Graphics Programming with Shaders Report – Daryl Grant

1. Scene Overview

My scene consists of 2 quad planes, a mesh containing several controls points arranged into a cube and an imported model.

All objects in the scene demonstrate correctly transformed normals, lighting calculations and shadows.

The 2 quad planes both demonstrate height mapping, the first one transforming the plane into mountain terrain and the second giving the plane a water surface look.

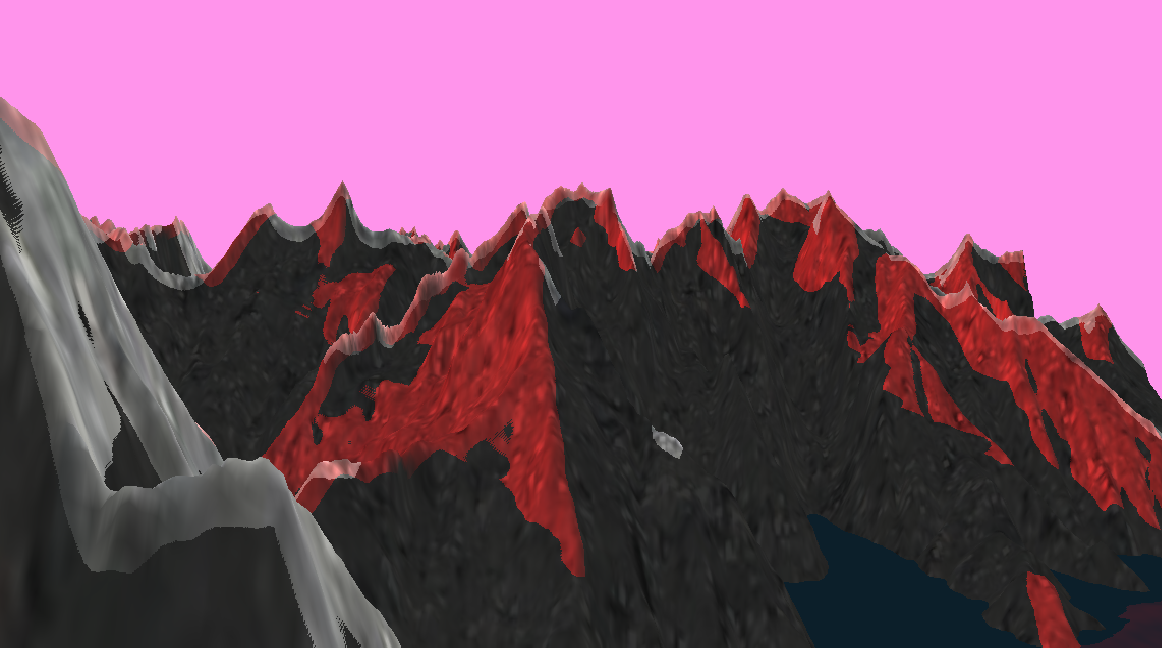
The control point mesh arranges many 2D meshes into a cube which are drawn so that they always face the camera. The meshes are then positioned under the water surface and their translations updated based on time to give them movement.

The imported model further shows that the calculations for multiple lights and the shadows they produce are correct.

Requirements

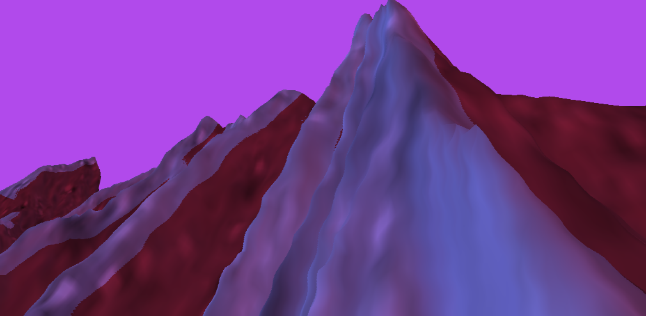
This section details how I have responded to each of the requirements in the Assessment Brief.

**Vertex Manipulation**



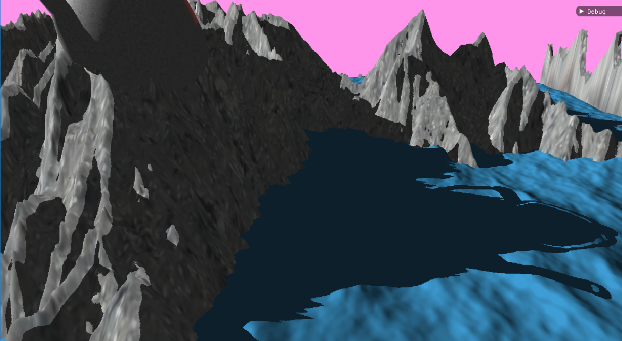
My scene demonstrates vertex manipulation by using height mapping to transform the 2 quad planes and give them the shapes of a mountain terrain and a water surface. A depth test is done on each of these transformed planes so their shadows can be calculated and normal maps are used so they can be correctly lit.

**Post Processing**



To show post processing, when the camera is below the surface of the water, the render texture that the final scene is rendered to is passed to a shader which manipulates each pixel in a sine wave pattern. This gives the whole scene an underwater effect, with everything appearing wavy.

**Lighting and Shadows**

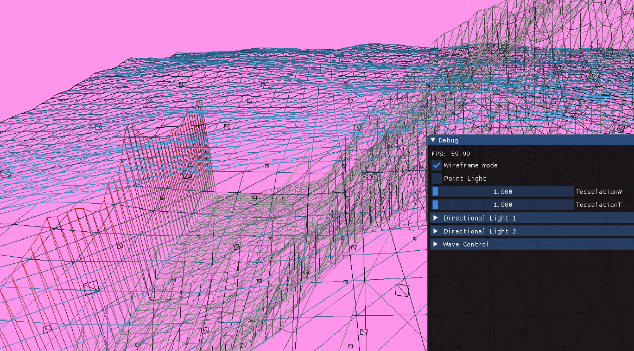


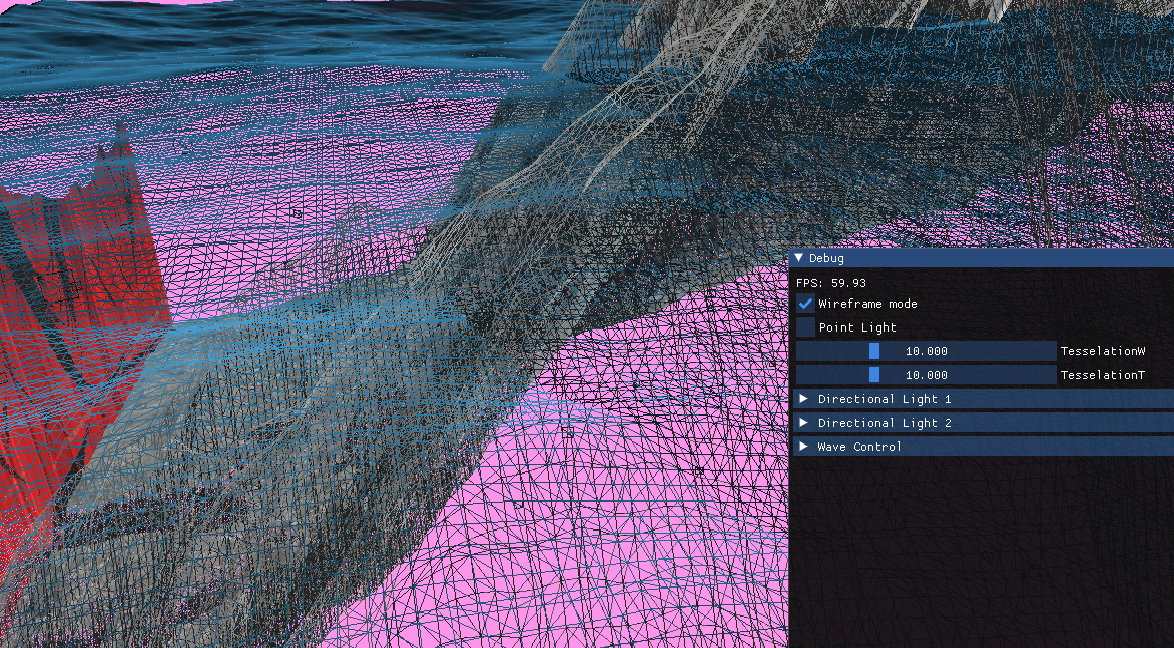
All objects in the scene are lit correctly by normals are calculated as well as transformed appropriately whenever any vertex manipulation happens. For the control point mesh the normals are calculated when each of the 2D meshes are created, these normals always point towards the camera. For the terrain and the water surface normal maps are used alongside the height maps to give each vertex the correct normal. The imported model has already calculated normals.

Each object in the scene is tested for depth from the viewport of any light it is lit by. Shadow mapping is then used to work out where the shadow should appear for that object.

The Lighting and shadows of all objects take into account both lights in the scene.

**Tessellation**





To show non-trivial tessellation, both height mapped objects in the scene, the water surface and the terrain, have a small amount of tessellation applied to them. The tessellation is fractional and its value can be decreased or increased using a slider in the UI.

The tessellated object both have their texture coordinates and normal updated after tessellation so they can be correctly textured and lit.

**Geometry Shader**

The geometry shader takes in a mesh of control points arranged into a cube and creates a 2D quad mesh at each of these points. The shader also takes in the camera position so that the quads can be drawn in a way that they always face the camera. These new quads are all given their own texture coordinates and a normal that always faces the camera in the XZ plane.

UI

UI controls in the scene include wireframe mode toggle, point light toggle, sliders to

control tessellation, the direction, position and colour of the lights and to control the height, speed and frequency of the wave used on the water.

Water and terrain have their own tessellation sliders and tessellation for each if fully controlled by that one value i.e. edges and inside tessellations amounts are all set equally.

Each directional light has a drop down menu which shows controls for their direction separately for X, Y and Z. The colour of these lights can also be changed.

The wave drop down menu shows controls for the speed, height and frequency of the water wave function. The height or any other factor of the terrain cannot be controlled through the UI.

1. Shaders

This section will go over and explain important calculations used, data passed and shader stages of the shader classes written.

Lighting and Shadows

Before any objects are rendered, they are first checked for depth from each of the lights in the scene. A depth map is created for each object and from the perspective of each light. Light view and perspective matrices are passed in to replace the camera matrices that are usually given. A depth value is then calculated by dividing the Z depth coordinate (Y coordinate in world space but the light is looking straight down in this case) by the W coordinate. The resulting map is later compared to other shadow maps to determine the shadows cast by each object.

The depth map creation for each object type is slightly different and must take into account any tessellation, vertex manipulation and created geometry before checking the depth.

Lighting for each object is done in the pixel shader stage. All height maps relevant to the object being lit are passed in and compared. If the pixel is out of range of all lights then no lighting is done and the sampled texture is returned. The maps are then checked individually if lighting is needed based on the depth from the light and all 3 light colours are added together and saturated to give the final colour of the pixel.

Height Mapping

The Height shader takes a plane made up from a set amount of quads and, after tessellating, samples a height map image and increases the y coordinate of each vertex based on the greyscale colour sampled. Since both height mapped objects in the scene are tessellated, height map calculations take place in the domain shader stage.

Normals for the newly transformed geometry are obtained by sampling a normal map. This normal map was generated from the height map by checking each pixel and all the pixel surrounding it to determine slope at each point. The normal map is sampled and the RGB colour given represents the normal value for that pixel in the x, y and z directions respectively.



For the water specifically the texture coordinate sampled is changed based on time and the texture set to wrap to make the water position move. The normal map sample must also be updated to match this.

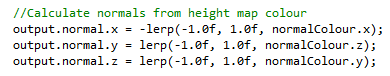
The height mapped water’s vertices are also manipulated using a sine wave over time, to give the water a wavier look.

Tessellation

Tessellation is applied to both height mapped objects in the scene to give them more detail and is done before the height mapping takes place.

The hull shader stage is passed 2 tessellation values, edge and inside. Each edge and both inside tessellation values for the quad to be tessellated could be set separately but keeping them to one value makes sure they don’t look mismatched. The tessellation amount passed in can be increased dynamically using the UI.

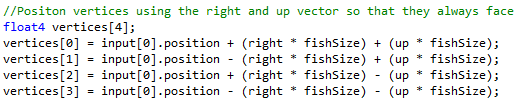
In the domain shader stage new vertex positions are calculated using linear interpolation on each of the patches passed to it by the hull shader. Normals are calculated by interpolating the X, Y and Z values sampled from the normal map to values between -1 and 1.



Billboarding

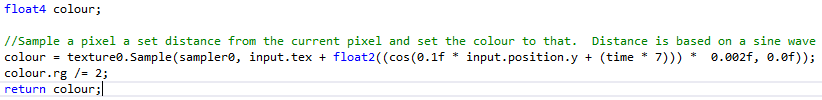
The Billboard shader is passed a cube mesh made up of controls points and turns each of these control points into a 2D quad that always faces the player.

The geometry shader stage is passed the position of the camera which is used to calculate how each mesh is drawn. To always face the camera an ‘up’ vector is set to (0, 1, 0) and the cross product of this and the vector point to the camera positon gives a ‘right’ vector. These up and right vectors can then be used to set the position of and append new vertices which will draw a quad made up of 2 triangles in a triangle strip.



Normals for these meshes are always pointing toward the camera in the XZ plane, but do not change in the Y direction.

Post Processing

Once all other techniques have been applied to all objects the scene is the rendered to a render texture. This texture is passed to the Underwater shader where, in the pixel shader stage, the render texture is sampled using a sine wave in the Y direction to sample a pixel a certain distance away from the current one based on the sine wave and current time, and return the colour of this pixel instead. This gives the texture a kind of wobble, and appearing underwater like. 

1. Critical Reflection

When comparing the final application to the week 7 proposal there are still a few things missing, and some that could be improved upon if I were to continue or do this project again.

The water surface was originally planned to use wave functions to give it a water look, either a ripple or a wave effect. In my application I decided to use height mapping to change the shape of the water. A sine wave is still passed through this height map but causes the normals to be incorrect. If I were to continue working on this I would improve the vertex manipulation of the water so that it also has correct normal even when changed with the sine wave.

In the proposal I said tessellation would be changed dynamically based in the distance of an object to the camera. I changed this to be dynamically tessellated via a slider. I think distance based tessellation would be an improvement and could be added if I were to work further on this application.

My scene contains 3 lights (when the point light is switched on) but only the 2 directional lights have working shadows. I could add more directional lights but the scene would be a lot better if more lights of different kinds were added. Multiple directional light do not look great together in the scene.

Currently the geometry shader only produces multiple quads to represent creatures in the sea. This could be improved by adding more vertices to make better shapes to look more like the fish or animals they are supposed to represent.

The edges of the terrain height map are calculated incorrect causing the edges to stick up. This has happened because I have incorrectly calculated the texture coordinates at the edge as well as setting the texture to wrap. I could change this by clamping the values but the water and the terrain use the same shader, and the water needs the wrapping to animate. A better solution would be to redo the quad plane mesh class and ensure texture coordinates stay within the 0 to 1 value.

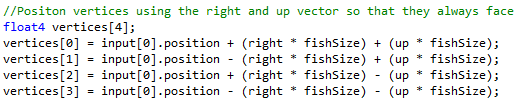
1. References

“jelly.png” Jellyfish texture

https://opengameart.org/forumtopic/looking-for-a-couple-2d-underwater-themed-sprites (Accessed: December 2018).

“rock.jpg” Rock texture for mountains

https://seamless-pixels.blogspot.com/p/free-seamless-ground-textures.html (Accessed: December 2018).



“terrainHeight.png” and “waveHeight.png” Height maps for terrain and water

Created at <https://terrain.party/>

(Accessed: November 2018).

“terrainNormal.png” and “waveNormal.png” Normal maps for terrain and water

Created at <http://cpetry.github.io/NormalMap-Online/>

(Accessed: November 2018).

“teapot.obj” Teapot model

Taken from lab examples