY,"Y",1,Xoption);
}
```

7.3.107   drawClipZBut

⟨*view3d*⟩+≡

```
  void drawClipZBut(void) {
    XClearArea(dsply,volumeWindow,clipZButX,clipZButY,
               midAxis+blank,midAxis+blank,False);
    GSetForeground(trashGC,(float)monoColor(toggleColor),Xoption);
    GDrawLine(trashGC,volumeWindow,clipZButTopEndX,clipZButTopEndY,
              clipZButBotEndX,clipZButBotEndY,Xoption);
    GDrawLine(trashGC,volumeWindow,clipZButTopEndX-4,clipZButTopEndY,
              clipZButTopEndX,clipZButTopEndY,Xoption);
    GDrawLine(trashGC,volumeWindow,clipZButTopEndX,clipZButTopEndY-4,
              clipZButTopEndX,clipZButTopEndY,Xoption);
    GDrawLine(trashGC,volumeWindow,clipZButBotEndX+4,clipZButBotEndY,
              clipZButBotEndX,clipZButBotEndY,Xoption);
    GDrawLine(trashGC,volumeWindow,clipZButBotEndX,clipZButBotEndY+4,
              clipZButBotEndX,clipZButBotEndY,Xoption);
    GSetForeground(trashGC,(float)monoColor(arcColor),Xoption);
    GFillArc(trashGC,volumeWindow,
             (int)(zClipMinN * midAxis * zFactor + clipZButTopEndX - 3),
             (int)(zClipMinN * midAxis * zFactor + clipZButTopEndY + 3),
             arcSize,arcSize,45*64,360*64,Xoption);/* 64 units per degree */
    GFillArc(trashGC,volumeWindow,
             (int)(zClipMaxN * midAxis * zFactor + clipZButTopEndX + 3),
             (int)(zClipMaxN * midAxis * zFactor + clipZButTopEndY - 5),
             arcSize,arcSize,45*64,360*64,Xoption);/* 64 units per degree */
    GSetForeground(volumeGC,(float)monoColor(toggleColor),Xoption);
    GDrawString(volumeGC,volumeWindow,clipZMessX,clipZMessY,"Z",1,Xoption);
  }
```

7.3.108 drawClipVolume

⟨*view3d*⟩+≡

```
void drawClipVolume(void) {
  float xminL,xmaxL,yminL,ymaxL,zminL,zmaxL;
  XClearArea(dsply,volumeWindow,backFaceX-1,backFaceY,
             lengthFace+deltaFace+2,lengthFace+deltaFace+1,False);
  GSetForeground(trashGC,(float)boxInline,Xoption);    /*boxOutline=133*/
  GSetLineAttributes(trashGC,0,LineSolid,CapButt,JoinMiter,Xoption);
  /* define corners of volume, clockwise, back to front */
  xminL = xClipMinN*lengthFace;
  xmaxL = xClipMaxN*lengthFace;
  yminL = yClipMinN*lengthFace;
  ymaxL = yClipMaxN*lengthFace;
  zminL = zClipMinN*zLength;
  zmaxL = (1-zClipMaxN)*zLength;  /* percentage upwards from bottom */
  flatClipBoxX[0]   = backFaceX + xminL + zminL;
  flatClipBoxY[0]   = backFaceY + yminL + zminL;
  flatClipBoxX[1]   = backFaceX + xmaxL + zminL;
  flatClipBoxY[1]   = flatClipBoxY[0];
  flatClipBoxX[2]   = flatClipBoxX[1];
  flatClipBoxY[2]   = backFaceY + ymaxL + zminL;
  flatClipBoxX[3]   = flatClipBoxX[0];
  flatClipBoxY[3]   = flatClipBoxY[2];
  flatClipBoxX[4]   = frontFaceX + xminL - zmaxL;
  flatClipBoxY[4]   = frontFaceY + yminL - zmaxL;
  flatClipBoxX[5]   = frontFaceX + xmaxL - zmaxL;
  flatClipBoxY[5]   = flatClipBoxY[4];
  flatClipBoxX[6]   = flatClipBoxX[5];
  flatClipBoxY[6]   = frontFaceY + ymaxL - zmaxL;
  flatClipBoxX[7]   = flatClipBoxX[4];
  flatClipBoxY[7]   = flatClipBoxY[6];
  /* now draw the volume */
  GDrawRectangle(trashGC,volumeWindow,
                 flatClipBoxX[0],flatClipBoxY[0],
                 flatClipBoxX[2]-flatClipBoxX[0],
                 flatClipBoxY[2]-flatClipBoxY[0],Xoption);
  GDrawLine(trashGC,volumeWindow,flatClipBoxX[0],flatClipBoxY[0],
            flatClipBoxX[4],flatClipBoxY[4],Xoption);
  GDrawLine(trashGC,volumeWindow,flatClipBoxX[1],flatClipBoxY[1],
            flatClipBoxX[5],flatClipBoxY[5],Xoption);
  GDrawLine(trashGC,volumeWindow,flatClipBoxX[2],flatClipBoxY[2],
            flatClipBoxX[6],flatClipBoxY[6],Xoption);
  GDrawLine(trashGC,volumeWindow,flatClipBoxX[3],flatClipBoxY[3],
            flatClipBoxX[7],flatClipBoxY[7],Xoption);
  GSetForeground(trashGC,(float)boxOutline,Xoption);
```

```
    GDrawRectangle(trashGC,volumeWindow,
                   flatClipBoxX[4],flatClipBoxY[4],
                   flatClipBoxX[6]-flatClipBoxX[4],
                   flatClipBoxY[6]-flatClipBoxY[4],Xoption);
    /* make sure volumeGC is set properly before calling these functions */
} /* drawClipVolume() */
```

7.3.109   drawHitherControl

⟨*view3d*⟩+≡

```
void drawHitherControl(void) {
  float xx,b,slope;
  int hitherTop, hitherBot;
  float b0x,b1x;
   /* draw box indicating minimum and maximum distance of projection */
  GSetForeground(trashGC,(float)hitherBoxColor,Xoption);
  b0x = (pzMin - clipPlaneMin)/(clipPlaneMax-clipPlaneMin);
  b0x = hitherMaxX - b0x*(hitherMaxX - hitherMinX);  /* screen x */
  b1x = (pzMax - clipPlaneMin)/(clipPlaneMax-clipPlaneMin);
  b1x = hitherMaxX - b1x*(hitherMaxX - hitherMinX);  /* screen x */
  GDraw3DButtonOut(trashGC,volumeWindow,
                  (int)(b0x),frusY(hitherBoxTop),
                  (int)fabs(b1x-b0x),hitherBoxHeight,Xoption);
    /* draw the hither plane */
  GSetForeground(trashGC,(float)hitherColor,Xoption);
  /* percentage x */
  xx = ((viewData.clipPlane-clipPlaneMin)/(clipPlaneMax-clipPlaneMin));
  xx = hitherMaxX - xx*(hitherMaxX - hitherMinX);  /* screen x */
  slope = ((float)frustrumY - frustrumMidY)/(frustrumX - frustrumVertex);
  b = ((float)frustrumX*frustrumMidY - frustrumVertex*frustrumY) /
      (frustrumX - frustrumVertex);
  hitherTop = slope * xx + b + 0.5;
  slope = (float)(frustrumBotY - frustrumMidY)/(frustrumX - frustrumVertex);
  b = ((float)frustrumX*frustrumMidY - frustrumVertex*frustrumBotY) /
      (frustrumX - frustrumVertex);
  hitherBot = slope * xx + b + 0.5;
  GDrawLine(trashGC,volumeWindow, frusX((int)xx),frusY(hitherTop),
           frusX((int)xx),frusY(hitherBot),Xoption);
  /* draw hither control box and bar */
  GDraw3DButtonOut(trashGC,volumeWindow,
                  frusX(hitherWinX),frusY(hitherWinY+5),
                  hitherWidth,hitherHeight,Xoption);
  GDrawLine(trashGC,volumeWindow,
           frusX(hitherMinX),frusY(hitherBarY+5),
           frusX(hitherMaxX),frusY(hitherBarY+5),Xoption);
  /* draw hither plane I/O pointer arrow */
  GDrawLine(trashGC,volumeWindow,
           frusX((int)xx),frusY(hitherBarY+2),
           frusX((int)xx),frusY(hitherBarY+8),Xoption);
   /* print string label */
  GSetForeground(volumeGC,(float)hitherColor,Xoption);
  GDrawString(volumeGC,volumeWindow,hitherMessX,hitherMessY,
             "Hither",6,Xoption);
```

```
  }
```

7.3.110   drawEyeControl

⟨*view3d*⟩+≡

```c
  void drawEyeControl(void) {
    float here;
    int there;
    GSetForeground(trashGC,(float)eyeColor,Xoption);
      /* draw the eyeDistance box & slide bar */
    GDraw3DButtonOut(trashGC,volumeWindow,frusX(eyeWinX),frusY(eyeWinY+5),
                       eyeWidth,eyeHeight,Xoption);
    GDrawLine(trashGC,volumeWindow,frusX(eyeMinX),frusY(eyeBarY+5),
              frusX(eyeMaxX),frusY(eyeBarY+5),Xoption);
    here = (viewData.eyeDistance - minEyeDistance) /
           (maxEyeDistance - minEyeDistance);
    here = pow((double)here,0.333333);
    there = here * (eyeMaxX - eyeMinX) + eyeMinX;  /* screen x */
    GDrawLine(trashGC,volumeWindow,frusX(there),frusY(eyeBarY+2),
              frusX(there),frusY(eyeBarY+8),Xoption);
      /* draw the eye */
    GSetLineAttributes(trashGC,2,LineSolid,CapButt,JoinMiter,Xoption);
    GSetForeground(trashGC,(float)monoColor(52),Xoption);
    GDrawLine(trashGC,volumeWindow,
              frusX(there),frusY(frustrumMidY-5),
              frusX(there+8),frusY(frustrumMidY),Xoption);
    GDrawLine(trashGC,volumeWindow,
              frusX(there+2),frusY(frustrumMidY+4),
              frusX(there+8),frusY(frustrumMidY-1),Xoption);
    GSetForeground(trashGC,(float)frustrumColor,Xoption);
    GDrawLine(trashGC,volumeWindow,
              frusX(there+4),frusY(frustrumMidY-3),
              frusX(there+2),frusY(frustrumMidY),Xoption);
    GDrawLine(trashGC,volumeWindow,
              frusX(there+4),frusY(frustrumMidY+2),
              frusX(there+3),frusY(frustrumMidY),Xoption);
    GSetLineAttributes(trashGC,0,LineSolid,CapButt,JoinMiter,Xoption);
    /* draw string label */
    GSetForeground(volumeGC,(float)eyeColor,Xoption);
    GDrawString(volumeGC,volumeWindow,eyeDistMessX,eyeDistMessY,
                 "Eye Distance",strlen("eye distance"),Xoption);
  }
```

7.3.111 drawFrustrum

⟨*view3d*⟩+≡

```
void drawFrustrum(void) {
  float normalizedEyeDistance;
  XClearArea(dsply,volumeWindow,
             control->buttonQueue[frustrumBut].buttonX,
             control->buttonQueue[frustrumBut].buttonY,
             control->buttonQueue[frustrumBut].buttonWidth+9,
             control->buttonQueue[frustrumBut].buttonHeight,False);
  GSetForeground(trashGC,(float)frustrumColor,Xoption);
  normalizedEyeDistance = (viewData.eyeDistance - minEyeDistance) /
                          (maxEyeDistance - minEyeDistance);
  normalizedEyeDistance = pow((double)normalizedEyeDistance,0.333333333);
  frustrumVertex = normalizedEyeDistance * (frustrumMax - frustrumMin) +
                   frustrumMin - 4;
  GDrawLine(trashGC,volumeWindow,
            frusX(frustrumX),frusY(frustrumY),
            frusX(frustrumX),frusY(frustrumY+frustrumLength),Xoption);
  GDrawLine(trashGC,volumeWindow,
            frusX(frustrumX),frusY(frustrumY),
            frusX(frustrumVertex),frusY(frustrumMidY),Xoption);
  GDrawLine(trashGC,volumeWindow,
            frusX(frustrumX),frusY(frustrumBotY),
            frusX(frustrumVertex),frusY(frustrumMidY),Xoption);
  /* draw controls */
  drawHitherControl();
  drawEyeControl();
} /* drawFrustrum() */
```

7.3.112   drawVolumePanel

⟨*view3d*⟩+≡
```
void drawVolumePanel(void) {
  int i,strlength;
              /* Draw some lines for volume panel. */
  GSetForeground(trashGC,(float)foregroundColor,Xoption);
  GSetLineAttributes(trashGC,3,LineSolid,CapButt,JoinMiter,Xoption);
  GDrawLine(trashGC, volumeWindow, 0, potA, controlWidth, potA, Xoption);

  GSetLineAttributes(trashGC,2,LineSolid,CapButt,JoinMiter,Xoption);
  GDrawLine(trashGC, volumeWindow, 0, volumeTitleA, controlWidth,
            volumeTitleA, Xoption);
  GDrawLine(trashGC, volumeWindow, 0, volumeTitleB, controlWidth,
            volumeTitleB, Xoption);
  writeControlTitle(volumeWindow);
  s = "Viewing Volume Panel";
  strlength = strlen(s);
  GSetForeground(anotherGC,(float)volumeTitleColor,Xoption);
  GDrawString(anotherGC,volumeWindow,
              centerX(anotherGC,s,strlength,controlWidth),
              volumeTitleA+18,s,strlength,Xoption);
  GSetForeground(anotherGC,(float)monoColor(toggleColor),Xoption);
  GDrawString(anotherGC,volumeWindow,
              control->buttonQueue[perspectiveBut].buttonX + 4,
              control->buttonQueue[perspectiveBut].buttonY - 17,
              "Settings", 8, Xoption);
  GSetForeground(trashGC,(float)monoColor(toggleColor),Xoption);
  GDraw3DButtonOut(trashGC,volumeWindow,
                  control->buttonQueue[perspectiveBut].buttonX - 7,
                  control->buttonQueue[perspectiveBut].buttonY - 36,
                  100,100,Xoption);
  for (i=0; i<strlen(clipMess); i++)
    GDrawString(trashGC,volumeWindow,clipMessX,clipMessY + i*clipMessDy,
                &(clipMess[i]),1,Xoption);
  for (i=0; i<strlen(eyeMess1); i++)
    GDrawString(trashGC,volumeWindow,eyeMess1X,eyeMess1Y + i*eyeMess1Dy,
                &(eyeMess1[i]),1,Xoption);
  for (i=0; i<strlen(eyeMess2); i++)
    GDrawString(trashGC,volumeWindow,eyeMess2X,eyeMess2Y + i*eyeMess2Dy,
                &(eyeMess2[i]),1,Xoption);
  GSetLineAttributes(trashGC,0,LineSolid,CapButt,JoinMiter,Xoption);
  GSetForeground(trashGC,(float)volumeButtonColor,Xoption);
  for (i=volumeButtonsStart; i<(volumeButtonsEnd); i++) {
    GSetForeground(trashGC,
                  (float)monoColor((control->buttonQueue[i]).textColor),
```

```
                  Xoption);
    switch (i) {
    case perspectiveBut:
    case clipRegionBut:
    case clipSurfaceBut:
      GSetForeground(volumeGC,(float)monoColor(toggleColor),Xoption);
      GDraw3DButtonOut(volumeGC,volumeWindow,
                       (control->buttonQueue[i]).buttonX,
                       (control->buttonQueue[i]).buttonY,
                       (control->buttonQueue[i]).buttonWidth,
                       (control->buttonQueue[i]).buttonHeight,Xoption);
      GSetForeground(volumeGC,
                       (float)monoColor((control->buttonQueue[i]).textColor),
                       Xoption);
      GDrawString(volumeGC,volumeWindow,
                    (control->buttonQueue[i]).buttonX +
                    (control->buttonQueue[i]).buttonWidth + 4,
                    (control->buttonQueue[i]).buttonY +
                    centerY(volumeGC,(control->buttonQueue[i]).buttonHeight),
                    (control->buttonQueue[i]).text,
                    strlen(control->buttonQueue[i].text),Xoption);
      if (i==perspectiveBut && viewData.perspective)
        GDrawString(volumeGC,volumeWindow,
                      (control->buttonQueue[i]).buttonX +
                      centerX(volumeGC,"x",1,
                              (control->buttonQueue[i]).buttonWidth),
                      (control->buttonQueue[i]).buttonY +
                      centerY(volumeGC,(control->buttonQueue[i]).buttonHeight),
                      "x",1,Xoption);
      else if (i==clipRegionBut && viewData.clipbox)
        GDrawString(volumeGC,volumeWindow,
                      (control->buttonQueue[i]).buttonX +
                      centerX(volumeGC,"x",1,
                              (control->buttonQueue[i]).buttonWidth),
                      (control->buttonQueue[i]).buttonY +
                      centerY(volumeGC,(control->buttonQueue[i]).buttonHeight),
                      "x",1,Xoption);
      else if (i==clipSurfaceBut && viewData.clipStuff)
        GDrawString(volumeGC,volumeWindow,
                      (control->buttonQueue[i]).buttonX +
                      centerX(volumeGC,"x",1,
                              (control->buttonQueue[i]).buttonWidth),
                      (control->buttonQueue[i]).buttonY +
                      centerY(volumeGC,(control->buttonQueue[i]).buttonHeight),
                      "x",1,Xoption);
      break;
```

```
        case clipXBut:
          drawClipXBut();
          break;
        case clipYBut:
          drawClipYBut();
          break;
        case clipZBut:
          drawClipZBut();
          break;
        case frustrumBut:
          break;
        default:
          GDraw3DButtonOut(trashGC,volumeWindow,
                          (control->buttonQueue[i]).buttonX,
                          (control->buttonQueue[i]).buttonY,
                          (control->buttonQueue[i]).buttonWidth,
                          (control->buttonQueue[i]).buttonHeight,Xoption);
          s = (control->buttonQueue[i]).text;
          strlength = strlen(s);
          GSetForeground(trashGC,
                          (float)monoColor((control->buttonQueue[i]).textColor),
                          Xoption);
          GDrawString(trashGC,volumeWindow,
                       (control->buttonQueue[i]).buttonX +
                       centerX(processGC,s,strlength,
                               (control->buttonQueue[i]).buttonWidth),
                       (control->buttonQueue[i]).buttonY +
                       centerY(processGC,(control->buttonQueue[i]).buttonHeight),
                       s,strlen(s),Xoption);
      }  /* switch */
    }  /* for i in volumeButtons */
    drawFrustrum();
    drawClipVolume();    /*** put in header ***/
    drawClipXBut();
    drawClipYBut();
    drawClipZBut();
  } /* drawVolumePanel() */
```

7.3.113   writeViewport

⟨*view3d*⟩+≡

```
int writeViewport(int thingsToWrite) {
  int                i, j, k, ii, code, *anIndex;
  LLPoint            *anLLPoint;
  LPoint             *anLPoint;
  viewTriple         *aPt;
  XWindowAttributes  vwInfo;
  FILE               *viewDataFile;
  char               viewDirName[80], viewDataFilename[80],
                     viewBitmapFilename[80], viewPixmapFilename[80],
                     command[80];
  XGetWindowAttributes(dsply,viewport->titleWindow,&vwInfo);
  sprintf(viewDirName,"%s%s",filename,".view");
  sprintf(command,"%s%s%s","rm -r ",viewDirName," >  /dev/null 2>&1");
  code = system(command);
  sprintf(command,"%s%s%s","mkdir ",viewDirName," > /dev/null 2>&1");
  system(command);
  if (0) {
    fprintf(stderr,"   Error: Cannot create %s\n",viewDirName);
    return(-1);
  } else {
          /*** Create the data file ***/
    sprintf(viewDataFilename,"%s%s",viewDirName,"/data");
    if ((viewDataFile = fopen(viewDataFilename,"w")) == NULL) {
      fprintf(stderr,"   Error: Cannot create %s\n",viewDataFilename);
      perror("fopen");
      return(-1);
    } else {
            /*** write out the view3dStruct stuff ***/
      fprintf(viewDataFile,"%d\n",viewData.typeOf3D);
      fprintf(viewDataFile,"%g %g %g %g %g %g\n",
              viewData.xmin,viewData.xmax,viewData.ymin,viewData.ymax,
              viewData.zmin,viewData.zmax);
      fprintf(viewDataFile,"%s\n",viewport->title);
      fprintf(viewDataFile,"%g %g %g %g %g %g %g %g\n",viewport->deltaX,
              viewport->deltaY,viewport->scale,
              viewport->scaleX,viewport->scaleY,viewport->scaleZ,
              viewport->theta,viewport->phi);
      fprintf(viewDataFile,"%d %d %d %d\n",vwInfo.x,vwInfo.y,vwInfo.width,
              vwInfo.height);
      fprintf(viewDataFile,"%d %d %d %d %d %d %d\n",viewport->haveControl,
              viewData.style, viewport->axesOn,
              viewport->hueOffset,viewport->numberOfHues,
              viewport->diagonals, viewData.outlineRenderOn);
```

```
      fprintf(viewDataFile,"%g %g %g %g\n",viewport->lightVector[0],
              viewport->lightVector[1], viewport->lightVector[2],
              viewport->translucency);
      fprintf(viewDataFile,"%d %g\n",viewData.perspective,
              viewData.eyeDistance);
      /* write out the generalized 3D components */
      fprintf(viewDataFile,"%d\n",viewData.numOfPoints);
      for (i=0; i<viewData.numOfPoints; i++) {
        aPt = refPt3D(viewData,i);
        fprintf(viewDataFile,"%g %g %g %g\n",aPt->x, aPt->y, aPt->z, aPt->c);
      }
      fprintf(viewDataFile,"%d\n",viewData.lllp.numOfComponents);
      anLLPoint = viewData.lllp.llp;
      for (i=0; i<viewData.lllp.numOfComponents; i++,anLLPoint++) {
        fprintf(viewDataFile,"%d %d\n",anLLPoint->prop.closed,
                anLLPoint->prop.solid);
        fprintf(viewDataFile,"%d\n",anLLPoint->numOfLists);
        anLPoint = anLLPoint->lp;
        for (j=0; j<anLLPoint->numOfLists; j++,anLPoint++) {
          fprintf(viewDataFile,"%d %d\n",anLPoint->prop.closed,
                  anLPoint->prop.solid);
          fprintf(viewDataFile,"%d\n",anLPoint->numOfPoints);
          anIndex = anLPoint->indices;
          for (k=0; k<anLPoint->numOfPoints; k++,anIndex++) {
            fprintf(viewDataFile,"%d\n",*anIndex);
          } /* for points in LPoints (k) */
        } /* for LPoints in LLPoints (j) */
      } /* for LLPoints in LLLPoints (i) */
      fclose(viewDataFile);
  }  /* else was able to open file under the given filename */
        /* write out special files */
  for (ii=1; ii<numBits; ii++) {   /* write.h is one-based */
    if (thingsToWrite & (1<<ii)) {
      switch (ii) {
      case aBitmap:
          /*** Create the pixmap (bitmaps need leaf name) ***/
          sprintf(viewBitmapFilename,"%s%s%s",viewDirName,"/","image.bm");
          XGetWindowAttributes(dsply,viewport->viewWindow,&vwInfo);
          code = XWriteBitmapFile(dsply,viewBitmapFilename,
                                  viewport->titleWindow,vwInfo.width,
                                  vwInfo.height+vwInfo.border_width+20,-1,-1);
          break;
      case aPixmap:
          /*** Create the pixmap (bitmaps need leaf name) ***/
          sprintf(viewPixmapFilename,"%s%s%s",viewDirName,"/","image.xpm");
          XGetWindowAttributes(dsply,viewport->viewWindow,&vwInfo);
```

```
                            write_pixmap_file(dsply,scrn,viewPixmapFilename,
                                              viewport->titleWindow,0,0,vwInfo.width,
                                              vwInfo.height+titleHeight);
                      break;
                  case anImage:
                      /*** Create the image (bitmaps need leaf name) ***/
                      writeImage = yes;
                      sprintf(viewPixmapFilename,"%s%s%s",viewDirName,"/","image.xpm");
                      XResizeWindow(dsply,viewport->titleWindow,300,300+titleHeight);
                      XResizeWindow(dsply,viewport->viewWindow,300,300);
                      viewport->hueTop = totalHues-1;  viewport->hueOffset = 0;
                      viewport->numberOfHues = viewport->hueTop - viewport->hueOffset;
                      firstTime = 1;
                      if (viewData.style == transparent) {
                        viewData.style = render;
                        viewData.outlineRenderOn = 1;
                      } else {
                        if (viewData.style == render) viewData.outlineRenderOn = 1;
                      }
                      drawViewport(Xoption);
                      writeTitle();
                      XGetWindowAttributes(dsply,viewport->viewWindow,&vwInfo);
                      write_pixmap_file(dsply,scrn,viewPixmapFilename,
                                              viewport->titleWindow,0,0,vwInfo.width,
                                              vwInfo.height+titleHeight);
                      viewport->monoOn = 1;
                      maxGreyShade = XInitShades(dsply,scrn);
                      firstTime = 1;
                      drawViewport(Xoption);
                      writeTitle();
                      sprintf(viewBitmapFilename,"%s%s%s",viewDirName,"/","image.bm");
                      code = XWriteBitmapFile(dsply,viewBitmapFilename,
                                              viewport->titleWindow,vwInfo.width,
                                              vwInfo.height+vwInfo.border_width+20,-1,-1);
                      writeImage = no;
                      break;
                  case aPostscript:
                      /*** Create postscript output for viewport (in axiom3d.ps) ***/
                   sprintf(PSfilename,"%s%s",viewDirName,"/axiom3d.ps");
                   if (PSInit(viewport->viewWindow,viewport->titleWindow) == psError)
                     return(-1);
                   drawViewport(PSoption);  /* write new script file in /tmp */
                   if (PSCreateFile(viewBorderWidth,viewport->viewWindow,
                                    viewport->titleWindow, viewport->title) == psError)
                     return(-1);  /* concat script & proc into axiom3d.ps */
                      break;
```

```
          } /* switch on ii */
        }  /* if thingsToWrite >> ii */
      }  /* for ii */
    return(0);
  }    /* else create directory okay */
 }
```

7.3.114 main

⟨*view3d*⟩+≡

```
int main(void) {
  XGCValues      controlGCVals;
  int            i, code;
  char property[256];
  char *prop = &property[0];
  char *str_type[20];
  XrmValue value;
  /**** Global inits ****/
  splitPoints = NIL(viewTriple);
  /**** Set up display ****/
  if ((dsply = XOpenDisplay(getenv("DISPLAY"))) == NULL)
    {fprintf(stderr,"Could not open display.\n");exit (-1);}
  scrn = DefaultScreen(dsply);
  rtWindow = RootWindow(dsply,scrn);
  XSetErrorHandler(theHandler);
  /*   XSynchronize(dsply,False); */
  /**** link Xwindows to viewports - X10 feature ****/
  table = XCreateAssocTable(nbuckets);
  /**** Create Axiom color map ****/
  totalShades = 0;
  totalColors = XInitSpadFill(dsply,scrn,&colorMap,
                            &totalHues,&totalSolidShades,
                            &totalDitheredAndSolids,&totalShades);
  if (totalColors < 0) {
    fprintf(stderr,"ERROR: Could not allocate all the necessary colors.\n");
    exitWithAck(RootWindow(dsply,scrn),Window,-1);
  }
  mergeDatabases();
  /*** Determine whether monochrome or color is used ***/
  if (XrmGetResource(rDB,"Axiom.3D.monochrome","",str_type,&value) == True){
    (void) strncpy(prop,value.addr,(int)value.size);
    }
  else {
    (void) strcpy(prop, "off");
    }
  mono = ((totalSolid == 2) || (strcmp(prop,"on") == 0));
  if (mono) maxGreyShade=XInitShades(dsply,scrn) ;

  if (XrmGetResource(rDB,"Axiom.3D.inverse","",str_type,&value) == True){
    (void) strncpy(prop,value.addr,(int)value.size);
    }
  else {
    (void) strcpy(prop, "off");
```

```
      }

  if (mono) {
    if (strcmp(prop,"on") == 0) {  /* 0 if equal - inverse video */
      foregroundColor = white;
      backgroundColor = black;
    } else {  /* off - no inverse video */
      foregroundColor = black;
      backgroundColor = white;
    }
  } else {  /* inverse of inverse in color (for some strange reason) */
    if (strcmp(prop,"on") == 0) {  /* 0 if equal - inverse video */
      foregroundColor = white;
      backgroundColor = black;
    } else {  /* off - no inverse video */
      foregroundColor = black;
      backgroundColor = white;
    }
  }
  /* read default file name for postScript output */
  if (XrmGetResource(rDB,"Axiom.3D.postscriptFile","",str_type,&value) == True){
    (void) strncpy(prop,value.addr,(int)value.size);
    }
  else {
    (void) strcpy(prop, "axiom3d.ps");
    }
  PSfilename = (char *)malloc(strlen(prop)+1);
  strcpy(PSfilename,prop);
  XSync(dsply,0);
  /**** Open global fonts ****/
   serverFont = XQueryFont(dsply,XGContextFromGC(DefaultGC(dsply,scrn)));
  if (XrmGetResource(rDB,"Axiom.3D.messageFont","Axiom.3D.Font",str_type,&value) =
    (void) strncpy(prop,value.addr,(int)value.size);
    }
  else {
    (void) strcpy(prop,messageFontDefault);
    }
  if ((globalFont = XLoadQueryFont(dsply, prop)) == NULL) {
    fprintf(stderr, "Warning:  could not get the %s font for messageFont\n",prop
    globalFont = serverFont;
  }
  if (XrmGetResource(rDB,"Axiom.3D.buttonFont","Axiom.3D.Font",str_type,&value) ==
    (void) strncpy(prop,value.addr,(int)value.size);
    }
  else {
    (void) strcpy(prop,buttonFontDefault);
```

```
      }
   if ((buttonFont = XLoadQueryFont(dsply, prop)) == NULL) {
      fprintf(stderr, "Warning:  could not get the %s font for buttonFont\n",prop);
      buttonFont = serverFont;
   }
   if (XrmGetResource(rDB,"Axiom.3D.headerFont","Axiom.3D.Font",str_type,&value) == True){
      (void) strncpy(prop,value.addr,(int)value.size);
      }
   else {
      (void) strcpy(prop,headerFontDefault);
      }
   if ((headerFont = XLoadQueryFont(dsply, prop)) == NULL) {
      fprintf(stderr, "Warning:  could not get the %s font for headerFont\n",prop);
      headerFont = serverFont;
   }
   if (XrmGetResource(rDB,"Axiom.3D.titleFont","Axiom.3D.Font",str_type,&value) == True){
      (void) strncpy(prop,value.addr,(int)value.size);
      }
   else {
      (void) strcpy(prop,titleFontDefault);
      }
   if ((titleFont = XLoadQueryFont(dsply, prop)) == NULL) {
      fprintf(stderr, "Warning:  could not get the %s font for titleFont\n",prop);
      titleFont = serverFont;
   }
   if (XrmGetResource(rDB,"Axiom.3D.lightingFont","Axiom.3D.Font",str_type,&value) == True){
      (void) strncpy(prop,value.addr,(int)value.size);
      }
   else {
      (void) strcpy(prop,lightingFontDefault);
      }
   if ((lightingFont = XLoadQueryFont(dsply, prop)) == NULL) {
      fprintf(stderr, "Warning:  could not get the %s font for lightingFont\n",prop);
      lightingFont = serverFont;
   }
   if (XrmGetResource(rDB,"Axiom.3D.volumeFont","Axiom.3D.Font",str_type,&value) == True){
      (void) strncpy(prop,value.addr,(int)value.size);
      }
   else {
      (void) strcpy(prop,volumeFontDefault);
      }
   if ((volumeFont = XLoadQueryFont(dsply, prop)) == NULL) {
      fprintf(stderr, "Warning:  could not get the %s font for volumeFont\n",prop);
      volumeFont = serverFont;
   }
 /**** Create widely used Graphic Contexts ****/
```

```
PSGlobalInit();
/* must initiate before using any G/PS functions */
/* need character name: used as postscript GC variable */
/* need to create ps GCs for all GCs used by drawing in viewWindow */
                        /* globalGC1 */
controlGCVals.foreground  = monoColor(axesColor);
controlGCVals.background  = backgroundColor;
globalGC1  = XCreateGC(dsply,rtWindow,GCForeground |
                         GCBackground ,&controlGCVals);
carefullySetFont(globalGC1,globalFont);
PSCreateContext(globalGC1, "globalGC1", psNormalWidth, psButtCap,
                psMiterJoin, psWhite, psBlack);
                        /* controlMessageGC */
controlGCVals.foreground  = controlMessageColor;
controlGCVals.background  = backgroundColor;
controlMessageGC       = XCreateGC(dsply,rtWindow,GCForeground |
                                    GCBackground ,&controlGCVals);
carefullySetFont(controlMessageGC,globalFont);
                        /* globalGC2 */
controlGCVals.foreground = monoColor(labelColor);
globalGC2 = XCreateGC(dsply,rtWindow,GCForeground,&controlGCVals);
carefullySetFont(globalGC2,buttonFont);
PSCreateContext(globalGC2, "globalGC2", psNormalWidth, psButtCap,
                psMiterJoin, psWhite, psBlack);
                        /* trashGC */
controlGCVals.function = GXcopy;
trashGC  = XCreateGC(dsply,rtWindow,0 ,&controlGCVals);
carefullySetFont(trashGC,buttonFont);
PSCreateContext(trashGC, "trashGC", psNormalWidth, psButtCap,
                psMiterJoin, psWhite, psBlack);
                      /* componentGC */
componentGC  = XCreateGC(dsply,rtWindow,0 ,&controlGCVals);
carefullySetFont(componentGC,buttonFont);
PSCreateContext(componentGC, "componentGC", psNormalWidth, psButtCap,
                psMiterJoin, psWhite, psBlack);
                      /* opaqueGC */
opaqueGC  = XCreateGC(dsply,rtWindow,0 ,&controlGCVals);
carefullySetFont(opaqueGC,buttonFont);
PSCreateContext(opaqueGC, "opaqueGC", psNormalWidth, psButtCap,
                psMiterJoin, psWhite, psBlack);
                      /* renderGC */
renderGC  = XCreateGC(dsply,rtWindow,0,&controlGCVals);
carefullySetFont(renderGC,buttonFont);
PSCreateContext(renderGC, "renderGC", psNormalWidth, psButtCap,
                psMiterJoin, psWhite, psBlack);
                      /* globGC */
```

```
globGC = XCreateGC(dsply,rtWindow,0,&controlGCVals);
carefullySetFont(globGC,headerFont);
PSCreateContext(globGC, "globGC", psNormalWidth, psButtCap,
                psMiterJoin, psWhite, psBlack);
            /* anotherGC */
controlGCVals.line_width = colorWidth;
anotherGC  = XCreateGC(dsply,rtWindow,GCBackground | GCLineWidth |
                        GCFunction ,&controlGCVals);
carefullySetFont(anotherGC,titleFont);
PSCreateContext(anotherGC, "anotherGC", psNormalWidth, psButtCap,
                psMiterJoin, psWhite, psBlack);
/* also create one for rendering (grayscale only for now) */
/* assign arbitrary number to renderGC as 9991 - see header.h */
PSCreateContext(GC9991, "renderGC", psNormalWidth, psButtCap,
                psRoundJoin, psWhite, psBlack );
            /* processGC */
gcVals.background  = backgroundColor;
processGC = XCreateGC(dsply,rtWindow,GCBackground |
                        GCFillStyle,&gcVals);
carefullySetFont(processGC,buttonFont);
            /* lightingGC */
controlGCVals.foreground = monoColor(axesColor);
controlGCVals.background = backgroundColor;
lightingGC  = XCreateGC(dsply,rtWindow,GCForeground | GCBackground
                        ,&controlGCVals);
carefullySetFont(lightingGC,lightingFont);
            /* volumeGC */
volumeGC = XCreateGC(dsply,rtWindow,GCForeground | GCBackground
                        ,&controlGCVals);
carefullySetFont(volumeGC,volumeFont);
            /* quitGC */
quitGC  = XCreateGC(dsply,rtWindow,GCForeground | GCBackground
                        ,&controlGCVals);
carefullySetFont(quitGC,buttonFont);
            /* saveGC */
saveGC  = XCreateGC(dsply,rtWindow,GCForeground | GCBackground
                        ,&controlGCVals);
carefullySetFont(saveGC,buttonFont);
            /* graphGC */
graphGC  = XCreateGC(dsply,rtWindow,GCForeground | GCBackground
                        ,&controlGCVals);
carefullySetFont(graphGC,buttonFont);
/**** Get Data from the Viewport Manager ****/
i    = 123;   /* Used in viewman, what is this for? */
code = check(write(Socket,&i,intSize));
/* Check if I am getting stuff from Axiom or, if I am viewalone. */
```

```
readViewman(&viewAloned,intSize);
readViewman(&viewData,sizeof(view3DStruct));
readViewman(&i,intSize);
if (!(viewData.title = (char *)saymem("main.c",i,sizeof(char)))) {
  fprintf(stderr,"VIEW3D: Fatal Error>> Ran out of memory trying to receive\
                         the title.\n");
  exitWithAck(RootWindow(dsply,scrn),Window,-1);
}
readViewman(viewData.title,i);
readViewman(&(viewData.lightVec[0]),floatSize);
readViewman(&(viewData.lightVec[1]),floatSize);
readViewman(&(viewData.lightVec[2]),floatSize);
viewData.scaleDown = yes;
switch (viewData.typeOf3D) {
/* Currently, the view3DType information doesn't get sent from
   Axiom - all surfaces are alike regardless of how they
   were created. We may revert back to receiving this information
   in case we want to take advantage of certain properties of
   certain surfaces (e.g. z=f(x,y)). */

    case view3DType:
    case viewTubeType:
            viewport = make3DComponents();
            viewData.box = no;
            viewData.pointSize = 3;
            break;
}; /* switch typeOf3D */
/**************************************************
 ** Do some temporary assignments that would    **
 ** later be coded in the makeViewport routines **
 ** when the corresponding code has been put     **
 ** into the viewalone, viewman and spad files. **
 **************************************************/
viewData.distortX    = viewData.distortY = viewData.distortZ = 1;
viewData.clipPlane   = clipPlaneMin;
viewData.clipStuff   = yes;
xClipMinN = yClipMinN = zClipMinN = 0.0;
xClipMaxN = yClipMaxN = zClipMaxN = 1.0;
control = viewport->controlPanel;
bsdSignal(SIGTERM,goodbye,DontRestartSystemCalls);
bsdSignal(SIGSEGV,goodbye,DontRestartSystemCalls);
/** send acknowledgement to viewport manager**/
i = 345;
sprintf(errorStr,"sending window info to viewport manager");
check(write(Socket,&(viewport->viewWindow),sizeof(Window)));
viewmap = XCreatePixmap(dsply,viewport->viewWindow,
```

```
                              vwInfo.width,vwInfo.height,
                              DisplayPlanes(dsply,scrn));
      viewmap_valid = 1;
      processEvents();
      goodbye(-1);
      return(0); /* control never gets here but compiler complains */
   }     /* main() */
```

Chapter 8

gdraws

This section contains a set of functions for handling postscript generation. This is handled by defining a set of cover functions for the X routines, as in GDrawArc versus XDrawArc. When the Xoption is set the X routine is called. When the PSoption is set the postscript routines are generated.

This directory consists of all the Gdraw functions. The subdirectory PS contains all the ps drawing functions in support of the Gdraw functions.

8.0.115   Gdraw

A Gdraw function, using a draw option dFlsg as input, provides drawings capability on different output devices. Currently, Gdraw supports routines both in Xwindows functions, and in postscript functions on a postscript printer/interpreter.

The general drawing functions are (see file for description of functionality)

```
 1.  Gdrawarc.c
 2.  Gdrawstring.c
 3.  Gdraw.c
 4.  Gdrawline.c
 5.  Gdrfilled.c
 6.  GdrawFrame.c
 7.  Gdrawlines.c
 8.  Gfillarc.c
 9.  GdrawIstr.c
10. Gdrawrect.c
11. Gmisc.c
12. GinitPS.c
13. GCreatePS.c
14. PSFill.c
15. ../include/g.h
    header information needed by all the Gdraw routines and
```

```
    view3d/view2d as well.
 16. ps.h
    header information needed by only the Gdraw routines.
```

\subsection{POSTSCRIPT}

```
A PostScript file can be generated by using a button on the control
panel of a viewalone picture, or in Axiom, with the command write,
option Postscript.  This file can be submitted to the postsctipt
printer to be printed, or viewed using the postscript interpreter.
It will draw a window, title, and picture clipped to fit inside of the
window.

In order to generate a postscript output, we first initializ file names and
paths to be used by our program by using
\begin{verbatim}
        InitPs(viewWindow, titleWindow)     [in GinitPS.c]
```

then call the G draw routines with option "PS", which would generate postscript drawing commands. Then create the output file (OUTPUT.ps by default) by using

```
        makePSfile(viewWindow, titleWindow, title)    [in GcreatePS.c]
```

The output file would be in local directory, i.e., the directory where Axiom or viewalone, etc., was started up.

The following routines are used to test out the Gdraw functions:

```
1.  main.c       creates windows, and titles and processes Xwindow events.
2.  data.c       creates data for drawing and call Gdraw routines.
3.  menu.c       draws menu when mouse clicks in viewWindow
4.  yesorno.c    determines if mouse click in menu means "yes" or "no".
5.  loadfont.c   loads font for display
```

The Gdraw routines have been written in such a way that they are usable by both view3d and view2d.

8.0.116   To use G Functions

In order to draw in postscript, use GSetForeground to set the foreground colors for the drawing (and fill) functions. Use GSetLineAttributes to set line attributes. And finally, replace the XDraw commands with the corresponding GDraw commands with the appropriate parameters. In addition, we need to: draw frame, set GC variable names, create GCs, initialize postscript data structures, cat all the procedures used together.

IMPORTANT:
In order to create postscript command file, we need environment variable DEVE (i.e., setenv DEVE /u/jimwen/3D/version28) or AXIOM (e.g., setenv AXIOM /spad/mnt/rt in case

gdraws directory has been installed on the server). Without this path, the program would not know where the postscript files (in gdraws/PS) are.

LIMITATIONS of current implementation:
A picture is printed with 1 inch x direction, and 1 inch y direction offset, and the largest complete picutre is the size of the page with the offest.

BUGS:
The region box is not drawn correctly when perspective is altered.

FUTURE DIRECTIONS:

- for view3d color rendering, may want to convert LINE drawing color to the appropriate grayscale. Right now, they're all drawn in black.

- for view3d's routines for color rendering, may want to implement color drawing for postscript functions just in case someone does have access to a color postscript printer.

- implement more functionality for attributes used in GC, i.e., dashed line, dotted line etc.

- implement a display postscript menu to set things like picture size, landscape/portrait orientation, picture centered, left, right, up, down, etc., reversed video.

- make font an attribute in postscripts's graphics context so the font can be set in user program. Right now, it uses only 1 font.

- for view2d's stuff, implement a smaller font for the drawing the units on the axes.

8.1 gfun.c

Indices for PostScript draw procedures.

The order of these defined variables are very important; they are used to create the OUTPUT file. Essentially, PSCreateFile() loops through the index of 0 to psDrawNo and checks if the file/procedure is used. If so, the file is copied to the output file.

⟨*gfun.c*⟩≡

```
#define output          0               /* output file */
#define headerps        1               /* postscript header file */
#define drawps          2               /* draw procedure */
#define drawarcps       3               /* draw arc procedure */
#define drawfilledps    4               /* draw filled procedure */
#define drawcolorps     5               /* draw color filled procedure */
#define drawpointps     6               /* draw point procedure */
#define fillpolyps      7               /* polygon filled procedure */
#define fillwolps       8               /* polygon filled with outline proc */
#define colorpolyps     9               /* polygon fill with color procedure */
#define colorwolps      10              /* polygon fill with color procedure */
#define drawlineps      11              /* draw line procedure */
#define drawlinesps     12              /* draw lines procedure */
#define drawIstrps      13              /* draw image string procedure */
#define drawstrps       14              /* draw string procedure */
#define drawrectps      15              /* draw rectangle procedure */
#define fillarcps       16              /* filled arc procedure */
#define setupps         17              /* setup, or pre-script */
#define GCdictps        18              /* grahphics context definition file */
#define scriptps        19              /* script file */
#define endps           20              /* wrap up, close down, procedure */
#define psDrawNo        21              /* for use in createPSfile() */
```

PostScript structures

⟨*gfun.c*⟩+≡
```
typedef struct _psStruct {      /* data structure for ps routines info */
        int     flag;
        char    filename[200];
} psStruct;

psStruct psData[psDrawNo];      /* need psDrawNo of them */
```

8.1.1 filecopy

Given 2 file pointers, this function copies file ifp to file ofp

⟨*gfun.c*⟩+≡

```
static void filecopy(FILE * ifp, FILE * ofp)
{ int c;
  while ((c = getc(ifp)) != EOF)
    putc(c, ofp);
}
```

8.1.2   PSCreateFile

PSCreateFile generates the output file by using the order of defined variables; they are used
to create the OUTPUT file. Essentially, PSCreateFile() loop through the index of 0 to
psDrawNo and checks if the file/procedure is used. If so, the file is included into the output
file.

⟨*gfun.c*⟩+≡

```
  int PSCreateFile(
          int bWidth,                /* border width of picture frame */
          Window vw, Window tw,  /* viewWindow, and titleWindow */
          char *title)               /* title of picture to be drawn in title bar */
  { FILE *ifp, *ofp, *fp;        /* input, output and temp file pointer */
    int i;                         /* index */
    /* last things to add to the script file */
    fp = fopen(psData[scriptps].filename, "a");
    fprintf(fp, "\n    grestore\t%% restore graphics state\n\n");
    fclose(fp);
    /* put procedures and script together into OUTPUT.ps */
    if ((ofp = fopen(psData[output].filename, "w")) == NULL) {
      fprintf(stderr, "Cannot open %s to write.\n", psData[output].filename);
      return (psError);
    }
    else {
      i = 1;
      while (i < psDrawNo) {  /* loops through each file/procedure */
        if (psData[i].flag) { /* if set, procedure/file is used */
          if ((ifp = fopen(psData[i].filename, "r")) == NULL) {
            if (i == GCdictps) {  /* GC dictionaries */
              fprintf(stderr, "Warning: missing GCdictionary.\n");
            }
            else {
              fprintf(stderr, "Cannot open %s to read.\n",psData[i].filename);
              fclose(ofp);
              return(psError);
            }
          }
          else {
            filecopy(ifp, ofp);
            fclose(ifp);
          }
        }
        i++;
      }
    }
```

```
    /* remove script file in tmp */
    unlink(psData[scriptps].filename);
    return (fclose(ofp));
}
```

8.1.3 GdrawsDrawFrame

This function draws the frame of the picture, which corresponds to the picture frame on the X display. In addition, it draws the title window as well as the title of the picture.

⟨*gfun.c*⟩+≡

```
  int GdrawsDrawFrame(
          int borderW,             /* border width */
          Window viewWindow, Window titleWindow,
          char *title)             /* title of picture */
{ FILE *fp;
  XWindowAttributes vwInfo, twInfo;
  /* choose 2 and "frameDict" for frame dictionary: can be anything else */
  PSCreateContext((GC)2, "frameDict", borderW, psButtCap, psMiterJoin,
                  psWhite, psBlack);
  fp = fopen(psData[scriptps].filename, "a");
  XGetWindowAttributes(dsply, viewWindow, &vwInfo);
  /* draw title window */
  XGetWindowAttributes(dsply, titleWindow, &twInfo);
  fprintf(fp, "\t%s\t%d\t%d\t%d\t%d\ttitle\n", "frameDict",
          twInfo.height - vwInfo.height, twInfo.width, 0, vwInfo.height);
  /* draw viewport window */
  fprintf(fp, "\t%s\tdrawFrame\n", "frameDict");      /* using GdrawsSetDimension() */
  /* draw title text */
  psData[drawIstrps].flag = yes;
  fprintf(fp, "\t%s\tloadFont\n\t%d\t(%s) stringwidth pop sub 2 div\n",
          "frameDict", twInfo.width, title);
  fprintf(fp, "\t%d\t(%s)\t(%s)\tpsDrawIStr\n", 15, title, "title");
  return (fclose(fp));
}
```

8.1.4   GdrawsSetDimension

GdrawsSetDimension sets the dimension of the picture.

⟨*gfun.c*⟩+≡

```
int GdrawsSetDimension(Window viewWindow, Window titleWindow) {
  FILE *fp;
  XWindowAttributes vwInfo, twInfo;
  float pageWidth, pageHeight, width;
  fp = fopen(psData[scriptps].filename, "w");
  XGetWindowAttributes(dsply, titleWindow, &twInfo);
  XGetWindowAttributes(dsply, viewWindow, &vwInfo);
  pageWidth = 575.0;
  pageHeight = 750.0;
  fprintf(fp, "\n    gsave\t%% save graphics state for clipping path\n\n");
  if ((vwInfo.height > pageWidth) || (vwInfo.height > pageHeight)) {
    width = (float) vwInfo.width;
    if (vwInfo.height > pageWidth) {
      width = pageWidth / width;
      fprintf(fp, "\t%f\t%f", width, width);
    }
    else {
      if (vwInfo.height > pageHeight)
        fprintf(fp, "\t%f\t%f", width, pageHeight / width);
    }
  }
  else {
    fprintf(fp, "\t%f\t%f", 1.0, 1.0);
  }
  fprintf(fp, "\tscale\n\n");
  fprintf(fp, "\t%d\t%d\t%d\tsetDim\n", twInfo.height - vwInfo.height,
          vwInfo.height, vwInfo.width);
  /* Write a Bounding Box for psfig etc. */
  fprintf(fp, "%%%%BoundingBox: 0 0 %d %d\n", vwInfo.height, vwInfo.width);
  fprintf(fp, "\tmaxX maxY\t0 0\trectangle\tclip\t%% set clip path\n\n");
  return (fclose(fp));
}
```

8.1.5 GDrawImageString

GDrawImageString draws an image text string See 9.1.4 on page 519

⟨*gfun.c*⟩+≡

```
int GDrawImageString(
                GC gc,           /* graphics context */
                Window wid,    /* window id */
                int x, int y,
                char *string,
                int length, int dFlag)
{ int s;
  switch (dFlag) {
    case Xoption:
      s = XDrawImageString(dsply, wid, gc, x, y, string, length);
      break;
    case PSoption: {
      FILE *fp;
      if ((fp = fopen(psData[scriptps].filename, "a")) == NULL) {
        fprintf(stderr, "GDrawImageString cannot open %s\n",
                psData[scriptps].filename);
        return (psError);
      }
      psData[drawIstrps].flag = yes;      /* set procedure flag */
      fprintf(fp, "\t%s\t%d\t%d\t(%s)\t(%s)\tpsDrawIStr\n",
              PSfindGC(gc), x, y, string, "window");
      s = fclose(fp);
    }
      break;
    default:
      fprintf(stderr,
              "GdrawImagestring request (%d) not implemented yet.\n", dFlag);
    return (psError);
  }
  return (s);
}
```

8.1.6 GDrawArc

GDrawArc draws an arc. See 9.1.6 on page 521

⟨*gfun.c*⟩+≡

```
  int GDrawArc(
          GC gc,                        /* graphics context */
          Window wid,                   /* window id */
          int x, int y,
          unsigned int wdth, unsigned int hght,
          int ang1, int ang2, int dFlag)
{ int s = 0;
  switch (dFlag) {
    case Xoption:
      XDrawArc(dsply, wid, gc, x, y, wdth, hght, ang1, ang2);
      break;
    case PSoption: {
      FILE *fp;
      if ((fp = fopen(psData[scriptps].filename, "a")) == NULL) {
        fprintf(stderr, "GDrawArc cannot open %s\n",
                psData[scriptps].filename);
        return (psError);
      }
      psData[drawarcps].flag = yes;
      fprintf(fp, "\t%s\t%d\t%d\t%d\t%d\t%d\t%d\tpsDrawArc\n",
              PSfindGC(gc), x, y, hght, wdth, ang1 / 64, ang2 / 64);
      s = fclose(fp);
    }
      break;
    default:
      fprintf(stderr, "Gdrawarc request (%d) not implemented yet.\n",dFlag);
    return (psError);
  }
  return (s);
}
```

8.1.7 GDrawLine

GDrawLine draws a line. See 9.1.3 on page 518

⟨*gfun.c*⟩+≡

```c
int GDrawLine(
        GC gc,                  /* graphics context */
        Window wid,             /* window id */
        int x0, int y0, int x1, int y1, int dFlag)
{ int s = 0;
  switch (dFlag) {
    case Xoption:
      XDrawLine(dsply, wid, gc, x0, y0, x1, y1);
      break;
    case PSoption: {
      FILE *fp;
      if ((fp = fopen(psData[scriptps].filename, "a")) == NULL) {
        fprintf(stderr, "GDrawLine cannot open %s\n",
                psData[scriptps].filename);
        return (psError);
      }
      psData[drawlineps].flag = yes;      /* sets procedure flag */
      fprintf(fp, "\t%s\t%d\t%d\t%d\t%d\tpsDrawLine\n",
              PSfindGC(gc), x1, y1, x0, y0);
      s = fclose(fp);
    }
      break;
    default:
      fprintf(stderr, "Gdrawline request (%d) not implemented yet.\n",dFlag);
      return (psError);
  }
  return (s);
}
```

8.1.8   GDrawLines

GDrawLines draws lines.

⟨*gfun.c*⟩+≡

```
int GDrawLines(
          GC gc,                    /* graphics context */
          Window wid,               /* window id */
          XPoint * points,      /* points */
          int numberOfPoints, int mode, int dFlag)
                                    /* number of points, mode and display flag */
{ int s = 0;
  switch (dFlag) {
    case Xoption:
      XDrawLines(dsply, wid, gc, points, numberOfPoints, mode);
      break;
    case PSoption: {
      FILE *fp;              /* not dealing with modes yet */
      int i = 0;
      if ((fp = fopen(psData[scriptps].filename, "a")) == NULL) {
        fprintf(stderr, "GDrawLines cannot open %s\n",
                psData[scriptps].filename);
        return (psError);
      }
      psData[drawlinesps].flag = yes;     /* set procedure flag */
      fprintf(fp, "\t%s\n", PSfindGC(gc));
      i = numberOfPoints - 1;
      while (i > 0)
      { fprintf(fp, "\t%d\t%d\n", points[i].x, points[i].y);
        i = i-1;
      }
      fprintf(fp, "\t%d\t%d\t%d\tpsDrawLines\n",
              numberOfPoints, points[i].x, points[i].y);
      s = fclose(fp);
    }
      break;
    default:
      fprintf(stderr, "Gdrawlines request (%d) not implemented yet\n",dFlag);
      return (psError);
  }
  return (s);
}
```

8.1.9 GDrawPoint

GDrawPoint draws a point. (See Section 9.1.2 on page 518),

⟨*gfun.c*⟩+≡

```
int GDrawPoint(
          Window wid,            /* window id */
          GC gc,                 /* graphics context */
          int x0, int y0, int dFlag)
{ int s = 0;
  switch (dFlag) {
    case Xoption:
      XDrawPoint(dsply, wid, gc, x0, y0);
      break;
    case PSoption: {
      FILE *fp;
      if ((fp = fopen(psData[scriptps].filename, "a")) == NULL) {
        fprintf(stderr, "GDrawPoint cannot open %s\n",
                psData[scriptps].filename);
        return (psError);
      }
      psData[drawpointps].flag = yes;      /* sets procedure flag */
      fprintf(fp, "\t%s\t%d\t%d\t%d\t%d\tpsDrawPoint\n",
              PSfindGC(gc), x0, y0, x0 + 1, y0 + 1);
      s = fclose(fp);
    }
      break;
    default:
      fprintf(stderr, "Gdrawpoint request (%d) not implemented yet\n",dFlag);
      return (psError);
  }
  return (s);
}
```

8.1.10   GDrawRectangle

GDrawRectangle draws a rectangle.

⟨*gfun.c*⟩+≡

```
  int GDrawRectangle(
                GC gc,
                Window windowId,
                short int x,short  int y,short  int width,short  int height,
                int dFlag)
  { int s = 0;
    switch (dFlag) {
      case Xoption:
        XDrawRectangle(dsply, windowId, gc, x, y, width, height);
        break;
      case PSoption: {
        FILE *fp;
        if ((fp = fopen(psData[scriptps].filename, "a")) == NULL) {
          fprintf(stderr, "GDrawRect cannot open %s\n",
                   psData[scriptps].filename);
          return (psError);
        }
        psData[drawrectps].flag = yes;
        fprintf(fp, "\t%s\t%d\t%d\t%d\t%d\tpsDrawRect\n",
                 PSfindGC(gc), width, height, x, y);
        s = fclose(fp);
      }
        break;
      default:
        fprintf(stderr, "Gdrawrect request (%d) not implemented yet\n",dFlag);
      return (psError);
    }
    return (s);
  }
```

8.1.11   GDraw3DButtonIn

GDraw3DButtonOut draws a rectangle with 3D shading on rhs and bottom

⟨*gfun.c*⟩+≡

```
int GDraw3DButtonOut(
                GC gc,
                Window windowId,
                short int x,short  int y,short  int width,short  int height,
                int dFlag)
{ /* draw regular rectangle */
  int s = GDrawRectangle(gc, windowId, x, y, width - 1, height - 1, dFlag);
  /* add extra line down rhs */
  GDrawLine(gc, windowId, x + width, y + 1, x + width, y + height, dFlag);
  /* add extra line across bottom */
  GDrawLine(gc, windowId, x + 1, y + height, x + width, y + height, dFlag);
  return (s);
}
```

8.1.12   GDraw3DButtonIn

GDraw3DButtonIn draws a rectangle with 3D shading on lhs and top

⟨*gfun.c*⟩+≡

```
int GDraw3DButtonIn(
                GC gc,
                Window windowId,
                short int x,short  int y,short  int width,short  int height,
                int dFlag)
{ /* draw regular rectangle */
  int s =
    GDrawRectangle(gc, windowId, x + 1, y + 1, width - 1, height - 1, dFlag);
  /* add extra line down lhs */
  GDrawLine(gc, windowId, x, y, x, y + height - 1, dFlag);
  /* add extra line across top */
  GDrawLine(gc, windowId, x, y, x + width - 1, y, dFlag);
  return (s);
}
```

8.1.13 GDrawPushButton

GDrawPushButton draws a push button whose appearance depends on "isOn."

⟨*gfun.c*⟩+≡

```
int GDrawPushButton(
                Display * display,
                GC gc1, GC gc2, GC gc3,
                Window windowId,
                short int x,short int y,short  int width,short  int height,
                int isOn, char *text,
                unsigned long buttoncolor, unsigned long color,
                int dFlag)
{
  int len = strlen(text);
  if (dFlag == Xoption)
    XClearArea(display, windowId, x, y, width + 1, height + 1, False);
  GSetForeground(gc1, (float) buttoncolor, dFlag);
  if (isOn)
    GDraw3DButtonIn(gc1, windowId, x, y, width, height, dFlag);
  else
    GDraw3DButtonOut(gc1, windowId, x, y, width, height, dFlag);
  GSetForeground(gc2, (float) color, dFlag);
  return GDrawString(gc2, windowId,
                        x + (isOn ? 2 : 0) + centerX(gc3, text, len, width),
                        y + (isOn ? 2 : 0) + centerY(gc3, height),
                        text, len, dFlag);
}
```

8.1.14 GDrawString

Draws a string. See 9.1.1 on page 517.

⟨*gfun.c*⟩+≡

```
  int GDrawString(
            GC gc,                /* graphics context */
            Window wid,           /* window id */
            int x, int y,
            char *string,         /* string to be drawn */
            int length, int dFlag)
{ int s;
  switch (dFlag) {
    case Xoption:
      s = XDrawString(dsply, wid, gc, x, y, string, length);
      break;
    case PSoption: {
      FILE *fp;
      if ((fp = fopen(psData[scriptps].filename, "a")) == NULL) {
        fprintf(stderr, "GDrawString cannot open %s\n",
                psData[scriptps].filename);
        return (psError);
      }
      psData[drawstrps].flag = yes;      /* sets procedure flag */
      fprintf(fp, "\t%s\t(%s)\t%d\t%d\tpsDrawStr\n",
              PSfindGC(gc), string, x, y);
      s = fclose(fp);
    }
      break;
    default:
      fprintf(stderr, "Gdrawstring request (%d) not implemented yet\n",dFlag);
      return (psError);
  }
  return (s);
}
```

8.1.15   GFillArc

Draws and fills an arc with foreground color. See 9.1.5 on page 520.

⟨*gfun.c*⟩+≡

```
int GFillArc(
        GC gc,                      /* graphics context */
        Window wid,                 /* window id */
        int x, int y,
        unsigned int wdth, unsigned int hght,
        int ang1, int ang2, int dFlag)
{ int s = 0;
  switch (dFlag) {
  case Xoption:                         /* angle: times 64 already */
    XFillArc(dsply, wid, gc, x, y, wdth, hght, ang1, ang2);
    break;
  case PSoption: {
    FILE *fp;
    if ((fp = fopen(psData[scriptps].filename, "a")) == NULL) {
      fprintf(stderr, "GFillArc cannot open %s\n",
              psData[scriptps].filename);
      return (psError);
    }
    psData[fillarcps].flag = yes;       /* sets procedure flag */
    fprintf(fp, "\t%s\t%d %d\t%d %d\t%d %d\t%d %d\tpsFillArc\n",
            PSfindGC(gc), x, y, hght, wdth, ang1 / 64, ang2 / 64,
            x + wdth / 2, y + hght / 2);
    s = fclose(fp);
  }
    break;
  default:
    fprintf(stderr, "GFillArc request (%d) not implemented yet\n", dFlag);
    return (psError);
  }
  return (s);
}
```

8.1.16 PSGlobalInit

Initializes the path and files to be used. This needs to be called only once each session.

⟨*gfun.c*⟩+≡

```
int PSGlobalInit(void) {
  char *tmp;
  /* path-independent global file name */
  psData[GCdictps].flag = yes;
  tmp = tempnam(NULL, "axPS");
  sprintf(psData[GCdictps].filename, "%s", tmp);
  free(tmp);
  psData[setupps].flag = yes;
  psData[scriptps].flag = yes;/* new script file name */
  psData[endps].flag = yes;
  /* path specific file names */
  if ((envAXIOM = getenv("DEVE")) != NULL) {  /* get env var AXIOM */
    psData[headerps].flag = yes;
    sprintf(psData[headerps].filename,"%s%s",envAXIOM,"/gdraws/PS/header.ps");
    sprintf(psData[drawps].filename, "%s%s", envAXIOM, "/gdraws/PS/draw.ps");
    sprintf(psData[drawarcps].filename, "%s%s", envAXIOM,
            "/gdraws/PS/drawarc.ps");
    sprintf(psData[drawfilledps].filename, "%s%s", envAXIOM,
            "/gdraws/PS/drwfilled.ps");
    sprintf(psData[drawcolorps].filename, "%s%s", envAXIOM,
            "/gdraws/PS/drawcolor.ps");
    sprintf(psData[fillpolyps].filename, "%s%s", envAXIOM,
            "/gdraws/PS/fillpoly.ps");
    sprintf(psData[colorpolyps].filename, "%s%s", envAXIOM,
            "/gdraws/PS/colorpoly.ps");
    sprintf(psData[fillwolps].filename, "%s%s", envAXIOM,
            "/gdraws/PS/fillwol.ps");
    sprintf(psData[colorwolps].filename, "%s%s", envAXIOM,
            "/gdraws/PS/colorwol.ps");
    sprintf(psData[drawpointps].filename, "%s%s", envAXIOM,
            "/gdraws/PS/drawpoint.ps");
    sprintf(psData[drawlineps].filename, "%s%s", envAXIOM,
            "/gdraws/PS/drawline.ps");
    sprintf(psData[drawlinesps].filename, "%s%s", envAXIOM,
            "/gdraws/PS/drawlines.ps");
    sprintf(psData[drawrectps].filename, "%s%s", envAXIOM,
            "/gdraws/PS/drawrect.ps");
    sprintf(psData[drawstrps].filename, "%s%s", envAXIOM,
            "/gdraws/PS/drawstr.ps");
    sprintf(psData[drawIstrps].filename, "%s%s", envAXIOM,
```

```
                    "/gdraws/PS/drawIstr.ps");
        sprintf(psData[fillarcps].filename, "%s%s", envAXIOM,
                    "/gdraws/PS/fillarc.ps");
        sprintf(psData[setupps].filename, "%s%s", envAXIOM, "/gdraws/ps/setup.ps");
        sprintf(psData[endps].filename, "%s%s", envAXIOM, "/gdraws/ps/end.ps");
    }
    else if ((envAXIOM = getenv("AXIOM")) != NULL) {
        psData[headerps].flag = yes;
        sprintf(psData[headerps].filename, "%s%s", envAXIOM,
                    "/lib/graph/header.ps");
        sprintf(psData[drawps].filename, "%s%s", envAXIOM,
                    "/lib/graph/draw.ps");
        sprintf(psData[drawarcps].filename, "%s%s", envAXIOM,
                    "/lib/graph/drawarc.ps");
        sprintf(psData[drawfilledps].filename, "%s%s", envAXIOM,
                    "/lib/graph/drwfilled.ps");
        sprintf(psData[drawcolorps].filename, "%s%s", envAXIOM,
                    "/lib/graph/drawcolor.ps");
        sprintf(psData[fillpolyps].filename, "%s%s", envAXIOM,
                    "/lib/graph/fillpoly.ps");
        sprintf(psData[colorpolyps].filename, "%s%s", envAXIOM,
                    "/lib/graph/colorpoly.ps");
        sprintf(psData[fillwolps].filename, "%s%s", envAXIOM,
                    "/lib/graph/fillwol.ps");
        sprintf(psData[colorwolps].filename, "%s%s", envAXIOM,
                    "/lib/graph/colorwol.ps");
        sprintf(psData[drawpointps].filename, "%s%s", envAXIOM,
                    "/lib/graph/drawpoint.ps");
        sprintf(psData[drawlineps].filename, "%s%s", envAXIOM,
                    "/lib/graph/drawline.ps");
        sprintf(psData[drawlinesps].filename, "%s%s", envAXIOM,
                    "/lib/graph/drawlines.ps");
        sprintf(psData[drawrectps].filename, "%s%s", envAXIOM,
                    "/lib/graph/drawrect.ps");
        sprintf(psData[drawstrps].filename, "%s%s", envAXIOM,
                    "/lib/graph/drawstr.ps");
        sprintf(psData[drawIstrps].filename, "%s%s", envAXIOM,
                    "/lib/graph/drawIstr.ps");
        sprintf(psData[fillarcps].filename, "%s%s", envAXIOM,
                    "/lib/graph/fillarc.ps");
        sprintf(psData[setupps].filename, "%s%s", envAXIOM,
                    "/lib/graph/setup.ps");
        sprintf(psData[endps].filename, "%s%s", envAXIOM,
                    "/lib/graph/end.ps");
    }
    else {
```

```
      fprintf(stderr,
        " need environment variable AXIOM or DEVE; process aborted\n");
      return (psError);
    }
    return (psInit = yes);
  }
```

8.1.17   PSInit

This needs to be called for every postscript file generated. It initializes the procedure flags.

⟨*gfun.c*⟩+≡

```
  int PSInit(Window vw, Window tw) {
    if (!psInit) {
      /* must have PSGlobalInit() called before this */
      fprintf(stderr,
        "Error: need initialization for ps data files: call PSGlobalInit().\n");
      return (psError);
    }
    sprintf(psData[output].filename, "%s", PSfilename); /* output file name */
    psData[drawps].flag = no;    /* ps procedures flags */
    psData[drawarcps].flag = no;
    psData[drawfilledps].flag = no;
    psData[drawcolorps].flag = no;
    psData[fillpolyps].flag = no;
    psData[fillwolps].flag = no;
    psData[colorpolyps].flag = no;
    psData[colorwolps].flag = no;
    psData[drawpointps].flag = no;
    psData[drawlineps].flag = no;
    psData[drawlinesps].flag = no;
    psData[drawrectps].flag = no;
    psData[drawstrps].flag = no;
    psData[drawIstrps].flag = no;
    psData[fillarcps].flag = no;
    sprintf(psData[scriptps].filename, "%s", tmpnam(NULL));      /* script file */
    return (GdrawsSetDimension(vw, tw));
  }
```

8.1.18   PSCreateContext

This procedure sets the line attributes; notice that lineWidth is not set for PS, this is because lineWidth of 0, the thinest line on the ps device, is device-dependent, thus, we'll leave it and use default. Also lineStyle is solid in ps by default, thus we don't need to set it. We'll leave out line style here since we really never used anything other than line solid which is the default line style in postscript.

⟨*gfun.c*⟩+≡

```
int PSCreateContext(
                GC gc,          /* graphics context */
                char *C_gc,      /* GC name to be used as postscript variable */
                int linewidth, int capstyle, int joinstyle,
                float bg, float fg)
{
  FILE *fp;
  GCptr newGC, curGC;
  /* get memory for new GC cell */
  if (!(newGC = (GCptr) malloc(sizeof(GCstruct)))) {
    fprintf(stderr, "Ran out of memory(malloc) trying to create a ps GC.\n");
    exit(-1);
  }
  /* attach newGC to chain */
  if (GChead == NULL)
    GChead = newGC;
  else {                          /* attach newGC to end of linked list */
    curGC = GChead;
    while (curGC->next != NULL)
      curGC = curGC->next;
    curGC->next = newGC;
  }
  /* fill newGC with information */
  newGC->GCint = gc;
  sprintf(newGC->GCchar, "%s", C_gc);
  newGC->next = NULL;
  if ((fp = fopen(psData[GCdictps].filename, "a")) == NULL) {
    fprintf(stderr, "PSCreateContext cannot open %s\n",
            psData[GCdictps].filename);
    return (psError);
  }
  fprintf(fp, "\t%d\t%d\t%d\n\t%f\t%f\t/%s\tmakeDict\n", joinstyle,
          capstyle, linewidth, bg, fg, C_gc);
  return (fclose(fp));
}
```

8.1.19 PSfindGC

Looks into GC linked list with gc (unsigned long) as index to find the character name.

⟨*gfun.c*⟩+≡

```
char *PSfindGC(GC gc) {
  GCptr curGC;
  curGC = GChead;
  while ((curGC != NULL) && (curGC->GCint != gc))
    curGC = curGC->next;
  if (curGC == NULL) {
    fprintf(stderr, "PSfindGC cannot find gc: %p.\n",gc);
    return (NULL);
  }
  else
    return (curGC->GCchar);
}
```

8.1.20   GSetForeground

Sets foreground color. See 9.1.7 on page 522.

⟨*gfun.c*⟩+≡

```
  int GSetForeground(
                GC gc,            /* graphics context */
                float color,      /* foreground color to be set */
                int dFlag)        /* display flag: PS, X,... */
{
  int s = 0;
  switch (dFlag) {
    case Xoption:
      XSetForeground(dsply, gc, (unsigned long) color);
      break;
    case PSoption: {
      FILE *fp;
      if ((fp = fopen(psData[scriptps].filename, "a")) == NULL) {
        fprintf(stderr, "GSetForeground cannot open %s\n",
                psData[scriptps].filename);
        return (0);
      }
      fprintf(fp, "\t%s\t%f\tsetForeground\n", PSfindGC(gc), color);
      s = fclose(fp);
      break;
    }
    default:
      fprintf(stderr,
        "GSetForeground request (%d) not implemented yet\n", dFlag);
      return (0);
  }
  return (s);
}
```

8.1.21 GSetBackground

Sets background color. See 9.1.8 on page 522.

⟨*gfun.c*⟩+≡

```
  int GSetBackground(
                GC gc,             /* graphics context */
                float color,       /* background color to be set */
                int dFlag)         /* display flag: PS, X,... */
  {
    int s = 0;
    switch (dFlag) {
      case Xoption:
        XSetBackground(dsply, gc, (unsigned long) color);
        break;
      case PSoption: {
        FILE *fp;
        if ((fp = fopen(psData[scriptps].filename, "a")) == NULL) {
          fprintf(stderr, "GSetBackground cannot open %s\n",
                  psData[scriptps].filename);
          return (0);
        }
        fprintf(fp, "\t%s\t%f\tsetBackground\n", PSfindGC(gc), color);
        s = fclose(fp);
        break;
      }
      default:
        fprintf(stderr,
         "GSetBackground request (%d) not implemented yet\n", dFlag);
      return (0);
    }
    return (s);
  }
```

8.1.22   GSetLineAttributes

(See 9.1.9 on page 522), Notice that we'll not setting line style for postscript. This is because solid is the ls in ps and in view3d and 2d, we really don't use anything else than solid.

⟨*gfun.c*⟩+≡

```
  int GSetLineAttributes(
                   GC gc,
                   int linewidth, int linestyle, int capstyle, int joinstyle,
                   int dFlag) {
    int s = 0;
    switch (dFlag) {
      case Xoption:
        XSetLineAttributes(dsply, gc, linewidth, linestyle,capstyle, joinstyle);
        break;
      case PSoption: {
        FILE *fp;
        int psCap, psJoin;
        switch (capstyle) {
          case 0:            /* CapNotLast is not implemented in ps */
          case 1:
            psCap = psButtCap;
            break;
          case 2:
            psCap = psRoundCap;
            break;
          case 3:
            psCap = psPSqCap;
            break;
          default:
            fprintf(stderr, "cap style: %d unknown, using default.\n", capstyle);
            psCap = psButtCap;
        }
        switch (joinstyle) {
          case 0:
            psJoin = psMiterJoin;
            break;
          case 1:
            psJoin = psRoundJoin;
            break;
          case 2:
            psJoin = psBevelJoin;
            break;
          default:
            fprintf(stderr,"join style: %d unknown, using default.\n",joinstyle);
```

```
            psJoin = psMiterJoin;
          }
          /*
           * width of zero is machine-dependent and is not recommended,
           * we'll use 1 as the thinest line available if (lineWidth < 1)
           * lineWidth = 1;
           */
          if ((fp = fopen(psData[scriptps].filename, "a")) == NULL) {
            fprintf(stderr, "GSetLineAttributes cannot open %s\n",
                    psData[scriptps].filename);
            return (0);
          }
          fprintf(fp, "\t%d\t%d\t%d\t%s\tsetLineAttributes\n",
                  linewidth, psCap, psJoin, PSfindGC(gc));
          s = fclose(fp);
      }
        break;
      default:
        fprintf(stderr,
          "GSetLineAttributes request (%d) not implemented yet\n", dFlag);
        return (0);
    }
    return(s);
  }
```

8.1.23 PSClose

This procedure frees the data structure used for GC information, and also unlinks the GC dictionary file.

⟨*gfun.c*⟩+≡

```
  int PSClose(void) {
    if (GChead != NULL) { /* free memory used by GC struct */
      GCptr curGC = GChead;
      while (curGC != NULL) {
        GCptr freeGC = curGC;
        curGC = curGC->next;
        free(freeGC);
      }
    }
    return (unlink(psData[GCdictps].filename)); /* remove GC dictionary file */
  }
```

8.1.24   centerX

⟨*gfun.c*⟩+≡

```
int centerX(GC viewGCx,char * theString,int strlength,int windowWidth)
{ XFontStruct *fontStruct;
  GContext con;
  int result;
  con=XGContextFromGC(viewGCx);
  fontStruct = XQueryFont(dsply,con);
  if(fontStruct == NULL) return(0);
  result = (windowWidth - XTextWidth(fontStruct,theString,strlength))/2 -
    fontStruct->min_bounds.lbearing;
  XFreeFontInfo(NULL,fontStruct,1);
  return(result);
}
```

8.1.25   centerY

⟨*gfun.c*⟩+≡

```
int centerY(GC viewGCy,int windowHeight) {
  XFontStruct *fontStruct;
  GContext con;
  int result;
  con=XGContextFromGC(viewGCy);
  fontStruct = XQueryFont(dsply,con);
  if (fontStruct == NULL) return(0);
  result = (windowHeight -
            (fontStruct->max_bounds.ascent
              + fontStruct->max_bounds.descent))/2 +
            fontStruct->max_bounds.ascent;
  XFreeFontInfo(NULL,fontStruct,1);
  return(result);
}
```

8.1.26 PSColorPolygon

PSColorPolygon draws and fills a polygon given data in XPoint. (See **??** on page **??**)

⟨*gfun.c*⟩+≡

```
int PSColorPolygon(
                float r, float g, float b,       /* red, green and blue color
                                                  * components */
                XPoint * points,          /* vertices of polygon */
                int numberOfPoints)       /* number of points */
{
  int i = 0;
  FILE *fp;
  if ((fp = fopen(psData[scriptps].filename, "a")) == NULL) {
    fprintf(stderr, "PSColorPolygon cannot open %s\n",
            psData[scriptps].filename);
    return (psError);
  }
  psData[colorpolyps].flag = yes;      /* sets procedure flag */
  fprintf(fp, "\t%f\t%f\t%f\tsetrgbcolor\n", r, g, b);
  i = numberOfPoints - 1;
  while (i > 0) {
    fprintf(fp, "\t%d\t%d\n", points[i].x, points[i].y);
    i = i-1;
  }
  fprintf(fp, "\t%d\t%d\t%d\tpsColorPoly\n",
          numberOfPoints, points[i].x, points[i].y);
  return (fclose(fp));
}
```

8.1.27   PSColorwOutline

PSColorwOutline draws and also outlines the colored polygon.

⟨*gfun.c*⟩+≡

```
int PSColorwOutline(
                float r, float g, float b,        /* red, green and blue color
                                                    * components */
                XPoint * points,          /* vertices of polygon */
                int numberOfPoints)      /* number of points */
{
  int i = 0;
  FILE *fp;
  if ((fp = fopen(psData[scriptps].filename, "a")) == NULL) {
    fprintf(stderr, "PSDrawFOL cannot open %s\n", psData[scriptps].filename);
    return (psError);
  }
  psData[colorwolps].flag = yes;       /* sets procedure flag */
  fprintf(fp, "\t%f\t%f\t%f\tsetrgbcolor\n", r, g, b);
  i = numberOfPoints - 1;
  while (i > 0) {
    fprintf(fp, "\t%d\t%d\n", points[i].x, points[i].y);
    i = i-1;
  }
  fprintf(fp, "\t%d\t%d\t%d\tpsFillwOutline\n",
          numberOfPoints, points[i].x, points[i].y);
  return (fclose(fp));
}
```

8.1.28 PSDrawColor

This function does what XDraw would do, notice that only a subset of attributes in GC is implemented – adequate for our purpose now

⟨*gfun.c*⟩+≡

```
int PSDrawColor(
    float r, float g, float b,  /* red, green and blue color components */
    XPoint *points,             /* point list */
    int numberOfPoints)         /* vertex count and display flag (X, PS,...) */
{
  int i = 0;
  FILE *fp;
  if ((fp = fopen(psData[scriptps].filename, "a")) == NULL) {
    fprintf(stderr, "GDraw cannot open %s\n",
            psData[scriptps].filename);
    return (psError);
  }
  psData[drawcolorps].flag = yes;      /* set procedure flag */
  fprintf(fp, "\t%f\t%f\t%f\tsetrgbcolor\n", r, g, b);
  i = numberOfPoints - 1;
  while (i > 0)
  { fprintf(fp, "\t%d\t%d\n", points[i].x, points[i].y);
    i=i-1;
  }
  fprintf(fp, "\t%d\t%d\t%d\tpsDrawColor\n",
          numberOfPoints, points[i].x, points[i].y);
  return (fclose(fp));
}
```

8.1.29   PSFillPolygon

PSFillPolygon draws and fills a polygon given data in XPoint. (See **??** on page **??**).

⟨*gfun.c*⟩+≡

```
int PSFillPolygon(
              GC gc,                  /* graphics context */
              XPoint * points,        /* vertices of polygon */
              int numberOfPoints)   /* number of points */
{
  int i = 0;
  FILE *fp;
  if ((fp = fopen(psData[scriptps].filename, "a")) == NULL) {
    fprintf(stderr, "PSFillPolygon cannot open %s\n",
            psData[scriptps].filename);
    return (psError);
  }
  psData[fillpolyps].flag = yes;       /* sets procedure flag */
  fprintf(fp, "\t%s\n", PSfindGC(gc));
  i = numberOfPoints - 1;
  while (i > 0)
  { fprintf(fp, "\t%d\t%d\n", points[i].x, points[i].y);
    i = i-1;
  }
  fprintf(fp, "\t%d\t%d\t%d\tpsFillPoly\n",
          numberOfPoints, points[i].x, points[i].y);
  return (fclose(fp));
}
```

8.1.30 PSFillwOutline

PSFillwOutline draws and also outlines the filled polygon.

⟨*gfun.c*⟩+≡

```
int PSFillwOutline(
               GC gc,              /* graphics context */
               XPoint * points, /* vertices of polygon */
               int numberOfPoints) /* number of points */
{ int i = 0;
  FILE *fp;
  if ((fp = fopen(psData[scriptps].filename, "a")) == NULL) {
    fprintf(stderr, "PSDrawFOL cannot open %s\n", psData[scriptps].filename);
    return (psError);
  }
  psData[fillwolps].flag = yes;       /* sets procedure flag */
  fprintf(fp, "\t%s\n", PSfindGC(gc));
  i = numberOfPoints - 1;
  while (i > 0)
  { fprintf(fp, "\t%d\t%d\n", points[i].x, points[i].y);
    i = i-1;
  }
  fprintf(fp, "\t%d\t%d\t%d\tpsFillwOutline\n",
          numberOfPoints, points[i].x, points[i].y);
  return (fclose(fp));
}
```

8.1.31 TrivEqual

⟨*gfun.c*⟩+≡

```
static int TrivEqual(Window s1,Window s2) {
  return ( s1 == s2);
}
```

8.1.32 TrivHashCode

⟨*gfun.c*⟩+≡

```
static int TrivHashCode(Window s,int size) {
  return (s % size);
}
```

8.1.33 XCreateAssocTable

⟨*gfun.c*⟩+≡

```
HashTable *XCreateAssocTable(int size) {
  HashTable * table;
  table = (HashTable *) malloc(sizeof(HashTable));
  hash_init(table,size,(EqualFunction)TrivEqual,
            (HashcodeFunction)TrivHashCode);
  return table;
}
```

8.1.34 XMakeAssoc

⟨*gfun.c*⟩+≡

```
void XMakeAssoc(Display * dsp, HashTable *table, Window w, int * p) {
  hash_insert(table,(char *) p, (char *) w);
}
```

8.1.35 XLookUpAssoc

⟨*gfun.c*⟩+≡

```
int *XLookUpAssoc(Display * dsp, HashTable *table,Window w) {
  return (int *) hash_find(table,(char *)w);
}
```

8.1.36   XDeleteAssoc

⟨*gfun.c*⟩+≡

```
void XDeleteAssoc(Display * dsp,HashTable * table, Window w) {
  hash_delete(table,(char *) w);
}
```

8.2   The postscript command definitions

8.2.1   colorpoly

```
operand stack configuration in order to use psColorPoly:
      psFillPoly
      XPoint[0].y
      XPoint[0].x
      n
      ...
      XPoint[n].y
      XPoint[n].x
      graphics-context dictionary
this draws a polygon by connecting all the points and fills the
region with foreground color
```

⟨*psfiles/colorpoly*⟩≡

```
  /psColorPoly
          {        gsave
                   newpath
                   yVal moveto
                   1 sub {
                           yVal lineto
                   } repeat
                   closepath
                   fill    %% fills with foreground color
                   grestore }
          def
```

8.2.2   colorwol

```
operand stack configuration in order to use psDrawFilled:
        psFillwOL
        XPoint[0].y
        XPoint[0].x
        n
        ...
        XPoint[n].y
        XPoint[n].x
        graphics-context dictionary
this draws lines connecting all the points and fills the
region with background color (default: 1, or white).
```

⟨*psfiles/colorwol*⟩≡

```
  /psColorwOutline
          {        gsave
                   newpath
                   yVal moveto
                   1 sub {
                           yVal lineto
                   } repeat
                   closepath
                   begin gsave fill grestore        %% fills with foreground color
                   0 setgray stroke grestore end } %% outline it with black
          def
```

8.2.3 drawarc

```
operand stack configuration in order to use psDrawArc:
      psDrawArc
      angle2
      angle1
      width
      height
      y
      x
      graphics-context dictionary
this draws an arc whose origin is at x, y, and whose width
and height specifies the rectangle which encases the arc.
Origin is at upper left corner of rectangle.
This function uses "scale" to make cricles and ellipses.
```

⟨*psfiles/drawarc*⟩≡

```
  /psDrawArc
          {       gsave
                  newpath
                  /sfactor 4 index 4 index div def %% scale factor
                  1 sfactor scale
                  6 5 roll                 %%       x on top of stack
                  3 index 2 div add        %% define x origin
                  6 5 roll                 %%       y on top of stack
                  6 5 roll                 %%       h on top of stack
                  2 div add yVal sfactor div   %% define y origin
                  5 4 roll                 %%       w on top of stack
                  2 div                    %% define radius
                  5 3 roll                 %%      a1 a2 on top of stack
                  1 index add
                  arcn                     %% draw clockwise arc
                  begin installGC stroke end
                  grestore }
          def
```

8.2.4   drawcolor

```
operand stack configuration in order to use psDrawColor:
        psDraw
        vlist[0].y
        vlist[0].x
        n
        ...
        vlist[n].y
        vlist[n].x
        graphics-context dictionary
to draw lines connecting points in vlist[0] to vlist[n]
```

⟨*psfiles/drawcolor*⟩≡

```
  /psDrawColor
          {       gsave
                  newpath
                  yVal moveto              %% set currentpoint
                  1 sub {                  %% loop to draw lines.
                          yVal lineto
                  } repeat
                  stroke %% draw in foreground color
                  grestore }
          def
```

8.2.5 drawIstr

```
operand stack configuration in order to use psDrawIStr:
      psDrawIStr
      window type: title or window
      string
      y
      x
      graphics-context dictionary
it draws a text string in foreground color on top of bounding box of
string, which is in background color.
```

$\langle psfiles/drawIstr\rangle\equiv$

```
  /psDrawIStr
          {       gsave
                  newpath                              %% for rectangle
                  loadFont

                  /window exch def                     %% get window type

                  %% draw bounding box with background color
                  /str exch def                        %% get text string
                  str stringwidth pop 1 sub            %% width
                  FontHeight 1 sub                     %% height
                  currentfont begin                    %% get font height
                          FontBBox
                  end
                  /ypos exch def pop                   %% define ypos
                  neg ypos add /offset exch def pop
                  /offset ypos offset div FontHeight mul def %% define offset
                  /h exch def /w exch def              %% define h
                  /y0 exch def                         %% define y0
                  /x0 exch def                         %% define x0
                  w h x0 y0 offset sub
                  window (title) eq
                  {hVal moveto drawRect}               %% draws in title window
                  {rectangle} ifelse                   %% draws in view window
                  begin
                  BGcolor setgray fill    %% set background box color

                  x0 y0
                  window (title) eq
                  {hVal}                               %% print title text
                  {yVal} ifelse                        %% print window text
                  moveto str
                  FGcolor setgray show                 %% set text color
                  end
```

```
                    grestore }
            def
```

8.2.6   drawline

```
operand stack configuration in order to use psDrawLine:
      psDrawLine
      y0
      x0
      y1
      x1
      graphics-context dictionary
this draws a line from (x0, y0) to (x1, y1).
```

⟨*psfiles/drawline*⟩≡
```
  /psDrawLine
            {           gsave
                        newpath
                        yVal moveto
                        yVal lineto
                        begin installGC stroke end
                        grestore }
            def
```

8.2.7 drawlines

```
operand stack configuration in order to use psDrawLines:
      psDrawLines
      points[0].y
      points[0].x
      n
      ...
      points[n].y
      points[n].x
      graphics-context dictionary
this draws lines connecting all the points.
```

⟨*psfiles/drawlines*⟩≡
```
  /psDrawLines
          {       gsave
                  newpath
                  yVal moveto
                  1 sub {
                          yVal lineto
                  } repeat
                  begin installGC stroke end
                  grestore }
          def
```

8.2.8 drawpoint

```
operand stack configuration in order to use psDrawPoint:
      psDrawPoint
      y0
      x0
      graphics-context dictionary
this draws a point at (x0, y0).
```

⟨*psfiles/drawpoint*⟩≡
```
  /psDrawPoint
          {       gsave
                  newpath
                  yVal moveto
                  yVal lineto
                  begin installGC stroke end        %%fills with foreground color
                  grestore }
          def
```

8.2.9   draw

```
operand stack configuration in order to use psDraw:
        psDraw
        vlist[0].y
        vlist[0].x
        n
        ...
        vlist[n].y
        vlist[n].x
        graphics-context dictionary
to draw lines connecting points in vlist[0] to vlist[n]
```

⟨*psfiles/draw*⟩≡

```
  /psDraw
          {         gsave
                    newpath
                    yVal moveto             %% set currentpoint
                    1 sub {                 %% loop to draw lines.
                            yVal lineto
                    } repeat
                    begin installGC stroke end  %% draw in foreground color
                    grestore }
          def
```

8.2.10   drawrect

```
operand stack configuration in order to use psDrawRect:
        psDrawRect
        y
        x
        height
        width
        graphics-context dictionary
this draws an outline of a rectangle whose origin is at (x,y) and is width
+ 1 wide and height + 1 tall.
```

⟨*psfiles/drawrect*⟩≡

```
  /psDrawRect
          {         gsave
                    newpath
                    rectangle
                    begin installGC stroke end
                    grestore }
          def
```

8.2.11   drawstr

```
operand stack configuration in order to use psDrawStr:
      psDrawStr
      y
      x
      string
      graphics-context dictionary
this function draws a text string at (x,y).
```

⟨*psfiles/drawstr*⟩≡
```
  /psDrawStr
          {         gsave
                    newpath
                    loadFont
                    yVal moveto
                    exch begin installGC show end
                    grestore }
          def
```

8.2.12   drwfilled

```
operand stack configuration in order to use psDrawFilled:
      psDrawFilled
      vlist[0].y
      vlist[0].x
      n
      ...
      vlist[n].y
      vlist[n].x
      graphics-context dictionary
this draws lines connecting all the points and fills the
region with background color (default: 1, or white).
```

⟨*psfiles/drwfilled*⟩≡
```
  /psDrawFilled
          {         gsave
                    newpath
                    yVal moveto
                    1 sub {
                            yVal lineto
                    } repeat
                    begin installGC fill end        %% fills with foreground color
                    grestore }
          def
```

8.2.13   end

⟨*psfiles/end*⟩≡

```
    cleartomark                                 %% clearing operand stack

  end              %% pops mainDict from dictionary stack

  showpage

  %------------------------ end ------------------------%
```

8.2.14 fillarc

operand stack configuration in order to use psFillArc:
```
      psFillArc
      y center of rectangle
      x center of rectangle
      angle2
      angle1
      width
      height
      y
      x
      graphics-context dictionary
```
this draws and fills an arc whose origin is at x, y, and whose width
and height specifies the rectangle which encases the arc.
Origin is at upper left corner of rectangle.
This function uses "scale" to make cricles and ellipses.

⟨*psfiles/fillarc*⟩≡
```
  /psFillArc
          {       gsave
                  yVal moveto
                  newpath
                  /sfactor 4 index 4 index div def
                  1 sfactor scale
                  6 5 roll                      %%        x on top of stack
                  3 index 2 div add             %% define x origin
                  6 5 roll                      %%        y on top of stack
                  6 5 roll                      %%        h on top of stack
                  2 div add yVal sfactor div    %% define y origin
                  5 4 roll                      %%        w on top of stack
                  2 div                         %% define radius
                  5 3 roll                      %%        a1 a2 now on top
                  1 index add
                  arc                           %% draw clockwise arc
                  begin installGC fill end      %% fills with foreground color
                  grestore }
          def
```

8.2.15   fillpoly

```
operand stack configuration in order to use psDrawFilled:
        psFillPoly
        XPoint[0].y
        XPoint[0].x
        n
        ...
        XPoint[n].y
        XPoint[n].x
        graphics-context dictionary
this draws a polygon by connecting all the points and fills the
region with foreground color
```

⟨*psfiles/fillpoly*⟩≡
```
  /psFillPoly
          {         gsave
                    newpath
                    yVal moveto
                    1 sub {
                            yVal lineto
                    } repeat
                    closepath
                    begin installGC fill end          %% fills with foreground color
                    grestore }
          def
```

8.2.16   fillwol

```
operand stack configuration in order to use psDrawFilled:
      psFillwOL
      XPoint[0].y
      XPoint[0].x
      n
      ...
      XPoint[n].y
      XPoint[n].x
      graphics-context dictionary
this draws lines connecting all the points and fills the
region with background color (default: 1, or white).
```

⟨*psfiles/fillwol*⟩≡
```
  /psFillwOutline
          {        gsave
                   newpath
                   yVal moveto
                   1 sub {
                           yVal lineto
                   } repeat
                   closepath
                   begin installGC
                           gsave fill grestore      %% fills with foreground color
                   0 setgray stroke grestore end } %% outline it with black
          def
```

8.2.17   header

⟨*psfiles/header*⟩≡
```
%!PS-Adobe-2.0
%%DocumentFonts: Times-Roman
%%Creator: Axiom
%%CreationDate: today
%%Pages: 1
%%processing (hard) limit: 250 pts or 500 values for the operand stack.
%%EndComments


%---------------------------- prologue ----------------------------%
%------------------------ support procedures ------------------------%


%--------- first create user dictionary with 100 entries max -----------%
%          (number can be changed to accomodate definitions)          %

100     dict    begin            %% using 100 entries in top level dictionary


/FontHeight     12 def


/inch
        {       72 mul }
        def


% yVal and hVal are necessary because the Xwindow display origin
% is at the upper left corner, while the postscript display
% origin is at the lower left hand corner.

/yVal           %% get Y value -- make upper left corner origin
        {       maxY exch sub  }        %% maxY is viewWindow height
        def


/hVal           %% get H value -- used for displaying title text
        {       maxH sub abs }  %% maxH is viewWindow height+titleWindow height
        def

% loads in the font

/loadFont
        {       /Times-Roman findfont FontHeight scalefont setfont }
        def

% draws a rectangle with input operand:
```

```
%               height
%               width
% notice that this function does not "draw" or ink the rectangle.
/drawRect
        {       1 index 1 add 0 rlineto         %% draw first side
                0 exch 1 add neg rlineto        %% draw second side
                1 add neg 0 rlineto             %% draw third side
                closepath }                     %% draw fourth side
        def


% create a rectangle with input operand in the view window:
%               y
%               x
%               height
%               width
% notice that this function does not "draw" or ink the rectangle.
/rectangle
        {       yVal moveto                     %% set currentpoint for line
                drawRect }                      %% draws the rectangle
        def


% These are global variables that every draw procedure uses
% THe operand should be as follows:
%               viewWindow width
%               viewWindow height
%               title height
/setDim
        {       /maxX exch def                  %% width of display
                /maxY exch def                  %% height of display
                /titleH exch def                %% height of title
                /maxH maxY titleH add def       %% height of display + title
        } def


%----------------------- major procedures -------------------------%


/title          %% draws a rectangle around the title of picture
        {       gsave
                newpath
                moveto                          %% lower left of title
                titleH 1 add 0 exch rlineto     %% draw first side
                1 add 0 rlineto                 %% draw second side
                1 add neg 0 exch rlineto
                begin installGC stroke end      %% draw third side
                grestore }
        def
```

```
/drawFrame        %% draw display frame
        {         gsave
                  newpath
                  maxX maxY 0 0 rectangle
                  begin installGC stroke end
                  grestore }
        def

% updates the foreground color of existing graphics-context dictionary:
%                 foreground color
%                 dictionary name
/setForeground
        {         /FGcolor exch put }
        def

% updates the background color of existing graphics-context dictionary:
%                 background color
%                 dictionary name
/setBackground
        {         /BGcolor exch put }
        def

% updates the line width, line style, cap style, join style of
% existing graphics-context dictionary:
%                 dictionary name
%                 join style
%                 cap style
%                 line width
/setLineAttributes
        {         begin
                  /JoinStyle exch def
                  /CapStyle  exch def
                  /LineWidth exch def
                  end }
        def

% creates a graphics context dictionary with the following information:
%                 /dictionary name
%                 foreground color
%                 background color
%                 line width
%                 cap style
%                 join style
% this creates different graphical contexts for different drawing functions.
/makeDict
        {         5 dict 2 copy def begin pop %% with dict name on top of stack
```

```
                    /FGcolor   exch def      %% define drawing attributes
                    /BGcolor   exch def      %% not heavily used
                    /LineWidth exch def
                    /CapStyle  exch def
                    /JoinStyle exch def
                    end }
            def

    % makes the current dictionary attributes effective
    % this function takes the values in the current dictionary to set the context
    % these are the values currently being used: foreground, cap, join, and width
    /installGC
            {
                    FGcolor currentgray ne
                    {FGcolor setgray} if          %% foreground color
                    CapStyle currentlinecap ne
                    {CapStyle setlinecap} if       %% cap style
                    JoinStyle currentlinejoin ne
                    {JoinStyle setlinejoin} if     %% join style
                    LineWidth currentlinewidth ne
                    {LineWidth setlinewidth} if }  %% line width
            def
```

8.2.18   setup

⟨*psfiles/setup*⟩≡

```
  %------------------------ script ------------------------%

  % 1 inch 1 inch translate

     mark                                 %% mark bottom of our stack
```

Chapter 9

The APIs

9.1 Graphics API

9.1.1 XDrawString

XDrawString draws an 8-bit text string, foreground only.

```
XDrawString (Display *display, Drawable drawable, GC gc,
             int x, int y, char *string, int length);
```

- `display` – Specifies a pointer to the Display structure; returned from XOpenDisplay.

- `drawable` – Specifies the drawable.

- `gc` – Specifies the graphics context.

- `x` – Specify the x coordinate of the baseline starting position for the character, relative to the origin of the specified drawable.

- `y` – Specify the y coordinate of the baseline starting position for the character, relative to the origin of the specified drawable.

- `string` – Specifies the character string.

- `length` – Specifies the number of characters in the string argument.

XDrawString draws the given string into a drawable using the foreground only to draw set bits in the font. It does not affect any other pixels in the bounding box for each character. The y coordinate defines the baseline row of pixels while the x coordinate is the point for measuring the ibearing, rbearing, and width from.

XDrawString uses these graphics context component: function, plane_mask, fill_style, font, subwindow_mode, clip x origin, clip_y_origin, and clip_mask. This function also uses these

517

graphics context mode-dcpendent components: foreground, tile, stipple, ts x origin, and ts_y_origin. Each character image, as defined by the font in gc, is treated as an additional mask for a fill operation on the drawable.

9.1.2   XDrawPoint

From the Xlib Drawing Primitives, XDrawPoint draws a point.

```
XDrawPoint (Display *display, Drawable drawable, GC gc, int x, int y);
```

Arguments

- `display` – Specifies a pointer to the Display structure; returned from XOpenDisplay.

- `drawable` – Specifies the drawable.

- `gc` – Specifies the graphics context.

- `x` – Specify the x coordinate of the point, relative to the corner of the drawable.

- `y` – Specify the y coordinate of the point, relative to the corner of the drawable.

XDrawPoint uses the foreground pixel and function components of the graphics con- text to draw a single point into the specified drawable. XDrawPoint uses these graphics context components: function, plane_mask, foreground, subwindow_mode, clip x origin, clip_y_origin, and clip_mask.

9.1.3   XDrawLine

From the Xlib Drawing Primitives, XDrawLine draws a line between two points.

```
XDrawLine(Display *display, Drawable drawable, GCgc,
          int x1, int y1, int x2, int y2);
```

- `display` – Specifies a pointer to the Display structure; returned from XOpenDisplay.

- `drawable` – Specifies the drawable.

- `gc` – Specifies the graphics context.

- `x1` – Specify the x coordinate of the start point of the line relative to the drawable origin.

- `y1` – Specify the y coordinate of the start point of the line relative to the drawable origin.

- `x2` – Specify the x coordinate of the end point of the line relative to the drawable origin.

- y2 – Specify the y coordinate of the end point of the line relative to the drawable origin.

XDrawLine connects point (xl, yl) to point (x2, y2). XDrawLine uses the components of the specified graphics context to draw a line between two points in the specified drawable. No pixel is drawn more than once.

XDrawLine uses these graphics context components: function, plane_mask, line_width, line_style, cap_style, fill_style, subwindow_mode, clip_ x_origin, clip_y_origin, and clip_mask. XDraw-Line also uses these graphics context mode-dependent components: foreground, background, tile, stipple, ts_x_origin, ts_y_origin, dash_offset, and dash_list. XDrawLine is not affected by tile or stipple in the GC.

9.1.4 XDrawImageString

XDrawlmageString draws image text characters.

```
XDrawImageString(Display *display, Drawable drawable, GC gc, int x, int y,
                char *string, int length);
```

- `display` – Specifies a pointer to the Display structure; returned from XOpenDisplay.

- `drawable` – Specifies the drawable.

- `gc` – Specifies the graphics context.

- `x` – Specify the x coordinate of the baseline starting position for the image text character, relative to the origin of the specified drawable.

- `y` – Specify the y coordinate of the baseline starting position for the image text character, relative to the origin of the specified drawable.

- `string` – Specifies the character string

- `length` – Specifies the number of characters in the string

XDrawImageString uses these graphics context components: plane_mask, foreground, background, font, subwindow_mode, clip x origin, clip_y_origin, and clip_mask. The function and fill_style defined in gc are ignored; the effective function is GXcopy and the effective fill_style is FillSolid. XDrawImageString first fills a destination rectangle with the background pixel defined in gc, and then prints the text with the foreground pixel. The upper-left corner of the filled rectngle is at Ix, y - font_ascent], the width is overall-width and the height is ascent + descent. The overall-width, ascent, and descent are as would be returned by XQuery-TextExtents using gc and string.

9.1.5   XFillArc

From the Xlib Drawing Primitives, XFillArc fills an arc.

```
XFillArc (Display *display, Drawable drawable, GC gc,
          int x, int y, unsigned int width, unsigned int height,
          int angle1, int angle2);
```

Arguments

- `display` – Specifies a pointer to the Display structure; returned from XOpenDisplay.

- `drawable` – Specifies the drawable.

- `gc` – Specifies the graphics context.

- `x` – Specify the x coordinate of the upper-left corner of the bounding box containing the arc, relative to the origin of the drawable.

- `y` – Specify the y coordinate of the upper-left corner of the bounding box containing the arc, relative to the origin of the drawable.

- `width` – Specify the width in pixels. These are the major and minor axes of the arc.

- `height` – Specify the height in pixels. These are the major and minor axes of the arc.

- `angle1` – Specifies the start of the arc relative to the three-o'clock position from the center. Angles are specified in degrees, multiplied by 64.

- `angle2` – Specifies the path and extent of the arc relative to the start of the arc. Angles are specified in degrees, multiplied by 64.

Description

XFillArc fills an arc according to the arc mode in the GC. The x, y, width, and height arguments specify the bounding box for the arc. See XDrawArc for the description of how this bounding box is used to compute the arc. Some, but not all, of the pixels drawn with XDrawArc will be drawn by XFilIArc with the same arguments. The arc forms one boundary of the area to be filled. The other boundary is determined by the arc_mode in the GC. If the arc_mode in the GC is ArcChord, the single line segment joining the endpoints of the arc is used. If ArcPieSlice, the two line segments joining the endpoints of the arc with the center point are used.

XFilIArc uses these graphics context components: function, plane_mask, fill_style, arc_mode, subwindow_mode, clip x origin, clip_y_origin, and clip_mask. This function also uses these graphics context mode-dependent components: foreground, background, tile, stipple, ts x origin, and ts_y_ origin.

9.1.6 XDrawArc

From the Xlib Drawing Primitives, XDrawArc draws an arc fitting inside a rectangle.

```
XDrawArc(Display *display, Drawable drawable, GC gc, int x, int y,
        unsigned int width, unsigned int height, int angle1, int angle2);
```

- `display` – Specifies a pointer to the Display structure; returned from XOpenDisplay.

- `drawable` – Specifies the drawable.

- `gc` – Specifies the graphics context.

- `x` – Specify the x coordinate of the upper-left corner of the rectangle that contains the arc, relative to the origin of the specified drawable.

- `y` – Specify the y coordinate of the upper-left corner of the rectangle that contains the arc, relative to the origin of the specified drawable.

- `width` – Specify the width in pixels of the major and minor axes of the arc.

- `height` – Specify the height in pixels of the major and minor axes of the arc.

- `angle1` – Specifies the start of the arc relative to the three-o'clock position from the center. Angles are specified in 64ths of a degree, (360 * 64 is a complete circle).

- `int angle2` – Specifies the path and extent of the arc relative to the start of the arc. Angles are specified in 64ths of a degree, (360 * 64 is a complete circle).

XDrawArc draws a circular or elliptical arc. An arc is specified by a rectangle and two angles. The x and y coordinates are relative to the origin of the drawable, and define the upper-left corner of the rectangle. The center of the circle or ellipse is the center of the rectan- gle, and the major and minor axes are specified by the width and height, respectively. The angles are signed integers in 64ths of a degree, with positive values indicating counter-clockwise motion and negative values indicating clockwise motion, truncated to a maximum of 360 degrees. The start of the arc is specified by angle1 relative to the three-o'clock position from the center, and the path and extent of the arc is specified by angle2 relative to the start of the arc.

By specifying one axis to be 0, a horizontal or vertical line can be drawn. Angles are computed based solely on the coordinate system and ignore the aspect ratio. In other words, if the bounding rectangle of the arc is not square and angle1 is 0 and angle2 is (45x64), a point drawn from the center of the bounding box through the endpoint of the arc will not pass through the corner of the rectangle.

XDrawArc uses these graphics context components: function, plane_mask, line_width, line_style, cap_style, join_style, fill_style, subwindow_ mode, clip x origin, clip_y_origin, and clip_mask. This function also uses these graphics context mode-dependent components: foreground, background, tile, stipple, ts x origin, ts_y_origin, dash_offset, and dash_list. XDrawArc is not affected by the tile or stipple in the GC.

9.1.7   XSetForeground

In Xlib, in the Graphics Context, XSetForeground m set the foreground pixel value in a graphics context.

```
XSetForeground (Display *display, GC gc, unsigned long foreground);
```

Arguments

- `display` – Specifies a pointer to the Display structure; returned from XOpenDisplay.

- `gc` – Specifies the graphics context.

- `foreground` – Specifies the foreground you want to set for the specified graphics context.

9.1.8   XSetBackground

In Xlib, in the Graphics Context, XSetBackground m set the background pixel value in a graphics context.

```
XSetBackground (Display *display, GC gc, unsigned long background);
```

Arguments

- `display` – Specifies a pointer to the Display structure; returned from XOpenDisplay.

- `gc` – Specifies the graphics context.

- `background` – Specifies the background you want to set for the specified graphics context.

9.1.9   XSetLineAttributes

In Xlib, in the Graphics Context, the function XSetLineAttributes sets four types of line characteristics in the GC: line_width, line_style, cap_style, and join_style.

- `line_width` – Specifies the line width you want to set for the specified graphics context. A line_width of zero (0) means to use the fastest algorithm for drawing a line of one pixel width. These lines may not meet properly with lines specified as width 1 or more.

- `line_style` – Specifies the line style you want to set for the specified graphics context. Possible values are LineSolid, LineOnOffDash, or LineDouble-Dash.

- `cap_style` – Specifies the line and cap style you want to set for the specified graphics context. Possible values are CapNotLast, CapButt, CapRound, or CapProjecting.

- `join_style` – Specifies the line-join style you want to set for the specified graphics con- text. Possible values are JoinMiter, JoinRound, or JoinBevel.

9.1.10 DefaultScreen

The DefaultScreen macro returns the default screen number referenced in the XOpenDisplay routine.

9.1.11 RootWindow

The RootWindow macro returns the root window.

9.1.12 XCreateAssocTable

Create a new association table (X10). Note that we have our own version of this function.

```
XAssocTable *XCreateAssocTable(int size)
```

Arguments

- `size` Specifies the number of buckets in the hashed association table.

Description XCreateAssocTable creates an association table, which allows you to associate your own structures with X resources in a fast lookup table. This function is provided for compatibility with X Version 10. To use it you must include the file ¡xll¿ and link with the library -loldX. The size argument specifies the number of buckets in the hash system of XAssocTable. For reasons of efficiency the number of buckets should be a power of two. Some size sugges- tions might be: use 32 buckets per 100 objects; a reasonable maximum number of object per buckets is 8. If there is an error allocating memory for the XAssocTable, a NULL pointer is returned.

Structures

```
typedef struct {
    XAssoc *buckets; /* pointer to first bucket in array */
    int size;        /* table size (number of buckets) */
  } XAssocTable;
```

9.1.13 XOpenDisplay

To open a connection to the X server that controls a display, use XOpenDisplay().

```
Display *XOpenDisplay(char *displayname);
```

Arguments

- `displayname` Specifies the hardware display name, which determines the display and communications domain to be used. On a POSIX-conformant system, if the display-name is NULL, it defaults to the value of the DISPLAY environment variable.

Description

The encoding and interpretation of the display name is implementation dependent. Strings in the Host Portable Character Encoding are supported; support for other characters is implementation dependent. On POSIX-conformant systems, the display name or DISPLAY environment variable can be a string in the format:

`hostname:number.screennumber`

`hostname` Specifies the name of the host machine on which the display is physically attached. You follow the hostname with either a single colon (:) or a double colon (::).

`number` Specifies the number of the display server on that host machine. You may optionally follow this display number with a period (.). A single CPU can have more than one display. Multiple displays are usually numbered starting with zero.

`screennumber` Specifies the screen to be used on that server. Multiple screens can be controlled by a single X server. The screennumber sets an internal variable that can be accessed by using the DefaultScreen() macro or the XDefaultScreen() function if you are using languages other than C.

For example, the following would specify screen 1 of display 0 on the machine named "dual-headed":

`dual-headed:0.1`

The XOpenDisplay() function returns a Display structure that serves as the connection to the X server and that contains all the information about that X server. XOpenDisplay() connects your application to the X server through TCP or DECnet communications protocols, or through some local inter-process communication protocol. If the hostname is a host machine name and a single colon (:) separates the hostname and display number, XOpenDisplay() connects using TCP streams. If the hostname is not specified, Xlib uses whatever it believes is the fastest transport. If the hostname is a host machine name and a double colon (::) separates the hostname and display number, XOpenDisplay() connects using DECnet. A single X server can support any or all of these transport mechanisms simultaneously. A particular Xlib implementation can support many more of these transport mechanisms.

If successful, XOpenDisplay() returns a pointer to a Display structure, which is defined in X11/Xlib.h. If XOpenDisplay() does not succeed, it returns NULL. After a successful call to XOpenDisplay() all of the screens in the display can be used by the client. The screen number specified in the displayname argument is returned by the DefaultScreen() macro (or the XDefaultScreen() function). You can access elements of the Display and Screen structures only by using the information macros or functions.

9.2   X11 API calls

- XSolidColor
  - moColor to define the foreground color

- – moColor_BG to define the background color

- void XMakeAssoc(Display *,HashTable *,Window,int *);

- int *XLookUpAssoc(Display *, HashTable *, Window);

- void XDeleteAssoc(Display * , HashTable * , Window);

- XID x_id; – The X Window System id

- XPoint *

- XLoadQueryFont(dsply,xDefault)

- XEvent

- XClearArea(dsply,control-¿controlWindow,0,0,controlWidth,potA,False);

- XWindowAttributes cwInfo;

- XGetWindowAttributes(dsply,cp-¿controlWindow,&cwInfo);

- XFlush(dsply);

- XQueryTree(dsply,tmpW,&rootW,&parentW,&childrenWs,&nChildren);

- XFree(childrenWs);

- XSetWindowAttributes cwAttrib, controlAttrib;

- XSizeHints sizehints;

- XColor foreColor, backColor;

- XCreateBitmapFromData(dsply,rtWindow,mouseBitmap_bits, mouseBitmap_width,mouseBitmap_height);

- XQueryColor(dsply,colorMap,&foreColor);

- XCreatePixmapCursor(dsply,mousebits,mousemask, &foreColor,&backColor, mouseBitmap_x_hot,mouseBi

- XCreateWindow(dsply,rtWindow, cXY.putX,cXY.putY,controlWidth,controlHeight,3, CopyFromParent,InputOutput,CopyFromParent, controlCreateMASK,&cwAttrib);

- XSetNormalHints(dsply,cw,&sizehints);

- XSetStandardProperties(dsply,cw,"2D Control Panel","2D Control Panel", None,NULL,0,&sizehints);

- XMakeAssoc(dsply,table,(control-¿buttonQueue[i]).self, &((control-¿buttonQueue[i]).buttonKey));

- XMapWindow(dsply,(control-¿buttonQueue[i]).self);

- XRaiseWindow(dsply,control-¿controlWindow);

- XMoveWindow(dsply,control-¿controlWindow,whereControl.putX, whereControl.putY);

- XClearArea(dsply,viewport-¿controlPanel-¿controlWindow, 0,controlMessageY-2,controlWidth
- XFontStruct *globalFont,*buttonFont,*headerFont,*titleFont,
- XrmDatabase rDB; – X resource database
- XrmValue value;
- XGCValues gcVals;
- (XArc *)malloc(xPointsNeeded * sizeof(XArc))
- XSetFont(dsply,gc,font-¿fid)
- XCreateAssocTable(nbuckets);
- XInitSpadFill(dsply,scrn,&colorMap, &totalHues,&totalSolidShades, &totalDitheredAnd-Solids,&totalShades);
- XQueryFont(dsply,XGContextFromGC(DefaultGC(dsply,scrn)));
- XrmGetResource(rDB, "Axiom.2D.messageFont", "Axiom.2D.Font", str_type, &value) == True)
- XLoadQueryFont(dsply, prop)
- XCreateGC(dsply,rtWindow,GCForeground — GCBackground , &controlGCVals);
- XrmInitialize();
- XrmGetFileDatabase(name);
- XrmMergeDatabases(applicationDB, &rDB);
- XResourceManagerString(dsply)
- XrmGetStringDatabase(XResourceManagerString(dsply));
- XResourceManagerString(dsply)
- XSync(dsply,False);
- XUnmapWindow(dsply,control-¿controlWindow);
- XWindowAttributes graphWindowAttrib;
- Xcon = ConnectionNumber(dsply);
- XEventsQueued(dsply, QueuedAlready)
- XPending(dsply)
- XNextEvent(dsply,event);
- (XButtonEvent *)event

- (XEvent *)event)
- XCheckWindowEvent(dsply, viewport-¿titleWindow, ExposureMask, &tempEvent);
- XGetWindowAttributes(dsply, whichWindow, &graphWindowAttrib);
- XResizeWindow(dsply, viewport-¿viewWindow, graphWindowAttrib.width, graphWindowAttrib.height-titleHeight);
- XMapWindow(dsply,whichWindow);
- XCheckMaskEvent(dsply, ButtonMotionMask, event));
- (XButtonEvent *)event)
- XFlush(dsply);
- XUnmapWindow(dsply,control-¿controlWindow);
- XLookUpAssoc(dsply,table,whichWindow));
- XMoveWindow(dsply,viewport-¿titleWindow,i1,i2);
- XResizeWindow(dsply,viewport-¿titleWindow,i1,i2+titleHeight);
- XFreeGC(dsply,globalGC1);
- XFreeFont(dsply,globalFont);
- XFreeColormap(dsply,colorMap);
- XCloseDisplay(dsply);
- XWindowAttributes attribInfo;
- XClearWindow(dsply,viewport-¿titleWindow);
- XCharStruct overall;
- XTextExtents(unitFont,"o",1,&dummyInt,&ascent,&descent,&overall);
- XSolidColor((int)(aPoint-¿hue),(int)(aPoint-¿shade)),
- XTextWidth(unitFont,aunit,strlength)/2;
- Xoption
- XSizeHints titleSizeHints,viewSizeHints;
- XSetWindowAttributes viewAttrib;
- XColor foreColor, backColor;
- XQueryColor(dsply,colorMap,&foreColor);

- XInternAtom(dsply, "WM_DELETE_WINDOW", False);

- XSetWMProtocols(dsply, viewTitleWindow, &wm_delete_window, 1);

- XSetNormalHints(dsply,viewTitleWindow,&titleSizeHints);

- XSetStandardProperties(dsply,viewTitleWindow,"Axiom 2D",viewport->title, None,NULL,0,&

- XWriteBitmapFile(dsply,viewBitmapFilename, viewport->titleWindow,vwInfo.width, vwInfo.h
  1,-1);

- XResizeWindow(dsply,viewport->titleWindow,300,300+titleHeight);

- XDeleteAssoc(dsply,table,(control->buttonQueue[i]).self);

- XDestroyWindow(dsply,control->controlWindow);

- XCreateImage(/* display */ dsply, /* visual */ DefaultVisual(dsply,scrn), /* depth */
  DefaultDepth(dsply,scrn), /* format */ ZPixmap, /* offset */ 0, /* data */ NULL, /*
  width */ vwInfo.width, /* height */ 1, /* bitmap_pad */ 32, /* bytes_per_line */ 0);

- XSizeHints sizehint;

- XGetImage(dsply, slicer_pixmap, 0, 0, slicer_width, slicer_height, AllPlanes, ZPixmap);

- XChangeShade(dsply, i);

- XShadeRectangle(dsply,cp->controlWindow, colormapX + colorOffsetX + i*shadeWidth,
  colormapY + colorOffsetY - 10, shadeWidth, 40);

- XFree(childrenWs);

- XPoint *quadMesh;

- XGetErrorText(display,event->error_code,buffer,511);

- XSetErrorHandler(the_handler);

- XCreateAssocTable(nbuckets);

- XInitSpadFill(dsply,scrn,&colorMap, &totalHues,&totalSolidShades, &totalDitheredAnd-
  Solids,&totalShades);

- XInitShades(dsply,scrn) ;

- XGContextFromGC(DefaultGC(dsply,scrn))

- XCreatePixmap(dsply,viewport->viewWindow, vwInfo.width,vwInfo.height, DisplayPlanes(dsp

- .Xdefaults

- XQueryPointer(dsply,rtWindow,&dummy,&dummy,&px,&py,&lx,&ly,&lbuttons);

- XPixelColor((int)colorindx-1)

- XCreatePixmap(/* display */ dsply, /* drawable */ viewport-¿viewWindow, /* width */ vwInfo.width, /* height */ vwInfo.height, /* depth */ DefaultDepth(dsply,scrn));

- XFillRectangle(dsply,viewmap,trashGC,0,0,vwInfo.width,vwInfo.height);

- XPutPixel(imageX,i,0,foregroundColor);

- XPutImage(dsply,viewport-¿viewWindow,trashGC,imageX,0,0,0, scanline,vwInfo.width,1);

- XDestroyImage(imageX);

- XCopyArea(dsply,viewmap,viewport-¿viewWindow,trashGC,0,0, vwInfo.width,vwInfo.height,0,0);

- XCopyArea(dsply,viewmap,viewport-¿viewWindow,trashGC,0,0, vwInfo.width,vwInfo.height,0,0);

- XFillPolygon(dsply, viewport-¿viewWindow, aGC, quadMesh, p-¿numpts, Convex,CoordModeOrigin);

- XPeekEvent(dsply,&peekEvent);

- XDrawLines(dsply, wid, gc, points, numberOfPoints, mode);

- XGContextFromGC(viewGCx);

- (impl) XCreateAssocTable(int size)

- (impl) XMakeAssoc(Display * dsp, HashTable *table, Window w, int * p)

- (impl) XLookUpAssoc(Display * dsp, HashTable *table,Window w)

- (impl) XDeleteAssoc(Display * dsp,HashTable * table, Window w)

- XSelectInput(dsply, menu, KeyPressMask—ButtonPressMask—ExposureMask);

Chapter 10

Makefile

⟨ *⟩≡
```
BOOK=${SPD}/books/bookvol8.pamphlet
WORK=${OBJ}/${SYS}/graph
OUTLIB= ${MNT}/${SYS}/lib
OUTBIN= ${MNT}/${SYS}/bin
LIB=    ${OBJ}/${SYS}/lib
TESTFILE=${MNT}/${SYS}/graph/parabola.view
PS=     ${MNT}/${SYS}/lib/graph

PSFiles= ${PS}/colorpoly.ps ${PS}/colorwol.ps  ${PS}/draw.ps      \
         ${PS}/drawIstr.ps  ${PS}/drawarc.ps   ${PS}/drawcolor.ps \
         ${PS}/drawline.ps  ${PS}/drawlines.ps ${PS}/drawpoint.ps \
         ${PS}/drawrect.ps  ${PS}/drawstr.ps   ${PS}/drwfilled.ps \
         ${PS}/end.ps       ${PS}/fillarc.ps   ${PS}/fillpoly.ps  \
         ${PS}/fillwol.ps   ${PS}/header.ps    ${PS}/setup.ps

CFLAGS  = ${CCF} -I${SRC}/include
LDFLAGS = ${LDF} -lX11 -lm

VLIBS=${LIB}/sockio-c.o ${LIB}/util.o  ${LIB}/bsdsignal.o

LIBS= ${VLIBS}      ${LIB}/pixmap.o    ${LIB}/halloc.o  ${LIB}/spadcolors.o \
      ${LIB}/hash.o ${LIB}/xspadfill.o ${LIB}/xdither.o ${LIB}/xshade.o

all: ${LIBS} dirs ${OUTLIB}/viewman ${OUTLIB}/view2d ${OUTLIB}/view3d \
     ${OUTBIN}/viewalone ${TESTFILE}/data ${PSFiles}
        @ echo 0 finished ${BOOK}

dirs:
        @ echo 1 making target directories for bookvol8
```

531

```
        @ mkdir -p ${WORK}
        @ mkdir -p ${OUTBIN}
        @ mkdir -p ${OUTLIB}
        @ mkdir -p ${TESTFILE}
        @ mkdir -p ${PS}


${OUTLIB}/viewman: ${BOOK}
        @ echo 2 making ${OUTLIB}/viewman from ${BOOK}
        @( cd ${WORK} ; \
           ${TANGLE} -R"viewman" ${BOOK} >viewman.c ; \
           ${CC} ${CFLAGS} -c viewman.c ; \
           ${CC} ${VLIBS} viewman.o -o ${OUTLIB}/viewman ${LDFLAGS} )


${OUTLIB}/view2d: ${BOOK}
        @ echo 3 making ${OUTLIB}/view2d from ${BOOK}
        @( cd ${WORK} ; \
           ${TANGLE} -R"view2d" ${BOOK} >view2d.c ; \
           ${CC} ${CFLAGS} -c view2d.c ; \
           ${CC} ${LIBS} view2d.o -o ${OUTLIB}/view2d ${LDFLAGS} )


${OUTLIB}/view3d: ${BOOK}
        @ echo 4 making ${OUTLIB}/view3d from ${BOOK}
        @( cd ${WORK} ; \
           ${TANGLE} -R"view3d" ${BOOK} >view3d.c ; \
           ${CC} ${CFLAGS} -c view3d.c ; \
           ${CC} ${LIBS} view3d.o -o ${OUTLIB}/view3d ${LDFLAGS} )


${OUTBIN}/viewalone: ${BOOK}
        @ echo 5 making ${OUTBIN}/viewalone from ${BOOK}
        @( cd ${WORK} ; \
           ${TANGLE} -R"viewalone" ${BOOK} >viewalone.c ; \
           ${CC} ${CFLAGS} -c viewalone.c ; \
           ${CC} ${VLIBS} viewalone.o -o ${OUTBIN}/viewalone ${LDFLAGS} )


${TESTFILE}/data: ${BOOK}
        @ echo 6 making ${TESTFILE} from ${BOOK}
        @ (cd ${TESTFILE} ; \
           ${TANGLE} -R"parabola.view/data" ${BOOK} >data ; \
           ${TANGLE} -R"parabola.view/graph0" ${BOOK} >graph0 )


${PS}/%.ps: ${BOOK}
        @ echo 7 ${PS}/$*.ps from ${BOOK}
        @ ${TANGLE} -R"psfiles/$*" ${BOOK} >${PS}/$*.ps
```

Bibliography

[1] Jenks, R.J. and Sutor, R.S. "Axiom – The Scientific Computation System" Springer-Verlag New York (1992) ISBN 0-387-97855-0

[2] Knuth, Donald E., "Literate Programming" Center for the Study of Language and Information ISBN 0-937073-81-4 Stanford CA (1992)

[3] Daly, Timothy, "The Axiom Website"
http://axiom.axiom-developer.org

[4] Watt, Stephen, "Aldor",
http://www.aldor.org

[5] Lamport, Leslie, "Latex – A Document Preparation System", Addison-Wesley, New York ISBN 0-201-52983-1

[6] Ramsey, Norman "Noweb – A Simple, Extensible Tool for Literate Programming"
http://www.eecs.harvard.edu/ ̃nr/noweb

[7] Daly, Timothy, "The Axiom Literate Documentation"
http://axiom.axiom-developer.org/axiom-website/documentation.html

[8] von Seggern, David "CRC Standard Curves and Surfaces" CRC Press, Inc. 1993 ISBN 0-8493-0196-3

Index

534