ASSESSMENT REPORT

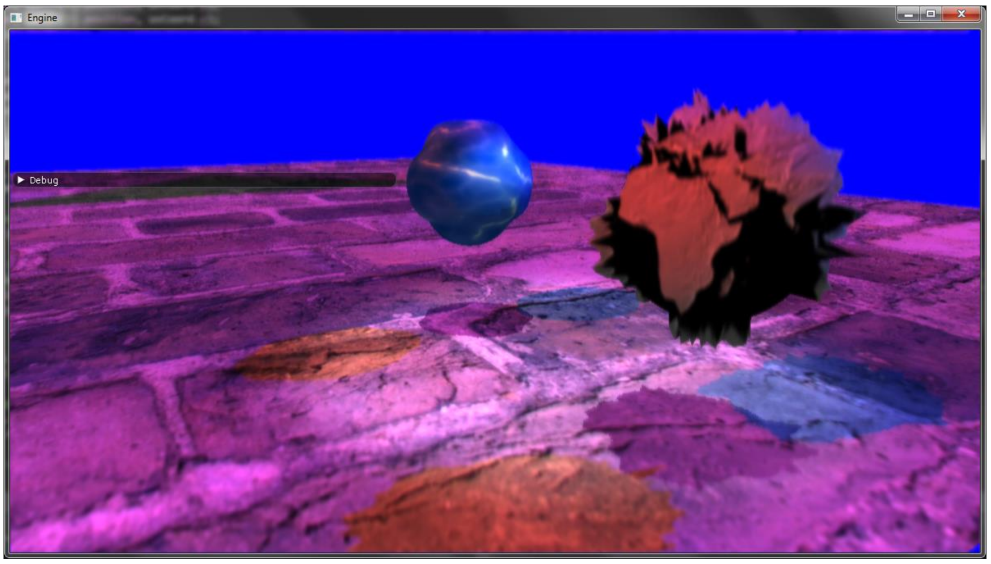
CMP301

FRASER BARKER
1600196
1600196@uad.ac.uk

Contents

Overview

Objects

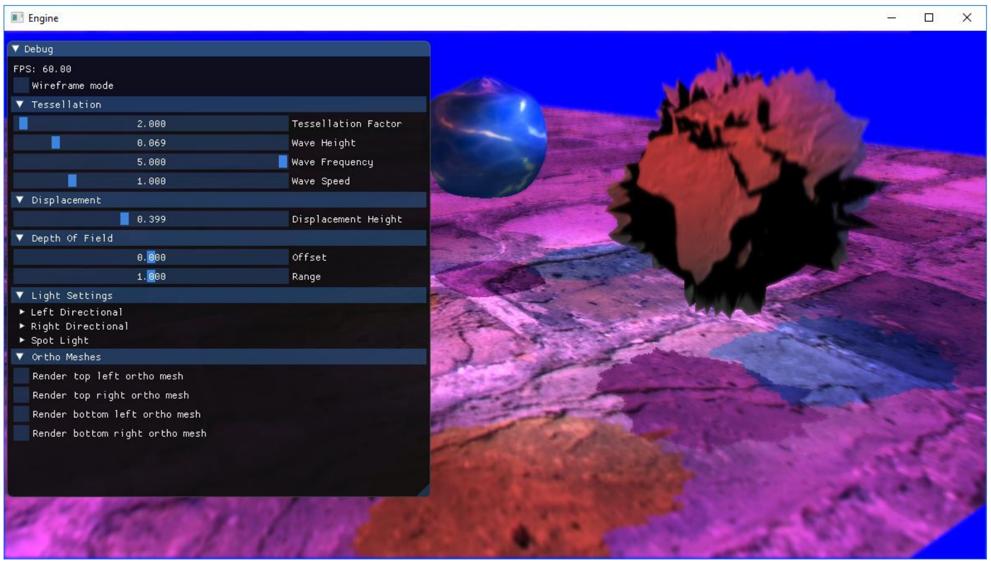


There are three objects within the scene. Two are spheres and the third is a plane. The two spheres both showcase vertex manipulation, correct lighting with shadows, and tessellation. Both spheres have their depth calculated from each light and the camera. This allows the spheres to have their shadows cast dynamically onto the plane below them. The plane is more basic and has a brick texture applied to it. It also has its depth calculated from each light and the camera, but its main use is to have the spheres shadows cast onto it.

Brief Response

Initially, a cohesive desert island scene was planned but changed due to being able to demonstrate the same process with much less hassle. Even straying from the proposal submitted in week seven although the project came close. Two spheres demonstrating most of the shader stages. Just missing the third (and arguably more desired, personally) sphere which would have showcased algorithmic morphing and reflection.

UI & Controls

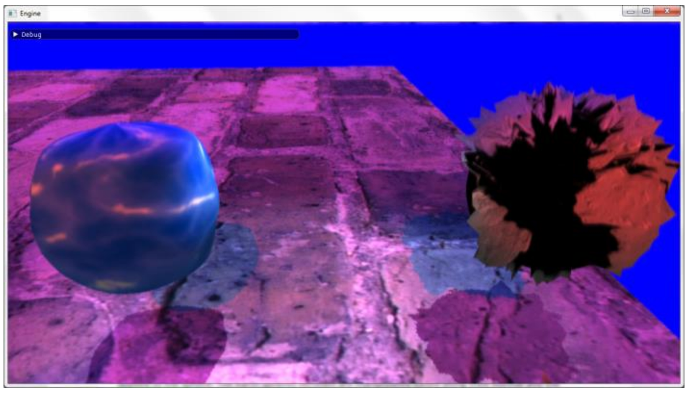


```
▼ Debug
FPS: 60.00
  Wireframe mode
▼ Tessellation
            2.000   Tessellation Factor
            0.069   Wave Height
            5.000   Wave Frequency
            1.000   Wave Speed
▼ Displacement
            0.399   Displacement Height
▼ Depth Of Field
            0.000   Offset
            1.000   Range
▼ Light Settings
 ▸ Left Directional
 ▸ Right Directional
 ▸ Spot Light
▼ Ortho Meshes
  Render top left ortho mesh
  Render top right ortho mesh
  Render bottom left ortho mesh
  Render bottom right ortho mesh
```

Header	Sub-Header	Range	Notes
Tessellation	Tessellation Factor	1 - 64	Alters tessellation factor applied to the inside and edges of each tessellated shape.
	Wave Height	0 – 0.5	Alters the height of the wave on the water sphere.
	Wave Frequency	0 – 5	Alters the frequency of waves on the water sphere.
	Wave Speed	0 – 5	Alters the speed of the waves on the water sphere.
Displacement	Displacement Height	0 - 1	Alters the vertex positions of the earth sphere based on the height map applied to the mesh.
Depth of Field	Offset	-1 – 1	Simple offset to alter the distance at which an object is perceived to be in focus.
	Range	0 - 2	Alters the focus diameter of the depth of field.
Light Settings	Left Directional	N/A	Contains options to alter the ambient and diffuse colour. Other options include changing the direction and position. Affects the left directional light.
	Right Directional	N/A	Contains options to alter the ambient and diffuse colour. Other options include changing the direction and position. Affects the right directional light.
	Spot Light	N/A	Contains options to alter the ambient and diffuse colour. Other options include changing the direction and position. Affects the spot light.
			Contains an advanced sub-header which allows the user to alter the angle, linear, constant and quadratic factors of the spot light.
Ortho Meshes	Render top left ortho mesh	True/False	Tick box to alter rendering of the top left ortho mesh which has the left directional lights shadow map applied to it.
	Render top right ortho mesh	True/False	Tick box to alter rendering of the top right ortho mesh which has the right directional lights shadow map applied to it.
	Render bottom left ortho mesh	True/False	Tick box to alter rendering of the bottom left ortho mesh which has the spot lights shadow map applied to it.
	Render bottom right ortho mesh	True/False	Tick box to alter rendering of the bottom right ortho mesh which shows the normals of the two spheres in the scene.

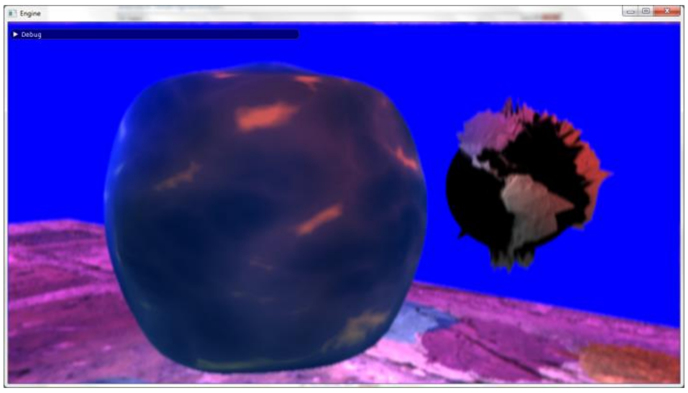
Algorithms and Data Structures

Vertex Manipulation



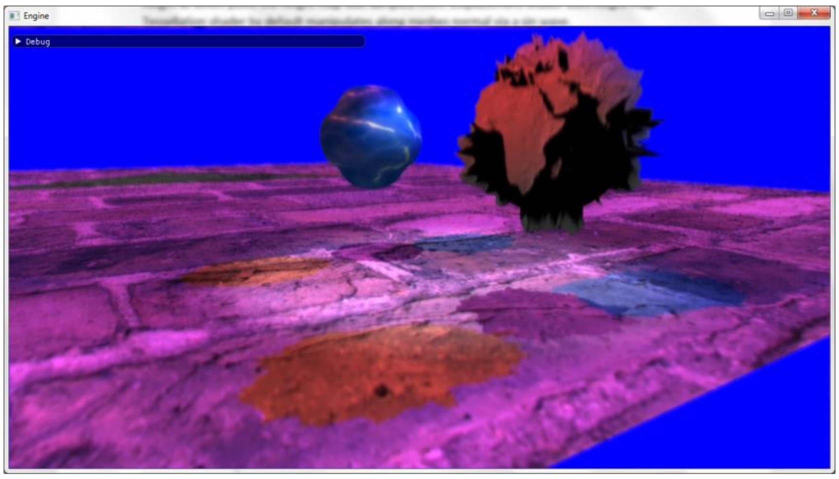
Both spheres within the scene showcase a different example of vertex manipulation. The water sphere has its position manipulated along its normals via a sin wave. The earth sphere showcases vertex manipulation via a height map which has been applied to the sphere. The earth spheres vertexes are manipulated based along their normal and the height at which point the height map was sampled from. Both spheres are tessellated which also affects the vertex manipulation as it provides a varying number of vertexes depending on the tessellation factor applied. In the water spheres case it makes the waves smoother. For the earth sphere, it allows more displacement due to the increased number of normals.

Post Processing

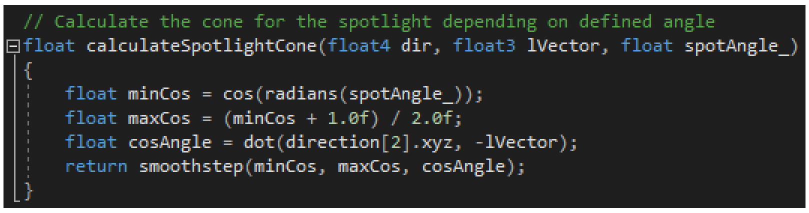


The scene utilises the vertical and horizontal Gaussian blur with down sampling to apply a final depth of field post processing effect. This depth of field shader was based on the XNA Shader Programming tutorial on depth of field. It takes in the normal scene texture, the combined blur texture after down sampling and the depth texture created from the cameras depth pass. The depth of field shader also takes in four variables, two of which can be altered by the user via ImGUI. The other two are the values of the near and far planes within the scene. These floats are passed into the pixel shader where they are used to calculate the blur factor. The three textures passed in are sampled based on the texture coordinates of the ortho mesh which this shader is applied to. The depth texture is sampled twice, once using the given texture coordinates and another based on the centre of the depth texture. These values are then flipped to put them in the 0 – 1 range. They are then both multiplied by the result of far planes value minus the near planes value to convert them into world space and the result of this is passed to the calculation of the blur factor. The blur factor is used to linearly interpolate between the two textures (normal and blurred), based on what the camera is looking at.

Lighting & Shadows



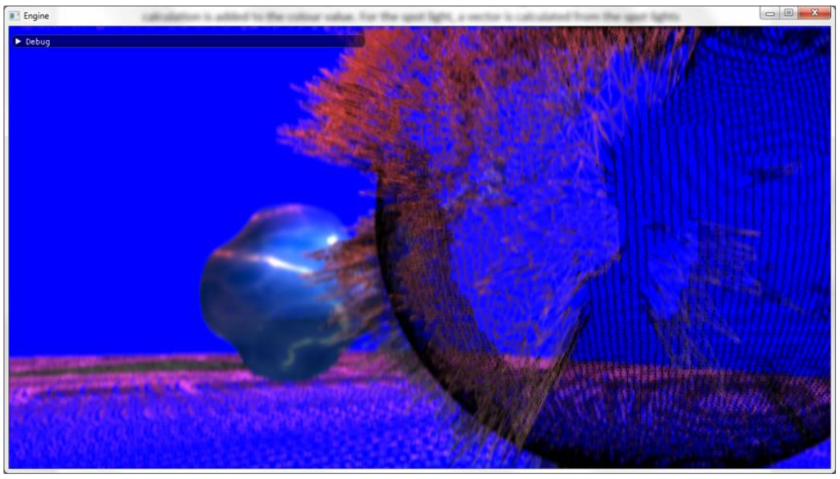
There are three lights within the scene, two are directional and the third is a spot light. Each light has its own shadow map generated via the three depth shaders based on what object is passed to it. These shaders are functionally like their non-depth counterparts with the main difference being that only the position and depth value are passed to their pixel shaders. These shadow maps are then used within the shadow, tessellation and displacement shaders. The displacement shader uses the same pixel shader as the tessellation shader. The pixel shader is more or less the same between the shadow shader and tessellation shader. For each light, the projected texture coordinates are calculated. If the projected texture coordinates are outside of the 0 -1 range then lighting is not calculated for the specific mesh. The depth value (e.g. the depth of the geometry) is calculated via sampling the shadow map of each light. The depth value of the light is based on its view of the object from the lights source, minus the shadow map bias. For the first two directional lights a simple diffuse calculation is added to the colour value based on their direction, the meshes normal and the lights diffuse value. For the spot light, a vector is calculated from the spot lights position and the meshes world position. The length of the spot lights vector is then used to calculate the attenuation factor of the spot light using the values passed in via a buffer (constant, linear and quadratic factors of the spot light). The spot lights vector is then normalised and is used for a basic diffuse calculation which is multiplied by the attenuation calculated earlier. The final spot light colour is multiplied by the intensity which is calculated based on the direction of the spot light, its normalised vector and its angle.



```
// Calculate the cone for the spotlight depending on defined angle
float calculateSpotlightCone(float4 dir, float3 lVector, float spotAngle_)
{
    float minCos = cos(radians(spotAngle_));
    float maxCos = (minCos + 1.0f) / 2.0f;
    float cosAngle = dot(direction[2].xyz, -lVector);
    return smoothstep(minCos, maxCos, cosAngle);
}
```

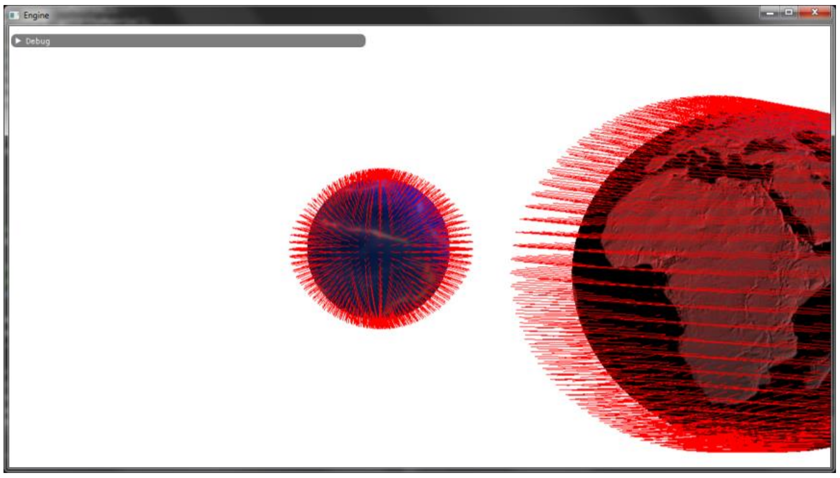
This results in a nice cone being calculated for the spot light. There is an issue with the spot light where it seems to think a part of the plane should be shadowed. From debugging it seems to be either an issue with the spot lights light view position or how lighting/shadowing is calculated in the pixel shader.

Tessellation



Both spheres within the scene are 4 control point patch lists. This means that until the domain shader, the vertex and hull shaders are purely passing along control points. The hull shader sets the tessellation factor of both spheres inside and edges. The tessellation factor which affects the inside and edges of both spheres can be altered via ImGUI. Both spheres are subject to distance based tessellation which divides the tessellation factor passed into the domain shader by the distance each mesh is away from the camera. The domain shader is used to convert the control points into vertex points by linearly interpolating between the patches given positions and the texture coordinates. The same is done for the texture coordinates and the normals of the mesh.

Geometry Shader Stage



A rather primitive use of the geometry shader stage but it is useful none-the-less. Functionally similar to their Displacement and Tessellation shader counterparts. Main difference being their geometry shaders output a line stream and have a max vertex count of 2 compared to outputting a triangle stream with a max vertex count of 4. Looping for each input point for the triangle, two points are appended before restarting the line strip. The first point of the line is at the vertex position, the end of the line is based on the vertex position plus the normals position multiplied by the length of the normal. This then creates lines around the sphere showcasing all of its normals. There is a weird issue though where the lines seem to rotate with the camera. The main suspect would have to be the calculation used for the normals before being given to the geometry shader.

Critical Reflection

In short, a lot was learned. During this module many new concepts have been introduced and a few took previous knowledge further. Vertex manipulation, tessellation and the geometry shader are all new concepts but after learning about them they are some of the most important. Furthering previous knowledge from OpenGL, the lighting and post processing effects that we can now implement in greater detail using DirectX can be re-implemented via glsl in OpenGL. There is an issue with the spot lights shadow calculation which could be resolved if there was more time. Reducing the branching within the shaders would be the next step for improving the application. Increasing the number of neighbours used in the blur shaders and possibly introducing a dynamic weight calculation for each neighbour would improve the blur shaders. Possible extension of the blur shader could be to utilise a circle of confusion as featured in the Nvidia depth of field gpu gems chapter. Use of texture arrays within shaders that have multiple textures being sent to them could simplify code and better utilise memory. Furthering the current implementation from pure diffuse calculations, a blinn-phong or cook-Torrance model could have been implemented to showcase specular highlights on the water sphere. The dynamic tessellation could be improved via ensuring that the vertices don't tear as a result of the distance-based tessellation. Also, sampling the points around a specific patch to check if tessellation would make much of a difference would help to reduce what areas need to be tessellated. Extensions for the application would include the implementation of the shader blob put forth in the proposal and doing bill boarding or a water droplet effect in the geometry shader. The shader blob would showcase reflection as a post processing effect, but refraction could also be applied to the water sphere. The reflection effect would be something like that which was featured in the Frank Luna book at the chapter on cube mapping. If the project was to be redone, the major points would be; to break down and plan as far as possible each shader that was being sought after, to gather information from multiple sources and utilise available time better.

References

Digitalerr0r. 2009. XNA Shader Programming. [ONLINE] Available at: https://digitalerr0r.wordpress.com/2009/05/16/xna-shader-programming-tutorial-20-depth-of-field/ [Accessed 09/12/2018]

Joao Paulo. 2017. Water 001. [ONLINE] Available at: https://3dtextures.me/2017/12/28/water-001/ [Accessed 09/12/2018]

Aster. 2004. GDEM V1 Black & White Map. [ONLINE] Available at: https://asterweb.jpl.nasa.gov/gdem.asp [Accessed 09/12/2018]

Ocornut. 2018. ImGUI. [ONLINE] Available at: https://github.com/ocornut/imgui [Accessed 09/12/2018]

Joe Demers. 2007. Depth of Field. [ONLINE] Available at: https://developer.nvidia.com/gpugems/GPUGems/gpugems_ch23.html [Accessed 09/12/2018]

Frank D. Luna. Introduction to 3D Game Programming with DirectX 11.0. Chapter 17 – Cube Mapping. [BOOK]

Gabriel Lacey – Gave some pointers about implementing the depth of field shader with regards to applying the shader based on the centre of the depth texture.

Paul Robertson – Texture, Shadow, Vertical and Horizontal Blur shaders which have either been extended or used outright.