Loading Textures with QuickTime

Author: Mark Szymczyk
Last Update: December 27, 2005

Most OpenGL games use texture mapping. There are many online articles and tutorials on OpenGL texture mapping but they gloss over loading the textures from disk. The reason many articles gloss over loading textures is that the code to load a texture depends on the texture's file type and the operating system on which the code is running. This article fills a gap for Mac OS X developers by showing how to use QuickTime to load textures.

Introduction

QuickTime is Apple's multimedia technology, and it is huge. It plays video and audio, which I covered in a previous article. But QuickTime can also load image files and draw the images by using graphics importers. QuickTime graphics importers can import JPEG, GIF, PICT, TIFF, PNG, TGA, and BMP (Windows bitmap) files. You can load textures in any of these formats with one set of code.

To use QuickTime's graphics importers to load textures, you must perform the following steps:

1) Open a graphics importer.
2) Retrieve information about the texture image, including its width, height, and color depth.
3) Create an offscreen buffer to draw the texture.
4) Draw the texture.

Before you can open a graphics importer you must locate the texture file and let QuickTime know where it is.

Setting Up Your Xcode Project

To use QuickTime to load textures, you must add the QuickTime framework and the texture files to your Xcode project. Choose Project > Add To Project to add files and frameworks to your project. The QuickTime framework should be in the directory `System/Library/Frameworks`. In your source code files you must add the header files `Movies.h` and `QuickTimeComponents.h`.

```
#include <QuickTime/Movies.h>
#include <QuickTime/QuickTimeComponents.h>
```

The code for this article uses the Carbon framework, but there's no reason why you can't use the code in a Cocoa application. The area of the code you'll change is the code that finds the file in the application bundle. The code uses the Core Foundation framework, which is the C equivalent of Cocoa's foundation classes. A Core Foundation string variable, `CFStringRef`, is the same as a pointer to a `NSString` object. A Cocoa application would replace the Core Foundation code (the variables and functions have the prefix CF) with the Cocoa equivalent.

Finding the File in the Application Bundle

To load the texture you must first find the texture's file in the application bundle. Finding the file requires you to call two functions. The first function to call is `CFBundleGetMainBundle()`, which returns the application's bundle.

```
CFBundleRef gameBundle = CFBundleGetMainBundle();
```

The second function to call is `CFBundleCopyResourceURL()`, which returns the file's location in the bundle. This function takes four arguments. The first argument is the bundle you retrieved by calling `CFBundleGetMainBundle()`. The second argument is the file's name. The third argument is the file's extension. The final argument is a subdirectory in the bundle where the operating system should search for the file. You can pass `NULL` as the final argument, which tells the operating system to search the entire bundle for the file.

```
CFBundleRef gameBundle;
CFStringRef filename;
CFStringRef fileExtension;
CFStringRef subdirectory;
CFURLRef fileLocation;

fileLocation = CFBundleCopyResourceURL(gameBundle, filename,
    fileExtension, subdirectory);
```

Suppose you want to load a file named `Background.png`. There are two ways you can call `CFBundleCopyResourceURL()`. First, you can use `Background` as the file name and `png` as the extension. Second, you can use `Background.png` as the file name and `NULL` as the extension. Both methods work. The first method is slightly faster if you have lots of files in your application bundle.

Creating a Data Reference

The easiest way to open a graphics importer is to use a data reference. The data reference tells QuickTime where to find the data it needs, which is the texture file for the purposes of this article. Call the function `QTNewDataReferenceFromCFURL()` to create a data reference. This function was introduced in QuickTime 6.4. QuickTime 6.4 shipped with Mac OS X 10.3, which means `QTNewDataReferenceFromCFURL()` works on Mac OS X 10.3 and later. People running Mac OS X 10.2 have to download and install QuickTime 6.4.

`QTNewDataReferenceFromCFURL()` takes four arguments. The first argument is the file location you retrieved by calling `CFBundleCopyResourceURL()`. The second argument is `flags`, which you should set to 0. The third argument is the data reference that `QTNewDataReferenceFromCFURL()` returns. The fourth argument is the data type of the data reference that `QTNewDataReferenceFromCFURL()` returns.

```
OSErr error;
Handle dataRef;
OSType dataRefType;
CFURLRef fileLocation;

error = QTNewDataReferenceFromCFURL(fileLocation, 0, &dataRef,
    &dataRefType);
```

The data reference `QTNewDataReferenceFromCFURL()` returns is a handle, a pointer to a pointer. Before calling `QTNewDataReferenceFromCFURL()`, you must allocate memory for the handle by calling `NewHandle()`. The size of the handle is `AliasHandle`.

```
Handle dataRef;
dataRef = NewHandle(sizeof(AliasHandle));
```

Opening the Graphics Importer

Call the function `GetGraphicsImporterFromDataRef()` to open a graphics importer. This function takes three arguments. The first two arguments are the data reference and the data reference type you created by calling `QTNewDataReferenceFromCFURL()`. The last argument is the graphics importer `GetGraphicsImporterFromDataRef()` returns.

```
ComponentInstance fileImporter;
OSErr error;

error = GetGraphicsImporterForDataRef(dataRef, dataRefType,
    &fileImporter);
```

Retrieving Information about the Texture

To draw a texture you need to know some information about the texture, such as its width, height, and color depth. The `ImageDescription` data structure stores relevant information about an image. You must create an `ImageDescription` handle and call the function `GraphicsImportGetImageDescription()` to get the information you need about the texture.

```
ComponentResult error;
ComponentInstance fileImporter;
ImageDescriptionHandle imageInfo;

imageInfo = (ImageDescriptionHandle)NewHandle(sizeof(ImageDescription));
error = GraphicsImportGetImageDescription(fileImporter, &imageInfo);
```

Getting the Texture's Size

The `ImageDescription` data structure's most important fields are `width`, `height`, and `depth`. You need these three fields to determine the image's size. After calculating the image's size, you must allocate a pointer to hold the image's data so you can use the texture in your OpenGL application.

```
int width;
int height;
int depth;

width = (**imageInfo).width;
height = (**imageInfo).height;
depth = (**imageInfo).depth

unsigned long imageSize;
Ptr imageData;

imageSize = width * height * depth / 8;
imageData = NewPtr(imageSize);
```

Getting the Texture's Boundary Rectangle

To draw a texture with QuickTime, you must create an offscreen buffer to hold the texture. The function to create an offscreen buffer takes a boundary rectangle as an argument. You want the buffer to match the image's boundary rectangle. Call the function `GraphicsImportGetNaturalBounds()` to get the image's boundary rectangle.

```
Rect imageRect;
ComponentResult error;

error = GraphicsImportGetNaturalBounds(fileImporter, &imageRect);
```

Drawing the Texture

After opening a graphics importer and retrieving the necessary information about the texture, you can go ahead and draw the texture. There are three steps to drawing the texture.

1) Create an offscreen buffer to hold the texture.
2) Draw the image in the offscreen buffer.
3) Specify the texture in OpenGL so you can use the texture in your OpenGL application.

Creating the Offscreen Buffer

QuickTime's graphic importers need offscreen buffers to draw images. Call the function `QTNewGWorldFromPtr()` to create an offscreen buffer. This function takes eight arguments.

- The buffer `QTNewGWorldFromPtr()` creates.
- The pixel format.
- The boundary rectangle for the buffer.
- A color table handle. You can pass `NULL`. Mac OS X has poor support for color depths that use color tables.
- A handle to a graphics device, which is a destination for drawing such as the screen or a printer. Passing `NULL` tells the operating system to use the main graphics device, which works well for Macs with one monitor. On Macs with multiple monitors, passing `NULL` uses the monitor that is set as the main monitor.
- Flags. If you're creating a universal binary, you should pass the flag `kNativeEndianPixMap`. If you don't pass this flag, the operating system creates a big-endian pixel map, which causes a performance hit on Intel Macs.
- A pointer to the image data.
- The number of bytes in one row of the image.

Pixel formats deserve more explanation. Each pixel has a color value with three color components: red, green, and blue. The color value may have a fourth component, alpha, that measures transparency. The pixel format defines the size of each color component. There are four pixel formats for `QTNewGWorldFromPtr()` on Mac OS X.

- `k16BE555PixelFormat` creates a 16-bit color buffer with PowerPC byte ordering. Each pixel has 5 bits of red, green, and blue. The last bit is not used.
- `k16LE555PixelFormat` creates a 16-bit color buffer with Intel byte ordering.
- `k32ARGBPixelFormat` creates a 32-bit color buffer with PowerPC byte ordering. Each pixel has 8 bits of alpha followed by eight bits of red, eight bits of green, and eight bits of blue. This pixel format is the Mac's native pixel format and is the one you'll end up using most.
- `k32BGRAPixelFormat` creates a 32-bit color buffer with Intel byte ordering.

Unless you're writing code specifically for Intel Macs, you should use one of the pixel formats with PowerPC byte ordering. Macs with an Intel processor can use pixel formats with PowerPC byte ordering, but PowerPC Macs cannot use pixel formats with Intel byte ordering. The following code demonstrates how to create an offscreen buffer using QuickTime:

```
Ptr imageData;
GworldPtr offscreenBuffer;
long bytesPerRow;
OSErr error;

bytesPerRow = width * depth / 8;
error = QTNewGWorldFromPtr(&offscreenBuffer, k32ARGBPixelFormat,
    &imageRect, NULL, NULL, kNativeEndianPixMap,imageData, bytesPerRow);
```

Drawing the Image in the Offscreen Buffer

After creating the offscreen buffer, you must set the graphics importer to use the offscreen buffer and draw the texture. Call the function `GraphicsImportSetGWorld()` to set the graphics importer. Call `GraphicsImportDraw()` to draw the texture.

```
ComponentResult error;

error = GraphicsImportSetGWorld(fileImporter, offscreenBuffer, NULL);
error = GraphicsImportDraw(fileImporter);
```

Specifying the Texture in OpenGL

After drawing the texture, call the OpenGL function `glTexImage2D()` (There are also versions of this function for one and three-dimensional textures) to specify the texture image. This function takes nine arguments.

- The target texture, which will normally be `GL_TEXTURE_2D`.
- The level of detail number. Pass 0 to use the base image. Numbers greater than 0 tell OpenGL to use a mipmap, which is a smaller version of the texture. Higher values indicate smaller mipmaps.
- The internal format, which specifies the number of color components in the texture.
- The width of the texture.
- The height of the texture. For the highest level of compatibility the width and height should be a power of 2 and should be at least 64 pixels.
- The width of the texture's border, which will be either 0 or 1. If you have a border width of 1, you must add 2 to the texture's width and height.
- The format, which specifies the pixel data.
- The data type of the pixel data.
- The image data.

For the sample code I included in this article, I used the internal format `GL_RGBA8`. This format has 8 bits of red, green, blue, and alpha. `GL_RGBA8` works well on Mac OS X if you're using 32-bit color.

The most common Mac offscreen buffer pixel format is `k32ARGBPixelFormat`, which has 8 bits of alpha, red, green, and blue. There is no `GL_ARGB` format in OpenGL. If you use `k32ARGBPixelFormat`, your format must be `GL_BGRA`, which is the reverse of `ARGB`. On a PowerPC Mac you must use the data type `GL_UNSIGNED_INT_8_8_8_8_REV`, which reverses the bytes. Intel Macs don't need to reverse the bytes so they use the data type `GL_UNSIGNED_INT_8_8_8_8`.

```
#if __BIG_ENDIAN__
    textureMap.SetType(GL_UNSIGNED_INT_8_8_8_8_REV);
#else
    textureMap.SetType(GL_UNSIGNED_INT_8_8_8_8);
#endif
```

The following code demonstrates the call to `glTexImage2D()` on Mac OS X:

```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8, width, height, 0,
    GL_BGRA, type, imageData);
```

Cleaning Up

When you're finished loading the texture you must dispose of the memory you allocated and close the graphics importer. You allocated a pointer to hold the image data and allocated two handles, one for the image information and one for the QuickTime data reference.

```
OSErr error;

error = CloseComponent(fileImporter);
DisposeHandle((Handle)imageInfo);
DisposeHandle(dataRef);

if (imageData != NULL) {
    DisposePtr(imageData);
    imageData = NULL;
}

if (offscreenBuffer != NULL) {
    DisposeGWorld(offscreenBuffer);
    offscreenBuffer = NULL;
}
```

`DisposeGWorld()` is a QuickDraw function, which was deprecated in Mac OS X 10.4. You will get a warning when you build your project.

Conclusion

I included some sample code that draws a texture on the screen. The most interesting code is in the `GameTexture.cpp` file. To learn more about OpenGL texture mapping, go to the OpenGL site. It includes online versions of the *OpenGL Programming Guide* (the red book) and the *OpenGL Reference Manual* (the blue book). Another good place for OpenGL information is the NeHe Productions website. It has a series of OpenGL tutorials that many people have found helpful.