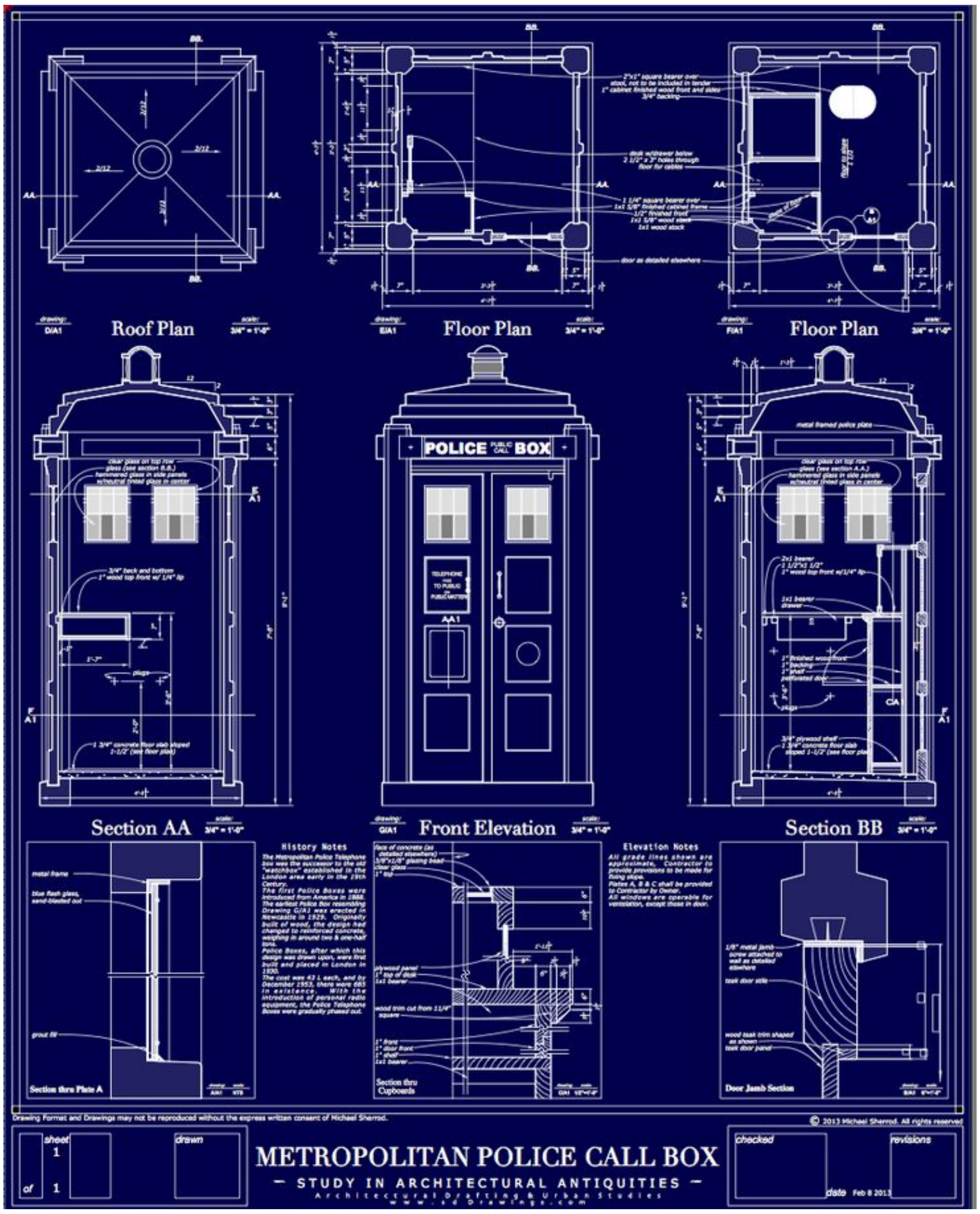
How To Build A Type 40 TARDIS 101 *(in OpenGL)* User Construction Manual



See Figure 1

Introduction

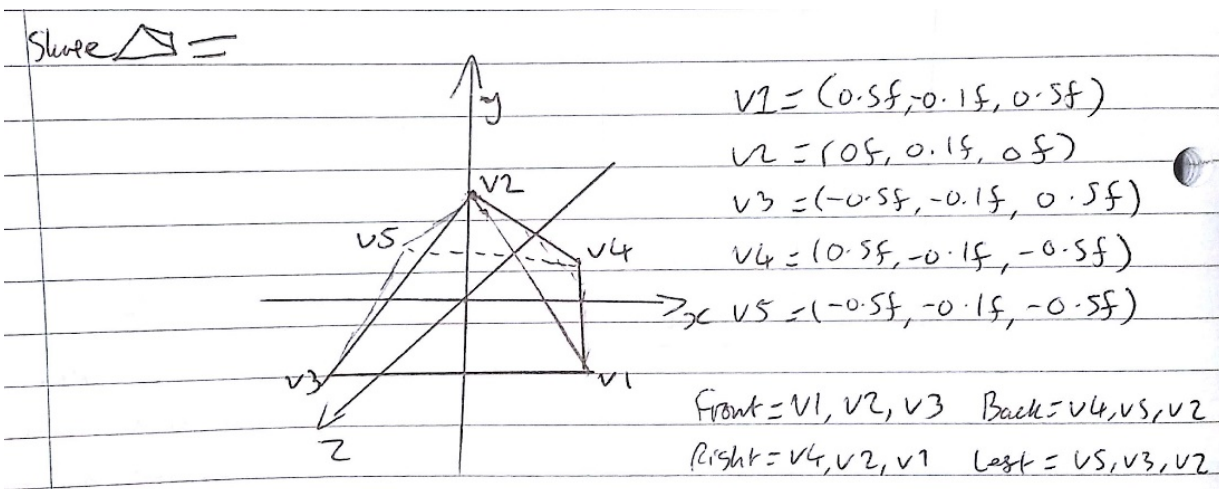
"I had a dream."

It suddenly hit me halfway through one lecture that a TARDIS is a machine. A very good machine at that. It evaporates into dust only to appear next to you seconds later. Genius! To design something like that in OpenGL would, although take a lot of time, effort and patience, look extremely awesome. So halfway through my lecture notes, I drew a box with a triangle on top and from that one image I feel like this has been an amazing journey. To present to you what I have today from that simple drawing a few weeks ago is exactly why I wanted to get into this industry. But enough of the background, let's get to the good stuff!

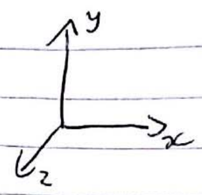
Pre-code planning

I instantly began to draw exactly what I wanted my TARDIS to look like. I must have drawn 10 pages of drawings before getting the exact right coordinates, direction of vertices and so on, but the finished product can be seen on the next page. As shown in my drawings, I already knew I was going to need to draw 3 shapes: A rectangle, a cube and a square pyramid. With these 3 shapes I should be able to draw the TARDIS and any backgrounds I liked since the plane can be textured and scaled to a background size.

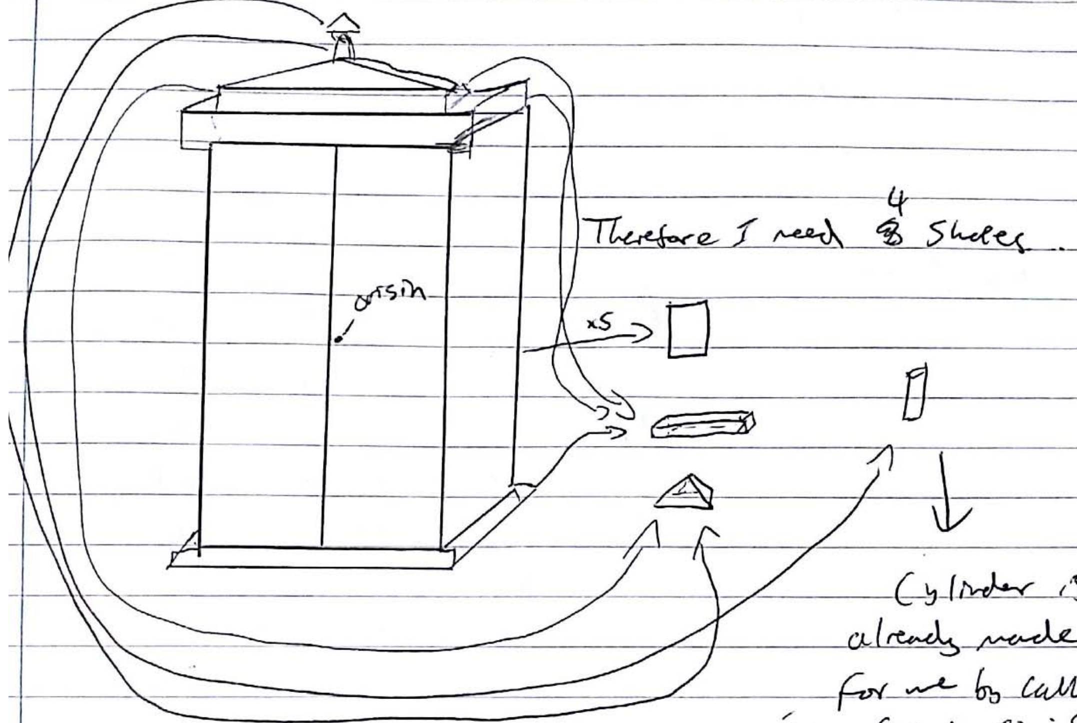
After understanding the coordinates of each shape, I then needed to know how they would be 'submitted' in OpenGL. This is simple enough when you realise that OpenGL works anticlockwise. The exact way I did it can be seen in my drawings below, but each set of coordinates are submitted in the right order so the shape can appear and fill correctly. My complex square pyramid object can be seen below:


Shape △ =
y
x
z
V1
V2
V3
V4
V5
V1 = (0.5f, -0.1f, 0.5f)
V2 = (0f, 0.1f, 0f)
V3 = (-0.5f, -0.1f, 0.5f)
V4 = (0.5f, -0.1f, -0.5f)
V5 = (-0.5f, -0.1f, -0.5f)
Front = V1, V2, V3 Back = V4, V5, V2
Right = V4, V2, V1 Left = V5, V3, V2

George's TARDIS Design



Full Model:

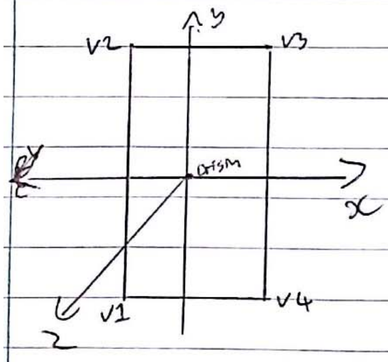


Therefore I need ~~8~~ 4 Shapes..

Cylinder is already made for me by calling 'new Cylinder();'

Shape =

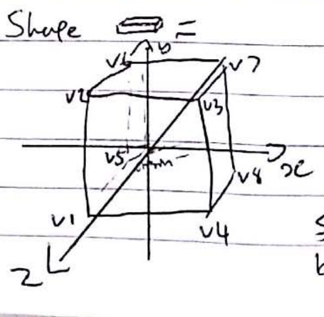
Therefore, 3 Shapes need to be made:



v1 = (-0.5f, -0.8f, 0f)
v2 = (-0.5f, ~~#~~0.8f, 0f)
v3 = (0.5f, 0.8f, 0f)
v4 = (0.5f, -0.8f, 0f)

AntiClockwise means Submitting like;
v4, v3, v2, v1

Shape =

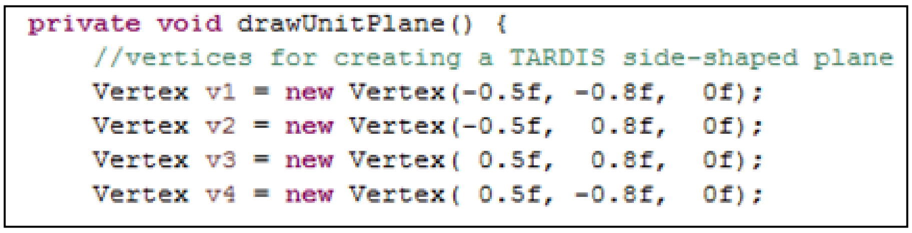


v1 = (-0.5f, -0.5f, 0.5f) v2 = (-0.5f, 0.5f, 0.5f)
v3 = (0.5f, 0.5f, 0.5f) v4 = (0.5f, -0.5f, 0.5f)
v5 = (-0.5f, -0.5f, -0.5f) v6 = (-0.5f, 0.5f, -0.5f)
v7 = (0.5f, 0.5f, -0.5f) v8 = (0.5f, -0.5f, -0.5f)

front = v4, v3, v2, v1 right = v8, v7, v3, v4 top = v7, v6, v2, v3
back = v8, v5, v6, v7 left = v5, v1, v2, v6 bottom = v4, v1, v5, v8

It all starts with a rectangle

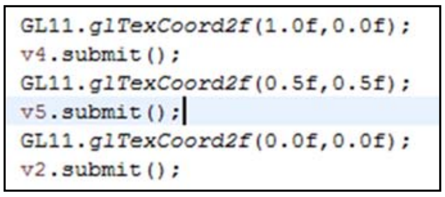
And yes, that is rectangle, not cuboid! My initial shape that I designed is in fact a 2D shape which has Z coordinates set to 0. I had planned this to be a cuboid and started off as the TARDIS being that shape, but quickly learnt I would run into problems. One such problem would be that I would only be able to apply one texture to the cube, in which I needed 2 (well, 3, but I'll come to that) because the front of the TARDIS has a different texture to the other sides. Drawing one 'plane' and translating and rotating it into the right place proved easier. In the code, this is therefore named "drawUnitPlane" as that's all it is. Each vertex is created in respect to the origin, which centres the plane on this point. The X coordinate difference is 1 between the two points and the y is 1.6. This explains how it has its rectangular-like shape and how it's not just a square. I could have just built a square and scaled it, but since I have no use for a square at all in this program, this really saves me time in scaling at a later point for each instance I use it. Speaking of that, I actually use this object 9 times; 3 for the right, left and back sides, 1 for the inside of the TARDIS, 2 for the front (each door) and 3 as the background and floor planes, since these are easily scaled, rotated, translated and textured.



```
private void drawUnitPlane() {
    //vertices for creating a TARDIS side-shaped plane
    Vertex v1 = new Vertex(-0.5f, -0.8f,  0f);
    Vertex v2 = new Vertex(-0.5f,  0.8f,  0f);
    Vertex v3 = new Vertex( 0.5f,  0.8f,  0f);
    Vertex v4 = new Vertex( 0.5f, -0.8f,  0f);
```

Complexity takes hold

After being able to draw a rectangle in OpenGL, I quickly shifted what I had learnt into more complex objects. As shown in my drawings, I needed a cube and a square pyramid, both of which followed the exact same principles which were outlined by my rectangle. I decided that the best way to create the pyramid was to set the value of y to 0.1 or -0.1, which means the top of the triangle is being drawn very narrowly. This helps me because I use this object as is when it comes to building the TARDIS roof and therefore I don't need to go into a ridiculously low scaling factor whenever I call it. The only time I need to scale it higher is when it becomes the object on top of the TARDIS, which isn't too hard to do. Because I am using textures for all of my objects other than the cylinder, I also needed to make sure I align the texture correctly. With textures, the bottom left coordinate is (0,0) and the top right coordinate is (1,1), so aligning these with each submit value wasn't difficult, as shown to the right.



```
GL11.glTexCoord2f(1.0f,0.0f);
v4.submit();
GL11.glTexCoord2f(0.5f,0.5f);
v5.submit();
GL11.glTexCoord2f(0.0f,0.0f);
v2.submit();
```

Setting up the scene and initial lighting

After creating all of the vertices correctly, in order for it to actually appear, each shape needs to be added to a glNewList function. After doing this I was able to call each shape, but without lighting nothing would appear. I therefore set the global ambient to an appropriate amount and colour so I could get working on where everything needed to be.

Getting a TARDIS to materialise for the first time

Once I had everything in lists and they were calling correctly and appearing on screen, I could begin the process of constructing a TARDIS. The bulk of the detail for this process can be seen in my scene graph or in the comments in the code, but simply put, everything is drawn around tardisMoveX and tardisMoveZ, which are defined at any time in the code. This means that moving the TARDIS is as easy as changing two numbers. At no point did I need to move it along the Y axis, but in the future the code could easily be refactored to support this. Once everything was set up, there I got a 3D untextured TARDIS, which looked pretty impressive.

Actually getting the TARDIS to move ("fly")

The way it moves in the first scene is basically formed around the java.util.Random package, which is why this is imported into my program. I created a method called randInt using this class to generate a randomNum variable which can be set to anything between a minimum and maximum number. I can then use this variable in another method called moveTardis which allows me to set the tardisMoveX variable to any random number between -3 and 3, unless it's itself in which case a while loop finds a variable which is different. This essentially means that on that first scene, the TARDIS will materialise at any random spot on the x axis which is viewable to the user, which is very cool!

Playing around with textures and more advanced lighting

Assigning textures was easy peasy. Even for the most complicated parts such as assigning it to a triangle, since all of my textures are of a 2 to the power of n format, OpenGL can automatically stretch it to its perfect size. All of my textures can be found within the 'textures' folder and they all seem to play nicely with how I have set them up, so they must be right.

The textures were easy.

The difficult bit was setting up appropriate lighting. My scene only has one light which is the cylinder on top of the TARDIS as the rest of the shading is done via the textures I'm using. It took me a good few hours to figure out what was happening, but I eventually realised the Emission light had to be applied at the top of the method call, not once it had been moved and rotated, or it would be spinning hopelessly around behind the animation in vortexMode. Luckily it works now. Phew.

I also realised that global ambience is no longer needed in my scene, which led me to turning off the global ambience at the top of my code which is commented.

It's not all doom and gloom (pun intended)

Although I don't specifically use any other form of lighting other than Emission, the way my TARDIS fades in and out is a form of using colour to change an object. It's called blending. Having really wanted to actually get a fade in and fade out working for the TARDIS, I researched into if this would be possible with OpenGL1.1. The answer is yes, and it works very very well. Blending works in the exact same way to lighting with the fact you need to assign your object a colour. Except this colour only gets applied when your objects Alpha is less than one. The perfect method for a straight fade out then fade in, using white as the interim colour. That' s exactly what you see when you press Space in my program. It fades out by converting the texture colour to white and subtracting the alpha slowly. I can even set the colour of the fade out between the white it is right now and black, which makes the blend look more transparent by losing all the textures immediately. This can be changed via the tardisFadeOutColour command, which is quite frankly awesome.

You spin me right round...

After getting the blending nailed, the final step was to create the second scene. Although it could be argued this scene is derived from the borgCube package that is part of this workspace, I ultimately took it a step further. Since the TARDIS has two global variables, tardisMoveX and tardisMoveZ, I could easily and quickly create two individual rotations which are both relative to each other. The TARDIS spins on its X axis whilst ALSO spinning on the Z axis around the origin. That means every object of the TARDIS is spinning on two different axis at the same time and pace. Now this is the best bit. If you press 'd' on the keyboard whilst in vortex mode, the TARDIS doors open. Which is a rotation (and a translation technically). A rotation whilst rotating whilst rotating. To add even more animation to this, the background is rotating anti clockwise as well!

Now that's a lot of rotation.

Known bugs :(

Unfortunately there are a few known bugs with the program that I couldn't quite stamp out in time for release. In terms of logic the program is pretty perfect; each variable and if statement work exactly as I intended them to do. Everything written that hasn't been commented out means something to something else.

I blame OpenGL.

I think I've gotten too used to the beautiful animation programs such as Adobe Flash where you can just delay frames freely to your hearts content. Basically, when it comes to anything that doesn't involve an animation, I cheat. Luckily this whittled down to only two keyboard commands: D and L.

I call something called "TimeUnit.*MILLISECONDS*.sleep(time)" which does exactly what it says. It sleeps the ENTIRE animation for that duration. Why do I call this? Because If I don't then the user will be able to open and close the doors of the TARDIS or turn on and off the light hundreds of times before they can even lift their finger off the button. This fixes that. It's cheap, it's nasty, but it does the job. Maybe in the future I'll learn a much better process to doing this, but for now it works and it's all I've got.

Other than that though, that's it! A lot of clever coding and many if statements later (all of which are in my documentation), you build your very own TARDIS scene. Good luck.

Animation References

Black, T. (2012). *tardisTextureMap.* Retrieved from Dormant Blog: http://dormantmind.com/wp-content/uploads/2012/05/tardisTextureMap.png

Google Maps. (2015). Retrieved from http://maps.google.co.uk

Figure 1. hdimagelib.com. (2015). *Tardis blueprint images.* Retrieved from http://hdimagelib.com/tardis+blueprints?image=365467188

Master-Matte-16. (2013). *TARDIS Console Room, Matt Smith's 2nd Tardis.* Retrieved from http://www.deviantart.com/art/TARDIS-Console-Room-Matt-Smith-s-2nd-Tardis-419632011