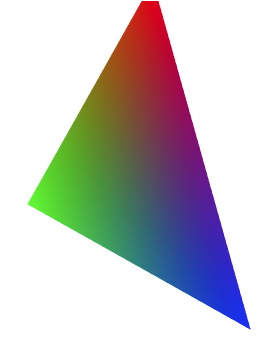
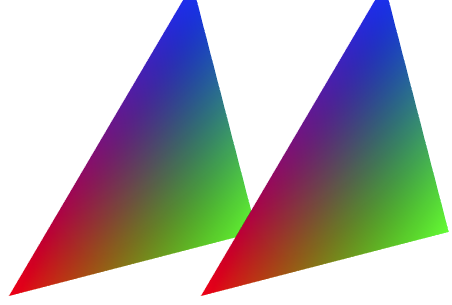
Week 1

Task 1 involved rotating a triangle around its centroid. This was achieved by setting a float uniform value (uRotation) to represent its’ current rotation and a vec2 uniform value (uPivot) to represent the local position of the centroid in the triangle. Vertices are first translated to the pivot, then rotated around the origin (using a matrix built in the shader), then translated back to their position (uPivot \* -1).

The program also needed to keep rendering frames rather than just rendering once at the start. To achieve this, I used window.requestAnimationFrame, which calls a function to update an animation (render in this case) at an appropriate time and passes it the time the application has been running for. This prevents rendering when the page can’t be seen, and essentially lets the browser control when frames should be rendered. Consistent movement is achieved by subtracting the previous time from the current time to create a deltaTime, and adjusting the rotation based on that delta time (fRotation += deltaTime \* speed;).

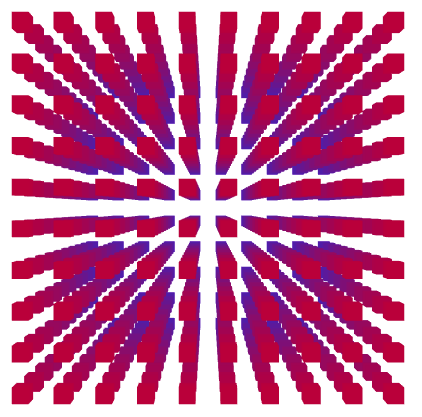


The achievement task was completed by adding an additional vec2 uniform (uPosition), setting it before 2 separate draw calls in the render function, and offsetting vertices with it after the rotation.

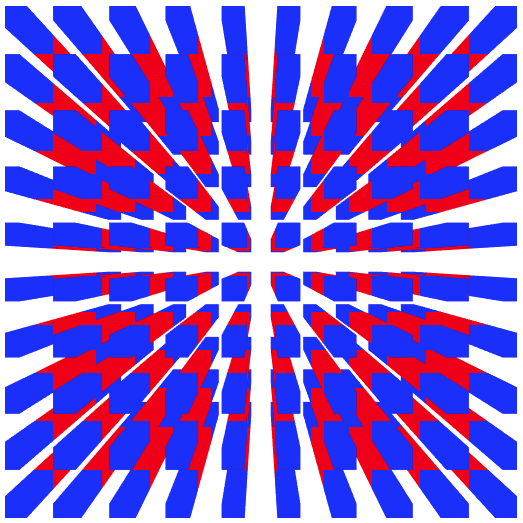


Week 2

Task 2 required only a single line change in the vertex shader to use the transformed Z position (depth into the screen) to determine the colour.



To complete the achievement task, I added a time uniform value (set from the render function) and a varying world position value (set from the vertex shader). In the fragment shader, the z value from the world position is shrank down to a 0..1 value, and another 0..1 value is calculated using the sin of the time uniform variable. The absolute value of the difference between the 2 is then calculated and scaled, then rounded to 0 or 1 which is used to interpolate between red and blue. To change the boxes shape, I removed the loop for the z position (so there’s only 1 layer on the Z plane), and changed the world matrix to scale and position the boxes.



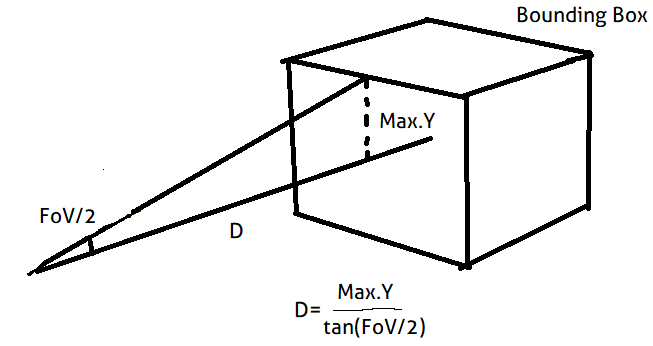
Week 3

I was a little confused by week 3’s task as after downloading the starting content from the interact, it appeared to already be done, so I decided to start with week 2’s content. I thought this was an opportune time to bring in a model class which I’d written for ITC363. There may be some extra functionality that isn’t currently used, but this gave me a basic structure meshes, mesh parts and shaders.

I then wrote a parser to download and parse an OBJ into the mesh class. Mesh parts are defined by material, so it’s relatively optimized but will ignore the part definitions in the OBJ file.

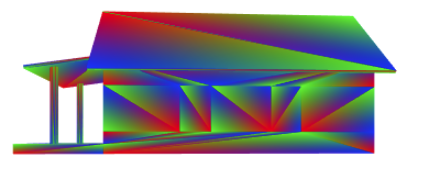
The mesh class supports hardware instancing and vertex normals, but neither are used yet.

The initial view was worked out by finding the bounding box for the mesh, and then finding the distance required to fit each axis inside the view frustum using trigonometry and using the largest one. The following shows how it was worked out for the Y/up axis:

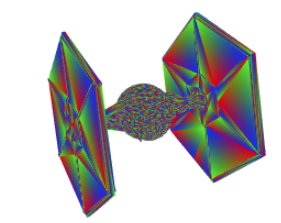
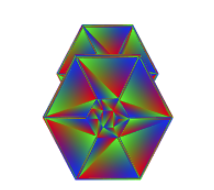


The camera supports movement with WASD and looking around with click/drag.

The result, after adding red/green or blue colours to each corner, is as expected:



I imported the Starwars.obj to complete the achievement:



Week 4

To import the textures, I used the method recommended by (Mozilla, 2018) (after converting them to PNG). I could have used the provided TGA loader or written my own (perhaps I was meant to do this?), but in reality, it seems like it makes more sense to use a format that the browser can parse internally rather than manually parsing one with JavaScript.

The colour mask was simple enough using the ‘discard;’ GLSL command. I used a pretty big epsilon to get rid of as much of the pink as possible, but some is still visible.



Alpha blending was more complicated. Because the house has multiple layers of partial transparency in the same mesh, triangles that render behind already rendered triangles would be occluded by depth testing:



While I didn’t find a good solution to this, I did find an article by Nvidia (Everitt, 2001) That described a method of using render targets to record depth and render the scene in multiple layers. While doing this is beyond the complexity I believe we’re expected to put in to this assignment, I implemented rendering in 2 layers defined by a camera facing plane at the centre of the building (by simply setting the near/far distance of the camera). The depth buffer is cleared after the back half of the house is drawn, then the front half is drawn. It still doesn’t work at all angles (because the plane isn’t perfectly representing what should be occluded) and is very case-specific, but it works better than pure alpha blending and from most angles shows what Nvidia’s method would look like if it were implemented.

The back and front layers:



The result (at a different angle to show both windows properly):



In reality I would send this mesh back to the artist and ask them to separate the windows if they need alpha blending.

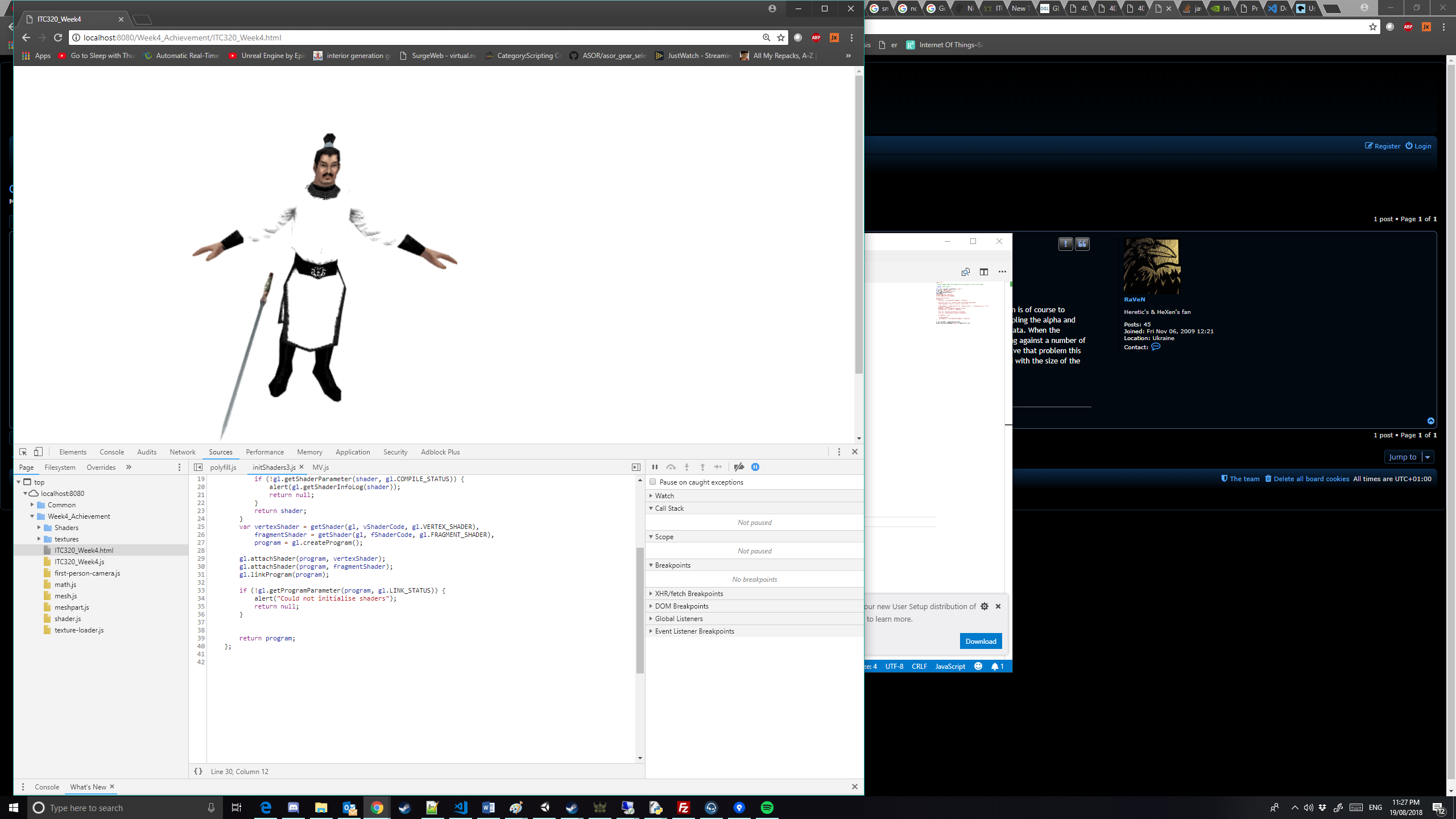
The final method I tried, and the one that I think looks best is alpha testing (discarding when alpha < 1). It still has quite ugly edges and no possibility of partial transparency, but it’s not as bad as the others as there’s nothing that could definitively be called an artefact.

The difficult part of the achievement task was separating the green from the rest of the texture (yellow/gold parts). The formula I ended up with for the ‘greenness’ interpolation value was (green – (blue + red) \* 0.5) ) ^ 2 \* 150. I’m not proud of having to work it out by fiddling, but it works:



The ‘greenness’ value being rendered directly:





# References

Everitt, C. (2001, May 5). *Nvidia*. Retrieved from Nvidia: https://www.nvidia.com/object/Interactive\_Order\_Transparency.html

Mozilla. (2018, Jun 29). *Using textures in WebGL*. Retrieved from MDN web docs: https://developer.mozilla.org/en-US/docs/Web/API/WebGL\_API/Tutorial/Using\_textures\_in\_WebGL