Building a Game – part 2 (3D Graphics)

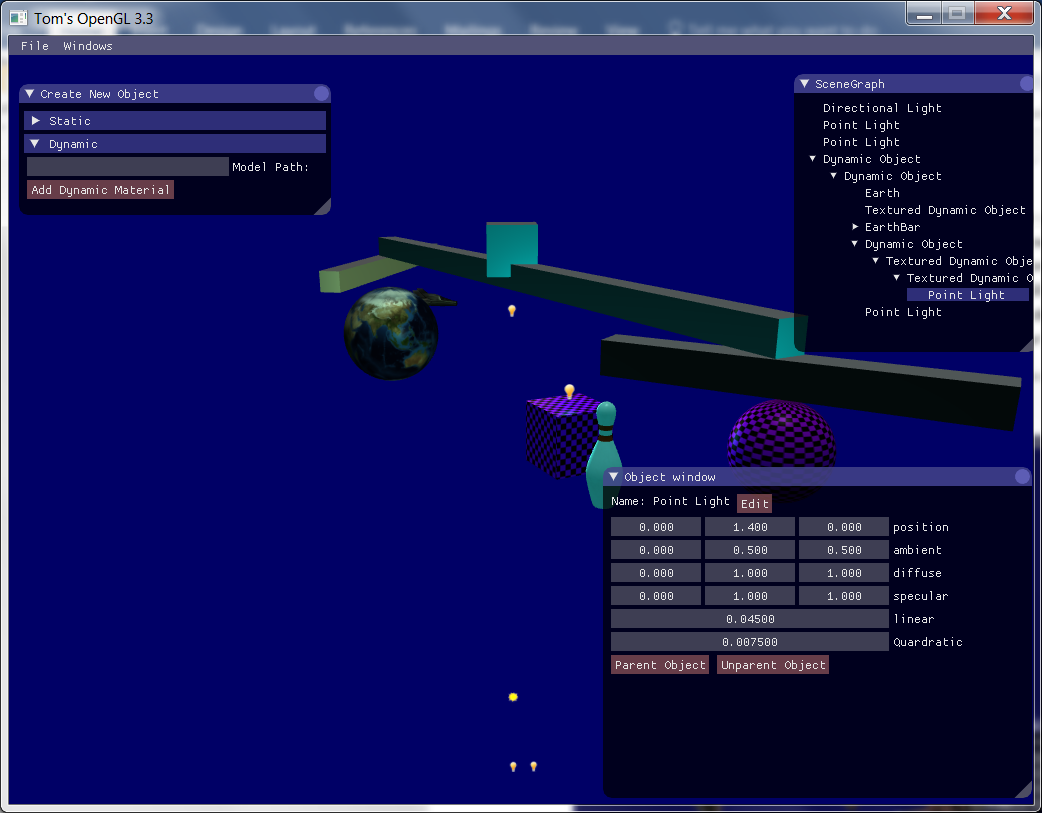
# Game concept

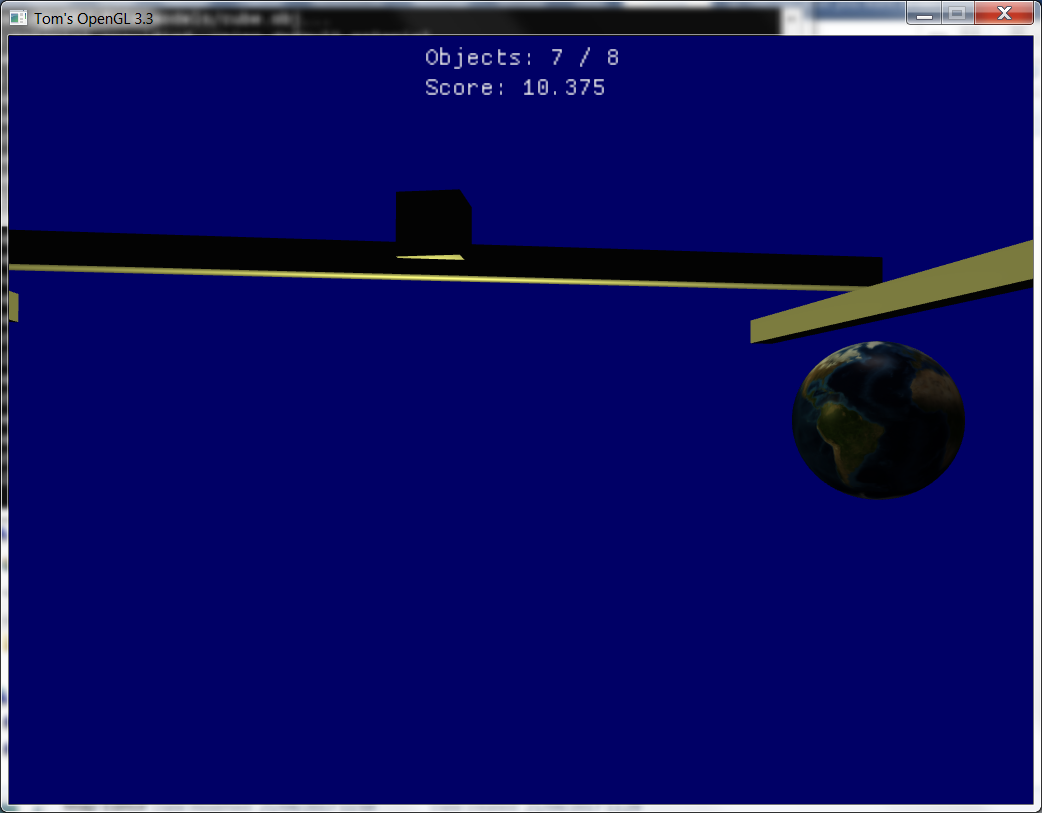
For this assessment, I started by creating an environment in which I could experiment with OpenGL 3.3 and ImGui. It started by only contained the basics such as a single directional light, one cube object and the ability to move and rotate it in 3 dimensions. After using the test environment for a while I decided that I was going to create a sandbox in which the user can create and share their own levels for their own and other’s enjoyment.

In my opinion the most enjoyable part of this project was creating the Map Editor, because being able to build a system of parented objects and point lights is extremely satisfying and rewarding. The Map Editor has been built so that you can create almost anything within the limits of 4 point lights.

The gameplay side of the Map Editor is testing the level that has been created. The objective of the game is to fly around the map and collect every object placed in the world, excluding directional and point lights. Once you enter the game testing state you’re given 20 seconds to collect the first object, if you collect an object within this time it will be destroyed and any attached children will be moved to a higher branch in the scene graph. Each time you collect an object you will be awarded an extra 10 seconds to collect the next object until either all objects have been collected, or your time has run out.

The final goal of the map testing is to find the best path to take to collect all objects and to achieve the highest score.

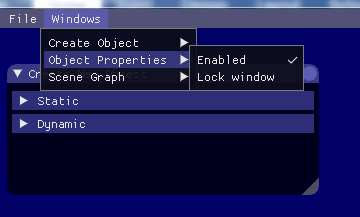
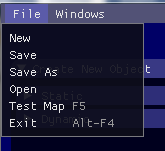




# Game controls

## Map editor

When controlling the camera use the mouse to direct it.

When your mouse is free, use it to select and manipulate objects with the GUI menus presented.

Use the top left file menu to create new maps, save your currently opened map, or open previously saved maps.

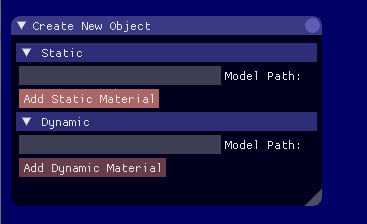
Use the top left window menu to show/hide and lock the GUI windows.

**Tip: Try some example maps such as “Example3”, “Lights”, “Velocity” and “Bowling”**

To create a new object, open the ‘create new object’ menu, and extend one of the two headers.

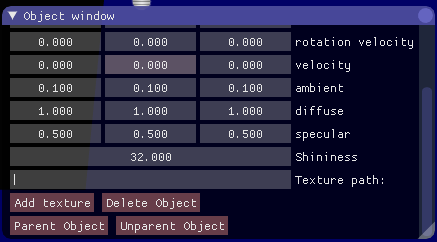
To load a custom .obj file, you must first add it to the “models/” file path within the installed directory. Then type in the name of the model, not including the file extension into the ‘model path’ box, and create the object. If a cube is created instead of the intended model, either your file is in the wrong directory, or in the wrong format (e.g. square faces, without normal included).

Some pre-existing models include “cube” and “sphere”



### Object properties

To edit object properties, you must first select an object from either the scene graph or by selecting it within the 3D environment. Any values can then be edited in the object properties window, initially located in the bottom right of the application.



To add a texture to a previously created object: select the object, scroll down to the bottom of the window, and select the texture path text field. Enter the textures file path excluding “materials/textures/”, but including the file extension. Note that only .tga and .bmp files are supported.

When ready click “add texture” and the model will be painted with either the desired texture or pink and black squares. A pink and black texture indicates that your texture file was in the wrong directory or falsely named.

To load a specular texture in combination with the diffuse texture, place it in the same file directory and add “\_specular” between the file name of the diffuse and it’s file extension.

To parent an object to another object select the desired child and press the “parent object” button. Then either select the parent in the 3D environment, or in the scene graph window, and the parenting will be completed.

**Note: to select a parent object in the scene graph it must first be closed.**

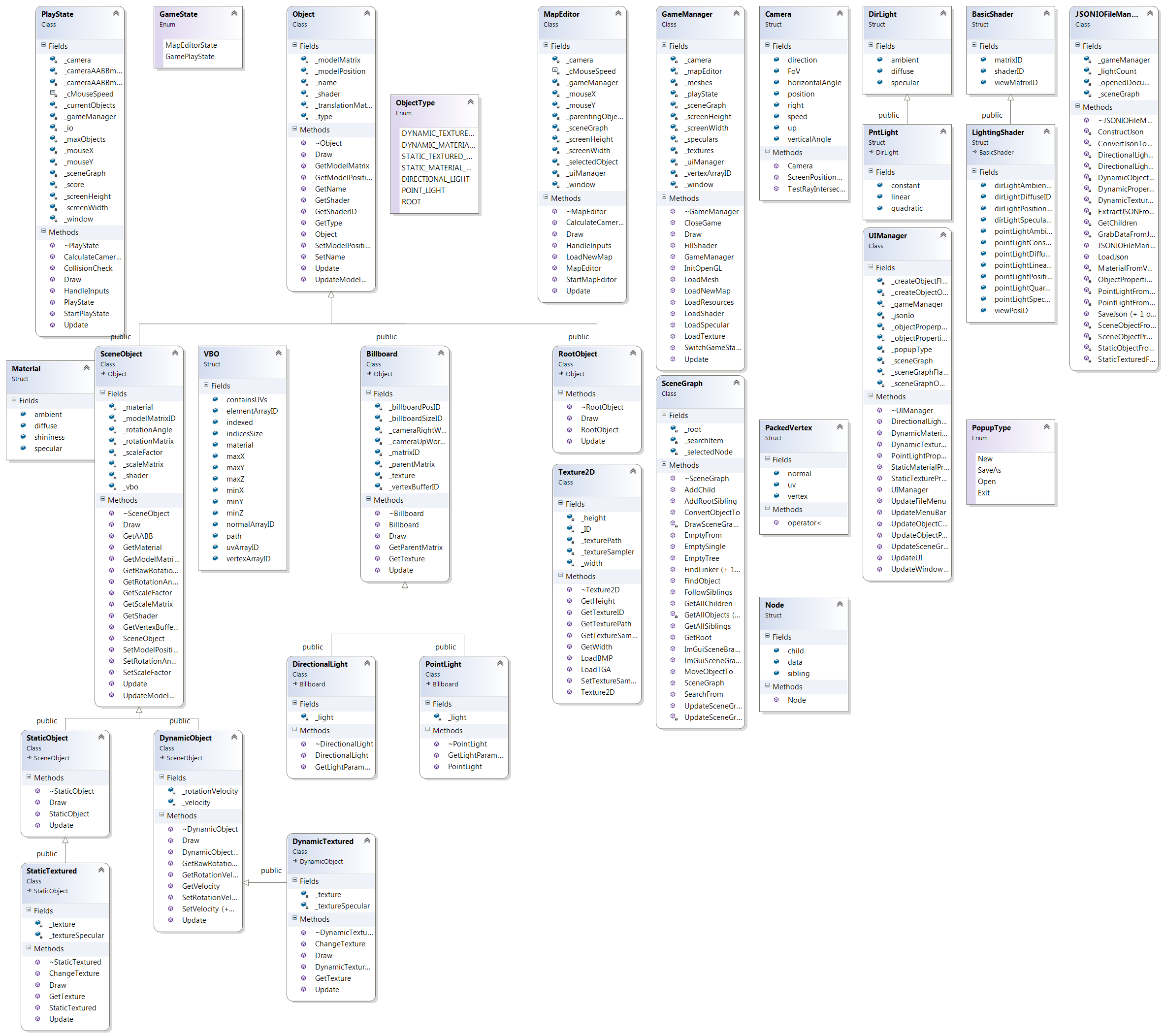
|  |  |
| --- | --- |
| **Key** | **Description** |
| W | Used to move in the direction the camera is facing |
| A | Used to move to the left of the direction the camera is facing |
| D | Used to move to the right of the direction the camera is facing |
| S | Used to move in the opposite direction the camera is facing |
| C | Used to toggle whether the mouse controls the camera |
| Alt-F4 | Used to quickly exit the application without saving |
| F5 | Used to save any changes and test the currently loaded map |
| Mouse 1 | Used to select the currently hovered object |
| Delete | Used to delete the currently selected object |

## Map testing

Use the mouse to alter the camera’s direction.

|  |  |
| --- | --- |
| **Key** | **Description** |
| W | Used to move in the direction the camera is facing (forwards) |
| A | Used to move to the left of the direction the camera is facing |
| D | Used to move to the right of the direction the camera is facing |
| S | Used to move in the opposite direction the camera is facing (backwards) |
| Escape | Used to exit map testing and return to the map editor |

# UML



In my program, I’ve used a scene graph to store all my game objects. This required me to create a Node data structure. Each node contains a pointer to the game object and two pointers to other Nodes. One of these points to a child of the node, and the other to a sibling.

PackedVertex is used in the loader methods to identify if any previously loaded indices have been duplicated. If this has occurred, it will remove the duplicates and reference the original instead.

VBO and Material are both structures used to hold a model’s properties. VBO holds refences to the model’s physical properties such as vertex data, normals and UVs. On the other hand, Materials holds direct values and is uniquely stored for each scene object so that they can be coloured separately.

LightingShader and BasicShader are structures used to store references to shader uniforms so that they don’t have to be relocated during every update.

PntLight and DirLight hold the direct values that are parsed to all LightingShader’s references during every update.

Camera is a structure holding the camera vectors, used to calculate the camera’s facing direction.

# Critical reflection

Looking at the result of the project I feel that there a few things I wanted to add to make it just a bit more user friendly. The main feature I thought would increase user enjoyment would be a copy/paste system, allowing the user to take a whole branch of objects and then paste them as a child of any selectable objects. The primary reason for not including this feature was time constraints. It wouldn’t be too complicated to add, but there were more important aspects such as gameplay that needed to be implemented.

Another feature I feel would have improved the project is the ability to undo any actions taken in the map editor. This could be done by keeping a stack of actions and then popping off the last action when undoing. I did not manage to implement this for the same reason as copy and pasting as I considered other features to be more important at this stage.

One of my personal achievements in this project is the scene graph class, it’s the first time I’ve used recursion in a way that didn’t seem inappropriate, and I feel it’s quite effective at searching and running through the tree. Building an effective scene graph was my first step towards getting my parenting system working properly, and I am extremely pleased with the result. However, my biggest enjoyment was creating the ImGui window that displayed the scene graph and getting this to successfully return a selected object. This meant that I could then introduce parenting.

The ability to load and save maps was something I thought would be extremely difficult until I found the JSON file format. After this I found the RapidJSON library, which was immensely helpful and easy to use. I could now save maps in a small file format, and easily debug the inputs and outputs thanks to visual studios built in JSON viewer.

I created quite a few structures to use in my application. These include VBO, which contained all the pointers to the mesh’s data buffers. VBO also contained a pointer to the mesh’s material, either grabbed from a .mtl file or loaded from default values. The VBO structure had to be optimized to facilitate its constant use.

During development, I ran into new ways of tackling problems that I otherwise wouldn’t have thought of. However, I found that as I progressed, some of the solutions interfered with other functions, causing unforeseen problems which I had to work around. On reflection, I blame my lack of planning for such a large project. If I had created a UML diagram from the start I could have foreseen some of these problems from the beginning, and saved time and effort trying to work around them.

Overall, I’m pleased with the result of this project as I’ve learnt a considerable number of new techniques and improved my problem-solving abilities. I also now understand why larger projects need more planning to produce successful outcomes.