Student information

Name: Cormac Somerville

Student ID: 2200592

Github Username: Corcso

Link to coursework repository:
https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1

Shared Files

The following files are used throughout the project and are relevant to all sections.

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/App1.h

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/App1.cpp

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/Common.hlsli

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/WorldObject.h

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/WorldObject.cpp

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/CommonStructs.h

Unused Files

The following files are not used and are excluded from the build. They still reside in the project to show my process.

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/DOFPart1_ps.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/DOFPart2_ps.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/DOF_ps.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/ShadowDepthShader.h

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/ShadowDepthShader.cpp

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/ShadowDepth_ps.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/ShadowDepth_vs.hlsl

Vertex Manipulation

Relevant Files:

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/HeightMap_vs.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/HeightMap_hs.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/HeightMap_ds.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/HeightMap_ps.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/Waves_ds.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/Waves_ps.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/HeightMapShader.h

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/HeightMapShader.cpp

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/WavesShader.h

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/WavesShader.cpp

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/TessPlaneMesh.h

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/TessPlaneMesh.cpp

Relevant Screenshots and Diagrams:

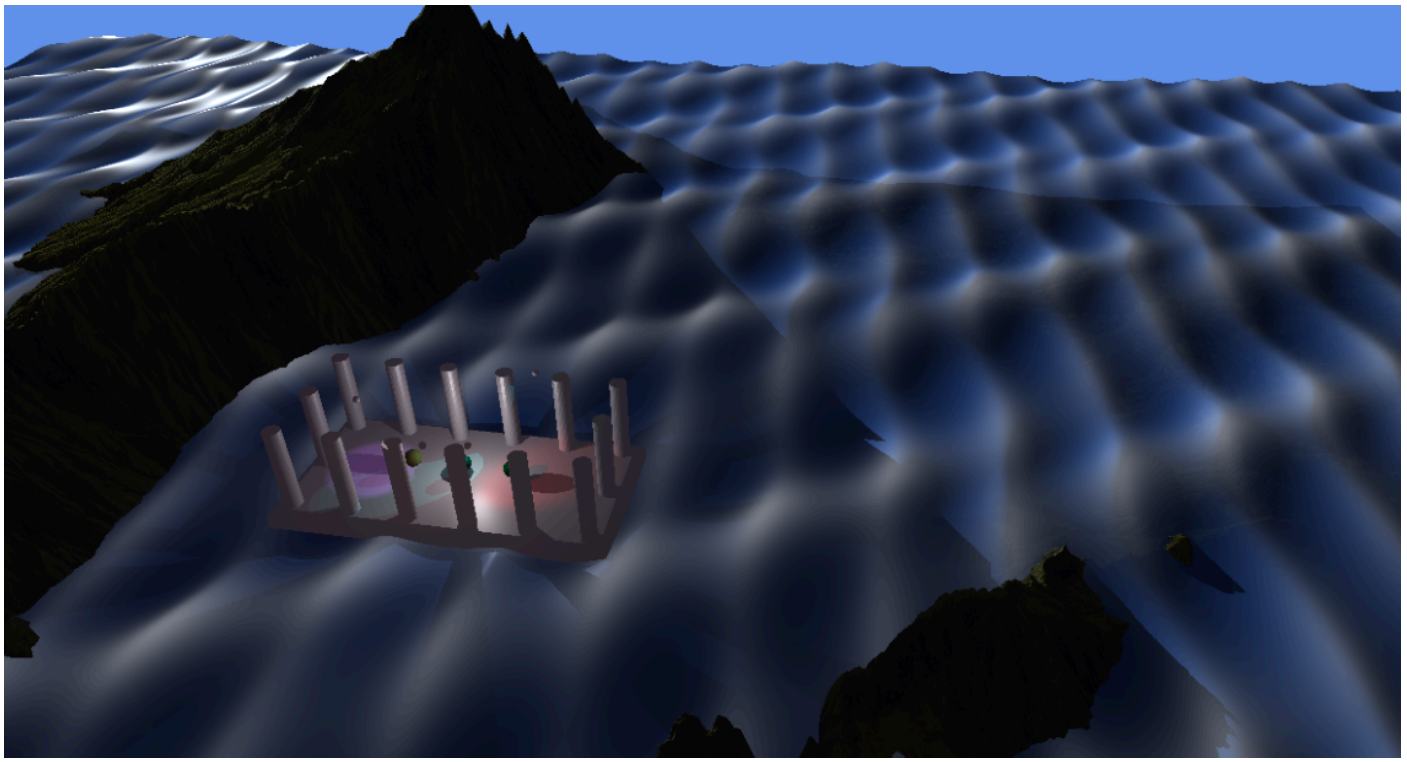


Figure 1 Overview of the scene with a low sun coming from the top left.

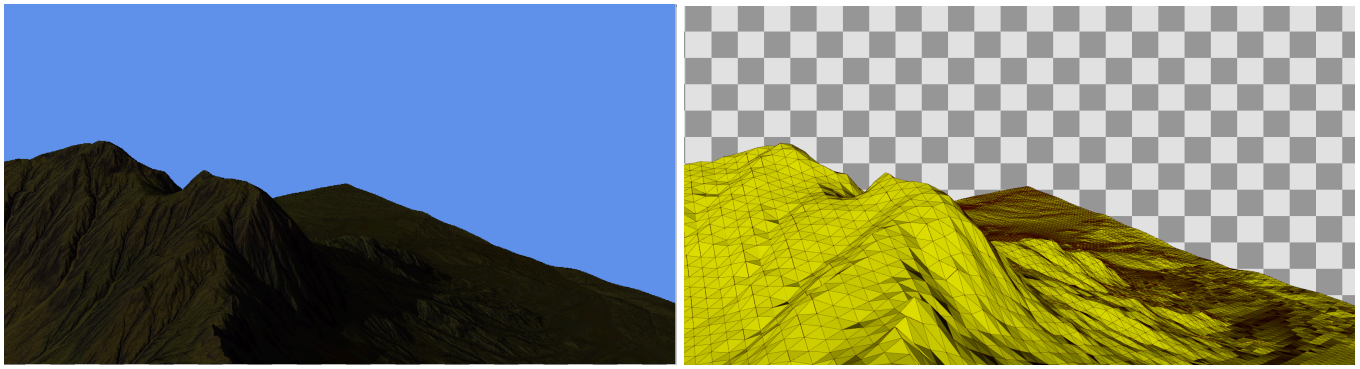


Figure 2 & 3 Render & Wireframe of Terrain

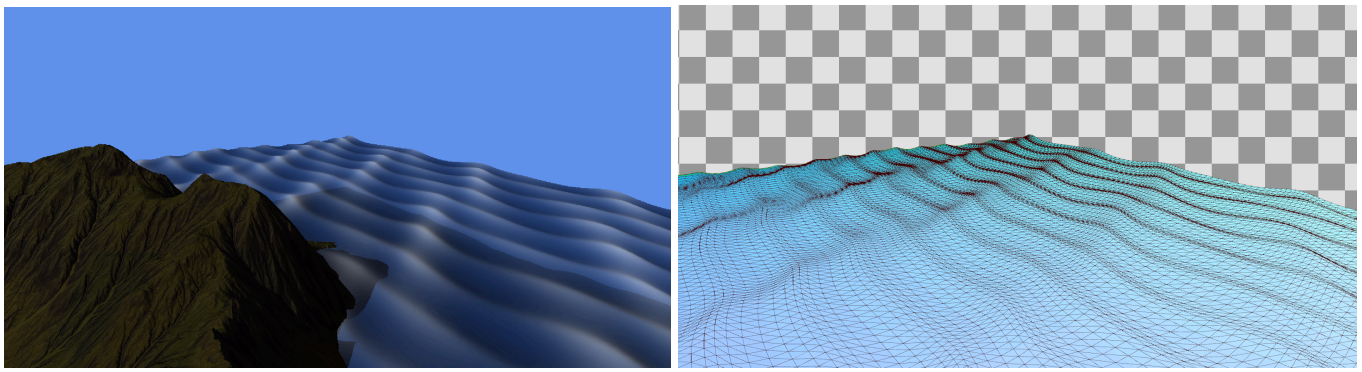


Figure 4 & 5 Render & Wireframe of Waves

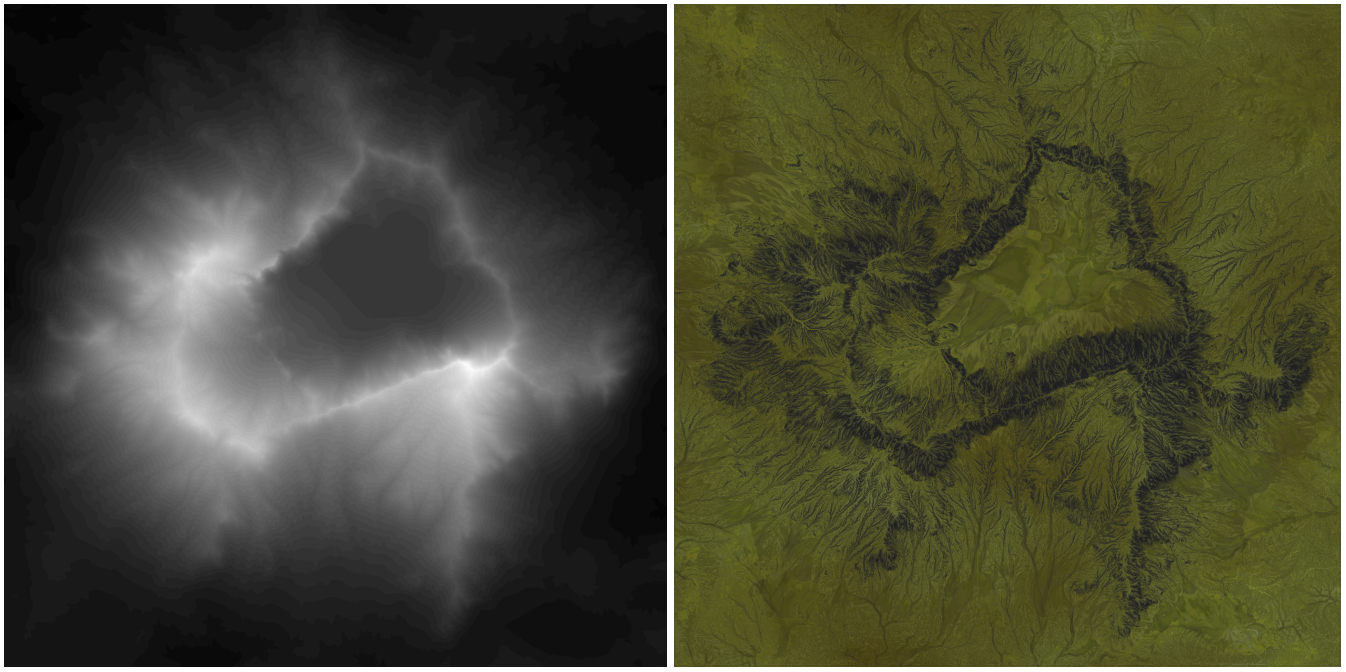


Figure 6 (Left) Height map for terrain (Demes, 2020) Figure 7 (right) Colour map for height map (Demes, 2020).

What was implemented:

Two vertex manipulation effects have been implemented, the first is a height map technique which takes data from an image to produce a height map. Height map normals are calculated via a rate of change approach. The height map can also be optionally smoothed at runtime (ananbd, 2020), this helps reduce sharp edges. The height map makes use of The Vertex, Hull, Domain, and Pixel shaders. With the majority of height map related calculations happening in the Domain and Pixel shaders. The Common.hlsli file additionally has the smoothing functions used for height map smoothing.

The second is an example of algorithmic vertex manipulation which uses a tri Gerstner waves approach (Fernando, 2004). Normals are calculated based on the partial derivative of the wave function. The wave function takes 3 wave parameter sets for more realistic and dynamic wave generation. The pixel shader performs lighting and specular calculations to make the waves shiny and additionally adds white to the crests of steep waves. The waves are blended with the scene to give a realistic water appearance. The Vertex, Hull, Domain and Pixel shaders are used, with Vertex and Hull shaders from the height map being reused as the functionality required is the same. The wave position and normal calculations happen in the Domain and Pixel shader.

Both additionally feature dynamic distance based tessellation which will be covered in the Tessellation section.

How was it implemented:

The height map samples a height map texture to decide the height of the vertex. The height map texture stores values from 0 to 1, these are remapped to -1 to 1 and then multiplied by an amplitude to produce the final height.
The normals for the heightmap are calculated using a rate of change approach. In the pixel shader, the height of texels left, right, above and below are compared to create a tangent (along positive U) and a bitangent (along positive V). These vectors are then crossed to produce a normal.

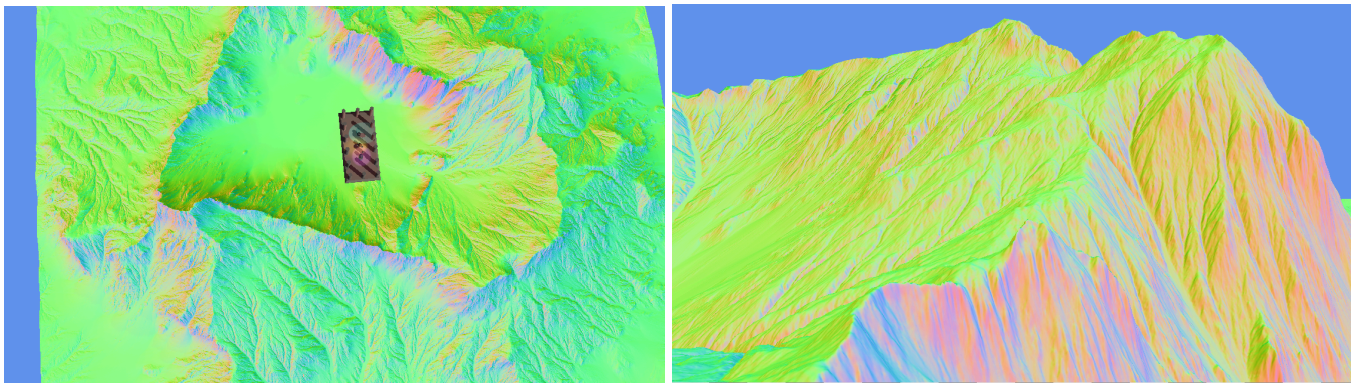


Figure 8 & 9 Normals of the terrain. In the top down view, down is positive Z, left is positive X.

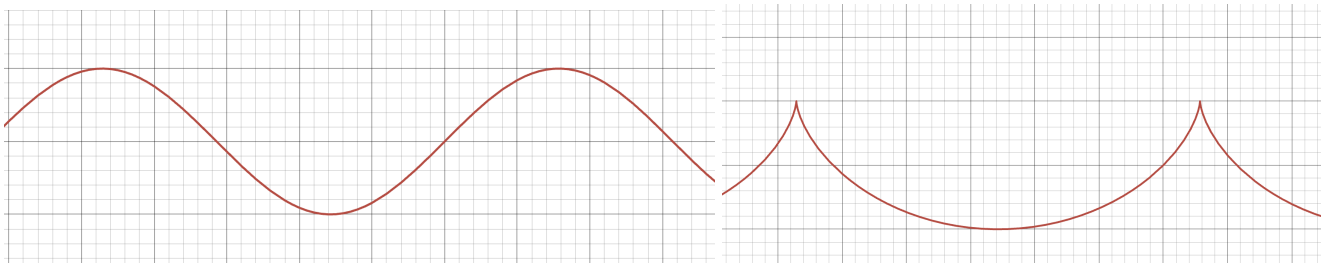
The images above showcase the normals outputted from the height map, I am able to validate the normals as correct as the red, green and blue regions map correctly to the X, Y and Z directions.

The diffuse colour for the height map is then based off of the colour texture for the height map. The height map correctly handles shadows and diffuse lighting, it does not perform any specular lighting calculations as the height map is not considered a smooth object.
The height map additionally features a smoothing option which removes harsh edges from the heightmap. This makes use of weighted samples around the heightmap similar to that of a gaussian blur (ananbd, 2020), where the weighted samples are sampled using a custom bilinear sampler. This custom bilinear sampler samples bilinearly over a certain amount of texels rather than just 1, to achieve a more smoothed average result. The custom bilinear sample uses index access of the texture to improve performance.

The waves use a tri Gerstner wave function (Fernando, 2004). This allows for 3 sets of wave parameters to be passed in. The image below shows the positional formula for calculating Gerstner waves. It features the use of a steepness parameter Q which separates it from normal wave calculations. The other parameters are (A) amplitude, (D) 2D directional vector, (t) time, (φ) speed, (ω) frequency, x & y the x and z position of the vertex (in the formula below z is up). (i) denotes the wave in which the parameters come from.

$$\mathbf{P}(x,y,t) = \begin{bmatrix} x + \sum\left(Q_i A_i \times \mathbf{D}_i.x \times \cos\left(w_i \mathbf{D}_i \cdot (x,y) + \varphi_i t\right)\right), \\ y + \sum\left(Q_i A_i \times \mathbf{D}_i.y \times \cos\left(w_i \mathbf{D}_i \cdot (x,y) + \varphi_i t\right)\right), \\ \sum\left(A_i \sin\left(w_i \mathbf{D}_i \cdot (x,y) + \varphi_i t\right)\right) \end{bmatrix}.$$

Figure 10 Positional Calculation formula for Gerstner waves from (Fernando, 2004). Please note Y and Z are flipped in this example.



Figures 11 & 12 Cross section of wave formula position where Q, steepness = 0 and = 1 (Desmos Studio, no date)

To calculate normals of an algorithmic vertex manipulation, we can partially derive the parametric position equation by X and Z to produce tangent and bitangent vectors. These can then be crossed together to obtain a normal.

$$\mathbf{N} = \begin{bmatrix} -\sum(\mathbf{D}_i.x \times WA \times C0), \\ -\sum(\mathbf{D}_i.y \times WA \times C0), \\ 1 - \sum(Q_i \times WA \times S0) \end{bmatrix}, \quad \begin{aligned} WA &= w_i \times A_i, \\ S0 &= \sin\left(w_i \times \mathbf{D}_i \cdot \mathbf{P} + \varphi_i t\right), \text{ and} \\ C0 &= \cos\left(w_i \times \mathbf{D}_i \cdot \mathbf{P} + \varphi_i t\right). \end{aligned}$$

Figure 13 The normal calculation formula for a Gerstner wave. (Fernando 2004)

The waves feature a dynamic alpha value, with lower pixels being less transparent than higher ones to give the illusion of more water occluding the light. When the wave is steep, higher parts also get a white tint to simulate wave foam.
The waves are then rendered last and blended with the scene using inverse source alpha additive blending.

Critical Analysis:

I believe the vertex manipulation has overall been a success. Geometry is correctly generated and normals are applied.

The terrain smoothing feature is quite performance heavy, when combined with high tessellation and depth of field render times become long and not suitable for real time graphics applications.

The water could have been improved with more accurate colouring, as well as depth independent blending. Nevertheless from a vertex manipulation standpoint the water wave simulation works well. One additional feature I could add to the waves would be the ability to remove and add the number of wave parameter sets which are summed to generate the waves.

Lighting and Shadows

Relevant Files:

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/PBR_vs.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/PBR_ps.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/PBRShader.h

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/PBRShader.cpp

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/TextureCubeShadowMaps.h

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/TextureCubeShadowMaps.cpp

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/WorldLight.h

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/WorldLight.cpp

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/UVSphereMesh.h

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/UVSphereMesh.cpp

Relevant Screenshots and Diagrams:

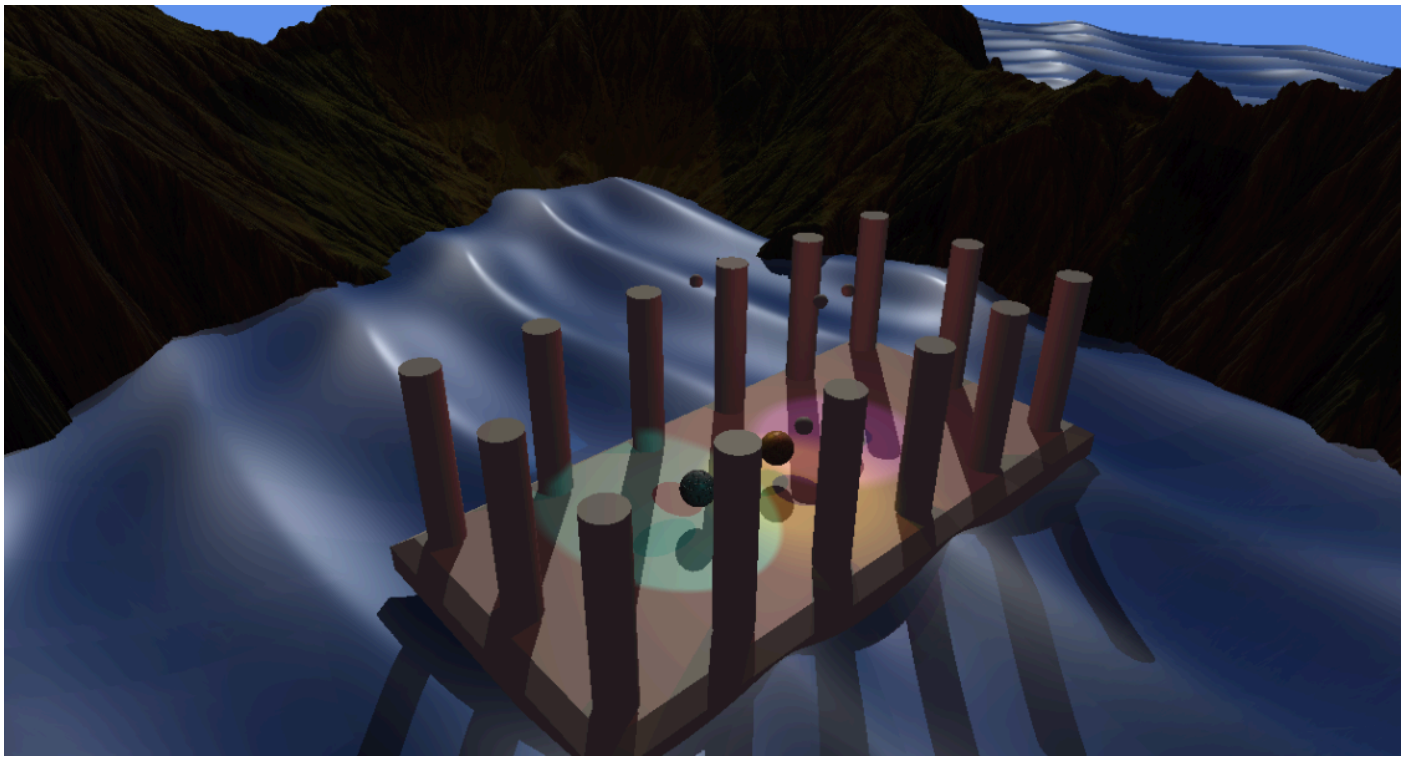
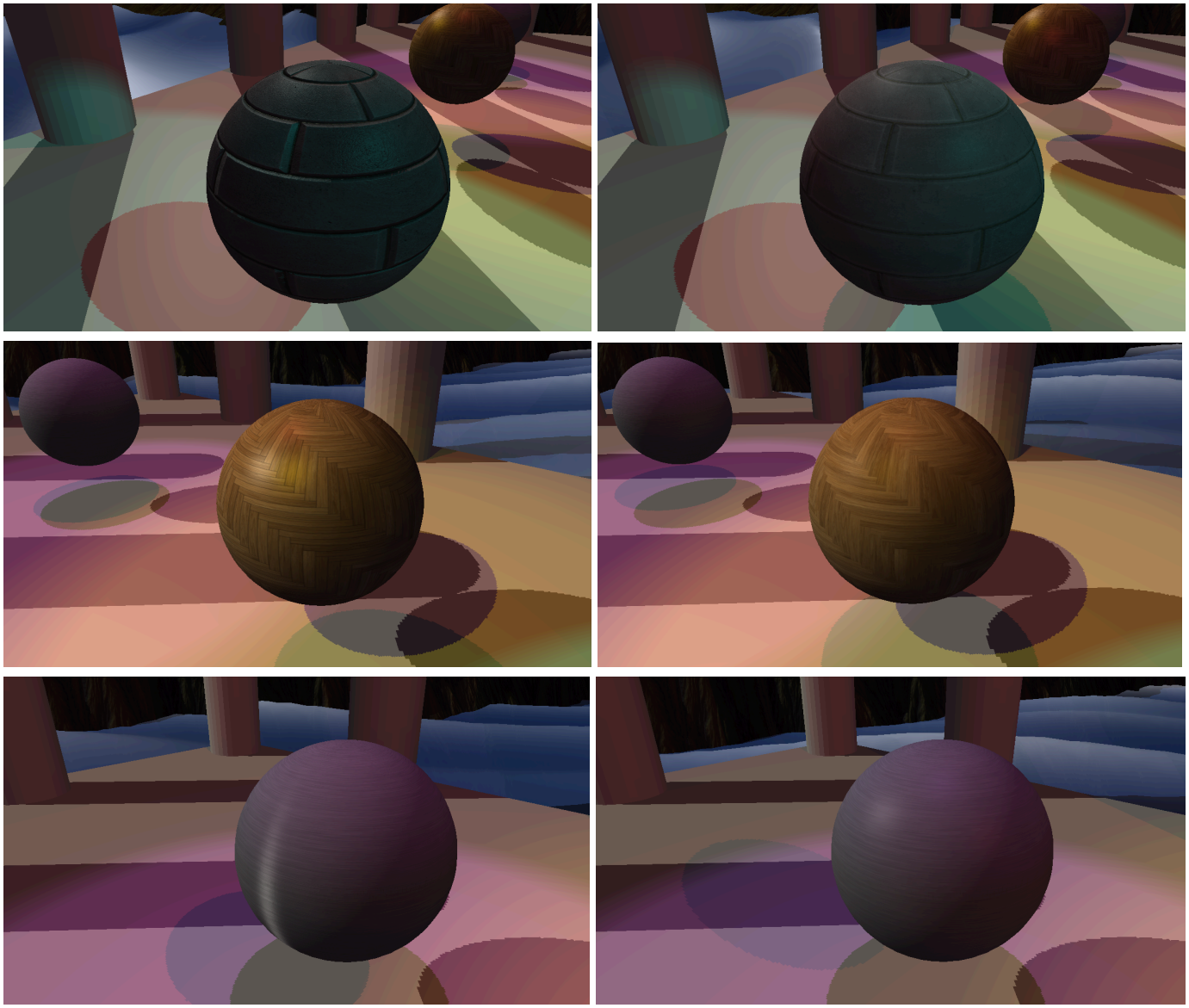


Figure 14 Temple which features PBR Objects and multiple lights.



Figures 15, 16, and 17 Comparisons between objects with PBR properties enabled (left) and only basic properties (right)

What was implemented:

The scene features 3 types of lights, point, directional and spotlights. Each type of light casts correct shadows, with point and spot casting perspective projected shadows and directional lights casting orthographic shadows.
The PBR (Physically based rendering) Shader makes use of the Heidrich–Seidel anisotropic (Wikipedia, no date b) specular lighting model combined with Blinn-Phong (for isotropic lighting). It also allows for the application of ambient occlusion, normal and roughness maps onto objects to produce details beyond vertex resolution.
Point and Spotlights cast omnidirectional shadows with the use of cube mapping (Luna, 2012).
The scene supports up to 8 active lights at a time, it contains 5 active lights which the user is able to control. 1 Directional light, 1 Point light and 3 Spotlights.

How was it implemented:

All shadows were implemented with a shadow mapping technique. A shadow map is a depth only view of the scene from the light's perspective. This view is then used within the pixel shader when rendering the colour output. Pixels world position's are projected with the same view and projection matrices used in the shadow map generation. The depth of the point on the shadow map is then compared to the projected depth of the world position. If the depth on the shadow map is closer than the one from the world position,

something is between the light and that pixel and it is counted as being in shadow. Otherwise it is not in shadow and diffuse & specular lighting calculations are performed.

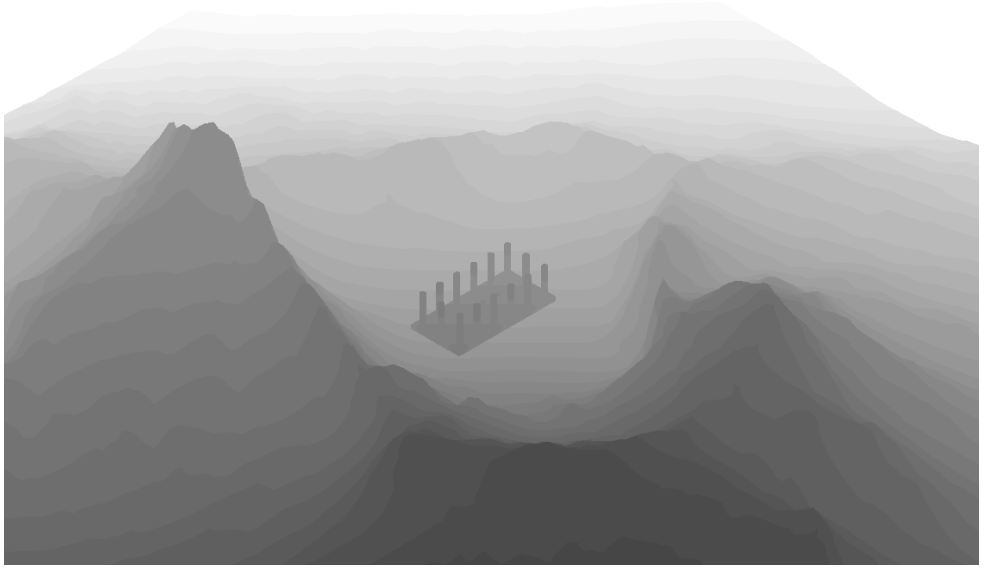
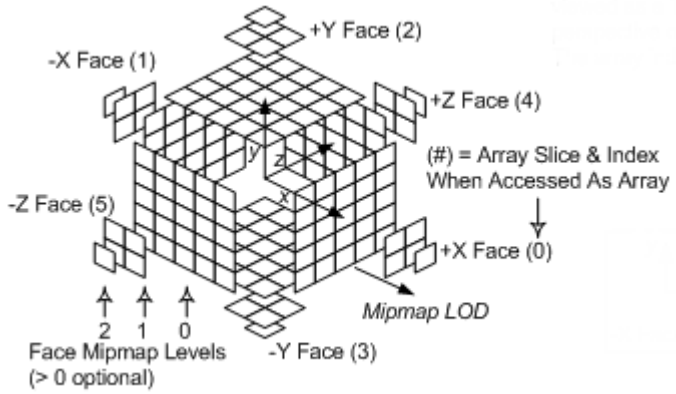


Figure 18 The Sun's shadow map. Directional light, Orthographic projection.

Directional shadows make use of a single orthographically projected shadow map. Since a point light's light radiates in all directions, and a spotlight's light can radiate in a hemisphere (When above 45 degrees a spotlight will cover 5/6 cube map faces), a cube map is used to achieve accurate shadows.

A cube map is a 6 faced shadow map which allows for omnidirectional sampling. This requires 6 depth passes, one for each face of the cube, which represent each cartesian direction. (Luna, 2012) This cube map shadow map can then be passed directly into a shader with the correct SRV flags and sampled with a 3D direction vector from the light to the pixel.



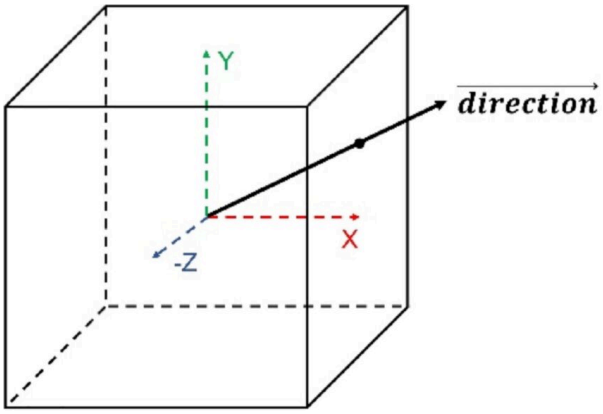


Figure 19 (left) Texture Cube layout (Microsoft, 2019) & Figure 20 (right) How a texture cube can be sampled. (Xiao et al, 2024)

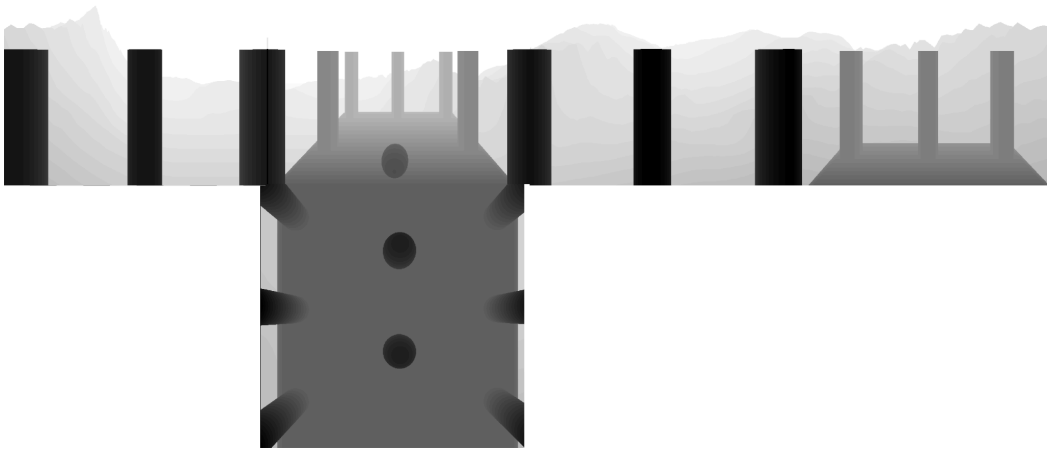


Figure 21 The point light shadow map for the point light in the scene. All faces shown.

When being rendered in the depth only pass, objects use their own shaders. There is not a specific set of shaders for shadow pass rendering. This is to ensure geometry is the same in the final scene view as in the shadow pass.
All shaders support receiving shadows as shadow check calculations are defined in the Common.hlsli HLSL header file.
Shadows are calculated per light, this allows for multiple lights to be added to the scene and cast shadows accurately. The below images show this. The image on the right showcases how light colours are accurately combined in and out of shadows.

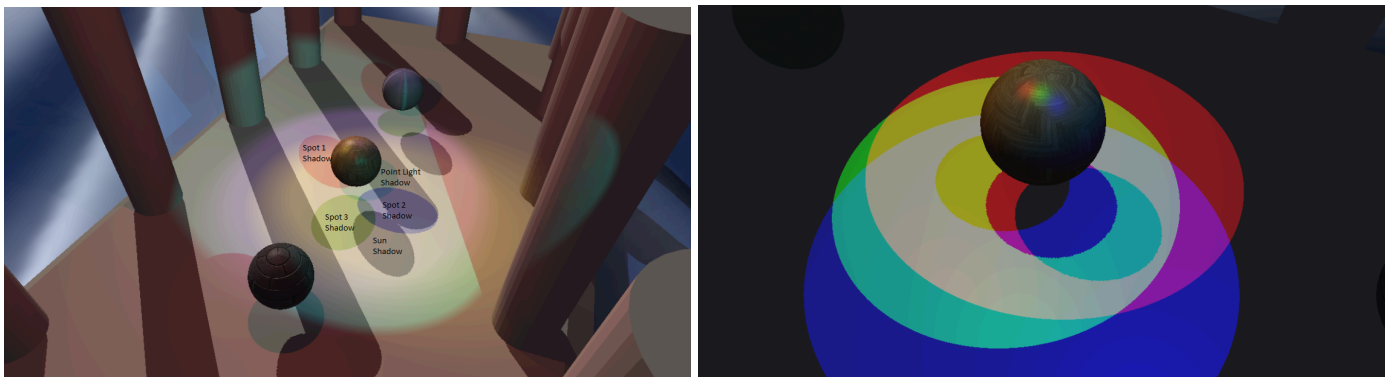


Figure 22 (left) Annotated scene diagram depicting 5 shadows. Figure 23 (right) 3 Spotlights on one sphere, red, green and blue.

All shaders support up to 8 lights, they take in a constant light buffer which contains an array of light structs as well as a light count integer, additionally they take in 8 TextureCubes and 8 Texture2Ds which will not all be bound but this allows for any combination of directional and point/spot lights. Lights contain properties such as position, direction, type, colour, power and attenuation.

The PBR shader takes in a material struct, this contains diffuse & specular colour, specularity, smoothness (inverse roughness), anisotropy, and texture flags. There are also up to 4 textures which can be assigned to an object which is using the PBR shader: A colour map, a normal map, an AO map and a roughness map.

Diffuse colour is the colour which ambient and diffuse light modulate with to produce an output colour, for example a red ball would have a diffuse colour of red.

Specular colour is the colour which specular highlights modulate with, this will typically be white for any real world objects.
Specularity is the sharpness of the specular highlight, lower values cause a larger specular reflection while higher values cause a smaller one. Lowering specularity does not produce a matte like effect on its own.
Smoothness is a value from 0 to 1 which affects how much specular highlight is added to the final output. 0 means no specular is added and 1 means specular is added. A value of 1 produces a shiny object whereas a value of 0 produces a matte one.

Anisotropy is used for surfaces which have parallel microfacets instead of regular ones (Wikipedia, no date b). When anisotropic, the surface will produce a specular highlight perpendicular to the object's tangent. This produces an effect seen on the brushed metal object in figure 17.

The specular highlight is calculated based off of the Heidrich–Seidel anisotropic distribution (Wikipedia, no date b) This takes in a normal (N), tangent (thread direction)(D), light to vertex vector (L), vertex to viewer vector (V) and specular power (S). These are then used to calculate the anisotropic specular power.

$$T = \frac{D + (-D \cdot N) * N}{\|D + (-D \cdot N) * N\|} \qquad P = \frac{L + (-L \cdot T) * T}{\|L + (-L \cdot T) * T\|}$$

$$R = \frac{-L + 2 * (L \cdot P) * P}{\| - L + 2 * (L \cdot P) * P\|} \qquad k_{\text{spec}} = (V \cdot R)^s$$

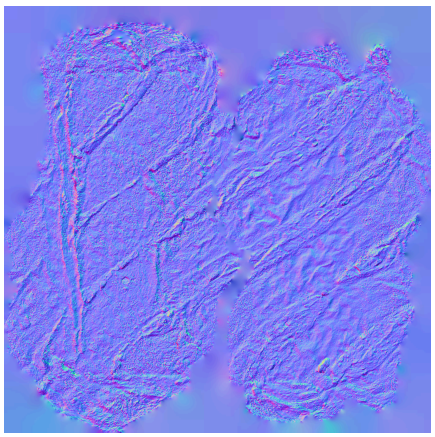
Figure 24 Formulae for the calculation of Heidrich–Seidel anisotropic specular light power (Wikipedia, no date b)

This is combined with an isotropic Blinn-Phong specular highlight. Blinn-Phong is calculated based on the angle between the viewer and the light from the vertex. It uses a half vector which is the normalized combination of the light to vertex and vertex to viewer vectors. This is then dotted with the normal and raised to the power of the specular power of the material.

The combination of the isotropic and anisotropic specular highlights is done with linear interpolated addition. At 0 a full isotropic no anisotropic highlight is returned, at 0.5 a isotropic + anisotropic highlight is returned, at 1 a full anisotropic no isotropic highlight is returned.

When using a colour map the diffuse colour is modulated with the sample from the colour map.

Normal maps are used to redefine normals at a per pixel level allowing for the illusion of more mesh detail than given. (Luna, 2012)



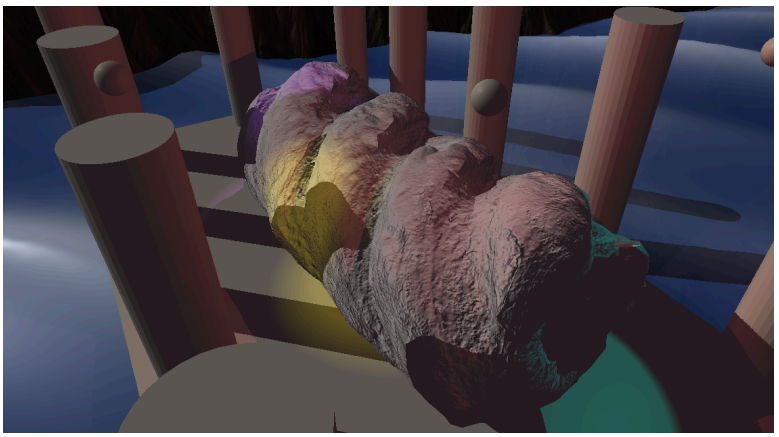


Figure 25 (left) The sausage roll models normal map (Demes, 2021 b) Figure 26 (right) The sausage roll rendered with only the normal map applied.

Normal maps are stored in tangent space (Luna, 2012) where (0, 0, 1) defines a normal perpendicular to the surface. After being remapped to the range -1 to 1 (from colour's 0 to 1) tangent space normals need to be converted into world space normals. This is done with a TBN matrix, where the tangent space normal is multiplied with a matrix made up of the pixels tangent, bitangent and normal (from the vertex data).

$$TBN = \begin{pmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \\ N_x & N_y & N_z \end{pmatrix}$$

Figure 27 TBN Matrix composition. (mtrebi, 2017)

This newly calculated normal is then used in lighting calculations. Below is an image comparing a sphere without a normal map to one with, it shows that the normal map produces correct normals. Positive X, Red, is to the right of the image.

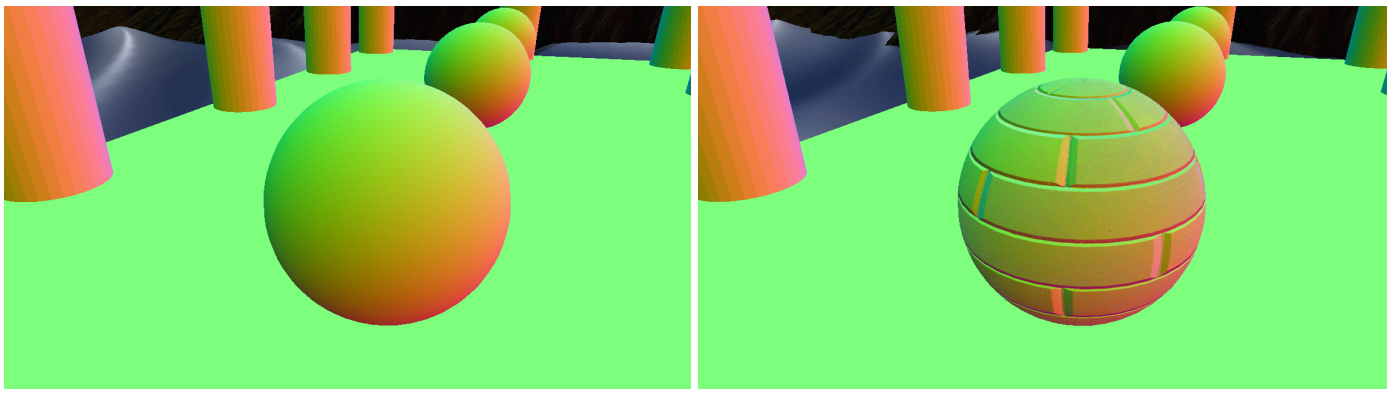


Figure 28 Brick normal map not applied (left) and applied (right).

The ambient occlusion (AO) map is used to simulate the areas where less ambient light is cast than normal, for example a crevice. The AO map represents float values, so they are black and white. A value from the AO map is sampled and then multiplied with the ambient & diffuse light due to be applied to that pixel (McReynolds and Blythe, 2005).

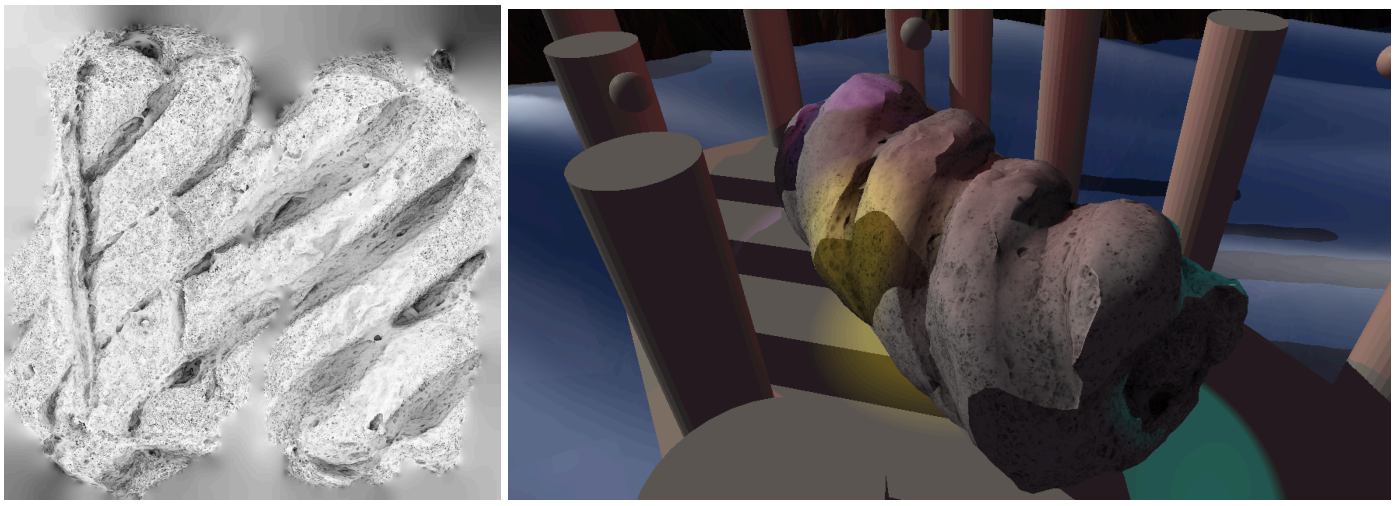


Figure 29 (left) The sausage roll model's Ambient Occlusion map (Demes, 2021 b) Figure 30 (right) The sausage roll rendered with only AO Map applied.

The roughness map allows for shiny and matte surfaces to exist on the same object and not require a separate material. Like AO, the roughness map is black and white so only represents float values. To apply the roughness map, the smoothness value from the constant buffer is overwritten with the inverse sampled value.

Critical Analysis:

Overall lighting & shadows have been successful. Lights produce accurate shadows and lighting, and materials can be applied to create detailed objects. Multiple lights also work well with each other, producing realistic shadows with lights correctly blending in and out of shadow.

PBR materials produce noticeable effects compared to being disabled, allowing for high quality objects with pixel level detail. More PBR parameters could have been added, like metallic properties, or clear coat.

Improvements could be made, perhaps adding smoothing on the end of shadows, especially for directional shadows where the shadow map needs to be of a very high resolution to produce reasonable quality results.

Additionally the code could be structured better with more of the lighting functions being part of the Common.hlsli header file and the shadow maps always being the first 16 texture buffers.

Post Process

Relevant Files:

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/BloomPart1_ps.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/BloomPart2_ps.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/BloomPart3_ps.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/BloomShader.h

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/BloomShader.cpp

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/DOFPart2_v2_ps.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/DOFPart3_ps.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/DepthOfFieldShader.h

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/DepthOfFieldShader.cpp

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/TextureShader.h

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/TextureShader.cpp

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/Texture_vs.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/Texture_ps.hlsl

Relevant Screenshots and Diagrams:

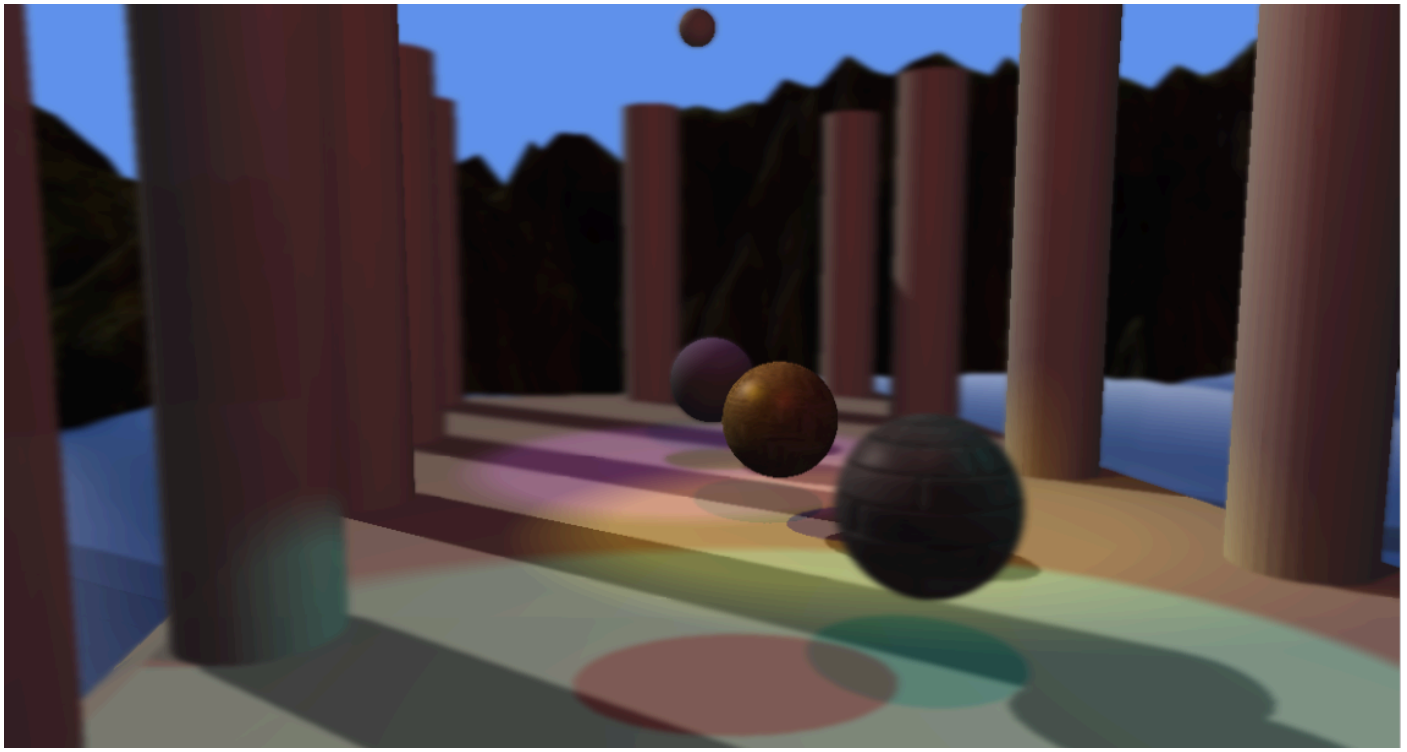


Figure 31 Scene with depth of field applied.

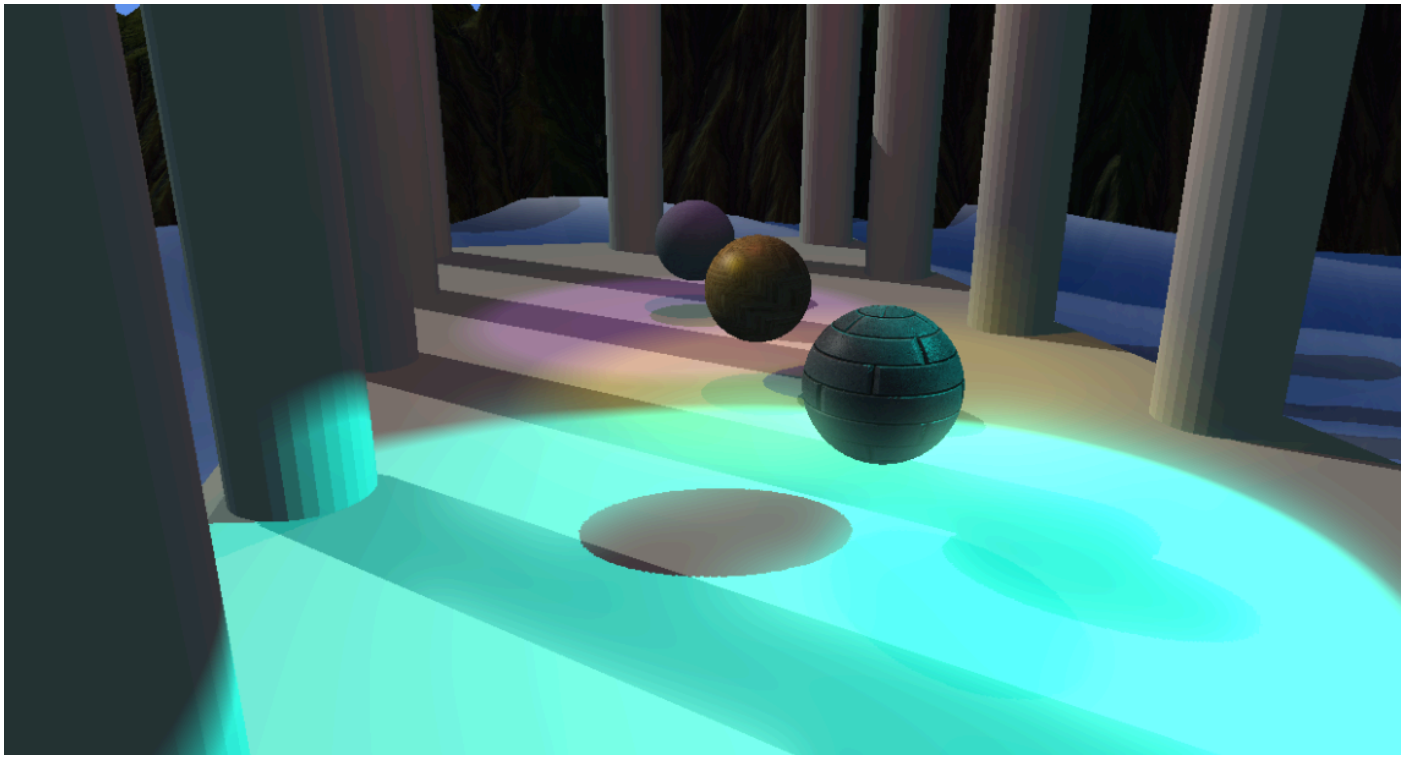


Figure 32 Scene with bloom applied.

What was implemented:

There are 2 post processing effects on this project. One is depth of field which makes use of a forward z buffer pass effect (Fernando, 2004) and the other is bloom (De Vries, 2014 a).
The depth of field uses a composition of 9 variably blurred layers to achieve the effect. These are then combined together to create the final render.
The bloom effect takes the brightest portion of the scene render, blurs it and adds it back on top of the render.

How was it implemented:

Both post processing effects make use of a gaussian blur (Wikipedia, no date a). To apply a gaussian blur to a pixel, the current pixel's value & neighbouring pixel's values are sampled and then multiplied by a weight before being combined. The weight is based on the distance from the current pixel and represents a normal distribution. This blur effect has O(n^2) complexity, but due to its symmetrical nature we can separate the horizontal and vertical blurs which allows us to achieve O(n) time complexity, allowing for larger blur kernels without performance hits.

The depth of field effect uses a forward pass z buffer effect as detailed in (Fernando, 2004). It renders the scene into 9 separate layers, split by their depth. These layers then get blurred based on their distance from the most in focus layer (layer 4). E.g. layer 4 isn't blurred, layer 5 is blurred, layer 2 is blurred twice as much as layer 5.

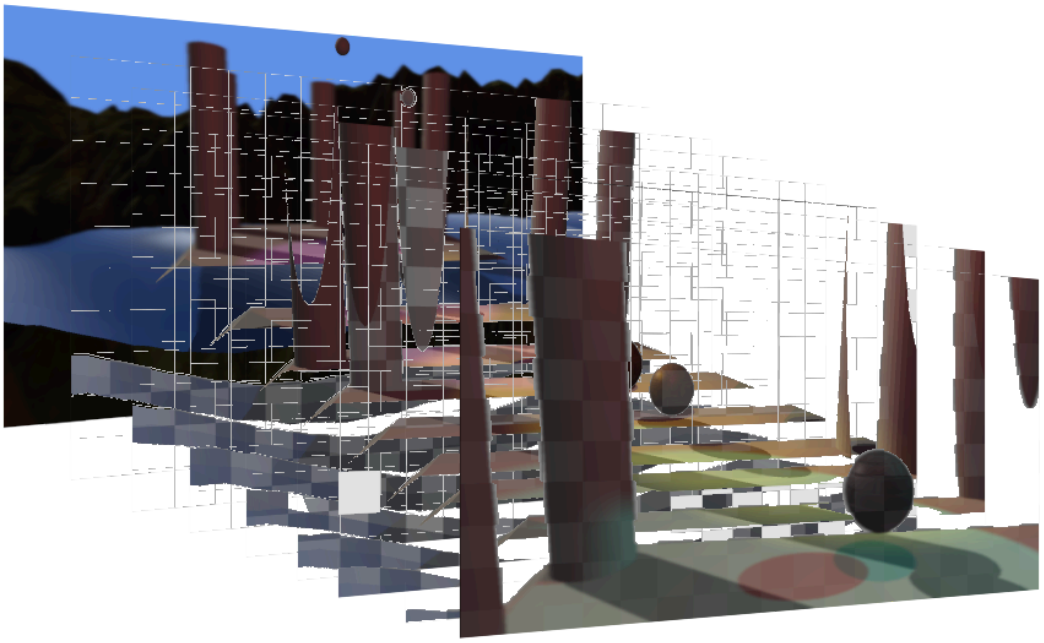


Figure 33 All 9 layers created by the DOF initial pass. The white lines are due to photo editing the background removal, they are not present in the layers.

The values which are used for these layers can be changed with the focus plane control in the GUI.

Each shader has a small function at the start which does any depth clipping for the layer. This happens in each shader rather than happening on a final render of the entire scene because if one object occludes another we aren't able to see the colour information behind the edges. Where edges meet this would cause sharp lines to extend upwards as the shader ceases to know about the colour behind the object in front.

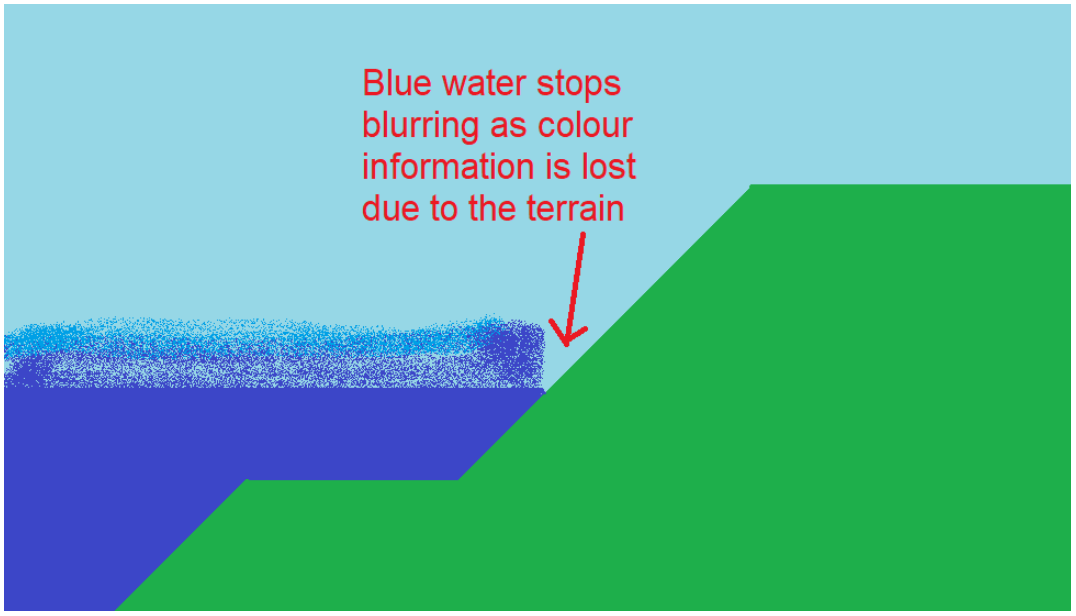


Figure 34 Diagram depicting sharp blur lines issue where colour information is lost behind objects.

To combat another artefact, each layer also stores the layer above it but not blurred, this stops ghosting issues around continuous edges, where 2 different blur distances meet the alpha values don't line up correctly causing you to be able to see through the more blurred layer onto much further behind layers.

The layers are blurred using a separate vertical and horizontal blur which makes use of a gaussian blur with dynamic weights (Wikipedia, no date a).

In the final pass the layers are combined using 1 - source alpha additive blending. The final colour is then corrected so its final alpha value is one.

Bloom uses a similar concept to depth of field, in which a section of the scene is rendered, then blurred then recombined. (De Vries, 2014 a).
The final render of the scene is rendered with the part 1 shader to produce a render of the scene where the overall luminosity (Anonymous, 2019) of the pixel is above a certain threshold.

When the final render of the scene is produced colour values are not clamped to 1. This allows the user to choose a threshold above one.

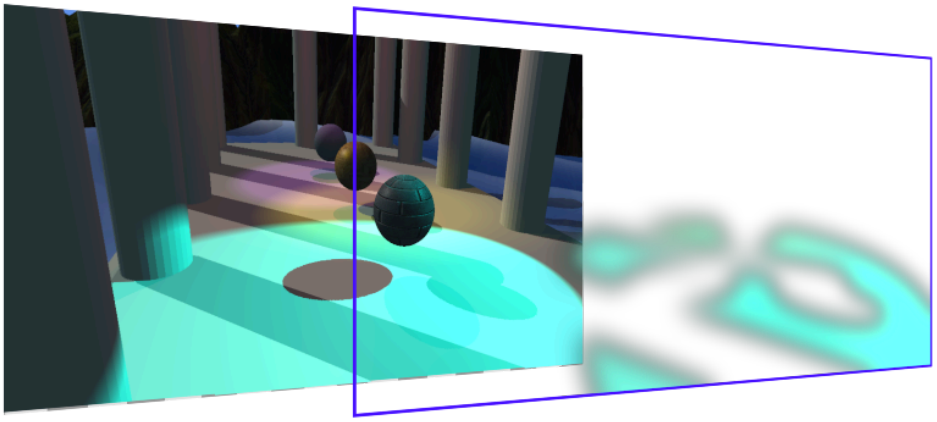


Figure 35 The final scene render and the bloom layer.

The blur is computed with the same 2 pass dynamic gaussian blur technique as seen in the depth of field effect. The user has control over the blur size and skip, both parameters increase the “Blurriness” of the bloom, blur size is more accurate but increasing it has performance cost, blur skip enlarges the blur less realistically but is computationally free. A blur skip of 1 creates a regular gaussian blur, a blur skip of 2 creates a gaussian blur but instead of sampling neighbouring pixels it samples them 2 away.

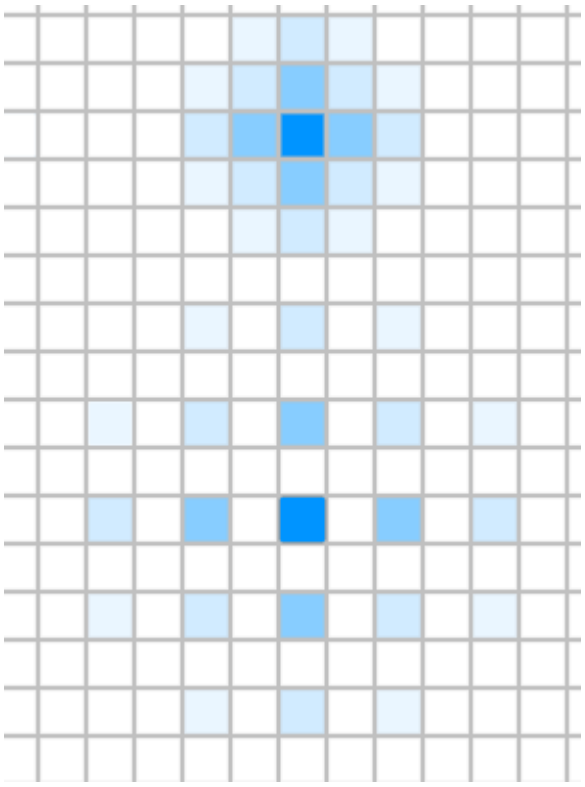


Figure 36 Blur skip diagram. Top is 1 Bottom is 2

The scene is then pasted together using 1 - source alpha additive blending similar to the depth of field shader.

Critical Analysis:

Post processing has overall been a success, with depth of field and bloom producing accurate results. Both post processing effects can also be combined successfully.

Both effects also feature a level of user control, with DOF the user is able to control the focus plane where the sharp layer resides, with bloom the user is able to control the luminosity threshold, blur size and blur skip.

Improvements could be made to the performance of the depth of field effect, due to the large number of passes and blurring required it causes the frame rate to drop noticeably.

Additionally a significant amount of time was spent trialing other depth of field effects and solving artifacts which could have been put to use on additional post processing effects. However this has strengthened the current depth of field solution.

Additional Features

Relevant Files:

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/HeightMap_ds.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/HeightMap_hs.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/Waves_ds.hlsl

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/TessPlaneMesh.h

https://github.com/Abertay-University-SDI/cmp301_coursework-Corcso-1/blob/master/Coursework/Coursework/TessPlaneMesh.cpp

Relevant Screenshots and Diagrams:

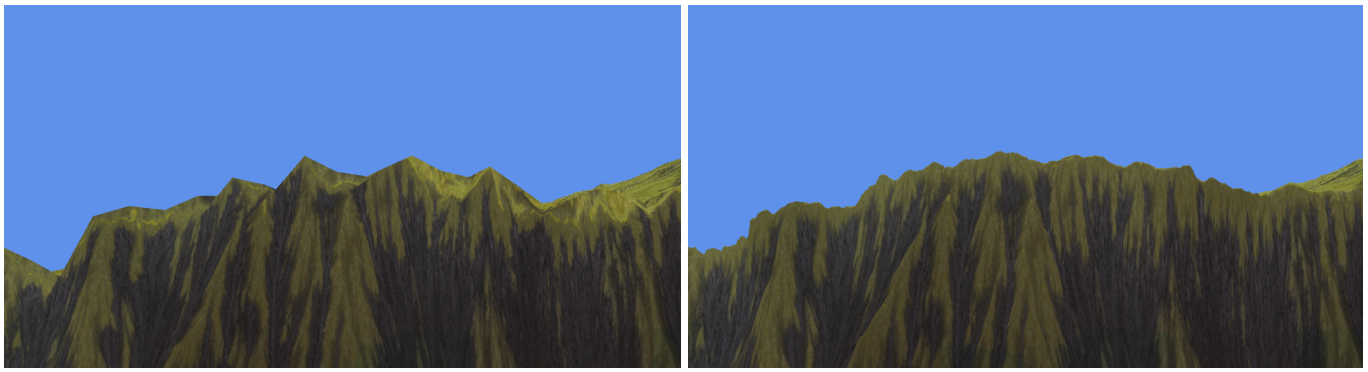


Figure 37 Comparison between non tessellated (left) and tessellated (right) terrain.

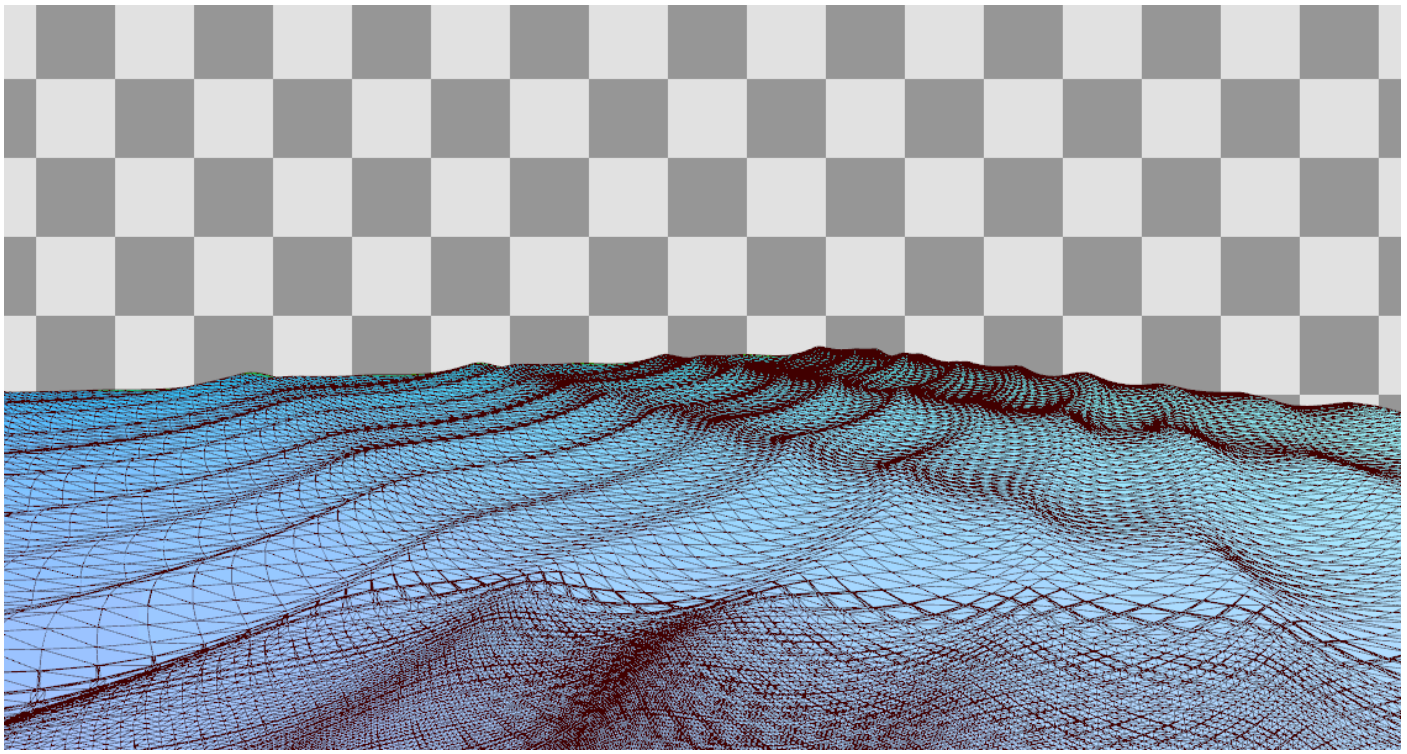


Figure 38 Dynamic distance based tessellation shown in the water manipulation.

What was implemented:

Dynamic distance based tessellation has been implemented with the water and terrain generation. This creates high quality terrain and water geometry with low CPU->GPU transfer costs. The tessellation minimum and maximum can be set by the user, as well as the distance at which these values are applied.

Tessellation is then linearly interpolated between the distance values set by the user.

The tessellation uses 4 control point patches and is seamless with edge based distance checks.

How was it implemented:

The terrain and water both use a TessellationPlaneMesh which formats vertices in a quad like fashion for use with a 4 control point topology type.

The hull shader then calculates the distance from each edge to the camera and uses this to determine the tessellation factor within the bounds. A Fractional tessellation factor is used to produce smoother results. Each edge's tessellation factor is calculated separately to guarantee seamless connection with neighbouring faces. Since neighbouring faces share an edge, the distance, therefore tessellation factor will be the same.

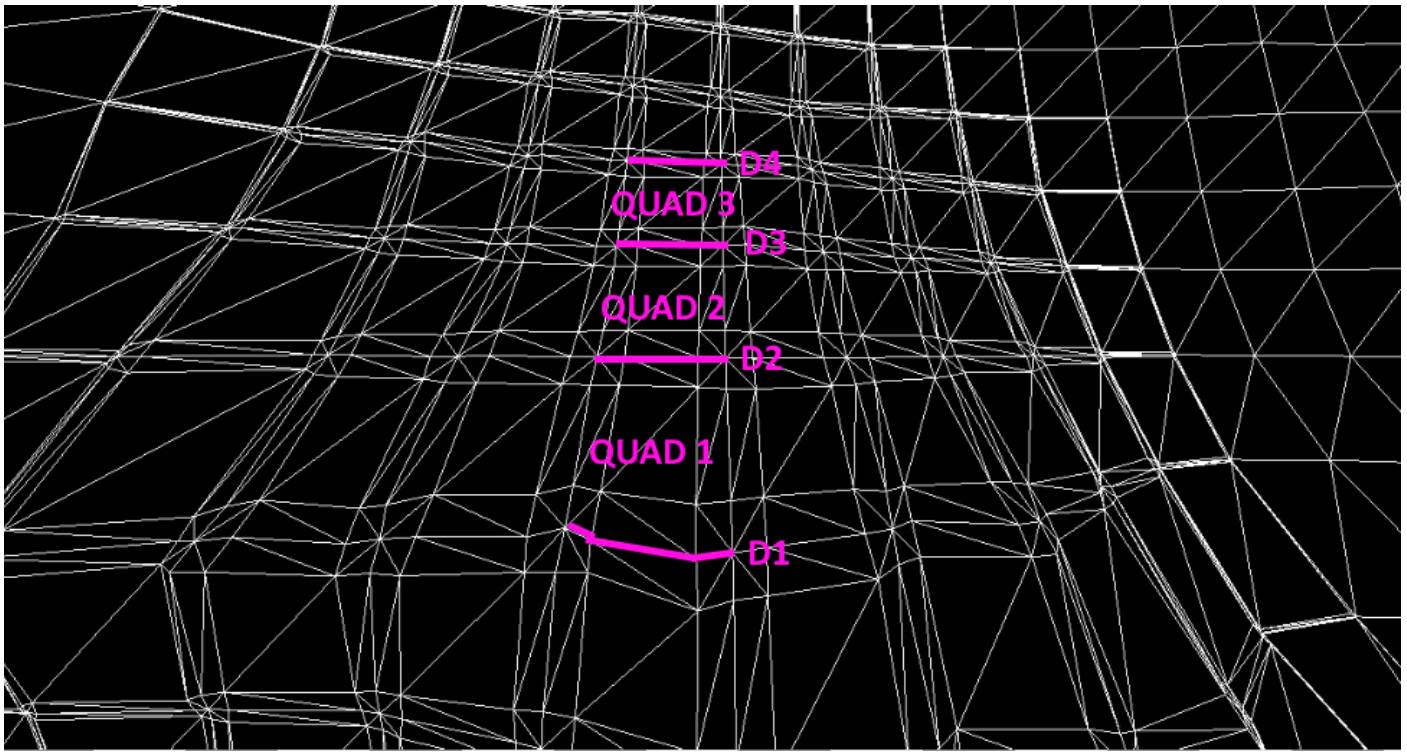


Figure 39 Quads with shared edges have shared edge distances.

The inner tessellation factor is calculated based on the average distance from each edge.
The user has the ability to control the minimum and maximum tessellation factor as well as the minimum and maximum distance these are applied at.

After tessellation factors have been set in the hull shader, the domain shader then calculates the new vertex attributes by using bilinear interpolation based on the domain location provided and the 4 control points the new vertex was derived from.

Critical Analysis:

I believe the dynamic tessellation has worked successfully, the terrain and water are able to be detailed closer to the user and less detailed further away. The edge based distance tessellation allows for seamless geometry with the use of only 4 control points, keeping the code in the hull shader simple and understandable.

I could perhaps further improve the tessellation by taking into account the user's viewing direction and deciding the distance based on the center of the user's screen in world space rather than the camera position. Building on from this, tessellation could be decided based on how much space the triangle or quad takes up on the user's screen, this allows for the most important areas to be tessellated efficiently. Alternatively I could base tessellation on the rate of change of control points, opting to tessellate more curved surfaces over flat ones.

References

ananbd (2020) *Clever ways to smooth a 2D height map?* Available at: https://www.reddit.com/r/gamedev/comments/h93vdv/clever_ways_to_smooth_a_2d_height_map/ (Accessed: 8 December 2024)

Anonymous (2009) *Formula to determine perceived brightness of RGB color* Available at: https://stackoverflow.com/questions/596216/formula-to-determine-perceived-brightness-of-rgb-color (Accessed: 5 December 2024)

Demes, L. (2018) *Metal 009* Available at: https://ambientcg.com/view?id=Metal009 (Accessed: 4 December 2024)

Demes, L. (2020) *Terrain 003* Available at: https://ambientcg.com/view?id=Terrain003 (Accessed: 9 December 2024)

Demes, L. (2021 a) *Bricks 066* Available at: https://ambientcg.com/view?id=Bricks066 (Accessed: 27 November 2024)

Demes, L. (2021 b) *Pastry 002* Available at: https://ambientcg.com/view?id=3DPastry002 (Accessed: 8 December 2024)

Demes, L. (2022) *Wood Floor 053* Available at: https://ambientcg.com/view?id=WoodFloor053 (Accessed: 4 December 2024)

Desmos Studio (no date) *Graphing Calculator* Available at: https://www.desmos.com/calculator (Accessed: 10 December 2024)

De Vries, J. (2014 a) *Bloom* Available at: https://learnopengl.com/Advanced-Lighting/Bloom (Accessed: 5 December 2024)

De Vries, J. (2014 b) *Light Casters* Available at: https://learnopengl.com/Advanced-Lighting/Bloom (Accessed: 5 December 2024)

Fernando, R. (2004) *GPU Gems* Boston: Addison Wesley

Luna, D. (2012) *3D Game Programming with DirectX 11* Herndon, VA: Mercury Learning and Information

McReynolds, T. and Blythe, D. (2005) *Advanced graphics programming using openGL*. 1st edition. San Francisco: Elsevier Morgan Kaufmann Publishers.

Microsoft (no date a) *Direct3D 11 Graphics* Available at: https://learn.microsoft.com/en-us/windows/win32/direct3d11/atoc-dx-graphics-direct3d-11 (Accessed: 26 November 2024)

Microsoft (no date b) *Direct3D 11 Graphics* Available at: https://learn.microsoft.com/en-us/windows/win32/api/_direct3d11/ (Accessed: 26 November 2024)

Microsoft (2019) *Texturecube representation* Available at: https://learn.microsoft.com/en-us/windows/win32/direct3d11/images/d3d10-resource-texturecube.png (Accessed: 10 December 2024)

mtrebi (2017) *Tangent Space To World Space (TBN Matrix)* Available at: https://stackoverflow.com/questions/42137397/tangent-space-to-world-space-tbn-matrix (Accessed: 27 November 2024)

Scaramozzino, M. (2018) *TestNormalMap.png* Available at: https://dreamlight.com/how-to-create-normal-maps-from-photographs/ (Accessed: 4 December 2024)

Wikipedia (no date a) *Gaussian blur* Available at: https://en.wikipedia.org/wiki/Gaussian_blur (Accessed: 19 November 2024)

Wikipedia (no date b) *Specular highlight*. Available at: https://en.wikipedia.org/wiki/Specular_highlight (Accessed: 30 December 2024)

Xiao, T., Deng, J., Wen, C. and Gu, Q. (2024) *Schematic of cube map texture sampling.* Available at: https://www.researchgate.net/figure/Schematic-of-cube-map-texture-sampling-Assuming-the-geometric-center-of-a-unit-cube-as_fig8_377728556 (Accessed: 10 December 2024)