An introduction to GLUT

by **Elie De Brauwer**

Goal of this file

This file offers a brief overview of what GLUT is and does and how you can use GLUT to create and environment that is usefull for the output of an OpenGL application and how to build these applications. This also contains a list of the most common GLUT functions and procedures, the document is ended with two examples including screenshots.
This file is targeted towards Linux users programming C++.

History of this file

- 26 june 2003: initial release
- 20 july 2003: updates, added new sample program illustrating the use of timers
- 11 august 2003: this file was merged with the **mindfuck** project

What is GLUT

Introduction

GLUT is the OpenGL Utility Toolkit. GLUT is used in the development of OpenGL applications. OpenGL is a graphics library which means that it can do all sorts of things with geometric forms, lightings, transformations etc. But OpenGL was designed to be portable so it can be used within various environments / operating systems (Linux, Unix, Solaris, Windows, Amiga,...) without changing the OpenGL code.
Within the OpenGL programming Guide (ISBN 0-201-60458-2, also known as " the red book ") one says the following about this subject:*The OpenGL Utility Toolkit (GLUT) is a window-system-independent toolkit written by Mark Kilgard, to hide the complexities of differing window system APIs.*So basically GLUT provides you with a utility that creates a window for you, in that window you can show the output of your OpenGL program. But glut is replacable, you can use the GtkGLArea or **GtkGLExt** for GTK, you can also use **OpenGL within Perl** without explicitly using GLUT.

Summary

Now, let's summarize, OpenGL is a standard, available on many platforms, it does everything that is related to graphics, it does not handle interaction with the window system. So a seperate library is needed, there are some alternatives but at this point I am using GLUT because the title of this document says so ;-).

Using GLUT

Installation

On a default **Debian** machine GLUT is provided by the libglut3-dev package, the OpenGL files are provided by the xlibmesa-gl-dev package.

```
helios@Kafka:~$ dpkg-query -S /usr/include/GL/glut.h
libglut3-dev: /usr/include/GL/glut.h
helios@Kafka:~$ dpkg-query -S /usr/include/GL/gl.h
```

```
xlibmesa-gl-dev: /usr/include/GL/gl.h
```

I presume you have the knowledge how to install these on your system or that you have these installed already.

Header files

There is a little extra advantage when using GLUT, if you were developing an OpenGL application you should include the header files gl.h and glu.h (glu is the OpenGL Utility Library) but when you are including glut.h, glut.h guarantees the inclusion of gl.h and glu.h. So when we start and OpenGL application we start by writing:

```
#include <GL/glut.h>
```

Compilation

The first easy programs using only some OpenGL commands and some glut function calls you should be able to compile using one of the following (normally the first one should work):

```
helios@Kafka:~$ g++ -Wall -lglut -O test.cpp
helios@Kafka:~$ g++ -Wall -lGL -lGLU -lglut -O test.cpp
```

Here -l is a linker option stating to include the library followed the -l, -Wall is a way of saying the compiler that it should output all the warnings and -O says the compiler to use some optimisation techniques. If you want more information about these commands I'd suggest you read the **g++/gcc** manpages.

Programming GLUT

Introduction

So this is what you have been waiting for, some action. Now I will cover the following subjects:

- What you should know about windows and GLUT
- What you should know about events and GLUT
- What you should know about creating 3D objects in GLUT
- What you should know about menus in GLUT
- What you should know about about actually starting the program and displaying the window

After you've been thru this you should have enough knowledge to use GLUT to create a window that is capable of showing your OpenGL visualisation. If you are still in the need for more information I would advise to visit the**glut 3** specification page on **OpenGL.org**, which contains information similar to the manpages.
One last note, there is a very easy trick to recognise glut functioncalls, all glut functions start with glutName where Name is the name of the function, all OpenGL functions start with gl and so on.

What you should know about windows and GLUT

*void glutInit(int *argc, char **argv);*
: glutInit() is the first function that should be called, it parses the parameters that are window system specific for example the -display parameter that is used to let X11 begin exported to other screens. Be sure to call this with &argc as first parameter, because the function wants a pointer not the plain argc. So you would call*glutInit(&argc,argv);*

glutInitDisplayMode(unsigned int mode);
: glutInitDisplayMode() sets the initial display mode, you should pass an OR'ed chain of modes, there are various options like GLUT_RGBA, GLUT_RGB, GLUT_INDEX, GLUT_SINGLE, GLUT_DOUBLE, GLUT_ACCUM, GLUT_ALPHA, GLUT_DEPTH, GLUT_STENCIL, GLUT_MULTISAMPLE,

GLUT_STEREO, GLUT_LUMINANCE. For more detailed information about these please consult the glutInitDIsplayMode manpage. I will just mention that GLUT_RGBA says you want to use the RGBA color model (default), GLUT_DOUBLE says you want to use double buffering, this means that you create 2 virtual screens one that is visible and one in the background, on this in the background changes are being made and you swap the buffers using the *glutSwapBuffers()* function. GLUT_SINGLE uses single buffering, which is not reallyadvised for an animation but this is default if you don't specify something else.
If you use GLUT_INDEX (color index mode) you should manually take care of the color map using the glutSetColor() function, refer to the manpage for more information.

glutInitWindowSize(int width, int height);
glutInitWindowSize() sets the initial size of the window.

glutInitWindowPosition(int x, int y);
glutInitWindowPosition() sets the initial position of the window but this is easily overwritten by your window manager.

void glutFullScreen(void);
glutFullScreen() requests that the window is made fullscreen

*int glutCreateWindow(char *name)*
glutCreateWindow() creates the window for you, the string you pass will be the title of the window, the window is only displayed when glutMainLoop() is entered (see *What you should know about about actually starting the program and displaying the window* for mor information). This returns an integer, this integer is the window identifier, some functions need this indentifier to know which window you mean, an example is the following: *glutDestroyWindow*

void glutDestroyWindow(int win)
glutDestroyWindow destroys a window, the win parameter is the window identifier mentioned above. The program isn't killed when you destroy the window, only the window is destroyed. (See example below).

What you should know about events and GLUT

In order to understand the following I will first explain what a callback is. When a certain event occurs you want a program to do something. So in case of event E call, method/function/procedure M. A callback just occured. Within this part you define which function has to be called when an event happens. (An event is something that happens: you move the mouse, press a key, resize the window,... mostly generated by the user). Now I will define the most commonly used GLUT callbacks.

*void glutDisplayFunc(void (*func)(void));*
This specifies the function that needs to be called when the window is redrawn. This function is called when the window if shown for the first time, when it is popped up, when the contents was damaged (by moving another window over it). It is also possible to force a redraw see glutPostRedisplay() below. An example might look like:

```
void doMagic(void){
   // put everything on the screen
}
```

To make doMagic() you displayfunction you would put the following line in you sourcecode:

```
glutDisplayFunc(doMagic);
```

void glutPostRedisplay(void);
When you put glutPostRedisplay(); somewhere in your code this means that the function defined using glutDisplayFunc() will be called at the next opportunity. It marks the current window as needing a redraw.

*void glutReshapeFunc(void (*func)(int width, int height));*
The callback defined using glutReshapeFunc is called whenever the window is being moved and/or resized. The arguments are the new width and the new height of the window.

*void glutTimerFunc(unsigned int msecs,void (*func)(int value), value);*
Defines a callback that is called periodically, you can register multiple callbacks but you can't unregister them, you should ignore these callbacks by setting the value so the method would ignore it. msecs is the number of milliseconds in which the callback will be trigger at least.

*void glutKeyboardFunc(void (*func)(unsigned char key,int x, int y));*
A callback for keyboard events, key is the ASCII value of the key pressed and x and y are the coordinates of the mouse at the moment of the keypress.

*void glutMouseFunc(void (*func)(int button, int state, int x, int y));*
A callback triggered on mouseclicks. Button can be GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON or GLUT_RIGHT_BUTTON. State defines what you actually did, did you press the button or release

the buttong, these are defined by GLUT_UP or GLUT_DOWN. And x and y are the coordinates of the mouse.

*void glutMotionFunc(void (*func)(int x, int y));*

Callback triggered when the mouse is moved while one or more buttons are pressed. x and y are the coordinates.

*void glutIdleFunc(void (*func)(void));*

As we will see in " What you should know about actually starting the programing ...." the program is started when you enter the main loop which waits for events, this can be compared to something like

```
while(true){
   // let the world move
}
```

But not everytime there occurs an event that needs to be handled that where the idling comes in. This is when the glutIdleFunc() comes in. This function is usefull when some continuous animation is needed.

Important note: You can undo most callbacks by passing NULL as the function name.

What you should know about creating 3D objects in GLUT

Another advantage of glut is that glut has some pre defined primitives that you can put on the screen in an easy way. I will list the primitives below, most parameters speak for themselves. Each primitive comes in a solid or in a wireframe taste.

- void glutSolidSphere(GLdouble radius, GLint slices, GLint stacks);
- void glutWireSphere(GLdouble radius, GLint slices, GLint stacks);

Renders a sphere centered around the origin, slices are the number of subdivision around the Z axis, stacks the number of subdivisions along the Z axis. Note that the double and int parameters are part of OpenGL (see the gl prefix) and not part of glut.

- void glutSolidCube(GLdouble size);
- void glutWireCube(GLdouble size);

Size is the length of the sides.

- void glutSolidCone(GLdouble base, GLdouble height, GLint slices, GLint stacks);
- void glutWireCone(GLdouble base, GLdouble height, GLint slices, GLint stacks);

base is the radius of the base.

- void glutSolidTorus(GLdouble innerRadius, GLdouble outerRadius, GLint nsides, GLint rings);
- void glutWireTorus(GLdouble innerRadius, GLdouble outerRadius, GLint nsides, GLint rings);

- void glutSolidDodecahedron(void);
- void glutWireDodecahedron(void);

Renders a solid or wireframe 12-sided regular solid. Centered around the origin and with a radius equal to the square root of 3.

- void glutSolidTetrahedron(void);
- void glutWiredTetrahedron(void);

Renders a solid or wireframe 4-sided regular solid. Centered around the origin and with a radius equal to the square root of 3.

- void glutSolidIcosahedron(void);

- void glutWireIcosahedron(void);

Renders a solid or wireframe 20-sided regular solid. Centered around the origin and with a radius equal to 1.0.

- void glutSolidOctahedron(void);
- void glutWireOctahedron(void);

Renders a solid or wireframe 8-sided regular solid. Centered around the origin and with a radius equal to 1.0.

- void glutSolidTeapot(GLdouble size);
- void glutWireTeapot(GLdouble size);

And last but not least, this renders a teapot. If you want to know more about the Utah teapot, try **Google**.

What you should know about menus in GLUT

This sections covers the function calls needed to display popup menus. For a more practical example how to use these menus I suggest you look down at the end of this document to the YATWM example.

*int glutCreateMenu(void (*func)(int value));*
: This function defines the callback that has to be called when when a menu item was selected. This callback function has one parameter, the value. (The programmer has to define a value for each menu item he defines see blutAddMenuEntry() for more information). This function returns and int, this is the menu identifier. This identifier is needed when you would want to attach this menu as a submenu.

void glutSetMenu(int menu);
: You can set the current menu by calling glutSetMenu and passing the menu identifier as parameter, using *int glutGetMenu(void);*you can query which menu is the current menu.

*void glutAddMenuEntry(char *name, int value);*
: This adds an entry to the menu with the label defined by name and the second parameter is the value that will be passed to the callback function. The menu is being added to the current menu. Each menu entry that is being added is added at the bottom of the current menu.

*void glutAddSubMenu(char *name, int menu);*
: This adds the menu identified by the menu identifier as a submenu with a given name to the current menu. The program won't work if it contains an infinite loop of menus.

void glutAttachMenu(int button);
: This attaches the current menu to a certain (mouse) event, you can let a menu listen to a specified mouse button, button can be one of the following: GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, and GLUT_RIGHT_BUTTON. You can also detach the menu using *void glutDetachMenu(int button);*.

What you should know about about actually starting the program and displaying the window

You can define all the callbacks you want, define the most beautiful scene filled with eyecandy but as long as you haven't entered the main loop nothing will happen, you won't even see the window. So basically you define the window, register the callbacks and enter the main loop by calling *void glutMainLoop(void);*. Now the window will appear, the idle function will be called and events will be registered.

Putting it all together: YAT

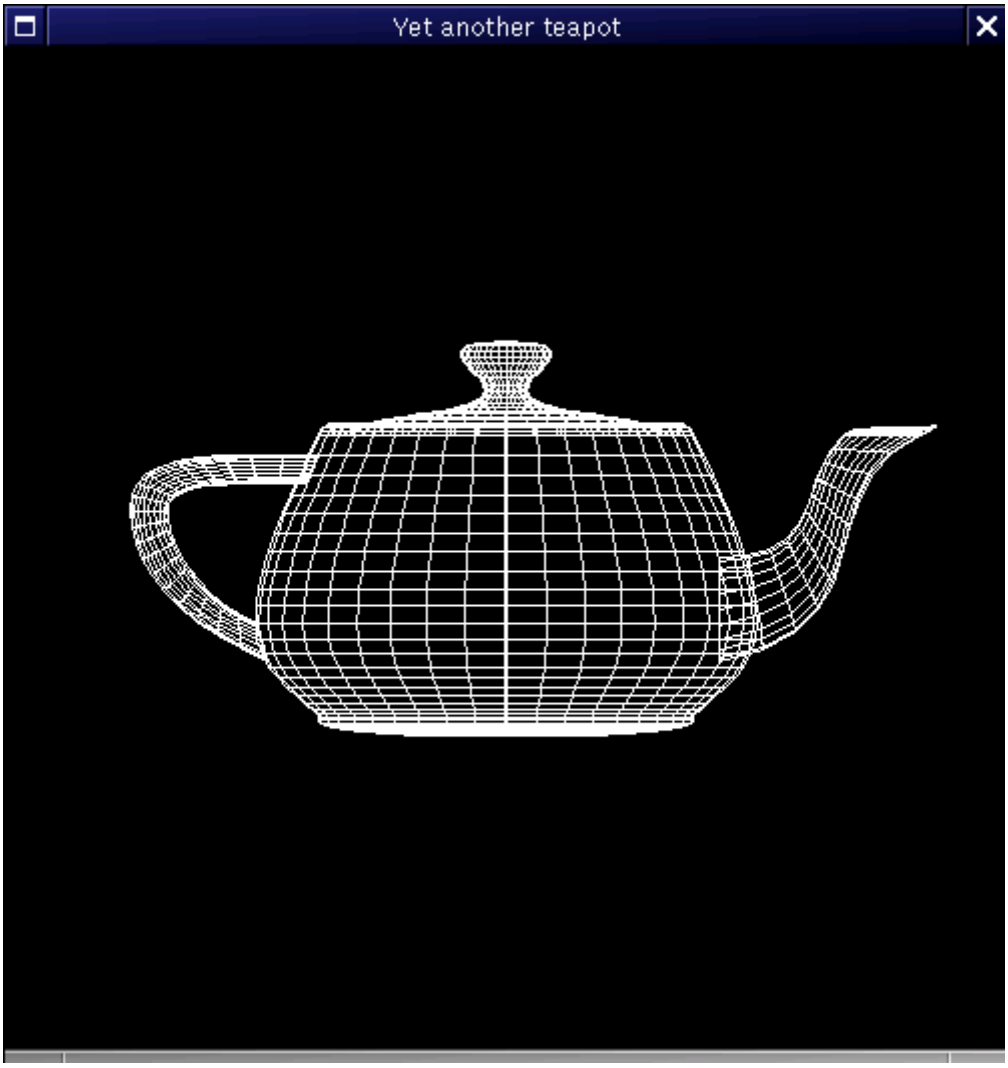
The source code below, opens a window, clears it with a black color and draws a teapot, it listens to keyboard presses and writes out the key that was pressed and the coordinates. The program terminates when the user presses "q". YAT is Yet Another Teapot. You can find some screenshots below showing some of the other primitives.

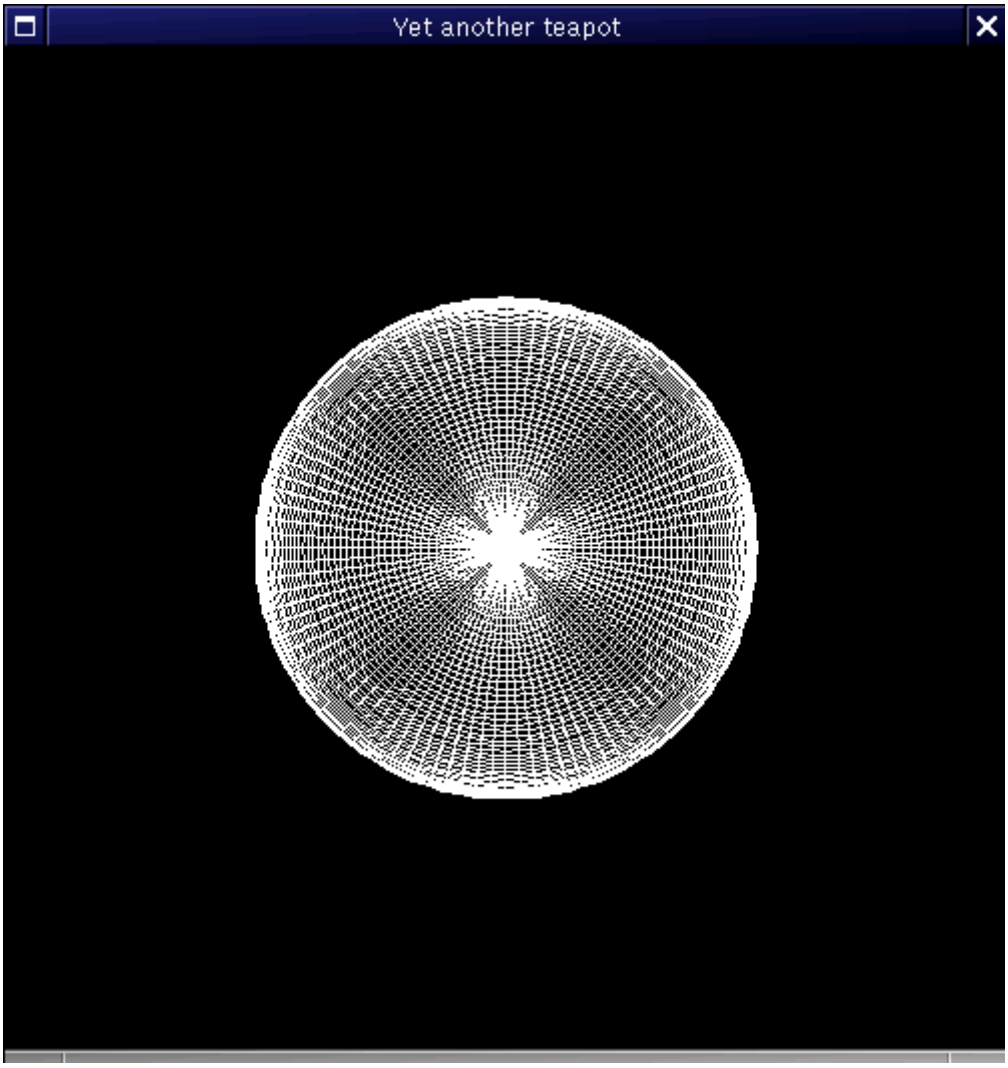
This program only uses 2 OpenGL functions, the first one is glClearColor() that sets the Clearcolor (background) to black and glClear() that actually clears the window. In this example no idlefunction is defined and I use single buffering (no double buffering) since no animation is needed.

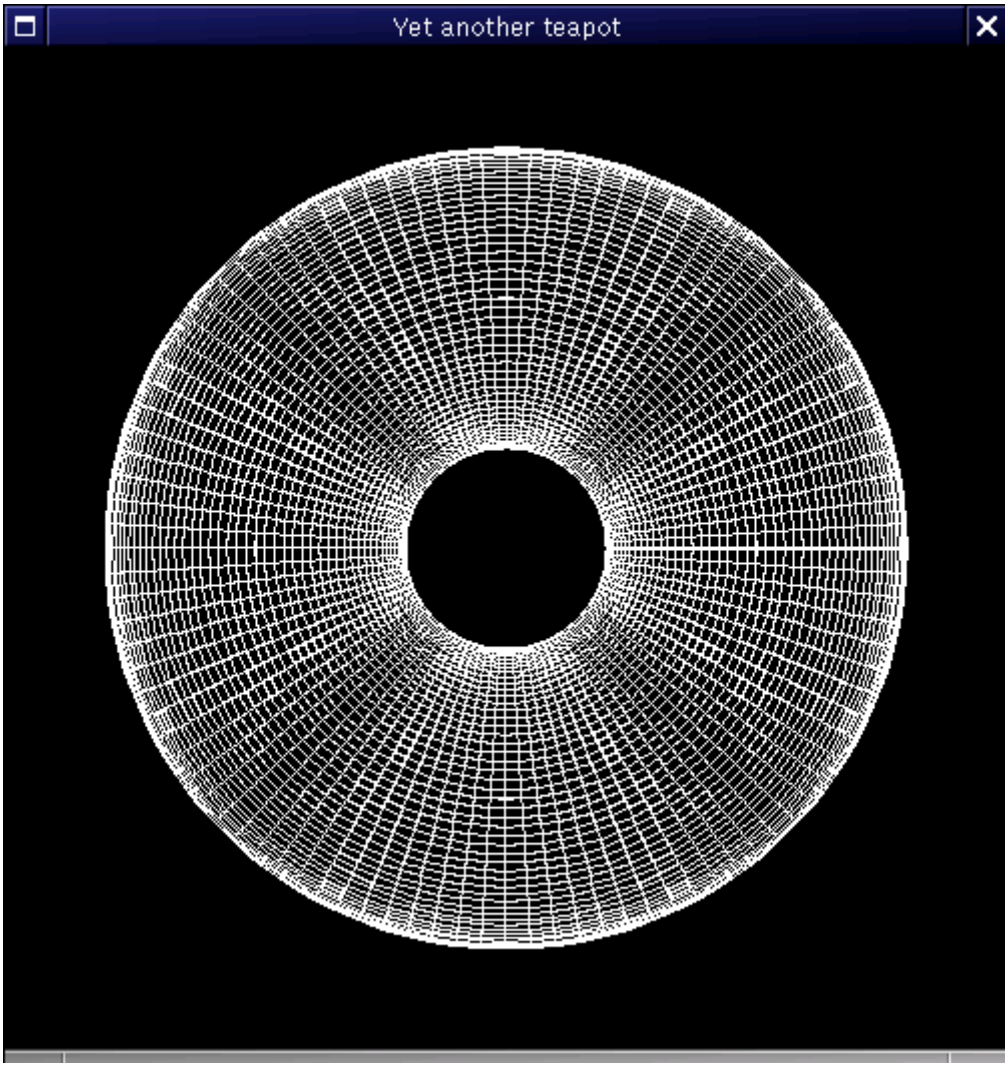
```
#include <iostream>
#include <cstdlib>
#include <GL/glut.h>
using namespace std;

// function prototypes
void disp(void);
void keyb(unsigned char key, int x, int y);


// window identifier
static int win;

int main(int argc, char **argv){

  //////////
  // INIT //
  //////////

  // initialize glut
  glutInit(&argc, argv);

  // specify the display mode to be RGB and single buffering
  // we use single buffering since this will be non animated
  glutInitDisplayMode(GLUT_RGBA | GLUT_SINGLE);

  // define the size
  glutInitWindowSize(500,500);

  // the position where the window will appear
  glutInitWindowPosition(100,100);


  // if we would want fullscreen:
  // glutFullScreen();

  // create the window, set the title and keep the
  // window identifier.
  win = glutCreateWindow("Yet another teapot");

  //////////////
  // CALLBACK //
  //////////////

  glutDisplayFunc(disp);
  glutKeyboardFunc(keyb);

  ////////////
  // OPENGL //
  ////////////

  // define the color we use to clearscreen
  glClearColor(0.0,0.0,0.0,0.0);



  // enter the main loop
```

```
  glutMainLoop();

  return 0;
}


void disp(void){

  // do  a clearscreen
  glClear(GL_COLOR_BUFFER_BIT);

  // draw something

  glutWireTeapot(0.5);
  // glutSolidTeapot(0.5);
  // glutWireSphere(0.5,100,100);
  // glutSolidSphere(0.5,100,100);
  // glutWireTorus(0.3,0.5,100,100);
  // glutSolidTorus(0.3,0.5,100,100);
  // glutWireIcosahedron();
  // glutSolidIcosahedron();
  // glutWireDodecahedron();
  // glutSolidDodecahedron();
  // glutWireCone(0.5,0.5,100,100);
  // glutSolidCone(0.5,0.5,100,100);
  // glutWireCube(0.5);
  // glutSolidCube(0.5);
}

void keyb(unsigned char key, int x, int y){
  cout << "Pressed key " << key << " on coordinates (" << x << "," << y << ")";
  cout << endl;
  if(key == 'q'){
    cout << "Got q,so quitting " << endl;
    glutDestroyWindow(win);
    exit(0);
  }
}
```

You can also download the **source**, some screenshots of the results are visible below:


Yet another teapot


Yet another teapot



Putting it even more together: YATWM

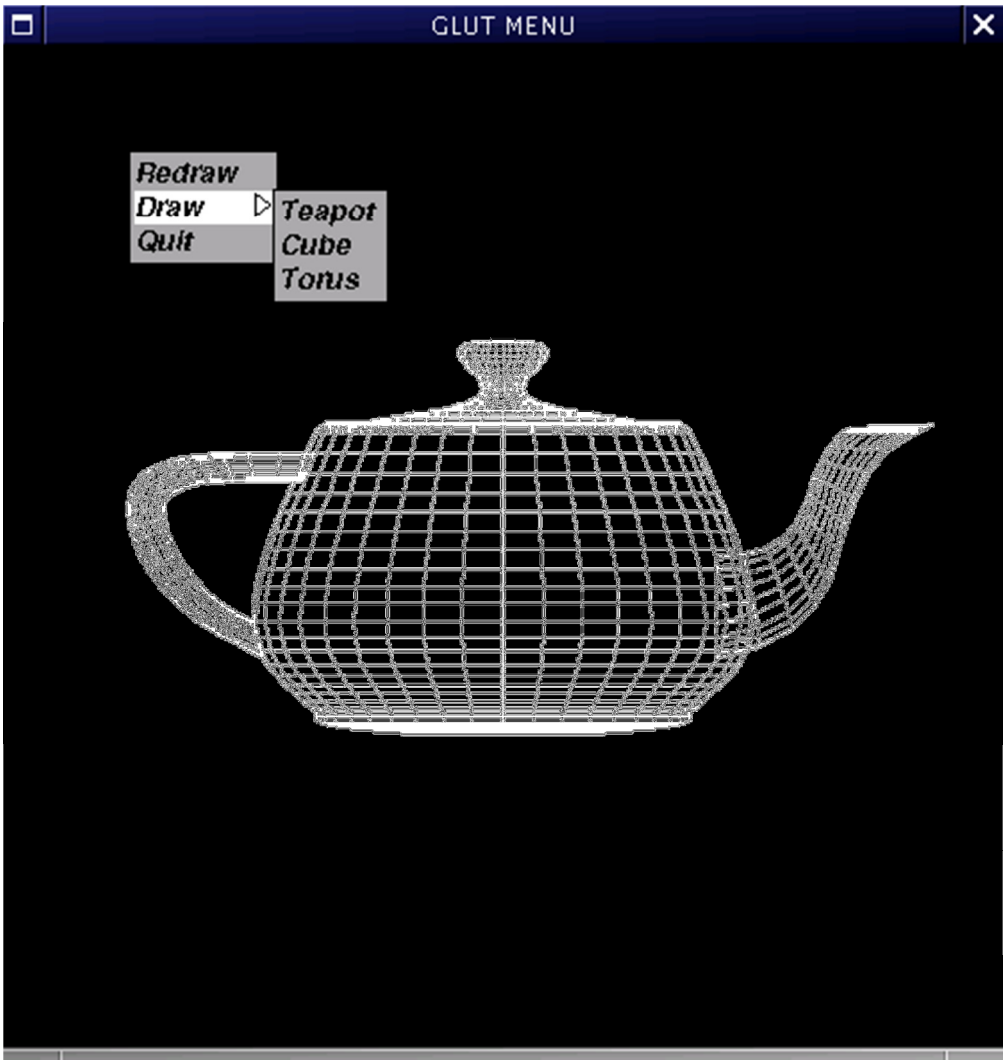
YATWM = Yet Another Teapot With Menus, the example illustrates how to create and use popup menus. Here we create a main menu that becomes active when the right mouse button is pressed, this menu also contains a submenu. This is basically the YAT example except the keyboard callback to quit is replaced by the menu callback and we can now choose which primitive to draw by selecting it from the menu instead of recompiling the code like we had to do with YAT. There is also a screenshot available below. And just like YAT 2 gl methods had to be used.
Not in fact three are used, a call to *glFlush()* is needed to flush the calculations to the X server.

```
#include <iostream>
#include <GL/glut.h>
#include <cstdlib>
using namespace std;

void createMenu(void);
void menu(int value);
void disp(void);

static int win;
static int menid;
static int submenid;
static int primitive = 0;
```

```
int main(int argc, char **argv){

  // normal initialisation
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_RGBA | GLUT_SINGLE);
  glutInitWindowSize(500,500);
  glutInitWindowPosition(100,100);

  win = glutCreateWindow("GLUT MENU");

  // put all the menu functions in one nice procedure
  createMenu();

  // set the clearcolor and the callback
  glClearColor(0.0,0.0,0.0,0.0);

  glutDisplayFunc(disp);

  // enter the main loop
  glutMainLoop();

}


void createMenu(void){
  //////////
  // MENU //
  //////////

  // Create a submenu, this has to be done first.
  submenid = glutCreateMenu(menu);

  // Add sub menu entry
  glutAddMenuEntry("Teapot", 2);
  glutAddMenuEntry("Cube", 3);
  glutAddMenuEntry("Torus", 4);

  // Create the menu, this menu becomes the current menu
  menid = glutCreateMenu(menu);

  // Create an entry
  glutAddMenuEntry("Clear", 1);

  glutAddSubMenu("Draw", submenid);
  // Create an entry
  glutAddMenuEntry("Quit", 0);

  // Let the menu respond on the right mouse button
  glutAttachMenu(GLUT_RIGHT_BUTTON);


}

void disp(void){
  // Just clean the screen
  glClear(GL_COLOR_BUFFER_BIT);

  // draw what the user asked
  if(primitive == 1){
    glutPostRedisplay();
  }else if(primitive == 2){
```

```
    glutWireTeapot(0.5);
  }else if(primitive == 3){
    glutWireCube(0.5);
  }else if(primitive == 4){
    glutWireTorus(0.3,0.6,100,100);
  }
  glFlush();
}


void menu(int value){
  if(value == 0){
    glutDestroyWindow(win);
    exit(0);
  }else{
    primitive=value;
  }

  // you would want to redraw now
  glutPostRedisplay();
}
```

You can also download the **sourcecode**, and a screenshot is available below.



The teapot strikes back: YATWT

YATWT = Yet Another Teapot With Timers, the example illustrates how to create and use timers. Here we create a main window which is wite and contains a white shape (initial values, it can be easily changed that a random figure in a random color will be used), after that two timers are initialised one that runs 1 second and changes the color tot a random color and another timer that expires after 5 seconds and takes a shape out of 4 predefined possibilities. Here we had to use one extra OpenGL command which is glColor3f which changes the drawing/foreground color to the color we create after a timer is expired. For more information about the random() functions please refer to man srand and man rand on your system. Please note that after a timer has expired we reinitialise him.

```
#include <iostream>    // for output
#include <cstdlib>     // for exit and rand
#include <GL/glut.h>   // for gl* glut*
#include <ctime>       // for seeding rand()
using namespace std;

#define TEAPOT 0
#define TORUS 1
#define CUBE 2
#define SPHERE 3

//////////////////
// global vars //
//////////////////

static int object=1;
static int win;
static float r=1.0;
static float g=1.0;
static float b=1.0;

//////////////////
// prototypes //
//////////////////

void display(void);
void keyboard(unsigned char key, int x, int y);
void timerColor(int value);
void timerShape(int value);

///////////
// main //
///////////

int main(int argc, char **argv){

  // init
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);
  glutInitWindowSize(500,500);
  glutInitWindowPosition(100,100);

  win = glutCreateWindow("YATWT");

  // callback
  glutDisplayFunc(display);
  glutKeyboardFunc(keyboard);
  // change color each second
  glutTimerFunc(1000,timerColor,0);
```

```
  // change the shape after five seconds
  glutTimerFunc(5000,timerShape,0);

  // clearcolor & main loop
  glClearColor(1.0,1.0,1.0,0.0);

  // seed random
  // see `man srand` for the user of srand and random
  time_t timer;
  srand(time(&timer));

  glutMainLoop();
  return 0;
}

//////////////
// display //
//////////////

void display(void){

  // clean te window
  glClear(GL_COLOR_BUFFER_BIT);

  // define the color in which te object has to be drawn
  glColor3f(r,g,b);
  // determine which object to draw
  switch(object){
  case TEAPOT:
    glutSolidTeapot(0.5);
    break;
  case TORUS:
    glutSolidTorus(0.25,0.50,100,100);
    break;
  case CUBE:
    glutSolidCube(0.5);
    break;
  case SPHERE:
    glutSolidSphere(0.5,100,100);
    break;
  }
  glutSwapBuffers();
}


//////////////
// keyboard //
//////////////

void keyboard(unsigned char key, int x, int y){
  if(key=='q'){
    cout << "Quitting" << endl;
    glutDestroyWindow(win);
    exit(0);
  }
}

////////////////
// timercolor //
////////////////
```

```
void timerColor(int value){
  // get new color or a value in [0,1]
  r = (1.0*(random()%256))/256.0;
  g = (1.0*(random()%256))/256.0;
  b = (1.0*(random()%256))/256.0;

  // say what we are doing
  cout << "New color is (" << r << "," << g <<  "," << b << ")" << endl;

  // draw it + reinitialise timer
  glutPostRedisplay();
  glutTimerFunc(1000,timerColor,0);
}

////////////////
// timerShape //
////////////////

void timerShape(int value){
  // generate new object
  object = random()%4;

  // say what we are doing
  cout << "Next object will be a ";
  switch(object){
  case TEAPOT:
    cout << "teapot" << endl;
    break;
  case TORUS:
    cout << "torus" << endl;
    break;
  case CUBE:
    cout << "cube" << endl;
    break;
  case SPHERE:
    cout << "sphere" << endl;
    break;
  }

  // draw it + reinitialise timer
  glutTimerFunc(5000,timerShape,0);
}
```

You can also download the **sourcecode**.
You can also see potential trouble in the example each five seconds the shape changes and no redraw is called, each second the color changes and a redraw is called. What happens at the fifth second ? Does this color first change or does the shape first change ? Now let us adapt the source code so that this issue disappears. (I included the issue above not because is it bulletproof but simply to illustrate how to use two timers). Now in this example the first shape and color is also random, only one timer is used. The keyboard and the display function were not change so I did not put them here. But they are in the sourcecode if you want them.

```
#include <iostream>    // for output
#include <cstdlib>     // for exit and rand
#include <GL/glut.h>   // for gl* glut*
#include <ctime>       // for seeding rand()
using namespace std;

#define TEAPOT 0
#define TORUS 1
#define CUBE 2
```

```
#define SPHERE 3

/////////////////
// global vars //
/////////////////

static int object;
static int win;
static float r;
static float g;
static float b;

/////////////////
// prototypes //
/////////////////

void display(void);
void keyboard(unsigned char key, int x, int y);
void timerColor(int value);
void changeShape();

//////////
// main //
//////////

int main(int argc, char **argv){

  // init
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);
  glutInitWindowSize(500,500);
  glutInitWindowPosition(100,100);

  win = glutCreateWindow("YATWT");

  // callback
  glutDisplayFunc(display);
  glutKeyboardFunc(keyboard);
  // change color each second
  timerColor(0);

  // clearcolor & main loop
  glClearColor(1.0,1.0,1.0,0.0);

  // seed random
  // see `man srand` for the user of srand and random
  time_t timer;
  srand(time(&timer));

  glutMainLoop();
  return 0;
}


////////////////
// timercolor //
////////////////

void timerColor(int value){
  static int counter=0;
  if(counter == 4){
```

```
    counter =0;
    changeShape();
  }else{
    counter++;
  }

  // get new color or a value in [0,1]
  r = (1.0*(random()%256))/256.0;
  g = (1.0*(random()%256))/256.0;
  b = (1.0*(random()%256))/256.0;

  // say what we are doing
  cout << "New color is (" << r << "," << g <<  "," << b << ")" << endl;

  // draw it + reinitialise timer
  glutPostRedisplay();
  glutTimerFunc(1000,timerColor,0);
}

//////////////////
// changeShape //
//////////////////

void changeShape(){
  // generate new object
  object = random()%4;

  // say what we are doing
  cout << "Next object will be a ";
  switch(object){
  case TEAPOT:
    cout << "teapot" << endl;
    break;
  case TORUS:
    cout << "torus" << endl;
    break;
  case CUBE:
    cout << "cube" << endl;
    break;
  case SPHERE:
    cout << "sphere" << endl;
    break;
  }
}
```