GLUT Tutorial

Roma

July 26, 2012

What is GLUT?

GLUT is the OpenGL Utility Toolkit.
It is a platform independant toolkit with utilty functions to help with basic window tasks.

Creating a Window

GLUT can be used to create a windows as follows

```
int main(int argc, char** argv){
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DEPTH| GLUT_DOUBLE
     | GLUT_RGBA);
  glutInitWindowSize(800,800);
  glutCreateWindow("Test Window");
  glutMainLoop();   //Never returns

  return EXIT_SUCCESS;
}
```

Problem anyone?

But Empty Windows Are Pointless

We can tell GLUT to use a given drawing function

```
...
glutDisplayFunc(display);
...
```

with a function pointer!

```
void display(){
  glClear(GL_DEPTH_BUFFER_BIT |
      GL_COLOR_BUFFER_BIT);
}
```

But Nothing Happens

The window making code in the previous slide used a double buffered window so the display code doesn't actually do anything. You need to swap the buffers before an image will appear at the end of display() using

```
glutSwapBuffers();
```

The Black Screen of Victory

So now we have a blank window. By adding in OpenGL commands into the display function we can make the window display what we want.

```
void display(){
  glClear(GL_DEPTH_BUFFER_BIT |
     GL_COLOR_BUFFER_BIT);
  glRenderCoolImage();
  glutSwapBuffers();
}
```

Moving Pictures

Now that we have a static image on the screen, we want to move to the next step. A changing image. We can specify a function for GLUT to use when nothing is happening to update the state of the world.

```
...
glutIdleFunc(changify);
...
```

with a function pointer!

```
void changify(){
  //Update the world state cooly
}
```

The Image Is Still Stationary

Even though we have updated what should be drawn we haven't actually told anything that the screen needs to redraw. As far as everything else is concerned redrawing would just draw the original thing again.

To get around this, we use the following command at the end of changify() to say that the screen needs to be redrawn because it has actually changed.

```
void changify(){
  //Update the world state cooly
  glutPostRedisplay();
}
```

So You Can Only Make Movies?

At the moment, we can show animations, but we can't interact with them. Essentially we have made a movie. As this is software, there should be a way of letting the user interact with your software somehow.

What sort of interaction should be supported?

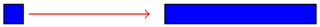
How to Speak Computer

GLUT supports several different kinds of input which we will look at

- Window re-size/shape
- Keyboard
- Mouse
- Right click menus
- Waiting

The Screen Ratio Diet

In terms of interaction dragging a window is pretty boring, but reshaping it might allow the user to change the shape of objects on the screen.



Enforcing Behaviours

There isn't a single correct option when dealing with changes in screen size and shape. Thankfully GLUT gives us a way to tailor the behaviour to our needs. We can assign a screen resize callback to make any appropriate adjustments

```
...
glutReshapeFunc(reshape);
...
void reshape(int width, int height){
 //Do something sensible
}
```

What different ways are there of sensibly dealing with a change is screen size and shape?

Keyboard Input - ASCII

Suppose we want to go to the extreme of having keyboard input. First we shall consider how to deal with events which correspond to the typing of an ASCII character. Note that you don't know at which point in the press/release cycle this is triggered. Also note that information about the mouse location is also provided.

```
...
glutKeyboardFunc(keypress);
...
void keypress(unsigned char key, int mouseX,
      int mouseY){
int mod = glutGetModifiers();
  if(key == 'i' && mod == GLUT_ACTIVE_SHIFT)
     {
      printf("The mouse is at %d %d\n", mouseX
         ,mouseY);
  }
}
```

What About The Arrow Keys?

GLUT has a separate callback for non ASCII keypresses. Similar to the previous, it also gives you the mouse position, but now you need to compare the result to special constants for each key. You still need glutGetModifiers() to find out about ALT, CTRL and SHIFT.

```
...
glutSpecialFunc(specialpress);
...
void specialpress(int key, int mouseX, int
    mouseY){
  if( key == GLUT_KEY_RIGHT){
    printf("The mouse is at %d %d\n", mouseX
        ,mouseY);
  }
}
```

So Seriously, What About the Mouse?

Just like we have callbacks for the keyboard, we have similar ones for the mouse. Much like how the keyboard can see the mouse, calling glutGetModifiers() is used in the mouse callback function to find out about the state of certain keyboard keys.

```
  ...
gluMouseFunc(mouse);
   ...
void mouse(int button, int state, int x, int y
    ){
  if(button == GLUT_LEFT_BUTTON && state ==
      GLUT_DOWN){
      printf("The mouse is at %d %d\n", x,y);
  }
}
```

What About Mouse Movement

The previous function told us only when mouse buttons are pushed. We need two different functions in order to track when the mouse is moving

```
    ...
glutMotionFunc(mouseMove);
glutPassiveMotionFunc(passiveMove);
   ...
void mouseMove(int x, int y){
 printf("A button is down and the mouse is at
    %d %d\n", x,y);
}

void passiveMove(int x, int y){
 printf("No buttons are down and the mouse is
    at %d %d\n", x,y);
}
```

Déjà Vu?

Looking at all these functions. Surely they remind you of something. Have you seen anything that looks similar to this before?

Context Menus: The One True Path

So we have mouse and keyboard interaction, but what about right click menus? These are standard things and yet making them with the existing functions would be a lot of work.
To deal with this GLUT has its own mechanism for creating right click menus.

```
int submenu = glutCreateMenu(violenceMenu);
glutAddMenuEntry ("Shoot", 0);
glutAddMenuEntry ("Explode", 1);

int menu = glutCreateMenu(mainMenu);
glutAddMenuEntry ("Fish", 0);
glutAddMenuEntry ("Meat", 1);
glutAddMenuEntry ("Chicken", 2);
glutAddSubMenu("Extra Functions",submenu);

glutAttachMenu(GLUT_RIGHT_BUTTON);
```

The Menus - They Do Nothing!

So we have a menu, but we don't actually have any code to deal with the menu. However, in hte previous slide we did specify which functions would be used to respond to each menu being clicked. So all we need is to define those functions.

```
void mainMenu(int item){
  if(item == 0) prinf("You clicked fish\n");
  if(item == 1) prinf("Moo\n");
  if(item == 2) prinf("Bawk\n");
}

void violenceMenu(int item){
  if(item == 0) prinf("Peow Peow\n");
  if(item == 1) {
   int* f;
   **f = 500;
  }
}
```

Waiting

Suppose we want to have something happen after a specified amount of time? Well GLUT has a function for that too. It takes a time in milliseconds, a function and a value that lets you identify where it came from. The function will be called at some point after the requested amount of time has passed. There is no mechanism for cancelling a request, but the value parameters allows you to work out that it should be ignored.

```
...
int millis = 1000;
glutTimerFunc(millis,coolStuff, 0);
...
void coolStuff(int value){
 if( value ==0){
   //Do something relevant
 }
}
```

Stoke Fonts

Stroke fonts are pretty much like everything else with GL. You use a transform to set where you want them to be drawn, and draw as normal. They will draw from where the translation moved to, and drawing each character translates the model view matrix to fit the next characted into place.

```
void display(){
 glTranslatef(0,10,0);
 glScalef(0.05,0.05,0.05);
 glColor3f(0,1.0,0);
 for (unsigned int i = 0; i < length; i++) {
  glutStrokeCharacter(GLUT_STROKE_ROMAN,s[i]);
 }
 glutPostRedisplay();
}
```

Stroke Fonts Killed Santa

There are in fact only 2 stroke fonts supported by GLUT: GLUT_STROKE_ROMAN and GLUT_STROKE_MONO_ROMAN.
On top of that, each drawing operation affects the model view matrix. This means that you need to undo all of the changes after you finish drawing, otherwise you may not be sure where exactly you are in the world.
Also you are writing words in the world. What if you just want your writing to be in a fixed position of the screen?

Journey to Flatland

We can also draw raster text to the screen. Raster cooridinates are transformed the same way that normal vertex coordinates are using the model view and projection matricies. Raster coordinates are however tracked independantly.

```
glColor3f(1.0,0,0);
glRasterPos2f(0, 0);
for (unsigned int i = 0; i < length; i++) {
  glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_
     24,s[i]);
}
```

Life in Flatland

Every call to glutBitmapCharacter(. . .) results in the current offset for drawing being moved by the exact amount of space necessary for that character to be drawn. This is good as it means that you can draw strings with an easy loop.
It has the following fonts

GLUT_BITMAP_8_BY_13
GLUT_BITMAP_9_BY_15
GLUT_BITMAP_TIMES_ROMAN_10
GLUT_BITMAP_TIMES_ROMAN_24
GLUT_BITMAP_HELVETICA_10
GLUT_BITMAP_HELVETICA_12
GLUT_BITMAP_HELVETICA_18

But What Can We Actually Do?

We have looked at a number of GLUT commands which allow us to respond to input and write text to the screen, but we still can't actually draw anything.
To this end we need OpenGL and GLU to actually have output on the screen. So lets recap some basic OpenGL and have a look at a basic program.

Look Mum - A Triangle!

We give co-ordinates for points inside of glBegin(). These are coordinates in the rendering world, and will be mapped back to the real world by OpenGL.

```
glBegin(GL_TRIANGLES);
glVertex3f( 0,1,0);
glVertex3f(-1,0,0);
glVertex3f( 1,0,0);
glEnd();
```

Drawing Shapes

Diffent parameters result in different shapes being drawn

GL_POINTS
GL_LINES
GL_LINE_STRIP
GL_LINE_LOOP
GL_TRIANGLES
GL_TRIANGLE_STRIP
GL_TRIANGLE_FAN
GL_QUADS
GL_QUAD_STRIP
GL_POLYGON