TITLE: PLOTSTUFF: A CLASS FOR PLOTTING STUFF FROM OVERTURE
BASED ON:
GL_GRAPHICSINTERFACE: A GRAPHICS INTERFACE BASED
ON OPEN GL
BASED ON:
GENERICGRAPHICS INTERFACE: A GENERIC GRAPHICS
INTERFACE
USER GUIDE, VERSION 1.00

AUTHOR(S): Bill Henshaw

SUBMITTED TO: For Electronic Distribution on the Web

RECEIVED
JAN 21 1997
OSTI

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

By acceptance of this article, the publisher recognizes that the U.S. Government retains a nonexclusive royalty-free license to publish or reproduce the published form of this contribution or to allow others to do so, for U.S. Government purposes.

The Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy.

Los Alamos

Los Alamos National Laboratory
Los Alamos New Mexico 87545

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

PlotStuff: A Class for Plotting Stuff from Overture

based on:

GL_GraphicsInterface: A Graphics Interface Based on OpenGL

based on:

GenericGraphicsInterface: A Generic Graphics Interface

User Guide, Version 1.00

Bill Henshaw

Computing, Information and Communications Division
Los Alamos National Laboratory
Los Alamos, NM, 87545

October 16, 1996

Abstract: **PlotStuff** can be used to interactively plot objects from Overture such as mappings, grids and grid functions. **PlotStuff** can be used to plot contours, surfaces, streamlines and grids. It can also be used to plot one-dimensional line plots. **GL_GraphicsInterface** is a class (from which **PlotStuff** is derived) that implements some standard plotting functions using OpenGL. **GL_GraphicsInterface** is itself derived from the class **GenericGraphicsInterface** which defines some standard plotting functions that are independent of any particular graphics package.

Contents

1	Introduction	3
2	Saving postscript files and including them in TeX documents	4
3	Making mpeg movies	4
4	Creating gif files to add to your www home page	5
5	PlotStuff Function Descriptions	6
5.1	Constructors	6
5.2	Plot a MappedGrid	6
5.3	Contour a realMappedGridFunction	6
5.4	StreamLines of a realMappedGridFunction	7
5.5	Plot a CompositeGrid	7
6	GL_GraphicsInterface Function Descriptions	8
6.1	Constructors	8
6.2	createWindow	8
6.3	destroyWindow	8
6.4	getMenu	9
6.5	hardCopy (save a Postscript File)	9

6.6	plot a Mapping	10
6.7	label: plot a String in normalized coordinates	10
6.8	xlabel: plot a String in 2D world coordinates	11
6.9	xlabel: plot a String in 3D world coordinates	11
6.10	plotAxes	12
7	GenericGraphicsInterface Function Descriptions	13
7.1	Constructors	13
7.2	erase	13
7.3	getMenu	13
7.4	inputString	14
7.5	label	14
7.6	pause	14
7.7	outputString	14
7.8	plot a Mapping	14
7.9	plotAxes	15
7.10	readLineFromCommandFile	15
7.11	readCommandFile	15
7.12	readCommandsFromStrings	15
7.13	redraw	16
7.14	resetView	16
7.15	saveCommandFile	16
7.16	setNormalizedCoordinates	16
7.17	stopReadingCommandFile	16
7.18	setNormalizedCoordinates	16
7.19	turnOnLighting	17
7.20	turnOffLighting	17
7.21	unsetNormalizedCoordinates	17
8	Setting Parameters: PlotStuffParameters, GraphicsParameters	18
9	Writing your own plotting routines with OpenGL	19

List of Figures

1	PlotStuff is used by the grid generator ogen	4
2	Plot of a 2D CompositeGrid	21
3	Contour lines and surface plot of a CompositeGridFunction	22
4	Stream lines of two components of a CompositeGridFunction	23

1 Introduction

This class can be used to plot various stuff. PlotStuff is implemented using OpenGL. Motif is used to create the graphical user interface. This class also supports the reading and writing of command files.

PlotStuff is not to be confused (who thought up these names anyhow !?) with the program plotStuff (little "p"), which is a post-processing routine that reads "show" files created by Ogshow and displays the results using PlotStuff.

Here some of the stuff that this class can do:

- plot a Mapping, MappedGrid or CompositeGrid
- plot contour, surface and streamlines of a 2D MappedGridFunction or CompositeGridFunction
- plot contours on cutting planes through a 3D MappedGridFunction or CompositeGridFunction
- plot 1D line plots of 1D grid functions.
- create and query menus
- read and save command files

It is quite simple to use this class to plot stuff from Overture. Here is an example

```
CompositeGrid cg = ... ;           // define a CompositeGrid somehow
Range all;
realCompositeGridFunction u(cg,all,all,all,2); // create a grid function
u=...                               // give values to u

PlotStuff ps;                       // create a PlotStuff object

ps.plot(cg);                        // interactively plot a CompositeGrid
ps.contour(u)                       // interactively plot contours and/or shaded surface of u
ps.streamLines(u)                   // interactively plot streamlines
...
```

The overlapping grid generator, ogen, used PlotStuff as its graphical interface as shown in figure (1).

Figure (1) shows some of the features of PlotStuff which include

- rotation buttons $x+r$, $x-r$, $y+r$, ... which rotate about the fixed x , y , and z axes (the x axis is to the right, the y -axis is up and the z ed axis is out of the screen.
- shift buttons $x+$, $x-$, $y+$ which shift the screen along a given axis.
- A bigger and smaller button
- An erase button to erase the screen.
- A reset button to reset the view.
- Pop-up menus that are defined by the application or user program.
- The pull-down file menu from which command files can be read/saved and from which the screen can be saved as a post-script file.
- A prompt line which shows a prompt. (Multiple levels of prompts can be concatenated).
- A command line where commands can be typed. Any currently valid command can be typed. Valid commands include any pop-up or pull-down menu item or any of the buttons, $x+r$, $x-r$, $x+$, $x-$, $y+$, bigger etc. When typing a command, only the first distinguishing characters need to be entered.

Optional parameters can be passed to the plotting routines. These parameters can be used to set titles on the plots, or to turn of the plotting of the axes, for example. All parameters are encapsulated in the PlotStuffParameters class. Full details on the parameters contained in a PlotStuffParameters object can be found in section 8

Figure 1: PlotStuff is used by the grid generator ogen

2 Saving postscript files and including them in TeX documents

You can use the `hardCopy` function to save a post-script file. The `save postscript` command is available from the file pull-down menu on the PlotStuff graphics screen.

If you have saved the an image as a postscript file you can use the `epsfig` command in a LaTeX or TeX file to embed the figure as in the following example:

```
\begin{figure} \label{fig:CompositeGrid}
  \begin{center}
    \epsfig{file=cg.ps,height=7.0in}
    \caption{Plot of a 2D CompositeGrid}
  \end{center}
\end{figure}
```

Note that the raster resolution is always the same, independent of the height and width values that you give to `epsfig`. If you make the figure too small, however, some thin lines may be lost on the plot depending on the resolution of the output device. Thus a plot may print ok even though it may look bad on the screen since a laser printer has more resolution than the typical screen.

3 Making mpeg movies

The `plotStuff` program can be used to make mpeg movies. The `movie` and `save` option allows one to create a sequence of ppm files that can be encoded into an mpeg movie using the perl script `Overture/bin/ppm2mpeg`.

You can also make your own ppm files by directly calling the hardCopy function with the ppm file option or by using the save ppm menu from the file pull-down menu.

4 Creating gif files to add to your www home page

To create a gif file (that can be included on one's www home page) one can first create a ppm file by calling the hardCopy function with the ppm file option or by using the save ppm menu from the file pull-down menu. You then need to convert the ppm file to a gif file. I use the convert utility which is now available on many machines. In principle you could also convert a postscript file to a gif file but this doesn't usually work very well because the resolution gets messed up.

5 PlotStuff Function Descriptions

5.1 Constructors

PlotStuff(const String & windowTitle)

Description: Default constructor; the default constructor will open a window

Author: WDH

PlotStuff(int & argc, char *argv[], const String & windowTitle)

Description: This Constructor takes the argc and argv from the main program – The GLUT window manager will strip off any parameters that it recognizes such as the size of the window. See the GLUT manual for further details.

argc (input/output): The argument count to main.

argv (input/output): The arguments to main.

PlotStuff(const bool initialize, const String & windowTitle)

Description: This Constructor will only create a window if the the argument initialize=TRUE To create a window later call createWindow()

initialize (input): If TRUE then a window will be created. If FALSE no window will be created and you will have to call createWindow to make the window.

Author: WDH.

5.2 Plot a MappedGrid

**void plot(const MappedGrid & mg,
PlotStuffParameters & parameters = defaultPlotStuffParameters)**

Description: Plot a MappedGrid and optionally supply parameters that define the plot characteristics. In two-dimensions grid-lines are plotted. In three dimensions, by default only the block boundaries of the grid are plotted. You may also plot grid lines on boundaries and/or plot shaded boundary surfaces.

Grids and boundary conditions are plotted with different colours. Grids are numbered and boundary conditions are numbered. For each number there corresponds a colour. The colour associated with each number is plotted in the lower left corner.

mg (input): MappedGrid to plot.

parameters (input/output): Supply optional parameters to alter plot characteristics.

Author: WDH

5.3 Contour a realMappedGridFunction

**void contour(const realMappedGridFunction & u,
PlotStuffParameters & parameters = defaultPlotStuffParameters)**

Description: Plot contours of a realMappedGridFunction in 2D or 3D. Optionally supply parameters that define the plot characteristics. In two-dimensions plotting options include

- plot shaded surface
- plot (colour) contour lines
- plot wire mesh surface (hidden lines are not supported here due to limitations in OpenGL).
- choose which component to plot

In 3D options include

- plot shaded surface contours on arbitrary planes that cut through the grid
- plot shaded surface contours on boundaries.
- plot (colour) contour lines on the planes or boundaries

u (input): function to plot contours of

parameters (input): supply optional parameters

Author: WDH

5.4 StreamLines of a realMappedGridFunction

```
void streamLines(const realMappedGridFunction & uv,
                 PlotStuffParameters & parameters = defaultPlotStuffParameters)
```

Description: Plot stream lines of a two-dimensional vector field. Optionally supply parameters that define the plot characteristics. This routine draws lines that are parallel to a vector field defined by two components of the grid function uv. By default component values 0 and 1 of the first component of uv are used for “u” and “v”. Plotting options include

- choose the components to use for “u” and “v”.
- choose new plot bounds (to zoom in on a particular region). In this case new streamlines are drawn on the new region as opposed to the plot being simply magnified.

uv (input): function to plot streamlines of.

parameters (input): supply optional parameters

Remarks: • The streamlines are coloured by the relative value of $u^2 + v^2$

- Streamlines that move too slowly are stopped
- There is a maximum number of steps used to integrate any streamline.
- To plot streamlines to cover a CompositeGrid, I make a rectangular background grid that covers some region (this region could be smaller than the entire grid if we are zooming). The number of points on this grid is nxg*nyg. The IntegerArray ig(nxg,nyg) is used to mark cells in the background grid. I draw streamlines starting at the midpoints of the background grid. Whenever a streamline pass through a cell of the background grid I increase the value of ig(i,j) by one. Only two streamlines are allowed per cell or else the streamline is stopped (or never started). In this way streamlines cover the domain in a reasonably uniform manner.

Author: WDH

5.5 Plot a CompositeGrid

```
void plot(const GridCollection & gc,
          PlotStuffParameters & parameters = defaultPlotStuffParameters)
```

Description: Plot a CompositeGrid and optionally supply parameters that define the plot characteristics. In two-dimensions grid-lines are plotted. Interpolation points can be plotted with small circles. In three dimensions, by default only the block boundaries of the grid are plotted. You may also plot grid lines on boundaries and/or plot shaded boundary surfaces.

Grids and boundary conditions are plotted with different colours. Grids are numbered and boundary conditions are numbered. For each number there corresponds a colour. The colour associated with each number is plotted in the lower left corner.

gc (input): CompositeGrid to plot.

parameters (input/output): Supply optional parameters to alter plot characteristics.

Author: WDH

6 GL_GraphicsInterface Function Descriptions

The class PlotStuff is derived from a GL_GraphicsInterface which is derived from a GenericGraphicsInterface. The GL_GraphicsInterface defines a graphics interface using OpenGL.

6.1 Constructors

GL_GraphicsInterface(const String & windowTitle)

Description: Default constructor; the default constructor will open a window

windowTitle (input): Title to appear on the window

Author: WDH

**GL_GraphicsInterface(int & argc,
char *argv[],
const String & windowTitle = nullString)**

Description: This Constructor takes the argc and argv from the main program – The window manager will strip off any parameters that it recognizes such as the size of the window.

argc (input/output): The argument count to main.

argv (input/output): The arguments to main.

windowTitle (input): Title to appear on the window

Author: WDH

GL_GraphicsInterface(const bool initialize, const String & windowTitle = nullString)

Description: This Constructor will only create a window if the the argument initialize=TRUE To create a window later call createWindow()

initialize (input): If TRUE then a window will be created. If FALSE no window will be created and you will have to call createWindow to make the window.

windowTitle (input): Title to appear on the window

Author: WDH.

6.2 createWindow

int createWindow(const String & windowTitle = nullString)

Description: Open the window (but only if one is not already open)

windowTitle (input): Title to appear on the window

Author: WDH

6.3 destroyWindow

int destroyWindow()

Description: destroy the window

Author: WDH

6.4 getMenu

```
int getMenu(const String *menu,
           String & answer,
           const String & prompt =nullString)
```

Description: Create a menu and return with a menu item. Display an optional prompt.

menu (input): The menu is an array of String's (the menu choices) with a empty String indicating the end of the menu choices. For example

```
PlotStuff ps;
String menu[] = { "plot",
                  "erase",
                  "exit",
                  "" };

String menuItem;
int i=ps.getMenuItem(menu,menuItem);
```

To create a cascading menu begin the string with an '>' To end the cascade begin the string with an '<' To end a cascade and start a new cascade begin the string with '<' followed by '>' Here is an example:

```
char *menu1[] = { "plot",
                  ">component",
                      "u",
                      "v",
                      "w",
                  "<erase",
                  ">stuff",
                      "s1",
                      ">more stuff",
                          "more1",
                          "more2",
                      "<s2",
                  "<>apples",
                      "apple1",
                  "<exit",NULL };;
```

answer (output): Return the chosen item.

prompt (input): display the optional prompt message

Errors: Some...

Return Values: On return "menuItem" is set equal to the menu item chosen. The function return value is set equal to the number of the item chosen, starting from zero. Thus, for example, in the above menu if the user picked "erase" the return value would be 1, if the user picked "plot" the return value would be 0.

Author: WDH

6.5 hardCopy (save a Postscript File)

```
int hardCopy(const String & fileName =nullString,
            const hardCopyTypes hardCopyType =postScript)
```

Description: This routines saves the graphics window in hard-copy form.

fileName (input): Optional name for the file to save the plot in. If no name is given then the user is prompted for a name.

hardCopyType (input): Only post-script is currently supported.

Return Values: 1: unable to open the file

Author: WDH

6.6 plot a Mapping

```
void plot(Mapping& map,  
          GraphicsParameters & parameters = nullGraphicsParameters)
```

Description: Plot a mapping. Plot curves, surfaces and volumes in 1,2 or 3 space dimensions. Plot grid lines. In 3D plot shaded surfaces and grid lines.

map (input): Mapping to plot.

parameters (input/output): supply optional parameters to change plotting characteristics.

Errors: Some...

Return Values: none.

Author: WDH

6.7 label: plot a String in normalized coordinates

```
void label(const String & string,  
           const real xPosition,  
           const real yPosition,  
           const real size =.1,  
           const int centering =0 ,  
           const real angle =0.)
```

Description: This routine plots a label in the normalized coordinate system where the screen has dimensions $[-1,1] \times [-1,1]$. This label does NOT rotate or scale with the plot.

string (input): String to draw.

xPosition (input): x coordinate of the string in normalized coordinates, $[-1,1]$. (See the centering argument).

yPosition (input): y coordinate of the string in normalized coordinates, $[-1,1]$. (See the centering argument).

size (input): Size of the characters in normalized coordinates (size=2.0 would fill the whole view).

centering (input): centering=0 means put the centre of the string at (xPosition,yPosition). centering=-1 means put the left end of the string at (xPosition,yPosition). centering=+1 means put the right end of the string at (xPosition,yPosition).

angle (input): Angle in degrees to rotate the string.

Errors: none.

Return Values: none.

Author: WDH

6.8 xlabel: plot a String in 2D world coordinates

```
void xlabel(const String & string,
            const real xPosition,
            const real yPosition,
            const real size =.1,
            const int centering =0 ,
            const real angle =0.)
```

Description: This routine plots a label with position and size in World coordinates, This label DOES rotate and scale with the plot. This version of xlabel plots the label in the $z = 0$ plane.

string (input): String to draw.

xPosition (input): x coordinate of the string in world coordinates. (See the centering argument).

yPosition (input): y coordinate of the string in world coordinates. (See the centering argument).

size (input): Size of the characters in NORMALIZED coordinates.

centering (input): centering=0 means put the centre of the string at (xPosition,yPosition). centering=-1 means put the left end of the string at (xPosition,yPosition). centering=+1 means put the right end of the string at (xPosition,yPosition).

angle (input): Angle in degrees to rotate the string.

Errors: none.

Return Values: none.

Author: WDH

6.9 xlabel: plot a String in 3D world coordinates

```
void xlabel(const String & string,
            const RealArray & x,
            const real size,
            const int centering,
            const RealArray & rightVector,
            const RealArray & upVector)
```

Description: This routine plots a label with position and size in World coordinates, This label DOES rotate and scale with the plot. This version of xlabel plots the string in the plane formed by the vectors rightVector and upVector.

string (input): String to draw.

x(0: 2) (input): x,y,z coordinates of the string in world coordinates. (see the centering entering argument).

size (input): Size of the characters in NORMALIZED coordinates.

centering (input): centering=0 means put the centre of the string at x, centering=-1 means put the left end of the string at x. centering=+1 means put the right end of the string at x.

rightVector(0: 2) (input): The string is drawn to lie parallel to this vector.

upVector(0: 2) (input): This vector defines the "up" direction for the characters. The characters are drawn in the plane defined by the rightVector and the upVector.

Errors: none.

Return Values: none.

Author: WDH

6.10 plotAxes

```
void plotAxes(const RealArray & xBound,  
              const int numberOfDimensions,  
              const String & xAxisLabel =nullString,  
              const String & yAxisLabel =nullString,  
              const String & zAxisLabel =nullString)
```

Description: This routines plots axes along the specified bounds, for the given number of space dimensions.

xBound(0:1,0:2) (input): Bounds to use for the axes. xBound(Start,axis1), xBound(End,axis1), ...

numberOfDimensions (input): Number of space dimensions. This determines how many axes to draw.

Errors: none.

Return Values: none.

Author: WDH

7 GenericGraphicsInterface Function Descriptions

The class PlotStuff is derived from a GL_GraphicsInterface which is derived from a GenericGraphicsInterface. The GenericGraphicsInterface defines functions for displaying a menu and getting a response, for inputting and outputting strings, for reading and writing command files as well as some generic plotting functions. The GenericGraphicsInterface does not actually know how to plot. The GL_GraphicsInterface implements the functions found in the GenericGraphicsInterface using OpenGL.

7.1 Constructors

GenericGraphicsInterface()

Description: Default constructor;

Author: WDH

GenericGraphicsInterface(int & argc, char *argv[])

Description: This Constructor takes the argc and argv from the main program – The window manager will strip off any parameters that it recognizes such as the size of the window.

argc (input/output): The argument count to main.

argv (input/output): The arguments to main.

Author: WDH

7.2 erase

void erase()

Description: Erase the graphics screen.

Author: WDH

7.3 getMenu

**int getMenuItem(const String *menu,
 String & answer,
 const String & prompt =nullString)**

Description: Create a menu and return with a menu item. Display an optional prompt.

menu (input): The menu is an array of String's (the menu choices) with a empty String indicating the end of the menu choices. For example

```
PlotStuff ps;  
String menu[] = { "plot",  
                  "erase",  
                  "exit",  
                  "" };  
  
String menuItem;  
int i=ps.getMenuItem(menu,menuItem);
```

answer (output): Return the chosen item.

prompt (input): display the optional prompt message

Errors: Some...

Return Values: On return "menuItem" is set equal to the menu item chosen. The function return value is set equal to the number of the item chosen, starting from zero. Thus, for example, in the above menu if the user picked "erase" the return value would be 1, if the user picked "plot" the return value would be 0.

Author: WDH

7.4 inputString

```
void inputString(String &answer,  
                const String & prompt =nullString)
```

Description: Input a string after displaying an optional prompt

answer (output): the string that was read

prompt (input): display an optional prompt.

Errors:

Return Values: none.

Author: WDH

7.5 label

```
void label(const String & string,  
          const real xPosition,  
          const real yPosition,  
          const real size,  
          const int centering,  
          const real angle )
```

Description: This utility routines plots a label, this label does NOT rotate or scale with the plot

7.6 pause

```
int pause()
```

Description: Pause and await a response, "continue" or "break"

7.7 outputString

```
void outputString(const String & message)
```

Description: Output a string.

message (input): the string to be output.

Errors:

Return Values: none.

Author: WDH

7.8 plot a Mapping

```
void plot(Mapping & map, GraphicsParameters & parameters)
```

Description: Plot a Mapping, pass optional parameters

Author: WDH

7.9 plotAxes

```
void plotAxes(const RealArray & xBound,
              const int numberOfDimensions,
              const String & xAxisLabel =blankString,
              const String & yAxisLabel =blankString,
              const String & zAxisLabel =blankString)
```

Description: This utility routines plot Axes

xBound(0: 1,0:2) (input): bounds on the axes

numberOfDimensions (input): Number of space dimensions. This determines how many axes to draw.

xAxisLabel (input) : label for x-axis

yAxisLabel (input) : label for y-axis

zAxisLabel (input) : label for z-axis

Author: WDH

7.10 readLineFromCommandFile

```
int readLineFromCommandFile(String & answer )
```

Purpose: Read a line from the command file. Lines beginning with a "*" are treated as comments.

Return values: number of characters read. A return value of zero means that an end-of-file was reached.

Author: WDH

7.11 readCommandFile

```
FILE* readCommandFile(const String & commandFileName =nullString)
```

Description: Start reading a command file.

commandFileName (input): If commandFileName is specified then this should be the name of the command file to read. It will first be opened. If commandFileName is not given then you will be prompted to enter the name of the file.

Errors: Unable to open the file.

Return Values: Pointer to the opened file or NULL if able to open the file.

Author: WDH

7.12 readCommandsFromStrings

```
int readCommandsFromStrings(const String *commands)
```

Description: Start reading commands from an array of Strings, commands terminated by the String=""

commands (input): A list of Strings (commands). There must be a null string, "", to indicate the end of the list.

Errors: unexpected results will occur if there is no null string to terminate the array.

Return Values: 0

Author: WDH

7.13 redraw

`void redraw(bool immediate = FALSE)`

Description: Redraw all graphics display lists. Use this option with `immediate=TRUE` to force a redraw in a command file or to force a redraw when making a "movie".

immediate (input): if `FALSE`, the screen will be redrawn at the next convenient time – this will normally be when the screen is waiting for user input. If `TRUE` the screen will be redrawn immediately.

7.14 resetView

`void resetView()`

Description: Reset the view to default.

7.15 saveCommandFile

`FILE* saveCommandFile(const String & commandFileName =nullString)`

Description: Start saving a command file.

commandFileName (input): If `commandFileName` is specified then this should be the name of the command file to save commands in. It will first be opened. If `commandFileName` is not given then you will be prompted to enter the name of the file.

Errors: Unable to open the file.

Return Values: Pointer to the opened file or `NULL` if able to open the file.

Author: WDH

7.16 setNormalizedCoordinates

`void setNormalizedCoordinates()`

Description: This routine sets the projection and modelview to a normalized system on $[-1,1] \times [-1,1]$. This normalized coordinate system is used to draw things on the screen that are not to be rotated. The `label` function, for example, draws characters in normalized coordinates.

Errors: none.

Return Values: none.

Author: WDH

7.17 stopReadingCommandFile

`void stopReadingCommandFile()`

Description: Stop reading the command file (and close the file)

7.18 setNormalizedCoordinates

`void stopSavingCommandFile()`

Description: Stop saving the command file (and close the file)

7.19 turnOnLighting

void turnOnLighting()

Description: Turn on lighting.

Errors: none.

Return Values: none.

Author: WDH

7.20 turnOffLighting

void turnOffLighting()

Description: Turn off lighting.

Errors: none.

Return Values: none.

Author: WDH

7.21 unsetNormalizedCoordinates

void unsetNormalizedCoordinates()

Description: This routines resets the view to the state before the call to **setNormalizedCoordinates**

Errors: none.

Return Values: none.

Author: WDH

8 Setting Parameters: PlotStuffParameters, GraphicsParameters

Optional parameters can be passed to the various plotting functions. PlotStuff parameters are encapsulated into the PlotStuffParameters Class which is derived from the GraphicsParameters Class. The GraphicsParameters Class defines plotting options for the GenericGraphicsInterface. ...more explanation needed here...

To pass optional parameters you must create an object of type PlotStuffParameters, set parameter values for this object and then pass the object to the plotting function. Here is an example of setting the title on a contour plot and turning off the plotting of the axes:

```
...
PlotStuff ps;                // create a PlotStuff object
PlotStuffParameters psp;     // create an object that is used to pass parameters

psp.set(GI_TOP_LABEL,"My Title"); // Set the title
psp.set(GI_PLOT_THE_AXES,FALSE);  // Do not plot the axes
ps.contour(u,psp);              // plot contours and pass parameters
...
```

Note that the PlotStuffParameters object, psp, will be changed within contour if parameters are changed interactively in the contour window. This means that you can set parameters interactively the first time you call a plotting function and then on subsequent calls the parameters will be remembered.

The following member functions of PlotStuffParameters can be used to set values:

```
int set(const PlotStuffOptions & option, const String & label)
```

```
int set(const PlotStuffOptions & option, int value)
```

```
int set(const PlotStuffOptions & option, real value)
```

```
int set(const PlotStuffOptions & option, const intArray & values)
```

```
int set(const PlotStuffOptions & option, const realArray & values)
```

Here is a list of the various parameters that can be set

option name	value to supply
GLBOTTOMLABEL	String or character array
GLBOTTOMLABEL_SUP_1	String or character array
GLBOTTOMLABEL_SUP_2	String or character array
GLBOTTOMLABEL_SUP_3	String or character array
GLCOLOUR.LINE.CONTOURS	bool, TRUE or FALSE
GLPLOT.COLOUR.BAR	bool
GLPLOT.CONTOUR.LINES	bool
GLPLOT.INTERPOLATION.POINTS	bool
GLPLOT.THE.OBJECT.AND.EXIT	bool
GLPLOT.SHADED.SURFACE	bool
GLPLOT.THE.AXES	bool
GLPLOT.THE.OBJECT	bool
GLPLOT.WIRE.FRAME	bool
GLSET.BACK.GROUND.GRID.FOR.STREAM.LINES	intArray dimension(0:1)
GLSET.COMPONENT.FOR.CONTOURS	int
GLSET.GRIDS.TO.PLOT	boolArray grids(0:?)
GLSET.MAPPING.COLOUR	String or character array
GLSET.MINIMUM.CONTOUR.SPACING	real
GLSET.MIN.AND.MAX.CONTOUR.LEVELS	realArray
GLSET.NUMBER.OF.CONTOUR.LEVELS	int
GLSET.PLOT.BOUNDS	realArray bounds(0:1,0:2)
GLUSE.PLOT.BOUNDS,	bool
GLUSE.PLOT.BOUNDS.OR.LARGER,	bool
GLSET.U.COMPONENT.FOR.STREAM.LINES	int
GLSET.V.COMPONENT.FOR.STREAM.LINES	int
GLTOP.LABEL	String or character array
GLTOP.LABEL.SUB.1	String or character array
GLTOP.LABEL.SUB.2	String or character array
GLTOP.LABEL.SUB.3	String or character array

Here are explanations of some of the less obvious options:

- **GLPLOT.THE.OBJECT.AND.EXIT:** Plot the object and exit without staying in the local menu.
- **GLSET.MINIMUM.CONTOUR.SPACING:** Force the contour spacing to be larger than this amount. This option is useful when the solution is nearly constant except for small variations caused by round-off errors, and you do not want to see the small variaions.
- **GLSET.PLOT.BOUNDS:** Set the plot bounds to be used. The plot bounds are set equal to the given realArray "bounds(0:1,0:2)". Use must also set GLUSE.PLOT.BOUNDS to TRUE if you want these bounds to be used.
- **GLUSE.PLOT.BOUNDS:** Use the current plot bounds found with the GraphicsParameters object.
- **GLUSE.PLOT.BOUNDS.OR.LARGER:** Use plot bounds that are least as large as the current plot bounds – increase the plot bounds if the current object requires it.

9 Writing your own plotting routines with OpenGL

You can easily write your own plotting routines if you know OpenGL. Take a look at the source code for the grid plotter or contour plotter for some examples.

OpenGL is a widely available package for three dimensional plotting. Versions of OpenGL exist for almost all workstations and PC's. Currently on Sun workstations I use Brian Paul's Mesa library – a public domain implementation of OpenGL that runs under X windows. See the OpenGL site on the WWW for further details, <http://www.sgi.com/Technology/OpenGL>. OpenGL is described in the OpenGL Programming Guide and the OpenGL Reference Manual, published by Addison Wesley.

Here is an example section of code that shows how the colour bar is drawn.

```

if( plotColourBar )
{
    // =====Draw the colour Bar=====
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL); // filled polygons
    glShadeModel(GL_SMOOTH); // interpolate colours between vertices

    // The colour bar is drawn in a way that is unaffected by rotations and scalings
    setNormalizedCoordinates(); // sets view to bounds [-1,1]x[-1,1]

    const int numberOfIntervals=50;
    real xLeft=.8, xRight=.85; // place the bar down the right side
    real yBottom=-.75, yTop=.75;
    glBegin(GL_QUAD_STRIP); // begin drawing a "quad-strip"
    for( int i=0; i<numberOfIntervals; i++ )
    {
        y=yBottom+i*(yTop-yBottom)/(numberOfIntervals-1);
        // the next routine chooses a colour from a colour table. It then calls glColor3f(red,green,blue)
        setColourFromTable((y-yBottom)/(yTop-yBottom));
        glVertex2f(xLeft, y);
        glVertex2f(xRight, y);
    }
    glEnd(); // end quad-strip colour bar
    // ---label colour bar---
    int numberOfLabels=numberOfContourLevels;
    glColor3f(0.,0.,0.); // label colour is black
    real size=.01;
    for( i=0; i<numberOfLabels; i++ )
    {
        y=yBottom+i*(yTop-yBottom)/(numberOfLabels-1);
        real alpha=(y-yBottom)/(yTop-yBottom);
        if( max(fabs(uMax), fabs(uMin)) < 10. )
            label(sprintf(buff, " %6.3f", uMin+alpha*(uMax-uMin)), xRight, y, size, -1); // flush left
        else if( max(fabs(uMax), fabs(uMin)) < 100. )
            label(sprintf(buff, " %7.2f", uMin+alpha*(uMax-uMin)), xRight, y, size, -1); // flush left
        else
            label(sprintf(buff, " %6e", uMin+alpha*(uMax-uMin)), xRight, y, size, -1); // flush left
    }
    // ---draw lines on colour bar corresponding to the contour levels
    glBegin(GL_LINES);
    for( i=0; i<numberOfContourLevels; i++ )
    {
        y=yBottom+i*(yTop-yBottom)/(numberOfContourLevels-1);
        glVertex3f(xLeft, y, .1); // raise the lines so we see them
        glVertex3f(xRight, y, .1);
    }
    glEnd();

    unsetNormalizedCoordinates();
}

```

Figure 2: Plot of a 2D CompositeGrid

Figure 3: Contour lines and surface plot of a CompositeGridFunction

Figure 4: Stream lines of two components of a CompositeGridFunction