Mac Pac

Game Programming

Assignment 1

Name: Stuart Bryson
Student #: 98082365

MacPac was written for assignment 1 of Game Programming at UTS. It is capable of reading COLLADA files and viewing them in an OpenGL window. The model can then be interacted with using the mouse, keyboard. The model can also be viewed using a demo mode which animates the camera along a spline.

Features

Model, View, Controller

The game is implemented using the “model, view, controller” paradigm. This is particularly useful in a 3D modelling or a game context. We are able to manipulate the scene individually from the view and visa versa. We could have many different views of the one scene with little effort. The controller holds the scene and view together. The controller is also responsible for hooking the GUI into the game.

COLLADA loader

The COLLADA loader is where most of my time on this assignment was spent.

COLLADA is a fantastic new 3D format. It has been developed as an open project led by the Research and Development team at Sony Computer Entertainment America. COLLADA is based on XML with a strong focus on element reuse. It is able to specify a scene or even an entire world in 3D. It handles a 3D scene heirachy/graph, vertices, normals, texture coordinates, skinning and weights, physics objects, and more. It also has the ability to specify shading language programs for objects.

Because of the complexity of this format I have spent many hours implementing the loader. The loader currently supports vertices, normals and UVs. The most complex part of this is triangulating the streams while ensuring that we have the correct combination of vertex, normal and UV values for each vertex index so we can send the streams to the video card. This is quite a difficult task. For more information on streams, see below.

Because of the extreme flexibility of COLLADA, choosing this format should not limit me in anyway when extending the functionality of my game in the future.

Streams

Streams are used to store verts, normals, uv's etc. The streams are then sent to the video card with glDrawElements. This is faster that using the glBegin and glEnd functions to

draw.

Each stream has a type. There is a position stream, a normal stream, uv streams and more. There is also an index stream which index's the other streams in order to create a face. I index the streams as triangles. While OpenGL (given that it is hardware accelerated) is fast at triangulating quads and n-gons, it is still more efficient for the triangulation of models to happen off-line. For this reason, my object loaders will triangulate all the polygons using a simple flower triangulation method.

The disadvantage of using streams is the potential for extra memory usage. This is due to the limitation of only having one index stream. For example, if you were to store 2 separate vertices which both have the same normal, you could not share that normal in memory. Eg:
Vertices: {1,1,1}{2,2,2}
Normal: {1,0,0},{1,0,0}
Notice the normals are the same however they must be duplicated as the vertices differ. It is generally felt in the industry that this small memory overhead is worth the gain in drawing efficiency.

Splines

The splines I implemented for the demo camera mode are not traditional bezier splines. Instead I am using a technique for splines developed by Thomas Lowe and described in “Game Programming Gems 4”. He summarises that all splines can be described using the t*H*G equation. The splines I have implemented are the Rounded Nonuniform Splines.

Timers

The game rendering and the camera interpolation are both implemented using timers. Currently you can specify the timer interval for rendering in the preferences. This is handy to see how the game will behave at a lower frame rate.

The camera is also running on a separate time. The camera position will update irrespective of the rendering. To illustrate this, turn on demo mode and then set the rendering interval to be longer (eg. 0.1). This can be done in the MacPac->Preferences menu. You can see that the camera is still receiving its update commands however the scene is rendering at a much lower frame rate. Another way to illustrate this is to turn on demo mode and then deactivate the render timer from the View menu. Then manually choose View->Refresh View.

In future development, I will update the scene entities in a similar fashion to the camera. It is important to update the scene, particularly physics etc, at a fixed rate and the rendering can render faster or slower with no effect on the scene entity updates.

Property Lists

Built into MacOSX is the NSDefaults database. Briefly, this is a standard way to save application preferences on a per user, computer or network basis. I use user preferences to store the application settings. Currently the only setting used is the

renderTimerInterval. These preferences are then saved into a property list that lives in the users home directory. Eg: /Users/sbryson/Library/Preferences/com.bryson.MacPac.plist

You can change these preferences either when the program is running from the MacPac->Preferences menu or editing the property list in the Property List Editor application.

I also use property lists to define my splines. Each property list contains an array of Vectors that specify the position of each control point. You can also edit these splines in the property list editor.

Graphical User Interface

Lastly a GUI has been set up with a standard, user resizeable window. You can disable the render timer and refresh the view manually. You also use the GUI to load scene files and camera splines.

At this time, the Fullscreen mode does not work.

Objective-C and C++

A lot of time has been spent on learning the Objective-C language that is used in MacOSX Cocoa programming. This is a very clean and elegant language.

Unfortunately, apart from Apple (or NEXTStep originally), not many people have adopted this language. For this reason, many libraries and utilities are not available in the native language.

Fortunately there is a compiler/language which bridges the two called Objective-C++. There are many limitations however to this language. The bridging of the two languages is not seamless and there are many things that can't be done. It does however allow libraries to be used. In my code for example I have used the tinyXML library, which is written in C++, to parse the COLLADA files.

Further to this, however, is the issue my game has with loading PLY files. I have used the RPly library written by Diego Nehab of Princeton University. This library uses static C callbacks as the PLY file is parsed. Unfortunately, the Objective-C objects that I wish to populate from within these callbacks, are left in a inconsistent state. It seems this is primarily a memory issue as I try to share objects from different languages. The logic of the code is fine. I have even been able to load a cube. However, more often than not, my PLY loader crashes. If I had more time I would further investigate my options in sharing Objective-C objects.