**Assessment Title: 3D Scene using DirectX**

***Note: The exe file can be found in the Debug folder named “DX11Proj.exe”. It can also be run via the DX11Proj.sln file in the main folder.***

**Water Effect**

**Discussion of Technique**

The technique is a collection of vertices that undulate forming the appearance of waves. The displacement is formed via the calculation of normal, tangent and bitangent. The wave structure consists of frequency, amplitude, phase, and the overall direction. The sine function is then used to work out the position for where the vertex should now be. Frequency would be how often the ‘wave’ pattern is repeated, amplitude being the height of said wave, phase being its relation to speed and the direction being which direction the wave will be ‘moving’ (though again technically this is just an illusion as the wave is only going up and down). The reason why the normal is also worked out is to allow for a rippling effect on the surface of the water. The cubemap previously made for the skybox is also used to show the reflections of the environment for the surface of the water.

Another improvement on the quality of the water texture is to use a Fresnel bias to effect how the reflective nature of the water is changed by the viewing angle. This allows a blending between the waters natural colour and the reflections. This is adjusted in the pixel shader. The delineation between its deep and shallow colours is also handled within the pixel shader.

The final addition is the creation of a composite of normal maps done at (in this project) three resolutions. This stops the water from having an easily spotted repeated pattern.

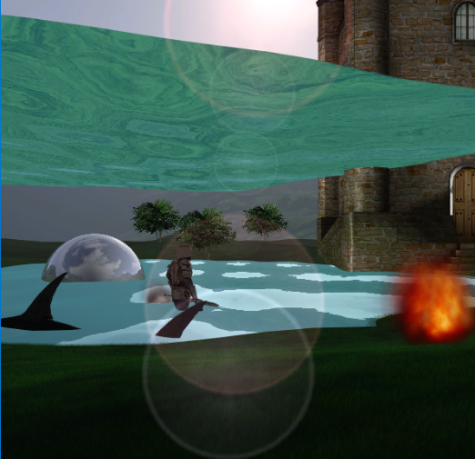
An improvement suggested in “Simulation of Realistic Water on 3D Game Scene”1 that could be made upon this system is a denaturing of the cubemap via bump maps in an animated scrolling texture.

**Discussion of Efficiency**

I recorded the Average FPS and SPF after five minutes on a scene that had the single water technique, then with two, and finally with ten.

|  |  |  |
| --- | --- | --- |
| Amount of Water | FPS | SPF |
| One | 248.468 | 0.00402482 |
| Two | 247.921 | 0.00403425 |
| Ten | 283.006 | 0.00353373 |

This shows a change of 34.5 FPS and a slowing down by 0.00049109 on the extremes. This effect could be mitigated via instancing. However, due to that resulting in identical wave patterns it would be a poor technique choice. As though with instancing the positions would be different the starting position of all values (and therefore the values created by the shader calculations) would be identical if not manually changed.

A screenshot of a video game

Description automatically generated with medium confidence

**Foliage Effects**

**Discussion of Technique**

This technique builds the form via multiple passes of a randomly dotted texture. Each layer will have a higher bias towards alpha. This helps simulate the general tapering found within grass (though could also apply to fur). Within the shader the length of the desired grass length effectively dictates how much alpha is blended into the object with each pass. A higher fade into pure alpha would create a much smaller blade of grass whereas one that faded slowly would show a much slower tapering of the grass blades.

Though initially paired with a simple grid the project instead couples it with a terrain feature. Animation has also been added but this technique will be discussed in the tree section below. The terrain takes both a height and normal map and alters the vertices in a way not too dissimilar to that of the waves. However, instead of reading the positions from a function derived from sine instead does it from the height/normal maps provided. As these maps are easily generatable and storable it is an efficient way to create a large variety of terrains. The harshness of a height map could allow a wide variant of terrains to be made.

**Discussion of Efficiency**

I recorded the Average FPS and SPF after five minutes on a scene that had the following sizes of the grass passes.

|  |  |  |
| --- | --- | --- |
| Number of passes | FPS | SPF |
| 80 | 248.281 | 0.00402779 |
| 160 | 129.001 | 0.00775214 |
| 800 | 26.874 | 0.03716940 |

This shows a change of 221.4 FPS and a slowing down by 0.03314161 on the extremes. The dramatic and evident effect of the number of passes shows a clear limitation on how much detail can/should be obtained via this technique. On this relatively simple scene it has a huge effect. Frustrum Culling, though applicable to many of the techniques discussed would have the largest benefit on this technique. This is due to the fact that grass by its nature is likely to encompass vast tracks of land within a game world.

An alternative technique that would be more efficient would have been billboarding which will be discussed in the following section.

**Foliage Effects (Trees)**

**Discussion of Technique**

Billboarding is the process of making sure a textured quad (textured with the appropriate assets) never can be seen side on by the view – having them rotate or move so they are always face on.

A slightly different variant of the technique that could be used is instead to have a collection of these quads that are arranged in a star shape – a vertex shader would then use to blend (dependant on view) which of the iterations would be shown more prominently to the viewer.

Both of these would suffer from failures of the Painters algorithm, a large number of objects resulting in slower sorting times as well as the inability to function with concave shapes. The solution – which also is faster than purely sorting them – is to instead clip all objects whose alpha reaches a threshold value, though this does cause aliasing artefacts. A remedy for those artefacts is enabling multi-sampling and alpha to coverage blending – in which a non-sorted collection of objects can combine to form something such as a bush by simply blending. Without the need for sorting and without the issues with aliasing artefacts. The ‘Alpha to coverage’ system using a method called stochastic transparency – “A reasonable, though noisy, image is created by using random stipple patterns to represent the alpha coverage of the fragment.”2.

**Discussion of Efficiency**

The efficiency of this technique is much higher than the previous one. However, it is not always applicable and has the issue of being repetitive which is why it is more suitable for instances such as the trees (in the project) or for example bushes.

**Particle Systems**

**Discussion of Technique along with relative efficiency**

There are two particle systems that will be discussed the parametric particle system and the iterative particle system. The first is a system in which all ‘particles’ are described by one (or a series of) calculations. For this reason, it is both more efficient computationally and more predictable. There is also the other benefit that due to them working irrespective of outside conditions or other stimulus they do not need any form of weight, movement, or kill parameters added (that are dependant on something not originally accounted for).

The second system is one in which outside effectors can influence the particles. For example, a prevalent wind could disrupt the shape dynamically produced by the smoke as well as even more complicated behaviours such as collision detection. This does, however, cause a higher amount of bloat and weight for the system as a whole. Each particle does not only need to be individually tracked in relation to outside objects but also in relation to any collision caused by other particles. The vast number of calculations endears itself to being parallel process mostly performed on the GPU – “By using some new features of Direct3D 11, we can maintain the particle system completely on the GPU and simply control the update and rendering of the simulation with the CPU.”3. The interactivity between particles an object is why this form of particle generation is often used for simulations or highly realistic games.

**Lighting and Glow Effects**

**Discussion of Technique**

The glowing technique used has two major components the blurring of the object followed by the recombining between the blurred objected and a light version of the original object. This gives the illusion of the source being a glowing object. The blurring, due to optimisation reasons, is done in two passes first horizontally then vertically. The general order of operations is to render the bright texture off-screen, down-sample to a smaller render target, perform the previously mentioned blending before recombining the blurred texture with the bright texture.

**References**

1. Xu, X. and Zou, K. (2012). Simulation of Realistic Water on 3D Game Scene. Procedia Engineering, 29, pp.1819–1823.
2. Akenine-Mo¨ller, Tomas, et al. *Real-Time Rendering, Fourth Edition*, CRC Press LLC, 2018.*ProQuest Ebook Central*,
3. Zink, J., Pettineo, M. and Hoxley, J. (2016). *Practical Rendering and Computation with Direct3D 11*. Natick Crc Press.