Drawing Tiles with OpenGL

Author: Mark Szymczyk
Last Update: August 7, 2006

This article shows you how to use OpenGL to quickly draw tiled backgrounds for 2D games.

Updates

Textures do not have to be square for widest compatibility. The width and height should be a power of 2. I misread a statement in the OpenGL Red Book. (August 7, 2006)

Introduction

Most of you are aware that OpenGL is a library for 3D graphics, but you can also use OpenGL for 2D graphics. Why use OpenGL for 2D graphics on Mac OS X instead of Quartz or Cocoa's graphics routines? There are several reasons to use OpenGL for 2D drawing.

- Speed. OpenGL takes advantage of your Mac's graphics accelerator, which makes drawing fast.
- Portability. Your OpenGL drawing code will run on Windows and Linux as well as Mac OS X.
- You can take your OpenGL knowledge and apply it to 3D games.

The fastest way to draw 2D graphics with OpenGL is to use texture maps. If your game has all of its action on one screen, you can draw the game's background on a texture map and draw the texture map on the screen.

If your game's world is large, covering multiple screens, storing the background in one texture map won't work. The maximum texture size depends on your video card, but it generally is no more than 4096 pixels. Even if you had one texture map per screen, you would eventually run out of video memory. This is where tiling comes in.

A tile is a reusable piece of artwork. If you ever played one of the Super Mario Brothers or Ultima series of games, you've seen tiles in action. By using tiles you can create large game worlds and have the graphics fit in video memory. Fitting your game's graphics in video memory is the key to fast drawing. The fastest drawing occurs when drawing from video memory to video memory because the pixels don't have to travel from the CPU to the video card.

The best strategy for using tiles in OpenGL is to place a set of tiles in one texture map. For the widest compatibility you should make the texture map a power of 2. A texture map size of 1024 by 1024 pixels enables your code to run on Macs with Rage 128 and better video cards, which is almost every Mac capable of running Mac OS X. This texture size allows you to store 1024 32-by-32 pixel tiles in one texture map. If you don't care about supporting the Rage 128, you can increase the texture size to 2048 pixels; a 2048 by 2048 texture map can store 4096 32-by-32 pixel tiles.

Setting Up the Viewing Volume

In OpenGL the viewing volume determines what you can see on the screen. An OpenGL viewing volume can have one of two projections: perspective and orthographic. 3D games normally use perspective projection. With perspective projection objects appear larger the closer they get to the camera. For a 2D game you want to use orthographic projection, where objects stay the same size no matter how far away they are from the camera.

To create a viewing volume with an orthographic projection, call the function `gluOrtho2D()`. This function is part of the OpenGL Utility Library (GLU). GLU is part of Apple's OpenGL framework, but you must include the header file `glu.h` to use GLU functions. The `gluOrtho2D()` function takes four arguments: left, right, bottom, and top.

How large a viewing volume should you supply to `gluOrtho2D()`? You can make the viewing volume as large as you want, but there are two common ways to create an orthographic viewing volume. First, you can decide how many tiles you want to appear on the screen and supply that to `gluOrtho2D()`. Specifying the viewing volume in terms of tiles will make your life easier for a tile-based game. You can use tiles as the basic unit of measurement in the game world, which makes things like keeping the player from walking through walls easier to code. The following call displays 40 tiles horizontally and 30 tiles vertically in the viewing volume:

```
gluOrtho2D(0.0, 40.0, 0.0, 30.0);
```

What is cool about orthographic projections for 2D games is the screen size is the same no matter what the player's screen resolution is. In the previous example, 40 tiles will fit across the screen on a 12-inch laptop display and on a 30-inch widescreen monitor.

The second common way to build a viewing volume is to decide how many "pixels" you want on the screen. The following call displays a 640 by 480 view volume:

```
gluOrtho2D(0.0, 640.0, 0.0, 480.0);
```

The reason why I put the term pixels in quotation marks in the last paragraph is that the viewing volume isn't specified in pixels. The monitor's screen resolution determines the number of pixels on the screen. If the player's screen resolution is 1280 by 1024 pixels, each horizontal unit in a 640 by 480 view volume corresponds to two pixels (640 units and 1280 pixels). A 640 by 480 view volume lets your game treat the screen as if it has a 640 by 480 screen resolution.

Creating the Texture Map

If you read my article on loading textures, you know the function `glTexImage2D()` creates a texture out of an image. There's a little more work to do to create an OpenGL texture map. I'm going to cover that material in this section.

Texture objects store texture data so you don't have to call `glTexImage2D()` every time you want to draw a texture map. Call the function `glGenTextures()` to create a texture object for the texture map. This function takes two arguments: the number of texture objects to create and either a texture name or a list of names, depending on the number of texture objects created. You normally create one texture object at a time.

```
GLuint textureName;
glGenTextures(1, &textureName);
```

Because OpenGL works on multiple screen resolutions, a texture is not going to be drawn exactly the same on all computers. Depending on the player's screen resolution, OpenGL may have to scale a texture to make it look right on the player's screen. You have to tell OpenGL what to do when it has to magnify or minify a texture. Call the function `glTexParameteri()`, which sets the values of various OpenGL texture parameters. This function takes three arguments.

- The type of texture, which is normally `GL_TEXTURE_2D`.
- The name of the parameter. It will be `GL_TEXTURE_MAG_FILTER` for texture magnification and `GL_TEXTURE_MIN_FILTER` for texture minification.
- The value of the parameter. For texture magnification and minification there are two possible values: `GL_NEAREST` and `GL_LINEAR`. `GL_NEAREST` is faster while `GL_LINEAR` looks better.

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```

Drawing a Tile

Drawing a tile on the screen requires the following steps:

1) Determine the tile to draw.
2) Locate the tile in the texture map holding the tiles.
3) Determine where to draw the tile on the screen.
4) Make the OpenGL calls.

Determining the Tile to Draw

2D game levels normally contain an array of numbers, with each number telling you the tile to draw at that location in the game world. Your game is responsible for finding the tile to draw at a specific location in the game world.

Locating a Tile in a Texture Map

Once you get the tile number, you have to find the tile in the texture map. The entire texture map coordinates range from 0 to 1, with (0.0, 0.0) being the upper left corner of the texture and (1.0, 1.0) being the lower right corner. If you weren't using tiles, drawing the background would be easy. Draw the entire texture map and have it fill the screen.

Tiles make the drawing situation a little more complicated. The texture map contains an entire set of tiles so you can't just draw the whole texture. You have to find the tile you want to draw so you draw only the part of the texture that contains the tile.

Suppose the texture map holds an 8 by 8 collection of tiles, which you can see in the figure below. Suppose you want to draw tile number 10. It resides in the second row and the third column. Because there are 8 tiles in each row and 8 tiles in each column, you move .125 units to move one tile down or one tile to the right (.125 = 1/8). To draw tile number 10, you would use the following texture coordinates:

```
Upper left: (.25, .125)
Upper right: (.375, .125)
Lower right: (.375, .25)
Lower left: (.25, .25)
```

Determining Where to Draw a Tile

Each texture corner corresponds to an OpenGL vertex. The vertex's x and y coordinates are where you want the corner to appear on the screen. Suppose you set up the viewing volume to fit 40 tiles horizontally and 30 tiles vertically, and you want to draw tile number 10 in the upper left corner of the screen. You would use the following vertex coordinates:

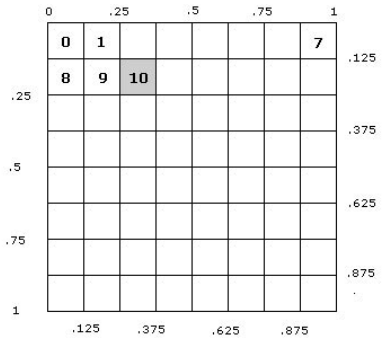
```
Upper left: (0.0, 30.0)
Upper right: (1.0, 30.0)
Lower right: (1.0, 29.0)
Lower left: (0.0, 29.0)
```

In OpenGL the y coordinate increases as you move up the screen. With a screen projection displaying 30 units vertically, 0 is the bottom of the screen, and 30 is the top.

Making the OpenGL Calls

To draw a tile in OpenGL, you draw a texture-mapped quadrilateral. Before you draw, you must make three OpenGL calls. Call `glEnable()` to turn on texture mapping. Call `glTexEnvf()` to tell OpenGL how to draw the texture. For tiles you want to use the mode `GL_REPLACE`. This mode tells OpenGL to draw the tile, overwriting what was previously on the screen. Call `glBindTexture()` to bind the texture so you can draw it.

```
glEnable(GL_TEXTURE_2D);
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
glBindTexture(GL_TEXTURE_2D, textureName);
```



After you make the setup calls, call `glBegin()` to start drawing the quadrilateral.

```
glBegin(GL_QUADS);
```

Each quadrilateral has four vertices. For each vertex you must specify a texture coordinate and a screen coordinate. Call `glTexCoord2f()` to specify the texture coordinate, and call `glVertex3f()` to specify where the vertex is on the screen. `glVertex3f()` takes three arguments: x, y, and z. Use a z value of 0 for 2D drawing.

The values you pass to `glTexCoord2f()` and `glVertex3f()` depend on how you store the tiles in the texture and how you set up the orthographic projection. The following code draws the upper left corner of tile number 10 in the ongoing example in this article:

```
glTexCoord2f(.25, .125);
glVertex3f(0.0, 30.0, 0.0);
```

Here's the code for the other three corners.

```
// Upper right corner
glTexCoord2f(.375, .125);
glVertex3f(1.0, 30.0, 0.0);

// Lower right corner
glTexCoord2f(.375, .25);
glVertex3f(1.0, 29.0, 0.0);

// Lower left corner
glTexCoord2f(.25, .25);
glVertex3f(0.0, 29.0, 0.0);
```

The order in which you specify the four corners of the tile is very important. Pick one corner and move either clockwise or counterclockwise to draw the other corners. If you crisscross, say drawing the lower left corner after the upper right corner, the drawing will get messed up.

After giving the four vertices texture and screen coordinates, call `glEnd()` to stop drawing.

```
glEnd();
```

Conclusion

Drawing tiles with OpenGL isn't too difficult. Store a set of tiles in a texture map. Draw a texture-mapped quadrilateral to draw the tile on the screen. Specify the texture coordinate and screen coordinate for each of the quadrilateral's four vertices.

The sample code that accompanies this article loads a game level on the screen. It provides a relatively simple example of an OpenGL 2D tile engine. Use the arrow keys to scroll around the level.