gltools - an OpenGL based on-line graphics toolbox

Jürgen Fuhrmann Hartmut Langmach

Date : 1999/12/2117 : 18 : 55 *Revision* : 2.14

gltools is a collection of OpenGL rendering utilities. It consists of three layers:

glwin: Interface to the windowing system.

glrnd: Management of an orthogonal rendering volume.

glmesh: Rendering utilities for functions on simplicial meshes including 3D isosurfaces and plane sections.

Further, parallel to the system dependent glwin module , there is the system independent gleps module which organizes vector postscript dump.
For Information, see also the gltools homepage[1].

1 Introduction

1.1 What is the intention of *gltools* ?

You have program and want to make it some pictures. You are developing 3D code and are unable to find any errors in your data arrays. You look for a replacement of GKS which no one seems to have anymore.
Many graphics packages, as the GLUTtoolkit[2] (which has nearly the same intentions as *gltools*), AVS[3] and GRAPE[4] provide a perfect environment.
But for them, one has to register existing code as a callback and/or one has to translate the existing data structures into those of the graphics package. This may be fairly time consuming and difficult. Also, one wants to control the graphics package via the existing code, not vice versa. This is where the *gltools* package comes in.
It should enable one to get easy access to the OpenGL world *on-line* from one's *own* data structures within *existing code*.
It has been tested with OpenGL on SGI Irix 6.x and Compaq Tru64 UNIX as well as with the Mesa package[5] of Brian Paul. It *should* compile with any ANSI-C compiler.

[1] http://www.wias-berlin.de/ gltools
[2] http://www.sgi.com/Technology/openGL/glut.html
[3] http://www.avs.com
[4] http://www.mathematik.uni-freiburg.de/Grape/grape.html
[5] http://www.ssec.wisc.edu/ brianp/Mesa.html

1.2 What is not the intention of *gltools* ?

It is intended to keep this package compact. There are two main reasons for this:

- the limited time of the authors
- the ease of use of *gltools* for the programmer.

So, it is until now not planned to incorporate menu control into the package. An easy (and most preferred by the author) way to provide menus would be a callback to a scripting language like *tcl* which could be equipped with a menu system, but this makes sense only when the whole code is embedded into such a language.

1.3 Sample code

glwexample-appctrl.c contains a simple test program with a GL window in application control mode.

glwexample-evctrl.c contains a simple test program with a GL window in event control mode.

glrexample.c contains a simple test program with a GL rendere.

glview.c contains a sample program which can be used to render function data on rectangular meshes. It is used as a test program for *glrnd* and *glmesh*.

1.4 To Do

- vector field rendering
- handle arbitrary clipping planes – this is only a question of a clever keyboard and mouse interface, *glmesh* already does everything.
- relate shown values in the title to real values.
- control rotation axes : keys X,Y,Z

2 glwin - A System Interface for OpenGL Applications

Revision : 2.43
Date : 2002/09/2408 : 55 : 44
Author: Jürgen Fuhrmann

This module provides access to basic facilities as X window management, event handling, off screen rendering etc. It has been derived from the OpenGL sample *tk* code by putting all global variables into a structure called `glWindow`. Not

everything has been checked, especially not the color index stuff. I is the intention of the author to keep track of changes in the *libtk* toolkit. There are three main difference to the *libtk* kit:

- *glwin* is able to to manage more than one window at one time. This has been achieved by wrapping all global variables of *libtk* into a (hidden) data type `glWindow` and letting the user provide data to all callback routines via a `void*` parameter.
- The event loop (called piecewise by `glwProcess()`) knows an application-controlled and an event-controlled (user-controlled) mode.

  In application-controlled mode it is possible to attach a GL window, let it process any pending events, render user data and exit without destroying the window contents. This enables the usage of *glwin* under the control of the application during transient simulations etc.

  In event-controlled mode, the event loop needs some user itervention (e.g. key press) to exit.
- The threaded version has an infitely running event loop. Setting the event controlled mode blocks the applicattion thread, giving a similar feeling.
- `glwGrab()` grabs the actual window contents and dumps it into a ppm file.
- `glwStartMPEGRecording()/glwStopMPEGRecording` record an MPEG stream using mpeg_encode version 1.5 or higher.
- `glwStartEPSRecording()/glwStopEPSRecording` record to a multipage eps or pdf file

2.1 Acknowledgements

The kernel of the code has been derived from various versions of the "tk nano window toolkit" which comes along as OpenGL demo software. Namely, code comes from versions on SGI and Digital machines, and from the MESA distribution. If that code wouldn't have been available in source form, gltools would not exist.

2.2 Installation

The installation should be fairly simple. You need an ANSI C compiler, the headers and libraries of X11 and GL and. Sorry for not providing Imakefiles and/or configure scripts - to me, these are a mess, and I try not to use anyting non-portable in my code.

2.2.1 Installation with vendor installed OpenGL

Everything should be smooth, you don't need to define any preprocessor options. Link with "`-lGL -lXext -lX11`".

2.2.2 Installation with MESA

Obtain MESA from `http://www.ssec.wisc.edu/ brianp/Mesa.html` and install it - the best way would be in `/usr/local/*`.
Then compile `glwin.c` with "`-I/usr/local/include`" and link with "`-L/usr/local/lib -lMesaGL -lXext -lX11`" (or whatever you have called the MESA libraries).

2.3 Imported packages

```
#ifndef GLW_HEADER
#define GLW_HEADER

#include <GL/gl.h>
#include <stdio.h>
```

2.4 Data Types

2.4.1 glWindow

```
typedef  struct glWindowStruct *glWindow;
```

This is the basic handle structure, all data in it are hidden from the user.

2.5 Constructors and Destructors

```
int glwDisableThreads(void);
int glwEnableThreads(void);
```

Any of these routines need to be called before any other call to glwin. If -DPTHREADS is defined during compile time, the user can still decide to disable threading at run time. Otherwise, these calls are meaningless.

2.5.1 glwInitDisplayMode

```
void glwInitDisplayMode(
                        GLenum mode
                        );
```

Set the display mode for next window to create. The default value is `GLW_RGB|GLW_DOUBLE|GLW_DIRECT|GLW_DEPTH`. Possible values are combinations of the following bitmasks:

```
#define GLW_RGB          0       /* RGB mode */
#define GLW_INDEX        1       /* Color index mode */
#define GLW_SINGLE       0       /* Single buffer mode */
#define GLW_DOUBLE       2       /* Double buffer mode */
#define GLW_DIRECT       0       /* Direct rendering */
#define GLW_INDIRECT     4       /* Indirect rendering */
#define GLW_ACCUM        8       /* Enable accumulation buffer */
#define GLW_ALPHA        16      /* Enable alpha calculations */
#define GLW_DEPTH        32      /* Enable depth buffer */
```

```
#define GLW_OVERLAY     64       /* Create Window with overlay */
#define GLW_UNDERLAY    128
#define GLW_STENCIL     512
#define GLW_PIXMAP     1024
```

At startup and after creating a window, all values are reset to the defaults. The values set are used only for the next window created.

2.5.2 glwInitPos

```
void glwInitPos(
                    int x,
                    int y
                    );
```

Set the initial position of the upper left corner of the window on the screen. Default: $(0, 0)$ (Upper left corner of the screen).

2.5.3 glwInitSize

```
void glwInitPos(
                    int x,
                    int y
                    );
#define GLW_SIZE_MPEG1_PAL  352,288
#define GLW_SIZE_MPEG2_PAL  704,576
#define GLW_SIZE_MPEG1_NTSC 352,240
#define GLW_SIZE_MPEG2_NTSC 704,480

#define GLW_SIZE_1 1024,768
#define GLW_SIZE_2 1280,1024
#define GLW_SIZE_3 1600,1200
```

Set the size of the window on the screen.
Default: GLW_PAL_HALF. The sizes are not mandatory. The MPEG-1 standard does not demand any frame size, the MPEG-2 standard demans a multiple of 16 in the frame size.
However it is useful to stick to the MPEG sizes given here if one wants to use MPEG in connection with hardware.

2.5.4 glwInitTitle

```
void glwInitTitle(
                  char * title
                  );
```

Set the title of the window to be created. Default: "gltools-1.0"
At startup and after creating a window, all values are reset to the defaults. The values set are used only for the next window created.
The maximum length of a name is defined here:

```
#define GLW_NAMELEN 128
```

2.5.5 glwInitDisplayModePolicy

```
void glwInitDisplayModePolicy(
                              GLenum type
                              );
```

Set visual match mode. Default: `GLW_MINIMUM_CRITERIA`.
Possible values:

```
enum {
    GLW_USE_ID = 1,
    GLW_EXACT_MATCH,
    GLW_MINIMUM_CRITERIA
};
```

At startup and after creating a window, all values are reset to the defaults. The values set are used only for the next window created.

2.5.6 glwInitOffscreen

```
void glwInitOffscreen(void);
```

Declare window to be created to be an offscreen window. This could be used for e.g. for creating frames in batch mode. The corresponding code is in beta status.
At startup and after creating a window, all values are reset to the defaults. The values set are used only for the next window created.

```
void glwInitToplevel(void *toplevel);
```

Declare the toplevel window of the system window to be created. Under X11, it is used as the target for the "transient_for" property.

2.5.7 glwInitAspectKeeping

```
void glwInitAspectKeeping(
                          int yesno
                          );
```

Keep aspect ratio of window when resizing. Default: 1 (yes).
At startup and after creating a window, all values are reset to the defaults. The values set are used only for the next window created.

2.5.8 glwGetDefault

```
char * glwGetDefault(char * resource, char *dflt);
```

Read resource from resource database, e.g. reads gltools*printerCommand from your .Xdefaults. Returns default if resource has not been found in database.

2.5.9 glwCreate

```
glWindow  glwCreate(
                    void
                    );
```

Create a window. Multiple windows may be created. OpenGL renders into the window freshly created or the window defined by `glwAttach` (2.6.1).

2.5.10 glwDestroy

```
void      glwDestroy(
                     glWindow win
                     );
```

Destroy the window.

2.6 Control

2.6.1 glwAttach

```
int       glwAttach(
                    glWindow win
                    );
```

"Attach" the window, i.e. make the corresponding OpenGL context current.

2.6.2 glwFlush

```
void       glwFlush(
                    glWindow win
                    );
```

Flush the window.

2.6.3 glwLock

```
void glwLock(void);
```

Lock gltools (in multithreading mode);

2.6.4 glwUnlock

```
void glwUnlock(void);
```

Unlock window (in multithreading mode);

2.6.5 glwRedraw(glWindow w,

```
void glwRedraw(glWindow w,
               void (*f)(
                         glWindow w,
                         void * user_data
                         ),
               void * user_data);
```

Redraw window with function f called with `user_data`. Equivalent to glwSetRedrawFunc(w,f); glwProcess(w,info);

2.6.6 glwProcess

```
void      glwProcess(
                     glWindow win,
                     void *user_data
                     );
```

Process event loop. User data are rendered by calling the redraw function set by `glwSetRedrawFunc` (2.7.1). The event loop is exited

- after `glwQuit()` call (user intervention) in event driven mode
- after processing pending events, *one* redraw and `glwQuit()` in application driven mode.

The parameter `info` is used to pass user data to the callback routines.
Thus, gltools can be used for rendering under control of the user code. void glwRunThreaded(glWindow w,void *info);

2.6.7 glwSetControlMode

```
void glwSetControlMode(
                       glWindow w,
                       int mode
                       );
```

Set control mode. Possible values:

```
#define GLW_EVENT_DRIVEN 1                /* default */
#define GLW_APPLICATION_DRIVEN 2
```

2.6.8 glwGrabPPM

```
void  glwGrabPPM(
                 glWindow win,
                 FILE *f
                 );
```

Grab actual window contents and write it into a *ppm* file.

2.6.9 glwGrabPPMAndPrint

```
void  glwGrabPPMAndPrint(
              glWindow win
              );
```

Grab actual window contents to ppm and print as postscript. You can influence the way the picture is printed if you specify the X resource `gltools*printerCommand`. The default is `pnmtops | lpr`.

2.6.10 glwStartMPEGRecording

```
void glwStartMPEGRecording(
              glWindow win,
              char *fileNameStub,
              int skip
              );
```

Start recording of frame data into an MPEG stream using mpeg_encode. Recording is done using `glwGrab` (**??**) which is called within `glwSwapBuffers` (2.6.18). You can influence the video recording if you specify in the X resource `gltools*mpegParameters` the name of an mpeg_encode parameter file. See the corresponding documentation to learn how to set up such a file. The parameter `skip` determines how many frames are skipped between the two dumps.

2.6.11 glwStopMPEGRecording

```
void  glwStopMPEGRecording(
              glWindow win
              );
```

Stop recording of frame data initiated by `glwStartMPEGRecording` (2.6.10).

2.6.12 glwStartEPSRecording

```
void glwStartEPSRecording(
              glWindow win,
              char *fileNameStub,
              int skip
              );
```

Start recording of frame data into an EPS file using the eps dump feature. The parameter `skip` determines how many frames are skipped between two dumps. If the file name ends with ".pdf" and ps2pdf is installed on the search path, pdf is recorded, instead.

2.6.13 glwStopEPSRecording

```
void  glwStopEPSRecording(
              glWindow win
              );
```

Stop recording of frame data initiated by `glwStartEPSRecording` (2.6.12).

2.6.14 glwDump

```
void  glwDump(
              glWindow win,
              void * info,
              char *fileName,
              int w,
              int h
              );
```

Render data into off screen pixmap of size $w \times h$ using the redraw function set by `glwSetRedrawFunc` (2.7.1) and create a *ppm* file. This code is in beta state. To be able to use this option, one has to create all OpenGL transformation data etc. in the redraw function.
If $w < 0$ or $h < 0$, we take the actual size of the window on the screen.

2.6.15 glwShowState

```
void      glwShowState(
                       glWindow win,
                       char *state
                       );
```

The title bar of a gltools window consists of two parts: the title itself and a status area. This function sets the status part.

2.6.16 glwSetTitle

```
void      glwSetTitle(
                      glWindow w,
                      char *title
                      );
```

The title bar of a gltools window consists of two parts: the title itself and a status area. This function sets the title part.

2.6.17 glwQuit

```
void glwQuit(
             glWindow w
             );
```

Stop event processing in `glwProcess` (2.6.6).

2.6.18 glwSwapBuffers

```
void    glwSwapBuffers(
                       glWindow w
                       );
```

Swap buffers in double buffer mode.

2.6.19 glwDebug

```
void glwDebug(void);
```

Toggle debugging output of `glwin` (which goes to stderr).

2.7 Callbacks

2.7.1 glwSetRedrawFunc

```
void glwSetRedrawFunc(
                      glWindow w,
                      void (*f)(
                                glWindow w,
                                void * user_data
                                )
                      );
```

Set user draw function. This function is called when a redraw is needed. Only this function should call OpenGL routines to draw something.

2.7.2 glwSetExposeFunc

```
void glwSetExposeFunc(
                      glWindow w,
                      void (*f)(
                                glWindow w,
                                void *user_data,
                                int width,
                                int height
                                )
                      );
```

Set function to be called after expose event. `width` and `height` contain the new size of the window.

2.7.3 glwSetReshapeFunc

```
void glwSetReshapeFunc(
                       glWindow w,
                       void (*f)(
                                 glWindow w,
                                 void *user_data,
                                 int width,
                                 int height
                                 )
                       );
```

Set function to be called after reshape event. `width` and `height` contain the new size of the window.

2.7.4 glwSetKeyDownFunc

```
void glwSetKeyDownFunc(
                        glWindow w,
                        GLenum (*f)(
                                    glWindow w,
                                    void *user_data,
                                    int key,
                                    GLenum button_shift_mask
                                    )
                        );
```

Set function to be called after key press. The `key` parameter can have the following values:

```
#define GLW_A                   'A'
#define GLW_B                   'B'
#define GLW_C                   'C'
#define GLW_D                   'D'
#define GLW_E                   'E'
#define GLW_F                   'F'
#define GLW_G                   'G'
#define GLW_H                   'H'
#define GLW_I                   'I'
#define GLW_J                   'J'
#define GLW_K                   'K'
#define GLW_L                   'L'
#define GLW_M                   'M'
#define GLW_N                   'N'
#define GLW_O                   'O'
#define GLW_P                   'P'
#define GLW_Q                   'Q'
#define GLW_R                   'R'
#define GLW_S                   'S'
#define GLW_T                   'T'
#define GLW_U                   'U'
#define GLW_V                   'V'
#define GLW_W                   'W'
#define GLW_X                   'X'
#define GLW_Y                   'Y'
#define GLW_Z                   'Z'

#define GLW_a                   'a'
#define GLW_b                   'b'
#define GLW_c                   'c'
#define GLW_d                   'd'
#define GLW_e                   'e'
#define GLW_f                   'f'
#define GLW_g                   'g'
#define GLW_h                   'h'
#define GLW_i                   'i'
```

```
#define GLW_j                   'j'
#define GLW_k                   'k'
#define GLW_l                   'l'
#define GLW_m                   'm'
#define GLW_n                   'n'
#define GLW_o                   'o'
#define GLW_p                   'p'
#define GLW_q                   'q'
#define GLW_r                   'r'
#define GLW_s                   's'
#define GLW_t                   't'
#define GLW_u                   'u'
#define GLW_v                   'v'
#define GLW_w                   'w'
#define GLW_x                   'x'
#define GLW_y                   'y'
#define GLW_z                   'z'

#define GLW_0                   '0'
#define GLW_1                   '1'
#define GLW_2                   '2'
#define GLW_3                   '3'
#define GLW_4                   '4'
#define GLW_5                   '5'
#define GLW_6                   '6'
#define GLW_7                   '7'
#define GLW_8                   '8'
#define GLW_9                   '9'

#define GLW_space               0x020
#define GLW_exclam              0x021
#define GLW_quotedbl            0x022
#define GLW_numbersign          0x023
#define GLW_dollar              0x024
#define GLW_percent             0x025
#define GLW_ampersand           0x026
#define GLW_apostrophe          0x027

#define GLW_quoteright          0x027
#define GLW_parenleft           0x028
#define GLW_parenright          0x029
#define GLW_asterisk            0x02a
#define GLW_plus                0x02b
#define GLW_comma               0x02c
#define GLW_minus               0x02d
#define GLW_period              0x02e
#define GLW_slash               0x02f
#define GLW_colon               0x03a
#define GLW_semicolon           0x03b
#define GLW_less                0x03c
#define GLW_equal               0x03d
```

```
#define GLW_greater             0x03e
#define GLW_question            0x03f
#define GLW_at                  0x040
#define GLW_bracketleft         0x05b
#define GLW_bracketright        0x05d
#define GLW_asciicircum         0x05e
#define GLW_underscore          0x05f
#define GLW_grave               0x060
#define GLW_braceleft           0x07b
#define GLW_bar                 0x07c
#define GLW_braceright          0x07d
#define GLW_asciitilde          0x07e
#define GLW_Return              0x0D
#define GLW_BackSpace           0x08
#define GLW_Escape              0x1B
#define GLW_Left                0xf5
#define GLW_Up                  0xf6
#define GLW_Right               0xf7
#define GLW_Down                0xf8
#define GLW_KP_Enter             0x8D
#define GLW_KP_Home              0x95
#define GLW_KP_Left              0x96
#define GLW_KP_Up                0x97
#define GLW_KP_Right             0x98
#define GLW_KP_Down              0x99
#define GLW_KP_Page_Up           0x9A
#define GLW_KP_Page_Down         0x9B
#define GLW_KP_End               0x9C
#define GLW_KP_Begin             0x9D
#define GLW_KP_Insert            0x9E
#define GLW_KP_Delete            0x9F
#define GLW_KP_Divide            0xAF
#define GLW_KP_Multiply          0xAA
#define GLW_KP_Add               0xAB
#define GLW_KP_Subtract          0xAD
#define GLW_KP_Decimal           0xAE
#define GLW_KP_0                 0xB0
#define GLW_KP_1                 0xB1
#define GLW_KP_2                 0xB2
#define GLW_KP_3                 0xB3
#define GLW_KP_4                 0xB4
#define GLW_KP_5                 0xB5
#define GLW_KP_6                 0xB6
#define GLW_KP_7                 0xB7
#define GLW_KP_8                 0xB8
#define GLW_KP_9                 0xB9
#define GLW_F1                   0xBE
#define GLW_F2                   0xBF
#define GLW_F3                   0xC0
#define GLW_F4                   0xC1
```

```
#define GLW_F5                  0xC2
#define GLW_F6                  0xC3
#define GLW_F7                  0xC4
#define GLW_F8                  0xC5
#define GLW_F9                  0xC6
#define GLW_F10                 0xC7
#define GLW_F11                 0xC8
#define GLW_F12                 0xC9
```

D instead of 5

```
#define GLW_Page_Up             0xD5
#define GLW_Page_Down           0xD6
#define GLW_Home                0xD0
#define GLW_End                 0xD7
```

E instead of 6

```
#define GLW_Insert              0xE3
#define GLW_Delete              0xFF
#define GLW_Print               0x61
#define GLW_Pause               0x13
#define GLW_Scroll_Lock         0x14
#define GLW_Tab                 '\t'
```

`shift_mask` can have the following values:

```
#define GLW_SHIFT               8
#define GLW_CONTROL             16
#define GLW_MOD1                32
#define GLW_LOCK                64
```

2.7.5 glwSetMouseDownFunc

```
void glwSetMouseDownFunc(
                        glWindow w,
                        GLenum (*f)(
                                    glWindow w,
                                    void *user_data,
                                    int pos_x,
                                    int pos_y,
                                    GLenum button_shift_mask
                                    )
                        );
```

Set function to be called after pressing a mouse button. `pos_x` and `pos_y` contain the current mouse position. `button_shift_mask` can have the following parameters:

```
#define GLW_LEFTBUTTON          1
#define GLW_RIGHTBUTTON         2
#define GLW_MIDDLEBUTTON        4
```

3 glrnd - Rendering Volume Management

Revision : 2.40
Date : 2003/01/2411 : 14 : 57
Author: Jürgen Fuhrmann, Hartmut Langmach

glrnd provides a framework for rendering data in a given rectangular rendering volume and manages all transformations, intersection planes and light sources, so that actual rendering modules do not have to care about this stuff. It is controlled by the keyboard and the mouse.

3.0.1 Keyboard User Interface

The main key to know when working with *glrnd* is the *state control* key. It allows to toggle between application-controlled and user-controlled mode. If the window is in user-controlled state, by pressing the state control key together with the shift key, you give control back to the application only until the next invocation of `glRender()`.
The following table may be incomplete. You can get the actual keyboard layout by pressing the help-Key.

glrnd key table, $Revision: 2.88 $ $Date: 2003/02/20 16:50:45 $
Backspace: Enter user control mode
tab: toggle state change mode
Return: Quit user control mode
SPACE: Mode control
+: Increase mouse sensitivity.
,: decrease control parameter.
-: Decrease mouse sensitivity.
.: increase control parameter.
<: Zoom out.
>: Zoom in.
?: This help.
B: Toggle background color (black/white).
d: Dump actual picture to ppm file (look for *-*.ppm)
F: Toggle rendering volume frame (bounding box) drawing.
I: Change number of isolines.
.: increase control parameter by a factor.
O: Toggle Ortho
P: Dump POV mesh
R: Reset to internal default.
S: Save actual state (look for .*-rndstate)
V: Start/Stop video recording
a: Switch to GUI
c: Toggle remembered lists
g: Toggle Gouraud/flat shading.
h: This help.
i: Toggle isoline mode.
,: decrease control parameter by a factor.
l: Toggle level surface mode.
m: Toggle model display when moving.
p: Dump actual picture to eps file (look for *-*.eps)
q: Mode control (Quit)
r: Restore last saved state.
v: Toggle vscale for plane sections
w: toggle wireframe mode
x: Show x orthogonal plane section.
y: Show y orthogonal plane section.
z: Show z orthogonal plane section.
prev: toggle state change mode
next: toggle state change mode
left:move left
up:move up
right:move right
down:move down
Backspace: Enter user control mode

3.0.2 Mouse interface

All actions can be performed with the left mouse button pressed down. Which action is performed depends on the state change mode. One can cycle through the state change mode using the next/prev keyboard keys. Wich state change mode is active, is shown in the window title.
Selected state change modes are bound to other mouse buttons and the combination of the shift key and a mouse button. Again, the information which mode is active when pressing a button is given in the window title.

3.0.3 Graphical User Interface

When the MOTIF option is active during installation, a graphical user interface can be used with glrnd. See the documentation of *glgui*.

3.1 Imported packages

```
#ifndef GLRND_H
#define GLRND_H
#include "glwin.h"

#define LuaBind
```

LuaBind typedef struct GLRenderer; LuaBindGC(GLRenderer,glrDestroy); LuaBindLua(glrnd.lua);

3.2 Data Types

3.2.1 glRenderer

```
typedef struct glRendererStruct *glRenderer, GLRenderer, glRendererData;
```

Hidden Data type which contains rendering data.

3.2.2 glrApplicationOption

```
#define GLR_MAX_APPLICATION_OPTIONS 10
typedef struct
{
  char key[32];
  int val;
} glrApplicationOption;
```

3.2.3 glRendererState

```
#define GLR_MAX_OBJECTS 9

#define GLR_DIR_X 0
#define GLR_DIR_Y 1
#define GLR_DIR_Z 2
```

```
typedef struct glRendererStateStruct
{
  double rotx;  /* rotation around x axis */
  double roty;  /* rotation around y axis */
  double rotz;  /* rotation around z axis */
  double tranx; /* translation in x direction */
  double trany; /* translation in y direction */
  double tranz; /* translation in z direction */
  double vscale; /* value scale [0.0-1.0] */
  double zoom;   /* zoom factor */
  double ctrl_prm;    /* control parameter for color scale */
  double ctrl_fac;    /* control parameter for color scale */
  int wireframe; /* show data as wire frame */
  int show_frame; /* show frame */
  int move_model;     /*switch on rendering when moving */
  double asp;
  int gouraud_shading; /* use gouraud shading */
  double sensitivity;  /* mouse sensitvity */
  int bg_black;        /* background color black */
  double ltx;          /* light position */
  double lty;          /* light position */
  double ltz;          /* light position */
  double plane_d[3];   /* depending on direction, between *min and *max */
  double scale[3];     /* scale factors for rendering volumes*/
  int plane_dir;       /* plane direction (x orth/y orth z orth)*/
  int level_surf;      /* show isolevel surfaces */
  double level;        /* isolevel (between fmin and fmax)*/
  int ortho;           /* orthogonal projection */
  int what_done;             /* what has been changed last */
  double isoline_distance; /* distance between isolines - obsolete*/
  int isoline_mode;         /* show isolines */
  int show_object[GLR_MAX_OBJECTS+1];        /* show remembered object list */

  double min[3],max[3];          /* renderer volume; read only! */

  double fmin,fmax;         /* min,max of current function */

  int transparency;
  int spacedim;
  int show_info;

  glrApplicationOption options[GLR_MAX_APPLICATION_OPTIONS];
  int noptions;
}  *glRendererState;
```

This public structure is desinged to hold all state data which are necessary for the interaction with the renderer.

```
#define GLR_ROTATE    1
#define GLR_TRANSLATE 2
#define GLR_LIGHT     3
```

```
#define GLR_ISOLEVEL  5
#define GLR_PLANE     6
#define GLR_INPLANE   7
#define GLR_PLANE_ASPECT    8
#define GLR_INPLANE_ASPECT    9
#define GLR_N_DO         10
```

this are the possible values for what_done

3.3 Constructors and Destructors

3.3.1 glrCreate

```
LuaBind GLRenderer* glrCreate(
LuaBind                     char *title,
LuaBind                     int xpos,
LuaBind                     int ypos,
LuaBind                     int width,
LuaBind                     int height
LuaBind                     );
```

Create renderer. It also calls `glwin` to create a window with corresponding data.

3.3.2 glrDestroy

```
LuaBind void glrDestroy(
LuaBind                glRenderer rnd
LuaBind                );
```

Destroy renderer (and corresponding window).

3.4 Setting/Getting Data

3.4.1 glrSetTitle

```
LuaBind void glrSetTitle(
LuaBind                 GLRenderer *rnd,
LuaBind                 char *title
LuaBind                 );
```

Set title.

3.4.2 glrReset

```
void glrReset(
              glRenderer rnd
              );
```

Reset to default all rotations etc.

3.4.3 glrSetVolume

```
LuaBind void glrSetVolume(
LuaBind           GLRenderer *rnd,
LuaBind                  double xmin,
LuaBind                  double xmax,
LuaBind                  double ymin,
LuaBind                  double ymax,
LuaBind                  double zmin,
LuaBind                  double zmax
LuaBind                  );
```

Define rendering volume. Everything drawn by the user function has to be placed within this volume to be visible.

3.4.4 glrSetUserInfo

```
void glrSetUserInfo(
                    glRenderer rnd,
                    char *user_info, /* format string */
                    ...              /* data according to user_info */
                    );
```

Set user information to be printed in the cooresponding field of the renderer.

3.4.5 void glrGetPoint

```
void glrGetPoint(
                glRenderer rnd,
                double *x, double *y, double *z
                );
```

get intersection point of main planes

3.4.6 void glrGetPlane

```
void glrGetPlane(
                glRenderer rnd,
                double *a,
                double *b,
                double *c,
                double *d
                );
```

3.4.7 void glrSetPlane

```
void glrSetPlane(
                glRenderer rnd,
                double a,
                double b,
                double c,
                double d
                );
```

Get/set data of intersection plane for 3D plane sections: the plane is defined by the equation

$$ax + by + cz + d = 0$$

in the three-dimensional space.

3.4.8 glrXSetPlane

```
void glrXSetPlane(
                  glRenderer rnd,
                  int dir,
                  double val
                  );
```

Alternative way to set intersection plane data: `dir` denotes the direction (0 = x, 1=y, 2=z) the plane should be orthogonal to, and val denotes the distanze from zero.

3.4.9 glrSetAxisName

```
void glrSetAxisName(glRenderer rnd,
                    char dir, char *name
                    );
char * glrGetAxisName(glRenderer rnd,
                      char dir
                      );
```

Set title.

3.4.10 glrSetAxisTics

```
void glrSetAxisTics(glRenderer rnd, char dir, int ntics, double *tics);
```

Set axis tics;

3.4.11 glrSetAxisPos

```
void glrSetAxisPos(glRenderer rnd, char dir, char *pos);
```

Set axis position

3.4.12 glrArrowList

```
int glrArrowList(glRenderer rnd);
```

3.5 Callbacks and Event Processing

3.5.1 glrDrawCallback

```
typedef void (*glrDrawCallback)(
                                glRenderer rnd,
                                void *data
                                );
```

Callback function for drawing data.

3.5.2 glRender

```
void glRender(
              glRenderer rnd,
              glrDrawCallback f,
              void *data
              );

void glRenderWithGUI(glRenderer rnd,
                     glrDrawCallback f,
                     void *info);
```

Render data with given callback function.

3.5.3 glrSetSecondaryCallback

```
void glrSetSecondaryCallback(glRenderer rnd,glrDrawCallback scb);
```

The user can call a secondary callback which e.g. calls a slave renderer.

3.5.4 glrSetInfoCallback

```
void glrSetInfoCallback(glRenderer rnd,glrDrawCallback info);
```

The user can draw information into the upper and right info areas.

3.5.5 glrDefaultInfoCallback

```
void glrDefaultInfoCallback(glRenderer rnd,void * thrash);
```

This is the default info callback, a user info callback can call this.

3.5.6 glrSetDataValid

```
void glrSetDataValid(
                     glRenderer rnd,
                     int valid
                     );
```

Tell renderer that all display lists it had compiled are still valid (for dumping and interaction with the GUI).

3.5.7 glrShowModel

```
void glrShowModel(glRenderer rnd,int ishow);
```

Tell renderer that it should not draw the model. This function can be used by the gui code. On slow displays this may make sense.

3.6 State file management

3.6.1 glrSaveState

```
void  glrSaveState(glRenderer rnd, char *filename);
```

Save actual transformation state using a state file. A default filename is generated when the second parameter is zero.

3.6.2 glrRestoreState

```
void  glrRestoreState(glRenderer rnd,char *filename);
```

Restore actual transformation state using a state file. A default filename is generated when the second parameter is zero.

3.6.3 glrSetStateFileNameStub

```
void glrSetStateFileNameStub(glRenderer rnd, char *name);
```

3.7 Key actions

3.7.1 glrKeyAction

```
typedef int (*glrKeyAction)(
                            glRenderer rnd,
                            int mask
                            );
```

3.7.2 glrRegisterKeyAction

```
void glrRegisterKeyAction(
                          glRenderer rnd,
                          int key,
                          glrKeyAction action,
                          char *help
                          );
```

Register key action callback routine

3.7.3 glrDumpHelpFile

```
void glrDumpHelpFile(glRenderer rnd);
```

Create help info for keys in LaTeX format

3.8 Frame dump

3.8.1 glrSetDumpFileNameStub

```
void glrSetDumpFileNameStub(
                            glRenderer rnd,
                            char *name
                            );
```

Set file name stub for data dump. Subsequent dumps get the corresponding number in the file name. The default name stub is derived from the title.

3.8.2 glrSetDumpPixmapSize

```
void glrSetDumpPixmapSize(
                          glRenderer rnd,
                          int w,
                          int h
                          );
```

Set size of dump pixmap.

3.8.3 glrDumpNext

```
void glrDumpNext(
                 glRenderer rnd
                 );
```

Dump data when invoking `glRender` next time.

```
void glrLock(glRenderer rnd);
void glrUnlock(glRenderer rnd);
```

3.9 Graphical User Interface

3.9.1 glrGUI

```
typedef void (*glrGUI)(glRenderer rnd);
```

3.9.2 glrSetGUI

```
void glrSetGUI(glrGUI gui);
```

Bring up a graphical user interface. It should interact with the renderer via the state structure `glRendererState` (3.2.3) and the `glrSetDataValid` (3.5.6) call.

3.9.3 glrGetRendererState

```
glRendererState  glrGetRendererState(glRenderer rnd);
```

3.9.4 glrGetWindow

```
glWindow glrGetWindow(glRenderer rnd);
```

3.9.5 glrGetInfo

```
void *glrGetInfo(glRenderer rnd);
```

3.10 Application options

Application options could be set using a keyboard callback. Alternatively, a pulldown menu in the GUI is defined which contains all the application options. They should be used to interactively influence user data which are application specific.

3.10.1 glrSetApplicationOption

```
void glrSetApplicationOption(glRenderer rnd,char *key, int val);
```

3.10.2 glrGetApplicationOption

```
int glrGetApplicationOption(glRenderer rnd,char *key);
```

3.11 Obsolete functions

These functions are considered to be obsolete. They are still maintained for backward compatibility.

```
void glrSetFlatshading(glRenderer rnd, int flat);
void glrGetFlatshading(glRenderer rnd,int *flat);

void   glrGetVScale(glRenderer rnd,double* vscale);
void   glrSetVScale(glRenderer rnd, double vscale);

void glrGetWireframe(glRenderer rnd, int *wireframe);

void glrGetLevelSurface(glRenderer rnd, int* mode);
void glrSetLevelSurface(glRenderer rnd,int mode);

void glrGetLevel(glRenderer rnd, double* lev);
void glrSetLevel(glRenderer rnd, double lev);

void glrGetIsolineMode(glRenderer rnd, int* mode);
void glrSetIsolineMode(glRenderer rnd,int mode);

FILE *glrGetPOV(glRenderer rnd);

void glrMoveWireframe(glRenderer rnd);
void glrMoveFrame(glRenderer rnd);
void glrMoveModel(glRenderer rnd);

void glrGetDialog(glRenderer rnd, int* dialog);
LuaBind void glrSetDialog(GLRenderer*,int dialog);

typedef void (*glrDrawCallback2)(
                                 glRenderer rnd,
                                 void *data1,
                                 void *data2
                                 );

void glRender2(glRenderer rnd, glrDrawCallback2 f,
               void *data1,void *data2);


int  glrLoadFont(glRenderer rnd, int font_number, char *fontName);
void glrSelectFont(glRenderer rnd, int font_number);
void glrSetFontSize(glRenderer rnd, double font_size);
void glrPrint(glRenderer rnd, char *text);
void glrPrintf(glRenderer rnd, char *format, ...);

#endif
```

4 glmesh - Function Drawing on Simplex Meshes

Author:Jürgen Fuhrmann
Revision : 2.19
Date : 2000/10/1816 : 43 : 32

glmesh contains rendering routines for triangular and tetrahedreal meshes based on callback functions invoking loops over simplices of a mesh in a data structure given by the user. No extra data structure has to be generated. Sure, this causes performance drawbacks, but instead the user gains great flexibility in using this interface on his/her data strucutures.
glmesh manages plane sections and level sets for 3D tetrahedral meshes.
The basic principle is a double callback mechanism - a loop callback function gets user data and a simplex callback function as parameter which has to be fed with number, node numbers and coordinates of a simplex.

4.1 Imported packages

```
#ifndef GLMESH_H
#define GLMESH_H

#include "glrnd.h"
```

4.2 Data Types

4.2.1 glMeshStruct

```
typedef struct glMeshStruct *glMesh;
```

This structure contains all necessary mesh data and is hidden from the user.
subsection:Constructors and Destructors

4.2.2 glmSimplexCallback

```
typedef void (*glmSimplexCallback)
     (
      glMesh m,
      int  number_of_this_simplex,
      int material_of_this_simplex,
      double *function_defined_on_this_this_simplex,
      int *index_in_funtion_on_this_simplex,
      double **coordinates_of_the_nodes
      );
```

4.2.3 glmLoopCallback

```
typedef void (*glmLoopCallback)
     (
      glMesh m,
      void *user_data,
      glmSimplexCallback call_this_on_every_simplex
      );
```

4.2.4 glmCreate

```
glMesh glmCreate(
                int number_of_mesh_nodes,
                int number_of_mesh_simplices,
                int space_dimension,
                void *user_data,
                glmLoopCallback loop_over_all_simplices
                );
```

Generate an instance of `glMesh`. This is cheap.

4.2.5 glmDestroy

```
void glmDestroy(
               glMesh m
               );
```

Destroy an instance of `glMesh`.

4.3 Setting Data

4.3.1 glmSetFunction

```
void glmSetFunction(
                    glMesh m,
                    double *f,
                    double min,
                    double max
                    );
```

Set piecewise linear function to plot, its minimum and its maximum. If `min` > `max`, they are automatically calculated.

```
double *glmGetNodeFunc(glMesh m);
```

```
void *glmGetUserData(glMesh m);
```

4.3.2 glmSetFunction

```
void glmSetCellFlux(glMesh m, double *values, double min, double max);
```

Set piecewise constant flux to plot

4.3.3 glmSetVoffset

```
void glmSetVoffset(
                   glMesh m,
                   int voffset
                   );
```

Set vector offset (0 or 1)

4.3.4 glmColorCallback

```
typedef float* (*glmColorCallback)
     (
      glMesh m,
      double fval,
      float *rgb
      );
```

4.3.5 glmSetColorCallback

```
void glmSetColorCallback(
                         glMesh m,
                         glmColorCallback col
                         );
```

Set color calculation function. `glmRBColor` is the default value.

4.3.6 glmMaterialColorCallback

```
typedef float* (*glmMaterialColorCallback)
     (
      glMesh m,
      int imat,
      float *rgb
      );
```

4.3.7 glmSetMaterialColorCallback

```
void glmSetMaterialColorCallback(
                         glMesh m,
                         glmMaterialColorCallback col
                         );
```

Set material color calculation function. `glmDefaultMaterialColor` is the default value.

4.3.8 glmDrawInfo

```
void glmDrawInfo(glRenderer rnd, glMesh m);
```

Info call back for mesh data.

4.4 Invocation

4.4.1 glmDraw

```
void glmDraw(
             glRenderer rnd,
             glMesh m
             );
```

The glmesh draw routines are invoked using `glmDraw` as a callback for `glRender`.

```
#endif
```

5 gleps - Encapsulated Postscript Dump

Revision : 2.8
Date : 2002/09/2408 : 55 : 44
Author: Jürgen Fuhrmann

This module provides the possibility to dump rendered graphics into vector postscript files using the feedback buffer rendering mechanism of OpenGL. It is in a beta state and possibly will remain there because it is not that easy to map correctly all the OpenGL functionality to Postscript. It should reasonably well render graphics which remains in the limits of the OpenGL features used by the other gltools parts.
This code would not exist if Mark Kilgard wouldn't have placed his rendereps sample code onto the net, and if there would not exist Frederic Delhoume's free gouraudtriangle postscript code. From Mark's code, handling of lines and polygons has been taken. String handling is new.
I tried to keep the generated postscript files as human readable as possible. Especially, there is a "tuning section" in the prolog where the user can change fonts, font sizes etc.
The main problems with the output are caused by the fact that OpenGL does not know alignment of bitmaps (bitmaps are used by glxUseXFont), and that there seems to be no simple possibility to align strings in OpenGL output. So only bottom left alignment of strings is possible, this is not quite beautiful. This problem also leads to the fact that Bounding box calculation in presence of strings may be incorrect.
Until now, I only see the possibility to tune the eps files by hand. In the ps header there are defined some alingned show commands which can be used to improve label placement in the ps file. Any idea to cope with this situation is appreciated.

5.0.1 glepsDumpUnSorted

```
void glepsDumpUnSorted(glWindow w, FILE *file,int crop);
```

Create vector postscript dump using the feedback buffer mechanism, but without hidden surface removal. The crop flag is used to decide whether to crop ps output to the actually drawn area or not.

5.0.2 glepsDumpSorted

```
void glepsDumpSorted(glWindow w, FILE *file, int crop);
```

Create vector postscript dump using the feedback buffer mechanism, with hidden surface removal based on sorting the feedback buffer before the dump. The crop flag is used to decide whether to crop ps output to the actually drawn area or not.

5.0.3 glepsSetOutputFormat

```
void glepsSetOutputFormat(char * coord_fmt, char *color_fmt);
```

Set the output format for floating point numbers (different for coordinates and colors) in the postscript file. (The more accurate the format, the longer the file...). There have to be a trailing spaces both formats.

```
void glepsEmitHeader(glWindow w, FILE *file);
void glepsEmitTrailer(glWindow w, FILE *file, int npages);
void glepsGrabEPSFrame(glWindow w, FILE *file, int npage);
```