An Evaluation of Volumetric Cloud Implementations in Video Games to Find a Good Balance Between Visual Reality and Performance

Aidan Murray

Computer Game Applications Development
(BSc Hons), 2021

School of Design and Informatics
Abertay University

Table of Contents

Table of Figures

Table of Tables

Acknowledgements

I would like to thank the University of Abertay, Dundee for allowing me to work on this project.

I would also like to thank my supervisor Dr Paul Robertson for the guidance and support he has provided throughout the project and Dr Jackie Archibald, the module leader, for keeping me on track throughout the academic year.

My thanks also go to my family and friends who supported me with their love and support throughout the project.

Finally, a special thank you to all of the people who were able to take part in the survey.

Abstract

Context:

Volumetric clouds aren't commonly used in real-time video game applications due to the performance issues they cause, including slow compute times and memory usage.

Aim:

To enhance the current standard of performance for volumetric clouds while either maintaining the visual realistic look or improving it using various optimisation techniques and different rendering methods.

Method:

A review of the current standard for real-time volumetric clouds was carried out to find the best methods used in terms of rendering and optimisation while also researching methods that had yet to be tested. Using this information, an application was created to test the performance and visuals of these methods. One method that had yet to be tested was if using a smaller box to render the clouds in, instead of a dome that covers the whole sky, would increase the performance as the box could be placed anywhere in the sky. Optimisations that were tested include decreasing the number of noise samples towards the clouds and towards the light in the scene and reducing the pixel count of the clouds as these were the attributes that contributed to the performance the most. For testing the performance of the application, multiple GPUs were used to calculate the compute time of the cloud shader while altering certain settings such as the coverage, distance from the clouds, ray-march step size and the number of steps towards the light to see what kind of effect they have on the performance. To assess the visuals, a survey was made for rating screenshots of the clouds in the application as well as other implementations found to compare how realistic they looked and if the application had achieved visuals fit for a video game.

Results:

This experiment resulted in having an application that could render volumetric clouds in real-time under 2ms compute time, which is the current standard for performance as the total render time for all passes must be below 16.6ms or below to have a game that runs at a minimum of 60FPS. The results for the survey showed that the application wasn't as realistic looking as the previous implementations found, but this could be due to participants judging the realism of the sky as well as the clouds, leading to worse results as the sky was set to be a solid colour in the application.

Conclusion:

This project was worth carrying out as it found that using these methods can achieve good enough looking clouds at a low enough cost but showed that using a render box instead of a dome didn't really affect the performance too much and that having a realistic looking sky also plays a big part in how the clouds look. It has also shown lots of ways that these clouds could be improved in the future to have better-looking clouds in video games.

Abbreviations, Symbols and Notation

GPU - Graphics Processing Unit

fBm - Fractal Brownian Motion

TP - Temporal Reprojection

ms - Milliseconds

FPS - Frames Per Second

1. Introduction

Rendering realistic looking clouds in real-time is essential for certain types of games such as flight simulators and virtual reality experiences as video games are supposed to immerse the player as much as possible and if the player realises that something in a game looks off it can detract from the full experience. In a flight simulator type game the player will most likely be flying through or close to the clouds which will make it much easier to see the problems with how they are rendered. With a virtual reality game, the player has a greater sense of depth which means they would easily notice if the clouds were just textures or if they weren't actually in 3D space.

There are many methods of using clouds in video games currently that wouldn't work in these situations, such as using an image of the sky that has clouds for the skybox. When clouds are put on the skybox texture nothing can pass through them or go behind them which isn't what you would want for the two types of games mentioned previously. For that reason, these games are more likely to use real-time volumetric clouds.

Volumetric clouds have been used in games more recently due to their close resemblance to real-life clouds and the way they form and move in 3D space. There is even a tool in CryEngine 5 that allows developers to quickly implement volumetric clouds into their project and customise aspects such as the cloud coverage, altitude, thickness etc.

However, many games don't use volumetric clouds due to how performance heavy they are, but there are areas for optimisation that have yet to be explored. To find these methods, research has to be done into this area to find the most common techniques used to develop an application that is close in terms of the current standard for performance and visuals for clouds. Once the application has been made with the new optimisations implemented, it can be tested against the current standard of performance and visuals for clouds. For testing the performance, multiple metrics will be considered as there are lots of settings that can affect the performance of volumetric clouds. The main focus for recording performance will be the compute time of the shader to measure only the operations that are happening on the GPU to avoid as many external factors impacting the results as possible. This will also be measured on lower to higher-end GPUs to find how the performance is depending on how powerful the card is. This is important as if the game was made for PC, not everyone that

plays it is going to have the same GPU, so it has to run well on a variety of devices. To judge the visuals a survey will be made, comparing screenshots from the application made to the previous implementations made and real clouds to see how close they end up in terms of realism. There will also be a rating section of the survey to get the average rating on how realistic the participants think the clouds look. This will determine if the clouds look good enough to be implemented in a video game. With this data, the study will conclude whether the researched techniques used were an improvement on the current standard and if they were viable for use in a video game or not based on the visuals and performance. The results will then be reflected on to come up with ideas for further research that can be done in this field.

2. Literature Review

Many steps need to be taken when generating volumetric clouds. One of the most important parts is getting the shape right. This is commonly done by combining layers of texture noise to attempt to recreate the billowy look of clouds. Rendering the clouds can be done with a variety of methods using different lighting models. Optimisation methods for these techniques also need to be considered to ensure the implementation will work in real-time and that it has the performance fit for a video game.

2.1 Shaping the Clouds

One technique commonly used for shaping clouds is using layered noise. In a talk about the video game *Horizon Zero Dawn* (Schneider, 2015), they explain the steps taken to shape their clouds. One being, creating the base shape of the clouds using a 3D texture. Each colour channel (RGBA) of the texture was used to store different sets of noise. The red channel was a combination of *Perlin* and *Worley* noise and the other 3 were used to store increasing frequencies of layered *Worley* noise. *Worley* noise was used as it has a billowy look similar to clouds and can be layered to create more detailed noise. The red channel's *Worley* noise was combined with *Perlin* as it maintains connectedness, making the clouds look wispy. A second 3D texture at a lower resolution with different frequencies of *Worley* noise in 3 different channels is used to add detail to the base cloud shape. Lastly, a 2D texture that uses *Curl* noise to simulate a fluid type motion caused by the turbulence that moves clouds around the sky is used. Depending on the cloud type, they use different height gradients to determine how high in the sky they should generate.

Another paper (Haggstrom, 2018), uses a similar approach where 3D textures are used for the base shape and detail, but they also use a different type of 2D weather map. The red/green colour channels are used for determining where the clouds can form. The red channel contains hand-drawn clouds for low cloud coverage whereas the green channel contains high frequency layered *Worley* noise. The blue channel is used to determine the clouds height and alpha for the density. Whereas, in Schneider's implementation (Schneider, 2015) the red channel was used for coverage, green for precipitation and blue for the cloud type.

The clouds in *Horizon Zero Dawn* (Schneider, 2015) look more realistic due to the *Curl* noise used but lacks when it comes to controlling the coverage due to only using one texture channel to control it. Implementing these techniques would impact performance as there are more textures to be accessed/sampled, however, it would increase the visual fidelity of the clouds. Haggstrom was able to reduce the compute time using their method, but this could be due to them not using *Curl* noise as this would reduce the number of texture accesses and they are also passing fewer textures to the shader.

2.2 Rendering Methods

Many techniques have been used to render volumetric clouds. One is based on how light is scattered within the clouds. When looking through the clouds towards the sun you will see a silver lining around the clouds which is calculated in computer graphics using a phase function. A research paper (Olajos, 2016) talks about comparing the *Henyey-Greenstein* and *Mie* functions as the former is a simpler phase function that is good for use in real-time applications, whereas the latter produces more realistic results, but is too complex for real-time applications. Meaning, for implementing volumetric clouds, using the *Henyey-Greenstein* function is more appropriate.

For 3D rendering with ray-marching/ray-tracing, a method was found using ray-box intersections for rendering within a box in the world-space of a scene, known as the *Kay-Kajiya slabs* method (Kay, T.L. and Kajiya, J.T., 1986). Using this means the area of clouds is determined by the size of the box. This could be better for performance as all ray-marching operations only need to be done within the area of the box and can be ignored if the box is outside of the view frustum of the camera. However, this limits the size of the cloud coverage to the bounds of the box.

To calculate the attenuation of light through a volume, there is a function known as the Beer-lambert law as demonstrated in this presentation (Mayaux, 2013). This calculates how the light energy decreases based on the density of a volume. To get the light extinction for clouds, a ray is marched from the camera to the scene to sample the density at each step along the ray. The amount of light energy is extinguished also needs to be calculated from a ray marching towards the light from each sample point. This can be very expensive as this is only to calculate a single pixel. However, Haggstrom (Haggstrom, 2018) found

that the number of marches towards the light source could be reduced to a single digit which should improve the performance significantly.

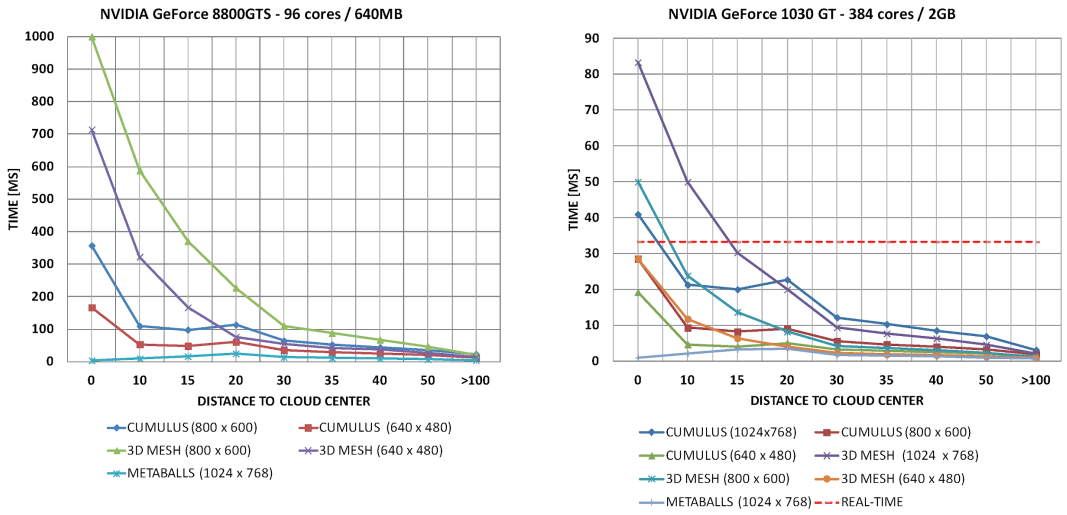
Another method (Schpok et al., 2003) is slice-based volume rendering. This works by creating several 2D textures using noise and applying them to planes that are layered with different opacities. This is a much cheaper alternative to Haggstrom's (Haggstrom, 2018) and Schneider's (Schneider, 2015) method of ray-marching and sampling although it does have many drawbacks. One of those being that the shadowing resolution is limited to the tesselation density of the plane and they also had an issue when using higher octaves of noise for subtracting details which led to the clouds being darkened more than they should have been.

A research paper was found (Bouthors, Neyret and Lefebvre, 2006) that uses a technique that can improve the realism of the clouds. This was colouring the bottom of the clouds based on the colour of the ground below. This was tested by having clouds above water, sand and snow to demonstrate the difference in visuals. Although this is a great way to improve the visuals of the clouds, the performance suffers a lot. They stated when trying to animate the clouds using this method, they could only achieve a maximum of 18 FPS and only 40FPS without animation. This isn't very practical for use in real-time as most games these days run at a constant 30 - 60FPS or above and this was only for the clouds themselves meaning that once the rest of the game is finished, it would perform badly.

Another method is using a mesh as the cloud's base shape, then using ray-marching to render them with some added noise to make them look more cloud-like (Bouthors et al, 2008). This method uses a similar approach with noise generation and sampling as Haggstrom's (Haggstrom, 2018) and Schneider's (Schneider, 2015) papers, but gives the artists more control of the shape of the clouds as they can use any type of mesh. The downside is that using meshes can make the clouds look unrealistic and if the meshes are high poly, it could cause some performance issues if many clouds are present. As this method also isn't procedurally generated, clouds would need to be placed manually unless a second script was written to randomly distribute the meshes in the scene.

2.3 Testing Visuals/Performance

Testing the performance of the clouds on different GPUs is important as when making a game for PC, users can have a range of different GPUs that run at different speeds meaning the clouds have to run well on a variety of different devices. A way to test this is to measure the compute time of the shader with different cloud types, different resolutions and having the camera at certain distances to the clouds as shown in Figure 1 (Parga and Palomo, 2018). Another thing that could be tested is the cloud coverage as this also affects performance.



(a) (b)

Figure 2.1: Cloud performance results calculated on different GPUs which take into account the distance from the clouds, compute time of the shader, window resolution and different cloud types (Parga and Palomo, 2018).

For testing visuals, a survey could be used to get people to compare pictures of real clouds, previous implementations and the new implementation which is proposed here and rate them on a scale of 1 to 5 on how realistic they feel the clouds look. This is important as the people playing the game are going to be looking at the clouds the most, meaning their opinions matter. A problem with this method is having certain participants responding to the survey carelessly, which can produce poor results overall (Ward and Meade, 2017). Data was found (Ward and Meade, 2017) that surveys that inflict cognitive dissonance in the participant increased the accuracy of responses, interest in the survey and how logical the responses were. Whereas in a survey that was hypocritical, the

responses were logically consistent, accurate, but less interest was shown in the survey. They also found that offering gifts to people for completing the survey such as free food and drink didn't increase the quality of results received. For a survey being made to compare cloud realism, providing cognitive dissonance to increase interest in the study is difficult as people don't usually have conflicting attitudes towards clouds, but the second point shows that encouraging more people to participate by offering gifts for completion doesn't provide more accurate results. Although, studies show (Fanning, 2005) that the design in terms of the layout and question order also have an effect on how people treat the survey and it's suggested that putting the most interesting questions at the beginning to pique the participants' interest as soon as possible can produce better results.

2.4 Optimisation Methods

One method when using a ray-marching algorithm is to reduce the sample count, but this introduces some issues such as visual artefacts. A way to combat these artefacts is using *Blue* noise as found in a talk about the visuals for the video game *INSIDE* (Gjoel and Svendsen, 2016). In the talk, they explain how they did volumetric lighting for flashlights and how reducing the samples increased the performance. Doing this introduced obvious looking artefacts. To eliminate these artefacts, spatial dithering was used with different types of noise. They concluded that Blue noise was best suited for the problem as it was random enough to stop humans from recognising patterns within it and it contained all values (between black and white) within small areas in the texture, meaning it is harder to spot the noise when using it for dithering.

Another method is using a compute shader rather than a pixel shader for deferred rendering, as you have much more control over a compute shader and it doesn't go through the entire render pipeline. An example of compute shader optimisations was found (Mallett and Yuksel, 2018) where they proposed a method called *Deferred Adaptive Compute Shading*. They gathered that they could have levels of quality for sampling textures which starts with a step value of 4. By only sampling every 4 pixels, they can then compare the pixels using a similarity criterion. If the pixels were similar enough, they would interpolate between them, otherwise, they would shade the new pixel. By doing this, they achieved up to 4 times the performance of other methods such as

checkerboarding. They also found that because it reduced the number of texture accesses, it increased the texture cache performance. This reduced the quality of the produced image slightly but increased the performance.
Another method of deferred rendering that is commonly used (Haggstrom, 2018) (Schneider, 2015) (Hillaire, 2016) (Toft et al., 2016) is *Temporal Reprojection.* The most commonly implemented version of this is rendering the clouds over 16 frames, where each frame 1/16th of the pixels are updated in a 4 by 4 grid of pixels. The pixel to render in the 4 by 4 grid is determined by a *Bayer matrix* to make it less obvious to the human eye which pixels are being updated. This leaves the issue of the rest of the pixels in each 4 by 4 grid not being updated that frame. This is fixed by reprojecting those pixels to the predicted location that should be based on the current view-projection matrix and the old view-projection matrix from the previous frame.

From what has been gathered from this research, it would seem that when it comes to shaping clouds, noise is the most frequently used method as is ray-marching when it comes to the rendering. Using these methods, lots of optimisations can be applied to increase performance such as GPU threading through a compute shader to interpolate certain pixels instead of sampling every time. Finally, for assessing the visuals, using a survey to rate the realism of clouds and having the most interesting questions at the beginning should produce more accurate results. This should indicate whether the clouds are feasible for use in a video game or not.

3. Methodology

To be able to test different methods of cloud generation by measuring their compute time and visual realism, an application was built which features some of the researched methods. This application was made using C++ along with a framework library for DirectX11 (Robertson, 2018) and using hlsl code for the shaders. This was used rather than a game engine to reduce the chances of other programs affecting the application's performance. The operating system used to develop and test this application was Windows 10 as it is one of the most widely used.

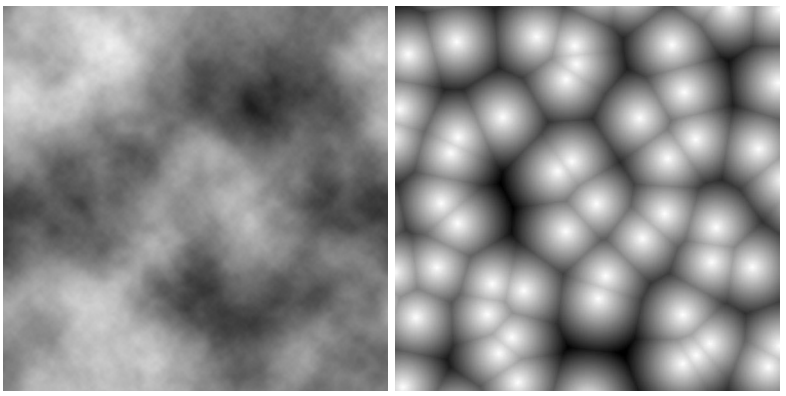
3.1 Shaping the clouds

One of the most important parts of making the clouds look natural is using some form of randomness. However, computers aren't very good at creating randomness and humans are good at recognising repeating patterns. This is where the noise comes in.

3.1.1 Use of Noise

Noise is usually a combination of different wave frequencies with varying amplitudes. These waves can be combined to form textures in many different dimensions (1D, 2D 3D, etc.).

For clouds implementation, as mentioned in section 2.1, it is best to use a combination of *Perlin* (Perlin, 1985) and Inverted *Worley* noise (Worley, 1996) as shown in Figure 3.1.



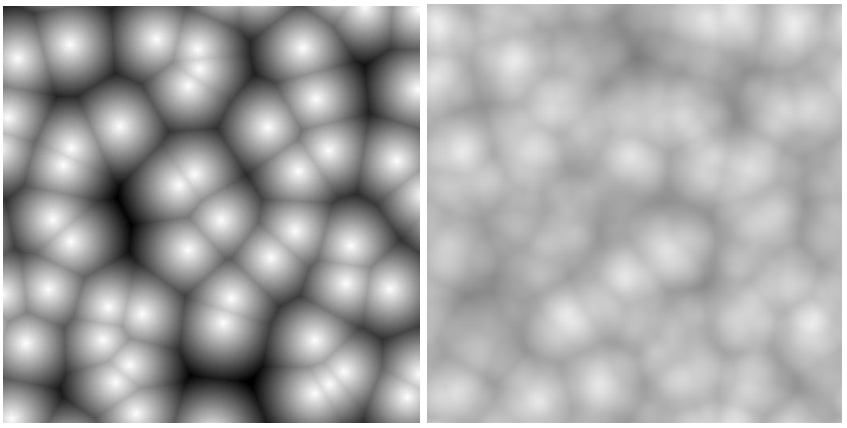
(a) Perlin noise **(b)** Inverted Worley noise

Figure 3.1: Examples Perlin noise and inverted Worley noise

To be able to control aspects of the noise, such as the frequency and number of octaves each texture contains, noise algorithms for 3D Perlin and Worley noise were implemented to generate the textures before running the cloud generation.

3.1.1.1 Layered Noise and Fractal Brownian Motion

When using Worley noise for clouds, many of the 'bubbles' in the texture will look pretty similar in size, but clouds usually feature a variety of large and small bubble-like shapes. To account for this, Worley noise can be generated with a higher number of randomly generated points at a different seed[1] to give a more varied cloud-like result as shown in Figure 3.2.

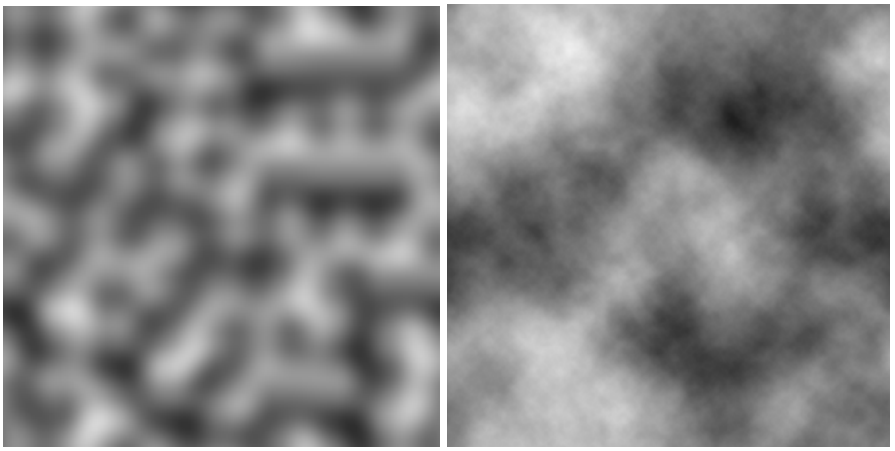


(a) Inverted Worley noise. **(b)** Layered Inverted Worley noise.

Figure 3.2: Comparing inverted Worley noise to layered inverted Worley noise.

Perlin noise can also be given more detail using a technique called Fractal Brownian Motion(Mandelbrot and Ness, 1968). As Perlin noise contains waveforms, these waveforms can be added onto each other at increasing frequencies and decreasing amplitude. The total number of extra frequencies added is referred to as the number of octaves contained within the noise. A demonstration of this can be seen in Figure 3.3. Doing this creates a more detailed wispy looking noise texture which is better for clouds.

[1] The *seed* is a term used to describe the number used to generate random numbers in a computer. Changing the seed changes what random numbers are generated and keeping it the same will always generate the same values.

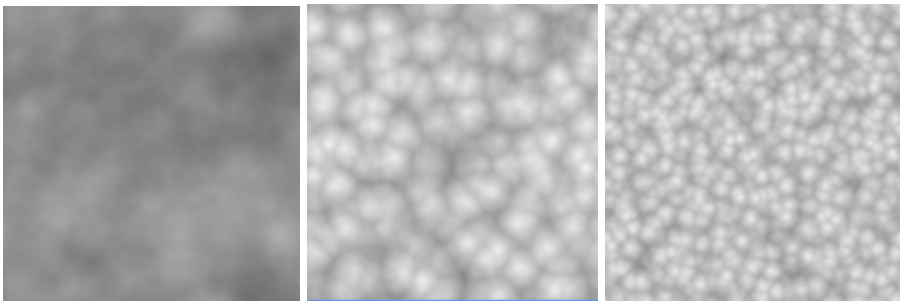


(a) Single octave Perlin noise. **(b)** Multiple octave Perlin noise.

Figure 3.3: Comparing Perlin noise with a single octave and multiple octaves

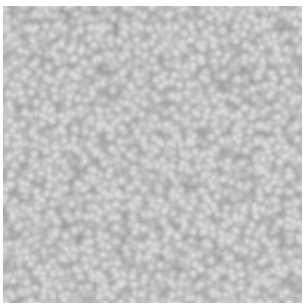
3.1.2 Shape Noise

To create the main shape of the clouds a similar method to Schnieder (Schneider, 2015) and Haggstrom (Haggstrom, 2018) was used. This consists of having a 3D texture and storing different noise values in each channel[2] of the texture. For the red channel, a combination of Perlin and Worley noise is used as Worley noise provides a billowy look while Perlin keeps the wispy connectedness intact. For the green, blue and alpha channels, increasing frequencies of Worley noise are used.



(a) Perlin-Worley **(b)** Medium-frequency Worley **(c)** High-frequency Worley

[2] Textures in computer graphics are most commonly made up of four colour channels, Red, Green, Blue and Alpha. Each of these channels are used to store data about the colour/transparency of a single pixel in the texture, but can also be used to store other data to be looked up by a shader.



(d) Higher-frequency Worley

Figure 3.4: The Shape noise with different frequencies of noise stored in each channel.

The channels are then sampled using fBm, combining them to give more detail to the shape. The persistence of the values from each channel decreases, the higher the frequency, meaning the red channel is most persistent and alpha the least. Figure 3.5 shows clouds rendered only with shape noise.

Figure 3.5: Using only shape noise for the density of the clouds.

However, this makes the clouds look too smooth, almost cartoon-like, which can be good if you want a more stylised look but isn't very realistic.

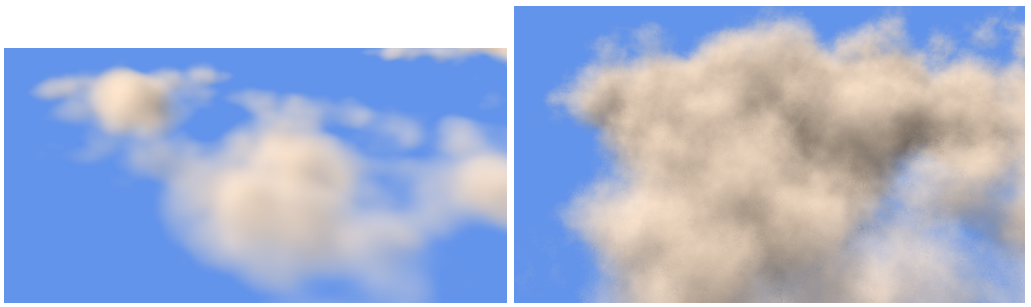
3.1.3 Detail Noise

To add more detail to the clouds, a similar approach to the Shape noise can be taken by storing different noise values in different channels and combining them with fBm. In this case, as shown in Figure 3.6, only three channels are used and Perlin noise is not used for the red channel.

(a) Low-frequency Worley. **(b)** Medium-frequency Worley. **(c)** High-frequency Worley.

Figure 3.6: Detail noise channels with increasing frequencies of Worley noise.

For performance reasons, this texture can be stored in a much lower resolution. Only three channels are used this time, as adding a fourth would deteriorate the features of noise. This texture is then used to erode parts off of the shape of the clouds as the comparison shows in Figure 3.7.



(a) Clouds without Detail noise. **(b)** Clouds with Detail noise.

Figure 3.7: How the Detail noise erosion affects the clouds.

3.1.4 Edge Fading

As this implementation uses a box to render clouds inside, a problem arises where the clouds are suddenly cut off at the edge of the box as shown in Figure 3.8.

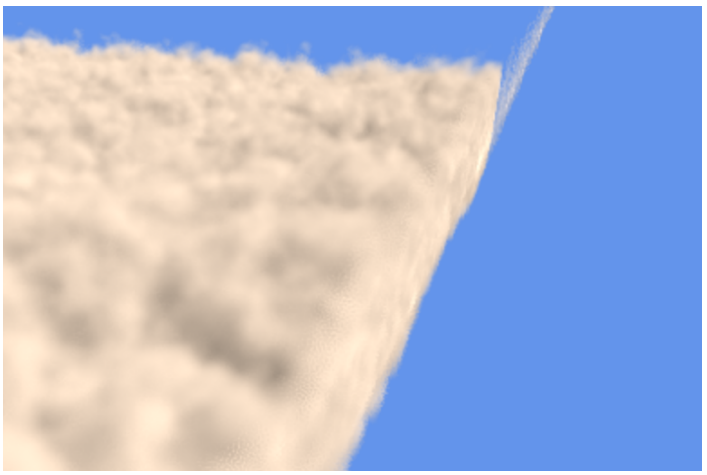


Figure 3.8: Clouds being cut off at the edge of the render box.

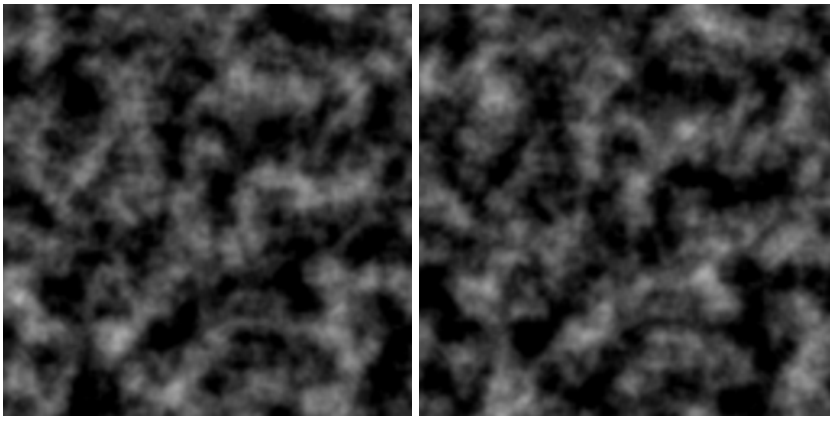
The sharp edges here make the clouds look unrealistic. To counter this, an algorithm was made to fade the height from a certain percentage in from the edges of the box in the X and Z axis. This is done by finding the sample position inside the box and fading it by the edge fade percentage if it's in the area where fading should be applied. This makes the clouds at the edge of the box look more natural as shown in Figure 3.9.

Figure 3.9: Clouds with fading at the container edges.

3.1.5 Weather map

The weather map is a 2D texture made for controlling the cloud coverage and height distribution across the sky. This is used to make the clouds look natural as it eliminates the repetition caused by sampling the tiled shape and detail noise.

The red channel controls the coverage in certain areas of the sky and the green channel controls the height value as shown in Figure 3.10.



(a) Coverage channel. **(b)** Height distribution channel.

Figure 3.10: Weather map channels made using different octaves of Perlin noise and subtracting them from each other.

The coverage and height channels use different frequencies of layered Perlin noise with different octaves (To make them look different from each other). The red channel is set to be the height noise subtracted by the coverage noise and the green channel is set to be the coverage noise subtracted by the height noise. This adds more randomness to the noise while keeping the clouds spaced out as shown in Figure 3.10.

These channels can also be hand-painted by artists to give more control over the composition of the clouds. Due to the algorithm using these textures as an additive blend, the clouds will still look natural when there are lines with sharp edges in the Weather texture.

Each channel has its own scale and intensity values. The intensities can be used to control how much each of these textures affects the final clouds generated.

3.2 Rendering

Objects in 3D space are usually positioned in the world-space coordinate system. For rendering volumetric materials such as clouds, there needs to be an area in world-space to render them, so that they can be positioned relative to the other objects in the scene. A ray-box intersection algorithm was used to specify these boundary points in the world that the clouds can be rendered between.

3.2.1 Ray-Box Intersections

Using the Kay and Kajiya slabs method (Kay and Kajiya 1986) a representation of a box can be created by defining 2 points in space. One of these points is the front bottom left corner of the box representing the minimum boundaries of the box and the other is the back top right corner representing the maximum boundaries of the box. Using these points, planes can be projected onto the axes that each edge of the box lies on. The intersection points are given by the location of the intersection between the ray and the planes. This should give the world space locations where the ray entered and exited the box.

Ray-marching can then be used to render the clouds within these boundaries. The number of marches is reduced too as it only needs to sample within the boundaries of the box.

An issue with this is that it doesn't curve at the horizon leading to it being slightly unrealistic as shown by the comparison in Figure 3.11.

(a) Real clouds curving towards the horizon.

(b) Implemented clouds not curving towards the horizon.

Figure 3.11: Comparison between real and implemented clouds and how the render box doesn't curve at the horizon.

However, having a smaller area of clouds to render should enhance performance as sampling is only happening within the box.

3.2.2 Ray-marching

Ray-marching is the process of casting a ray and incrementing the ray length for a set number of steps to sample something at that ray position.
The starting ray distance is set to the distance at which the ray first intersects the box in section 3.2.1. If the ray has left the render box, the ray-marching can be terminated early as there is no need to sample outside the box. An example of what this would look like is shown in Figure 3.12.

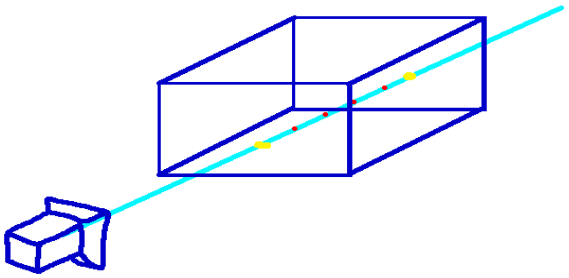


Figure 3.12: Ray from the camera marching through the render box and sampling at the points marked in red. The yellow points are the points where the ray intersects with the box.

3.2.3 Sampling the Density

The density is calculated by sampling the Shape and Detail textures using the world-space position of the ray at each step through the box. This position is scaled and offset by user-defined values so the size of the clouds and how much detail is eroded from them can be altered. The offset can then be used to scroll along textures which can be used to animate the clouds.

3.2.3.1 Edge and Height Density Distribution

To get the shape density of the clouds, the edge fade value is calculated using the current position value as explained in section 3.1.4. The height gradient value is then calculated to prevent the clouds from being cut off at the top of the

render box and so different height values are used depending on the height values from the weather map.

This is done by getting the height percentage of the ray-march position in the container. Equation 1 is used to calculate this height percentage, where Hp is the height percentage, py is the Y value of the ray position in world-space, bmy is the Y position of the bottom edge of the render box in world-space and ch is the container's height.

$$Hp = (py - bmy) / ch \quad (1)$$

Minimum and maximum values for the height are set. The minimum is set to 20% whereas the maximum is set to the minimum value added to the height sampled from the weather map. These are used for determining density based on the height within the container. This is done using the saturate and remap functions. The saturate function clamps a value between 0 and 1. The remap function remaps a value from a specified range to a new range. Using these functions, the height percentage is remapped from between 0 and the minimum value to between 0 and 1, creating a linear interpolation of depth going from 0 to 1 when the height percentage is between 0% and the minimum value(20%). If the height percentage is greater than the minimum value, then the output value of the function would be above 1 which is why the output is fed into the saturate function, to clamp it back to 1.

The same is done with the height percentage and the maximum value, except this time it's remapping the value from between 1 and the maximum to between 0 and 1. This makes it so when the height percentage is greater than the maximum percentage it fades out towards 0 once the height percentage reaches 1.

These two saturated values can then be multiplied together to give the height gradient value which determines the depth percentage based on the height of the sample point in the render box. The height gradient is then multiplied by the edge fade value and height percentage subtracted from 1 to make the clouds look wispier towards the bottom (Schneider, 2015). Doing this gives the result shown in Figure 3.13.

Figure 3.13: Clouds with edge fading and height distribution applied.

3.2.3.2 Cloud Coverage

The shape noise texture is sampled using the coordinates calculated with the offset and scale of the world-space ray position. The colour channels are then combined using fBm by getting the dot product between the channel values and the normalised weights set for each channel. This is multiplied by the current density percentage from the edge and height gradients to make the density depend on the values stored in the shape noise texture.

The final shape density is calculated using the global coverage value and the coverage sample from the weather map.

If the shape density is less than or equal to 0, the Detail noise isn't sampled as should be no clouds at this location. The Detail noise is then sampled using an offset and scale and combining the final result together with fBm. Although, instead of adding this onto the shape density, it is subtracted to erode parts of the clouds as explained in section 3.1.3.

3.2.4 Lighting

For performance reasons, if the density is less than or equal to 0, no light calculations take place as there shouldn't be clouds at this position. To calculate the amount of light that gets absorbed through the clouds, a simplified version of the Beer-Lambert Law is used.

3.2.4.1 Beer-Lambert Law

The Beer-Lambert Law is used for calculating the extinction of light energy as it travels through a medium. The simplified version used here is shown in Equation 2 where d is the sampled density at the current ray-point, e is Euler's number, Le is light extinction and ab_c is the absorption of light through the medium (Haggstrom, 2018).

$$Le = e^{-d \times ab_s} \tag{2}$$

The extra value for the absorption of light through the clouds towards the observer is a user-defined value between 0 and 1 which can be altered to change the strength of the absorption. Figure 3.14 shows how the Beer-Lambert law looks in graph form.

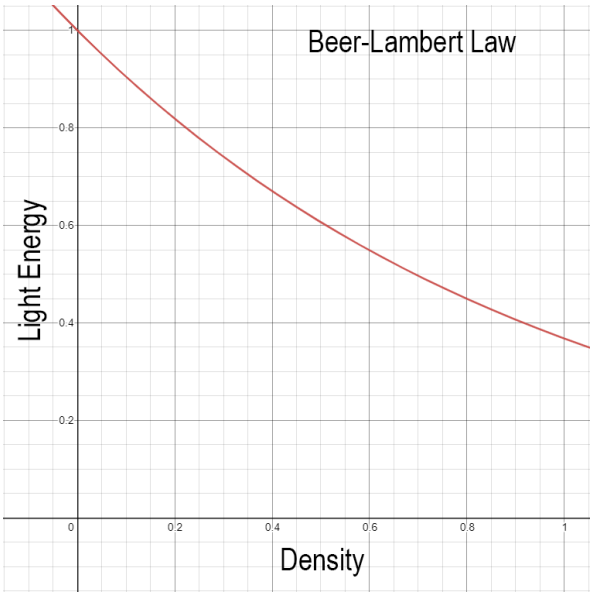


Figure 3.14: How the light energy decreases based on the density of the medium.

Figure 3.14 shows that as the density increases, the light energy that gets through decreases exponentially.

Applying this to the ray-marching algorithm that was made and accumulating the light extinction per step gives the transparency value for the clouds as shown in Figure 3.15.

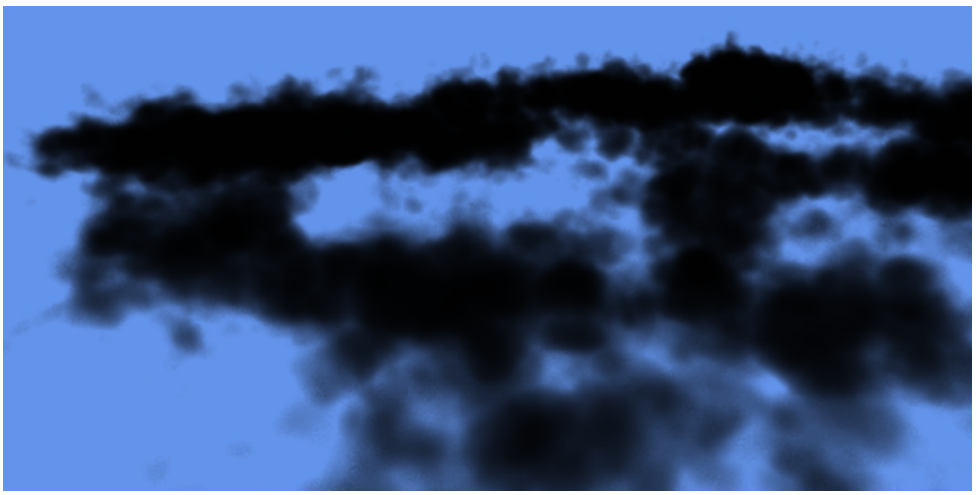


Figure 3.15: Extinction of light energy based on the density of the clouds in space.

If the light extinction is below 0.01, the ray stops marching as sampling beyond this point won't make much difference. This is also better for performance as it doesn't march all the way through the container.
This is not enough information to light the clouds. To do this, a secondary ray-march has to take place, marching towards the light source.

3.2.4.2 Light Marching

The light-march marches a ray from the current sample point to the direction of the light source/sun. This samples the density of the clouds that the light has to travel through to get to the sample point which affects how much light energy is transmitted. An example of this can be seen in Figure 3.16.

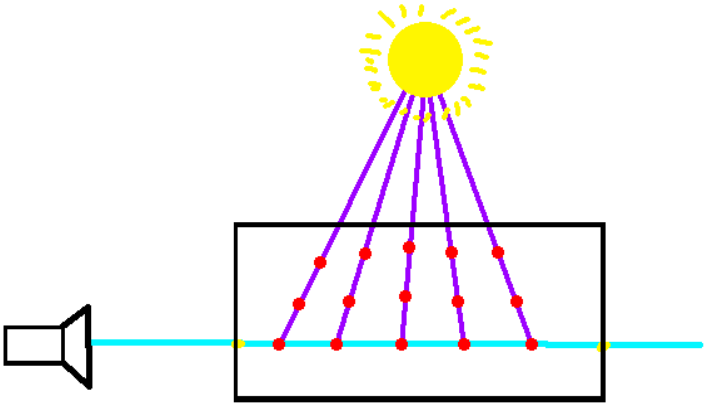


Figure 3.16: Marching through the container towards the sun at every sample point.

This can be expensive as, for each pixel in the render texture, several points are sampled from the inside of the container from along the view-vector and for each sampled point, multiple density samples are being taken towards the light. Even though this is expensive, this was the best method found for the most realistic looking lighting and there are many ways to optimise this method.

The light-march uses the Beer-Lambert Law the same way as the other ray-march, except it has its own absorption value.

This still isn't enough information to light the clouds light isn't just absorbed it is also scattered.

3.2.4.3 Scattering and Phase functions

As clouds are made up of small water droplets, only a small amount of light is absorbed and the rest is refracted/reflected instead. Two commonly used terms for volumetric lighting are, in-scattering and out-scattering.

In-scattering is when the light waves being reflected/refracted, end up in phase with other light waves causing the light energy to be greater on the opposite end than the light is from the medium. This effect is similar to how a magnifying glass can be used to concentrate light waves into a single point to burn objects (Mayaux, 2013). Due to the reflective/refractive properties of water/ice, the clouds appear to be very bright. An example of in-scattering is shown in Figure 3.17.

Figure 3.17: In-scatter through clouds when looking through towards sun

Out-scattering is when the light waves bounce back in the direction of the light which creates areas of the cloud that, when looking through towards the sun, are darker as shown in Figure 3.18.

Figure 3.18: Out-scatter causing darker areas of clouds where light bounces away from the camera.

A phase function is used to calculate the probability of the light waves being in phase after bouncing. As mentioned in section 2.2, the *Mie* phase function is a complex model used for calculating real physical properties of phasing light waves but is too performance heavy to be implemented in real-time (Olajos, 2016). *Henyey-Greenstein* function is an approximation of the Mie function which is much less complex and better for performance. The *Henyey-Greenstein*

function can be seen in Equation 3, where θ represents the angle between the view-vector and the direction to the sun and g represents the scatter value which goes between -1 and 1 (out-scatter amount and in-scatter amount).

$$P_{hg}(\theta) = \left(1 - g^2\right)/(4\pi(1 + g^2 - 2gcos(\theta))^{3/2}) \quad (3)$$

To make the in-scatter more defined, a value for the silver lining intensity and exponent are set by the user (Haggstrom, 2018). An example of the silver lining effect can be seen in Figure 3.19.

Figure 3.19: Areas on the edge of the clouds appearing brighter. This is the silver lining effect.

Equation 4 shows how this is calculated with is_{extra} representing the extra phase value that is going to be calculated, sl_i representing the silver lining intensity and sl_e representing the silver lining exponent. Changing the silver lining exponent will change how sharp the drop off of the silver lining effect is.

$$is_{extra} = sl_i \times saturate(cos(\theta))^{sl_e} \quad (4)$$

The final in-scatter value is then given by the maximum value between Equation 4 and the result of the *Henyey-Greenstein* function using the in-scatter as the g value.

The final phase value is then calculated by interpolating between the final in-scatter value and the phase value from the out-scatter using a blend value set

by the user. This is calculated outside of the ray-marching loop for performance reasons and is added to the result of the light march.

3.2.4.4 Calculating the Pixel Colour

The pixel colour is initially set to the source image from the scene. It is then multiplied by the light extinction to remove colour where the clouds are going to be rendered. The accumulated light energy through the light-march and initial ray-march is multiplied by the diffuse colour of the light in the scene and added onto the final pixel colour.

3.3 Optimisation Methods

The application so far will produce nice looking clouds but at the cost of performance. The method used for ray-marching is similar to Haggstrom's implementation (Haggstrom, 2018), therefore the equation that they used to estimate the execution time can be used as a guide on what should be altered to enhance the performance. Equation 5 is the equation used by Haggstrom, where t_{tot} is the total execution time, p is the number of pixels to render, n is the number of steps marched by the ray inside the container, s_n is the number of steps towards the sun, t_{d0} is the time its takes to sample the density at each sample point from the viewing ray and t_{d1} is the time it takes for each step towards the light source.

$$t_{tot} = p \times (n \times t_{d0} + n \times s_n \times t_{d1}) \quad \text{(Haggstrom, 2018)} \qquad (5)$$

3.3.1 Rendering at Half the Resolution

As you can see in Equation 5, the pixel count has the largest effect on the performance. The simplest way to quickly improve the performance is to half the resolution of the clouds and scale them up to the game window (Schneider, 2015) as they don't need to be the same resolution as the rest of the game and can be used as an overlay. This doesn't affect the visuals of the clouds too much as they already have blurry/wispy like properties as seen in the comparison in Figure 3.20.

(a) Full resolution clouds.

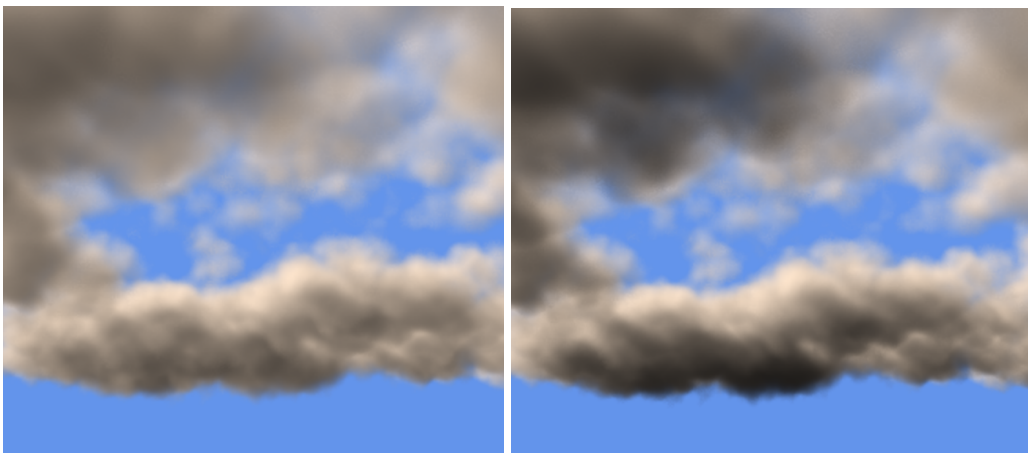
(b) Half resolution clouds.

Figure 3.20: Comparing the clouds after halving the resolution.

The next step is to reduce the other parameters such as the number of steps towards the sun.

3.3.2 Reducing the Steps Towards the Sun

When altering the number of steps taken towards the sun, a lot of the details in the clouds are lost due to the ray not travelling through enough of the clouds and not getting enough samples, but there is a point where increasing the number of light steps seems to make little difference as shown in the comparison in Figure 3.21.



(a) Using 4 steps. **(b)** Using 30 steps.

Figure 3.21: Comparing the visuals of clouds based on the number of marches towards the sun.

As shown in Figure 3.21, the clouds still look good when there are only 4 steps towards the sun, which should improve the performance significantly. The only difference being the one with more steps is slightly darker. This can be adjusted using the brightness settings of the clouds to make clouds with fewer steps darker if needed.

3.3.3 Using Blue Noise

Increasing the step size should also increase the performance as the ray will exit the container faster if the steps are larger. This causes the artifacting present in Figure 3.22.

Figure 3.22: Artifacting caused by large step sizes. (More clear when the camera is in motion as you can see the clouds moving when the camera moves)

This is due to there being larger distances between the samples, which even worse when the camera is in motion as the distance is dependent on the camera's location.
To counter this, *Blue* noise is used. The reason for using this type of noise over other types is explained in section 2.4. The Blue noise texture used for this implementation is shown in Figure 3.23.

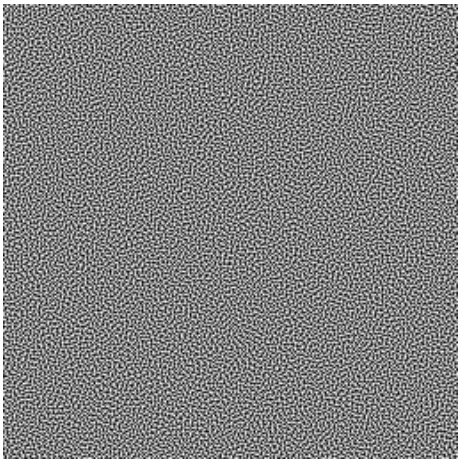
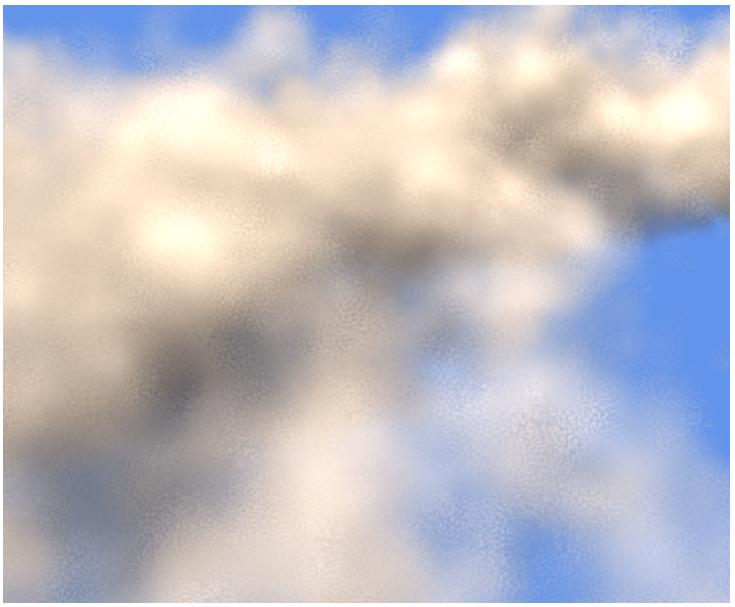


Figure 3.23: Blue noise texture used to reduce artifacting.

This is sampled using the screen-space coordinates of the cloud render texture which are scaled to maintain the aspect ratio of the texture. The colour sampled from this texture is then multiplied by a strength value to increase/decrease how much it affects the clouds. This new colour value can be used to offset the starting distance of the ray-march which eliminates the artifacting as a bit of randomness has been added to the distance. This is also done with the light marching too. Figure 3.24 shows a comparison of the clouds with a longer step size with and without the *Blue* noise.

(a) Artifacting caused by large step size.



(b) Reduced artifacting using Blue noise.

Figure 3.24: Comparing artifacting before and after using Blue noise.. (Again, the difference is much easier to see when in motion)

3.3.4 Dithering and Temporal Reprojection

One of the most common optimisation methods used for volumetric clouds as mentioned in section 2.4 is *temporal reprojection*. First, a value is set keeping track of how many frames have passed and is reset every time it reaches 16, as this is the number of pixels present in the 4 by 4 area of pixels in the render

texture. This value is used as the index for a 4 by 4 Bayer Matrix array[3]. In the cloud shader, it is compared to see if the current index of the thread within the group has the same value. If it does, that pixel is calculated, if not, it's skipped over for that frame.

Theoretically, this should be 16 times faster as now only 1/16 of the pixels are being calculated each frame.

The issue with this is that it introduces a ghosting effect when the clouds are moving or when the camera is moving as seen in Figure 3.25.

Figure 3.25: Ghosting effect caused by only rendering 1/16th of the pixels per frame.

This is caused by the other 15 pixels per group each frame that aren't being updated and this only gets worse with faster movement and slower frame rates. The method for fixing this is to have a separate pass that reprojects the pixels to the correct place.

This is done by first using the same size thread groups and Bayer matrix with the same values to calculate if the current pixel is the one being calculated by cloud shader. If it is, the resulting pixel is set to be the colour of the pixel sampled from the current frame. If not, the pixel has to be reprojected.

Using the render texture and view-projection matrix from the previous frame, the screen-space coordinate of the old pixel can be found and used to create a motion vector.

[3] A *Bayer Matrix* or *Bayer Filter* is commonly used for dithering in computer graphics due to the structure it uses, making it less detectable to the human eye.

To find the coordinate of the old pixel, the screen-space coordinate for the current pixel is converted to world-space using the inverse of the current view-projection matrix, assuming the pixel is at the maximum depth point (on the far plane). This world-space position can then be converted back into screen-space using the previous view-projection matrix to get the old screen-space coordinates.

Instead of setting the pixel colour to be the same as the old one, the vector between the current screen-space coordinates and the previous ones is calculated. Using the length of this vector, it can be divided up into a number of steps. This way there can be multiple samples towards the old coordinates by stepping the ray towards it and sampling the colour along the way. The colour is then divided by the number of steps to get the final colour value. The result of this can be seen in Figure 3.26.

Figure 3.26: Results after reprojecting the pixels.

This isn't perfect as there is still some artifacting when the camera moves fast, but this should be able to be fixed using effects like motion blur or anti-aliasing.

The other method of dithering explained in section 2.4, *Deferred Adaptive Compute Shading* (Mallett and Yuksel, 2018), wasn't used as this would end up being more expensive due to the use of the slight difference in the colours between the pixels caused by the use of Blue noise. If the pixels aren't similar enough, the algorithm renders more pixels, leading to worse performance than the other method where all 1/16th of the pixels are being drawn each frame.

There's also the option of applying this method over temporal reprojection, but this could make the performance worse as the temporal reprojection method is already fast enough when calculating the motion vector for each pixel and sampling the colours. Adding more dithering could add extra complexity leading to slower compute times.

3.4 Testing

The two main aspects of this project that need to be tested are the compute time of the shader in different scenarios and how realistic the visuals of the clouds are.

3.4.1 Measuring Compute Times

As mentioned in section 2.4 (Parga and Palomo, 2018), the cloud shader can have varying performance depending on the distance from the centre of the clouds, the coverage, step size, the number of light steps and even the GPU used.

Each attribute was tested by increasing the values by an increment value and recording the average of 100 compute results at each step. These values were then stored in a CSV file as they can then be used as a spreadsheet and converted to graphs. Before recording any of these values, the camera was positioned looking directly up into the clouds to act as a constant. These tests were performed with and without temporal reprojection enabled to see how it affects the results. It was also important to test these on different GPUs to see how the results differed as if these clouds were being made for a PC game, they would need to be able to run well on a variety of different devices. For these tests, the GPUs used include:

- NVIDIA RTX 2070 8GB
- NVIDIA GTX 1080 8GB
- NVIDIA GTX 1660 Super 6GB
- NVIDIA 1060 6GB

These were selected as they range from high end to low end in terms of performance and memory.

These graphs can then conclude how much these values actually impact the performance and how dependent the performance is on the GPU used.

The compute times were recorded using DirectX11's GPU queries to only measure the performance of the shader itself. The frame rate of the program is

not being used here as a performance metric as there are too many external attributes that can affect this such as background applications running while recording the results.

3.4.2 Survey Design

To evaluate how realistic the clouds look, a survey was made. This survey contains two sections. The first section has the participant rating images of clouds from 1 to 5 based on how realistic they feel the clouds look. In total there are 13 images:

- 4 of this implementation.
- 3 of real clouds
- 3 of Schneider's implementation (Schneider, 2015).
- 3 of Haggstrom's (Haggstrom, 2018).

Images from Schneider's and Haggstrom's implementations were used as they are the closest to the current standard of graphics for volumetric clouds that were found.

The purpose of this section was to see how well people were able to differentiate real and fake clouds using the rating system. The other implementations were put in to see how well people rate them which can then be used to judge how close this implementation is to them in comparison. The order of these images was also randomised to stop people from realising which clouds were real and fake. However, the first image was set to be one of the real cloud images so that the participant would mainly base their judgement of the other images off of this one.

The second section of the survey is set up to compare different implementations. There are 12 questions in total:

- 2 comparing this implementation to real clouds.
- 2 comparing this implementation to Schneider's.
- 2 comparing this implementation to Haggstrom's.
- 2 comparing Haggstrom's to real clouds.
- 2 comparing Haggstrom's to Schnieder's.
- 2 comparing Schneider's to real clouds.

This way there are two comparisons for every combination. There are two for each as a different picture is used for each second comparison to make the judgment fairer, as judging the implemented clouds based on a single image may not show the clouds from the best angle. The results from these answers

can then be averaged out to find the most realistic looking clouds to the least realistic looking clouds. This differs from the first section as the participant is directly comparing the clouds whereas the first section is about rating the realism. This means that even the lowest-rated clouds in the second section could still have a high enough rating in the first section to be considered realistic enough to be implemented in a video game even if it doesn't look as real as the other images.

4. Results

This section is to show the results gathered from testing the application and the survey mentioned in section 3.4.2.

4.1 Application Tests

All tests were recorded using the mean average of 100 compute times for each recording. Each graph has 20 of recordings which increments the subject value for that graph. Table 4.1 shows the specifications of the GPUs that were used to test the compute times of the shaders.

Brand	Model	Base Clock Speed (MHz)	Memory	Cores
NVIDIA	GTX 1060	1506	6GB	1280
NVIDIA	GTX 1660 Super	1530	6GB	1408
NVIDIA	GTX 1080	1607	8GB	2560
NVIDIA	RTX 2070	1410	8GB	2304

Table 4.1: GPUs used to test the application.

4.1.1 Coverage Test

The results for these tests can be found in A.1 of the Appendix.

This shows the increasing coverage against the average compute time recorded. These graphs display a bell curve-like shape where compute time starts low and at about half coverage, it peaks then begins to fall again as the coverage increases more.

The compute time when using temporal reprojection is better in most cases, but seems to be more expensive for the GTX 1060 card.

4.1.2 Distance Test

The results for these tests can be found in A.2 of the Appendix.

This shows the distance from the camera to the centre of the render box against the compute time.
The graphs show that as the distance from the box increased the compute time increased. The high-end GPUs seem to handle the increasing distance better than the lower-end as their compute times begin to plateau whereas the lower ones look like they will continue to increase.

4.1.3 Light Steps Test

The results for these tests can be found in A.3 of the Appendix.
This shows the light steps starting at 0 and increasing towards 10 against the compute time. The graphs show similar results to the distance test where the number of light steps increases, the compute time also increases. The GTX 1060 didn't handle this test well as it stays at above 5 ms for most of the test even with temporal reprojection enabled.

4.1.4 Step Size test

The results for these tests can be found in A.4 of the Appendix.
This shows the step size of the marching ray increasing towards 10 against the compute time. The graphs here show that the compute time decreases at an exponential rate as the step size increases and plateaus at a step size of ~4.5.

4.2 Survey Answers

The first question was asking the participant if they were visually impaired or not as this could affect how they rate the images. As shown in Figure 4.1, the majority of the people who took part in the survey were not visually impaired meaning that the results should have less chance of error.

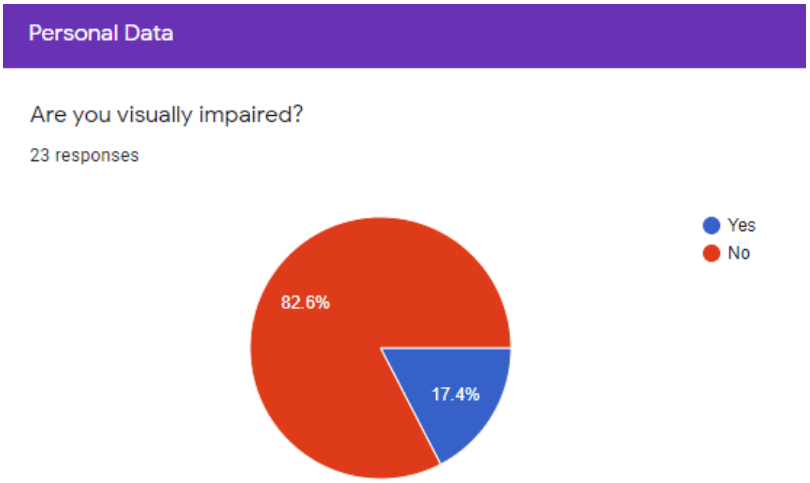


Figure 4.1: Checking whether the survey participant is visually impaired.

4.2.1 Survey Section 1

The full results for this section can be found in B.1 of the Appendix.

This data was processed by getting the weighted average from each question and adding it to Table 4.2. (This application's cloud pool has one extra image to gather as much data as possible about it.)

Image Pool	Image 1 Weighted Average Rating	Image 2 Weighted Average Rating	Image 3 Weighted Average Rating	Image 4 Weighted Average Rating
This Application's Clouds	2.90	3.09	2.64	2.64
Haggstrom's Clouds	4.27	3.86	3.64	
Schneider's Clouds	4.41	4.32	4.27	
Real Clouds	4.41	4.50	4.50	

Table 4.2: Weighted Ratings from survey section 1.

This data can then be used to calculate the average rating for each image pool. The average ratings are shown in Table 4.3.

	This Application's Clouds	Haggstrom's Clouds	Schneider's Clouds	Real Clouds
Score:	2.82	3.92	4.33	4.47

Table 4.3: Final average ratings for survey section 1.

4.2.2 Survey Section 2

The full results for this section can be found in B.2 of the Appendix.
As this section has questions comparing images from the different image pools, a point system was made. This is done by assigning 1 point to the image in the question that had the highest number of votes and for questions that had the same number of votes for each image, both the images in the question get half a point each. Table 4.4 shows the results of this processed data.

	This Application's Clouds	Haggstrom's Clouds	Schneider's Clouds	Real Clouds
Score:	0	2	4.5	5.5

Table 4.4: Processed data from survey section 2.

5. Discussion

This section will go over the results of section 4, analysing the data found and explaining why the results ended up this way.

5.1 Application Test Results

These tests demonstrated a wide range of results from the different scenarios given, meaning they all affect the performance in their own way. As mentioned before, temporal reprojection theoretically should make the algorithm 16 times faster when enabled as it's only rendering 1/16th of the pixels. However, due to the optimisations used, such as exiting early from the ray-marching algorithms, the results won't show the times being 16 times faster as some pixels take a shorter amount of time than others because of this.

5.1.1 Coverage Tests

These graphs mostly consist of bell curve-like shapes. This is because when coverage is low, the majority of the ray-marches won't sample any density meaning it skips the Detail noise and lighting code for that pixel. As the global coverage increases towards ~0.5 the timings peak due to there being more pixels to sample the Detail noise and perform marches towards the light for. The timings then start to fall again to roughly the same time or lower than before. This is due to the extinction value being so low that the ray-march exits early meaning there is less to calculate for each pixel. This means that to maximise the performance, the coverage value has to either be low and high.

The results with and without temporal reprojection were mostly expected as having it enabled should reduce the compute time. However, the results for the GTX 1060 card show that when it was enabled it performed worse until the coverage was over ~ 0.7. This could be due to assigning a GPU thread to each pixel in the render texture which in this case, the size of the render textures for the clouds and the temporal reprojection passes was 600x338 which is a total of 405,600 threads. So running temporal reprojection uses twice the number of threads. As this GPU had the lowest number of cores, it also had the lowest number of threads to handle the clouds. External attributes such as having multiple monitor setups with different refresh rates may also affect this due to the GPU having more tasks to do.

5.1.2 Distance Tests

These graphs showed the timings increasing the further away the camera gets from the centre of the cloud render box. This is because the render box is filling more of the screen, meaning there are more samples to take and more lighting calculations to do for the pixels where clouds form further out from the centre of the box.

The results for this section were mostly expected, except for the spike in compute time at ~ 55 units from the box. This is most likely caused by an area of clouds with a lot of transparency (an area with about half coverage as explained in section 5.1.1) coming into the camera's view-frustum[4] causing a sudden spike in samples due to lack of light extinction.

Using temporal reprojection here almost halved the timings in each graph. For the high-end cards, this resulted in a maximum compute time of 1.5ms when temporal reprojection was enabled.

5.1.3 Light Steps Tests

These graphs show the timings increasing as the step count towards the light increased. This is due to there being more density samples done for every step towards the light.

There were some interesting results from the GTX 1060 and GTX 1660 Super cards. For the GTX 1660 Super, when temporal reprojection was enabled, the timings quickly plateaued at around 6 steps whereas without temporal reprojection it continued to increase (this can be seen in the GTX 1080 and RTX 2070 results, just not as prominently). This is due to the ray towards the light leaving the render box meaning the number of marches is limited by the step size. This happening when temporal reprojection is enabled and not when it is disabled is most likely due to the fact that temporal reprojection is rendering 1 pixel per 4x4 group of pixels on the render texture each frame, meaning these pixels are more spaced out. There will be some areas of clouds that require more sampling (again due to the transparency/coverage being about 50%), but with temporal reprojection, it jumps past most of these pixels making for a much lower compute time.

[4] The view-frustum is a frustum in 3D space used to calculate perspective and convert it to 2D screen-coordinates to display it on a screen/window.

5.1.4 Step Size Tests

These graphs show that the timings decrease at an exponential rate when the step size increases. This happens as when the steps are bigger, the ray leaves the render box quicker when marching towards the clouds and towards the light from the sample point. Due to the sample count towards the clouds and the sample count towards the light decreasing, this creates an exponential decay in the timings.

The results shown across the different GPUs in these tests were pretty consistent with the high-end cards being only slightly faster than the lower-end. Again, using temporal reprojection seems to be better in terms of performance for this, with the compute time staying lower throughout the experiments.

5.1.5 Results From Final Settings

With this, better compute times were achieved than in Schneider's (Schneider, 2015) and Haggstrom's (Haggstrom, 2018) implementations, but this isn't enough to say the application performs better as there are settings which enhance performance but hinder the visuals. These include:

- Large step size where even using the blue noise would look bad.
- Low step count towards the light, making the lighting look flat.
- Too much cloud coverage, making the sky look unrealistic.
- Too little coverage, meaning you will barely be able to see the clouds.
- Being too close to the render box to the point where you either can't see any clouds or the screen is obscured by a cloud.

To have the clouds look nice and perform well, a balance between these settings had to be found. The settings that were chosen, which were used when taking the screenshots for the survey, are shown in Table 5.1.

	Coverage (0 - 1)	Distance (Metres)	Light Steps	Step Size (Metres)	Temporal Reprojection
Value Used:	0.4	~100	4	2.5	Enabled

Table 5.1: Final cloud settings used.

With these settings, Table 5.2 shows the performance of each of the GPUs used, displaying the clouds at 1200x675 pixels upscaled from 600x338.

	GTX 1060	GTX 1660 Super	GTX 1080	RTX 2070
Compute Time (ms) No TR:	3.85	3.10	1.98	2.11
Compute Time (ms) TR:	5.71	2.12	1.16	1.15

Table 5.2: Final performance information.

These results show that the timings are very reliant on the GPU used. In Haggstrom's experiments (Haggstrom, 2018) they use a GTX 980 TI with 2816 cores, 6GB of memory and a base clock speed of 1000MHz and they were able to get their clouds running at under 2ms at 960x540 pixels. As for Schneider's clouds (Schneider, 2015), it isn't mentioned which GPU is used, but they mention that they had it running under 2ms compute time on the Playstation 4 which has a GPU with 8GB memory, 1280 cores, 800Mhz clock speed and this was recorded at a resolution of 960x540 pixels.

Overall, using methods such as a render box instead of a dome that covers the entire sky and computing the in-scatter/out-scatter phase values outside of the ray-marching algorithm did manage to reduce the compute time to be under 2ms, but failed to match the performance of prior implementations.

This could be caused by a variety of different factors. For example, when recording the final compute time they may have been using different values for each of the settings which would have resulted in different timings. It could also be due to the GPUs working on background applications while taking the recordings.

5.2 Survey Results

After compiling the data from the survey, clear ratings were found from the results on both sections which show how realistic the participants found the images and which image pools they found more realistic than others.

5.2.1 Section 1

The section 1 survey results show that the participants didn't find the application to look as realistic as some of the other implementations, but the average rating for them shows that they didn't think they looked unrealistic either, but somewhere in between real and fake as they were given a rating of 2.82 on average, which is almost exactly in the middle between 1 and 5. The next lowest rating was Haggstrom's clouds (Haggstrom, 2018) which were rated only slightly higher at a score of 3.92 on average, followed by the most realistic implementation, Schneider's (Schneider, 2015), at a rating of 4.33 on average. However, the real clouds were only rated a 4.47 on average meaning that not all participants were able to tell they were real, which impacts how they rated the other images.

The ratings for this application are most likely due to the lack of atmosphere present, whereas the other image sets had an atmosphere. This wasn't included in the application as the clouds were the main focus, but after doing this survey it is clear that having an atmosphere really affects how realistic the clouds look when in the sky and the further away the clouds are, the more they should fade to the colour of the atmosphere. Most participants may have picked up on this as the background for this application was a solid cornflower blue colour instead of a gradient and this could have easily skewed their judgment towards them.

5.2.2 Section 2

In this section, each of the clouds in the data sets was compared against each other to find the order of realism. This resulted in this application having a score of 0, Haggstrom's clouds having a score of 2, Schneider having a score of 4.5 and the real clouds having a score of 5.5.

These results were expected due to the reasons explained in section 5.2.1 about the atmosphere playing a big part in the judgment of the clouds. Another attribute that could have affected this was that the compared images weren't of the same type of cloud. This could sway the judgement of the participants as they may not be used to seeing certain types of clouds as they may live in an area where those types of clouds don't form. Also, the sky in this application doesn't dynamically change colour when the sun is setting like in the other implementations, so people may have been able to tell that the colour of the clouds compared to the sky's colour looked slightly off similar to how the

atmosphere wasn’t present. Again, this was overlooked mainly to focus on how the clouds looked rather than how they looked with the sky.

6. Conclusion

This investigation set out to find if the current standard for performance for volumetric clouds in video games could be improved by combining different methods used for rendering graphics. These methods included using a ray-box intersections algorithm to create a box to render the clouds in for creating smaller areas of clouds in the sky instead of the more common method of using a dome that covers the whole sky that curves towards the horizon line (Schneider, 2015). Another optimisation tested was calculating the phase value for the in-scatter/out-scatter outside of the ray-marching loop to save compute time instead of calculating it in the lighting function within the loop, using the current density to alter how it looks (Haggstrom, 2018).

However, the performance ended up slightly worse which could be due to the settings used, the hardware, the number of monitors and their refresh rates that were used when recording test results. Even though it wasn't as good as the implementation mentioned previously, the application was still able to produce good looking clouds with under 2ms compute time.

As for the visuals, the survey showed that they don't look completely unrealistic, but they don't look very realistic either. This is most likely due to the fact that there was no work done to the sky that the clouds occupied as they weren't the focal point of the study, but most participants likely picked up on the sky not really fitting how the clouds looked as mentioned in section 5.2.

6.1 Future Work

There is still a lot of work that can be done to improve this application.

To improve the visuals, an atmosphere could be used to make the sky look more real. The clouds can then be faded into the atmosphere the further away they get as this is more realistic. Implementing the clouds into the depth of a rasterized scene could be done as this application currently doesn't work with the depth of the scene. If this was done, shadows from the clouds could also be cast onto the landscape to make them feel more like they are part of the scene. To further emphasize this, the method of having the bottom of the clouds coloured based on what is underneath them as mentioned in section 2.2 (Bouthors, Neyret and Lefebvre, 2006) could be used. Having a sun in the scene which, when moving it, changes the light direction and colour. For example, when the sun is setting, the light colour would be more red.

A weather system could be made to have areas of precipitation in the clouds that cause rainfall in certain areas.

More research could be done into the effects of the refresh rate and the number of monitors used when testing compute times for shaders to see how much of an impact it can have on the performance. Also, research into how overclocking a GPU affects the results could be done.

The survey results could be improved by getting an artist to take the screenshots of clouds and set the settings to make them look as realistic as possible while testing to make sure the performance is still balanced. Comparing images of the same types of clouds would also make for a fairer comparison.

Adding a blur to the temporal reprojection algorithm to eliminate some of the artifacting would also make the clouds look nicer when in motion.

The clouds could be animated using the offset values for the Detail and Shape noise textures.

The lighting model could be improved to make the clouds look more realistic by experimenting with different phase functions and their performance.

Bending the render box as it gets closer to the horizon could be experimented with to try and achieve a similar effect to the dome method.

Changing the size and fade distances of the render box could also be used as metrics to measure the compute time as they should provide a variety of results.

To achieve more accurate results, the same GPUs as used in previous implementations could be used as well as recording the results with render textures of the same resolutions.

List of References

Bouthors, A., Neyret, F., Lefebvre, S. (2006) Real-time realistic illumination and shading of stratiform clouds.*Eurographics Workshop on Natural Phenomena* [Online]. [Viewed 15 Nov 2020] Available from: http://www-evasion.imag.fr/Publications/2006/BNL06/bnl06-elec.pdf

Bouthors, A. et al (2008) Interactive multiple anisotropic scattering in clouds. *ACM Symposium on Interactive 3D Graphics and Games (I3D).* [Online]. [Viewed 15 Nov 2020]. Available from: http://www-evasion.imag.fr/Publications/2008/BNMBC08/clouds.pdf

Fanning, E. (2005) Formatting a Paper-based Survey Questionnaire: B*est Practices. Practical Assessment, Research, and Evaluation* [Online]. **10**(12). [Viewed 28 Mar 2021]. Available from: doi: : https://doi.org/10.7275/s84t-8a63

Gjoel, M. and Svendsen, M. (2016) Low Complexity, High Fidelity: The Rendering of INSIDE [PowerPoint Presentation]. *Game Developers Conference*. 21st December. [Viewed 25 Mar 2021]. Available from: https://www.youtube.com/watch?v=RdN06E6Xn9E

Haggstrom, F. (2018) *Real-time Rendering of Volumetric Clouds*. Degree Project in Computing Science Engineering, 30hp, Umea˚ University. [Online]. [Viewed 14 October 2020]. Available from: http://www.diva-portal.org/smash/get/diva2:1223894/FULLTEXT01.pdf

Hillaire, S. (2016) *Physically Based Sky, Atmosphere and Cloud Rendering in Frostbite* [Powerpoint Presentation] [Online]. SIGGRAPH 2016 Course - Physically Based Shading in Theory and Practice. July. [Viewed 5 May 2021]. Available from: https://media.contentapi.ea.com/content/dam/eacom/frostbite/files/s2016-pbs-frostbite-sky-clouds-new.pdf

Kay, T.L. and Kajiya, J.T. (1986). Ray Tracing Complex Scenes. *SIGGRAPH* [Online]. **20**(4) [Viewed 15 Feb 2021]. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.124.4731&rep=rep1&type=pdf

Mallett, I. and Yuksel, C. (2018) Deferred Adaptive Compute Shading. *Eurographics/ ACM SIGGRAPH Symposium on High Performance Graphics.* [Online]. [Viewed 27 Mar 2021]. Available from: doi: 10.1145/3231578.3232160

Mandelbrot, B.B. and Ness, J.W.V. (1968). Fractional Brownian Motions, Fractional Noises and Applications. *SIAM Review.* [Online]. **10**(4), 422-437. [Viewed 7 May 2021]. Available from: https://doi.org/10.1137/1010093

Mayaux, B. (2013). Real-Time Volumetric Rendering. *Revision(Demoparty)*. [Online]. [Viewed 24 Feb 2021]. Available from: http://patapom.com/topics/Revision2013/Revision%202013%20-%20Real-time%20Volumetric%20Rendering%20Course%20Notes.pdf

Olajos, R. (2016). Real-Time Rendering of Volumetric Clouds. Masters thesis, LUND University [Viewed 14 Feb 2021]. Available from: http://lup.lub.lu.se/luur/download?func=downloadFile&recordOId=8893256&fileOId=8893258

Parga, C. J. and Palomo, S. R. G. (2018) Efficient Algorithms for Real-Time GPU Volumetric Cloud Rendering with Enhanced Geometry. *Software Engineering and Computer Systems Department, National Distance Education University (UNED)*. [Online] [Viewed 14 October 2020] Available from: https://www.mdpi.com/2073-8994/10/4/125/htm

Perlin, K. (1985). An Image Synthesizer. *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques.* [Online]. Pages 287-296. [Viewed 7 May 2021]. Available from: https://doi.org/10.1145/325334.325247

Robertson, P. (2018). DirectX11 Framework. Scotland: Abertay University

Schneider, A. (2015) *Advances in Real-Time Rendering in Games*. August 2015, SIGGRAPH, Los Angeles, CA [Online].[Viewed 14 October 2020]. Available from: http://killzone.dl.playstation.net/killzone/horizonzerodawn/presentations/Siggraph15_Schneider_Real-Time_Volumetric_Cloudscapes_of_Horizon_Zero_Dawn.pdf

Schpok, J. et al. (2003). A Real-Time Cloud Modeling, Rendering, and Animation System. *Eurographics/SIGGRAPH Symposium on Computer Animation*. [Online]. [Viewed 27 Feb 2021] Available from: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.884.2816&rep=rep1&type=pdf

Toft, A., Bowles, H. and Zimmermann, D. (2016) Optimisations for Real-Time Volumetric Cloudscapes. [Online] [Viewed 5 May 2021] Available from: https://arxiv.org/pdf/1609.05344.pdf

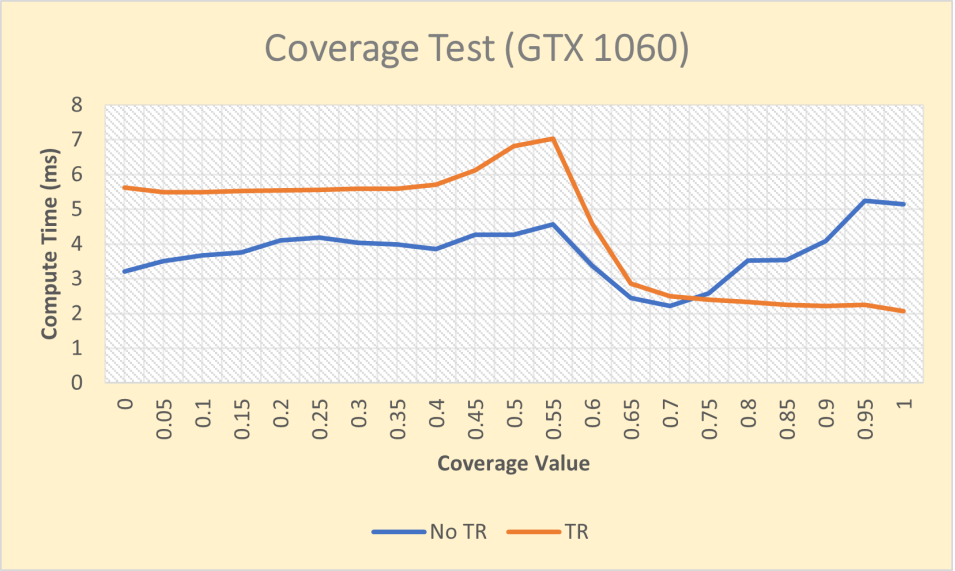
Ward, M.K. and Meade, A.W. (2017) Applying Social Psychology to Prevent Careless Responding during Online Surveys. *Applied Psychology* [Online]. **67**(2), 255-366. [Viewed 26 Mar 2021]. Available from: doi: https://doi.org/10.1111/apps.12118

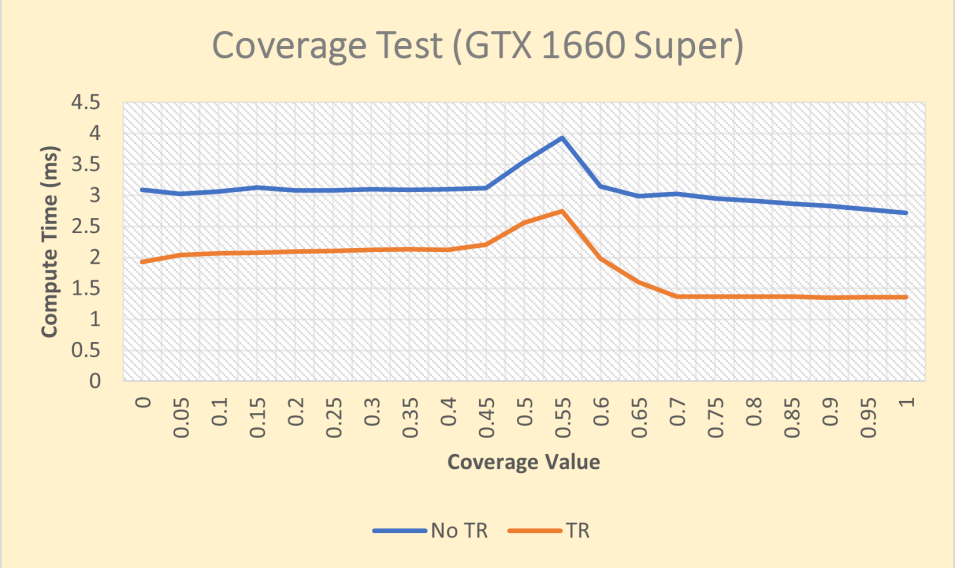
Worley, S. (1996). A Cellular Texture Basis Function. *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques.* [Online]. Pages 291-294. [Viewed 7 May 2021]. Available from: https://doi.org/10.1145/237170.237267

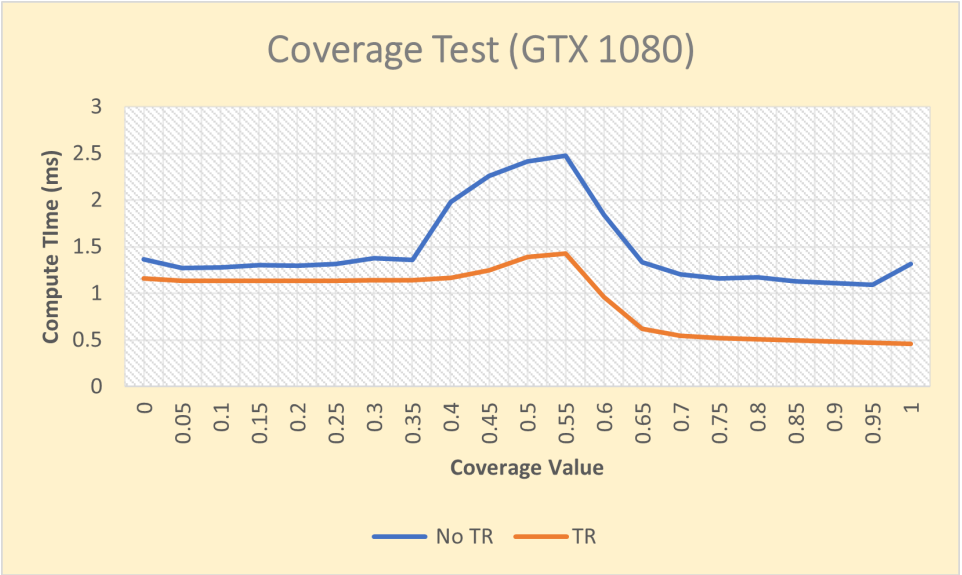
Appendices

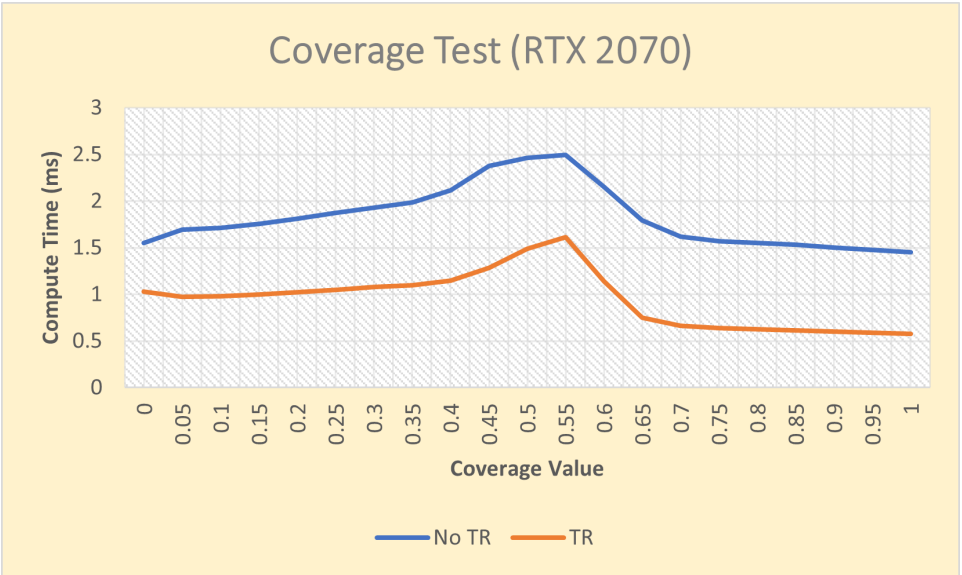
Appendix A - Application Testing Results

A.1 Coverage Test Results:

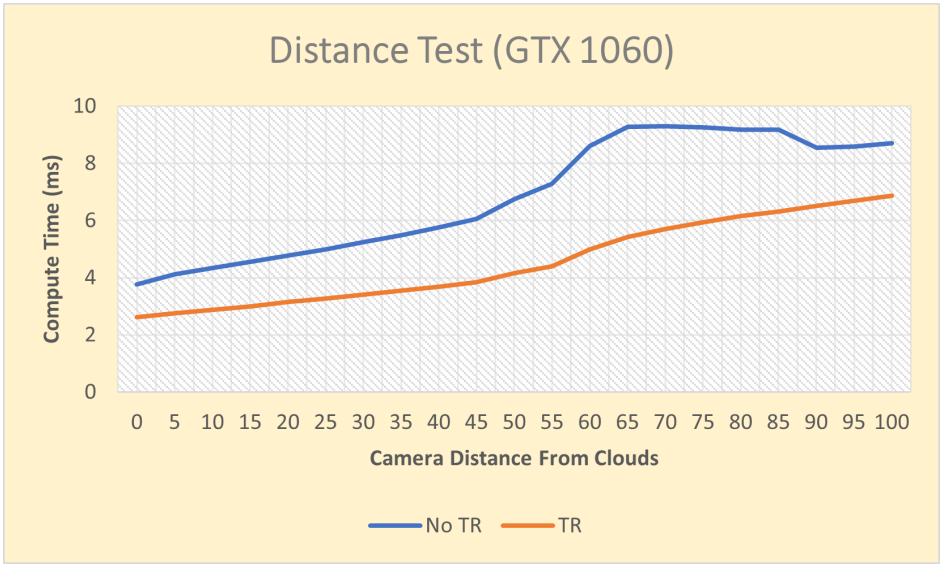


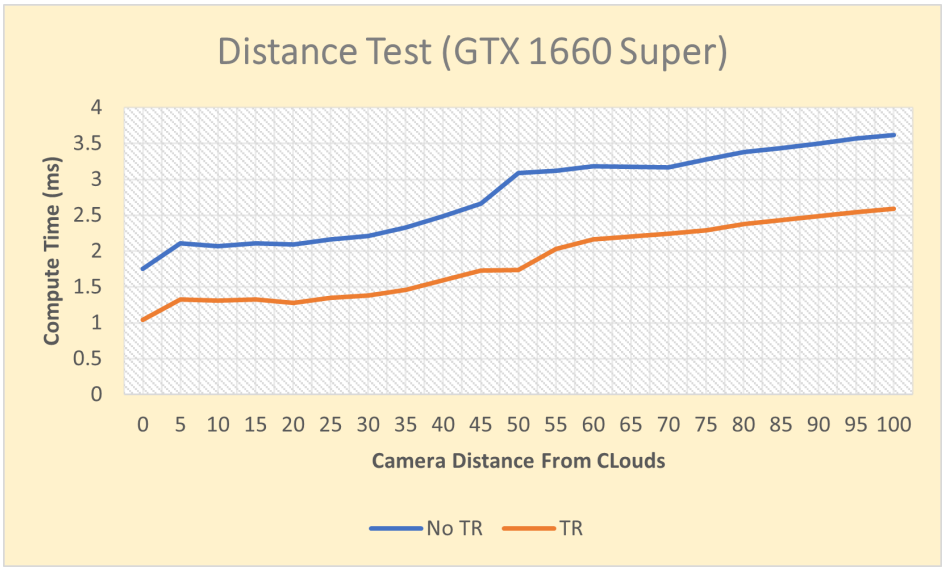


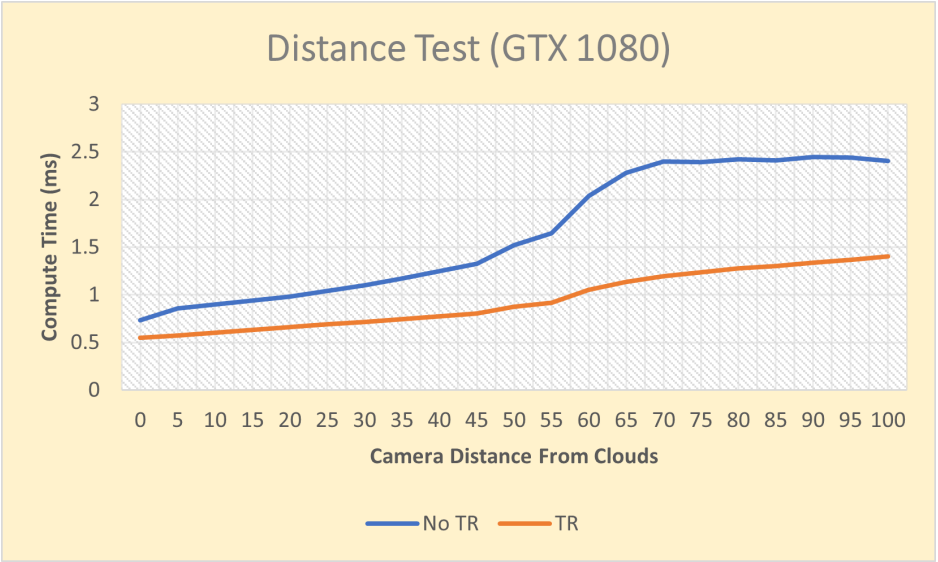

Coverage Test (GTX 1080)
Compute TIme (ms)
3
2.5
2
1.5
1
0.5
0
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
0.55
0.6
0.65
0.7
0.75
0.8
0.85
0.9
0.95
1
Coverage Value
No TR
TR

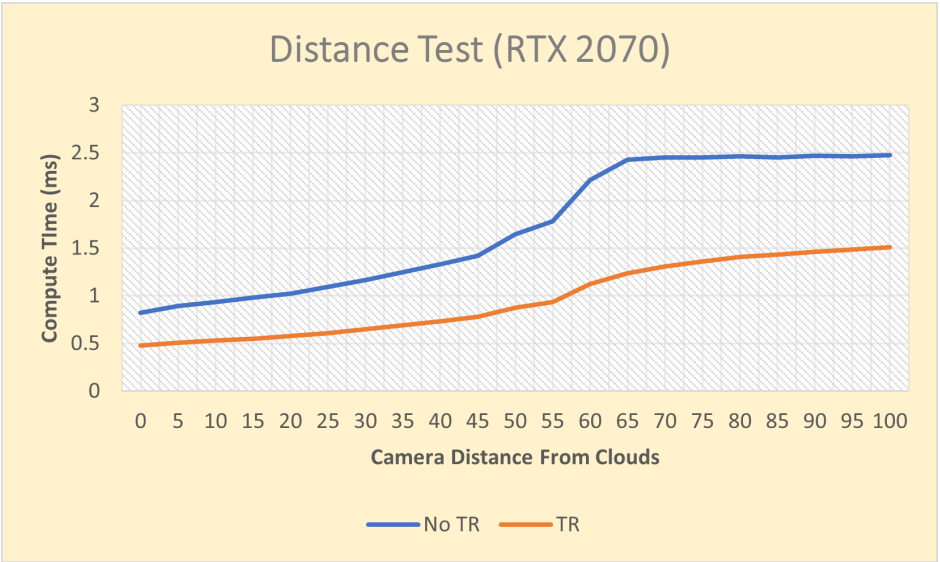

Coverage Test (RTX 2070)
Compute Time (ms)
3
2.5
2
1.5
1
0.5
0
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
0.55
0.6
0.65
0.7
0.75
0.8
0.85
0.9
0.95
1
Coverage Value
No TR
TR

A.2 Distance Test Results:

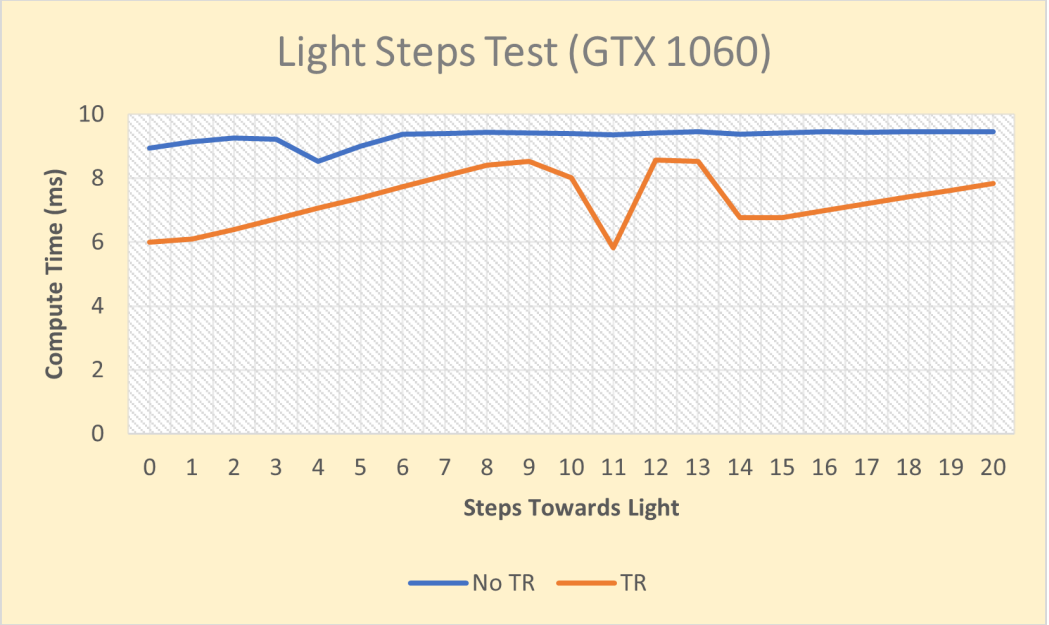


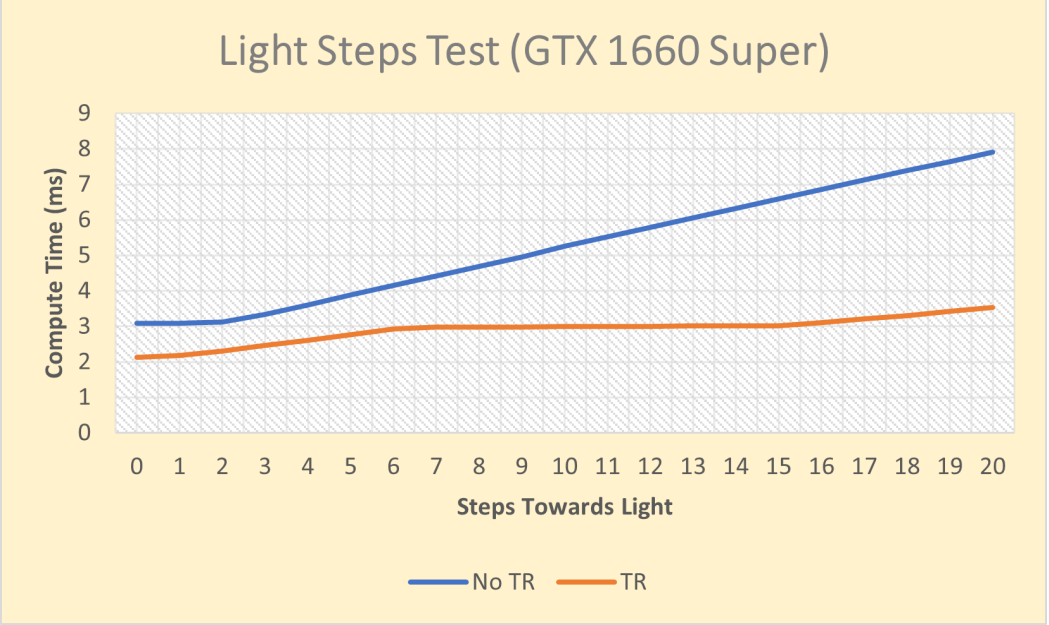


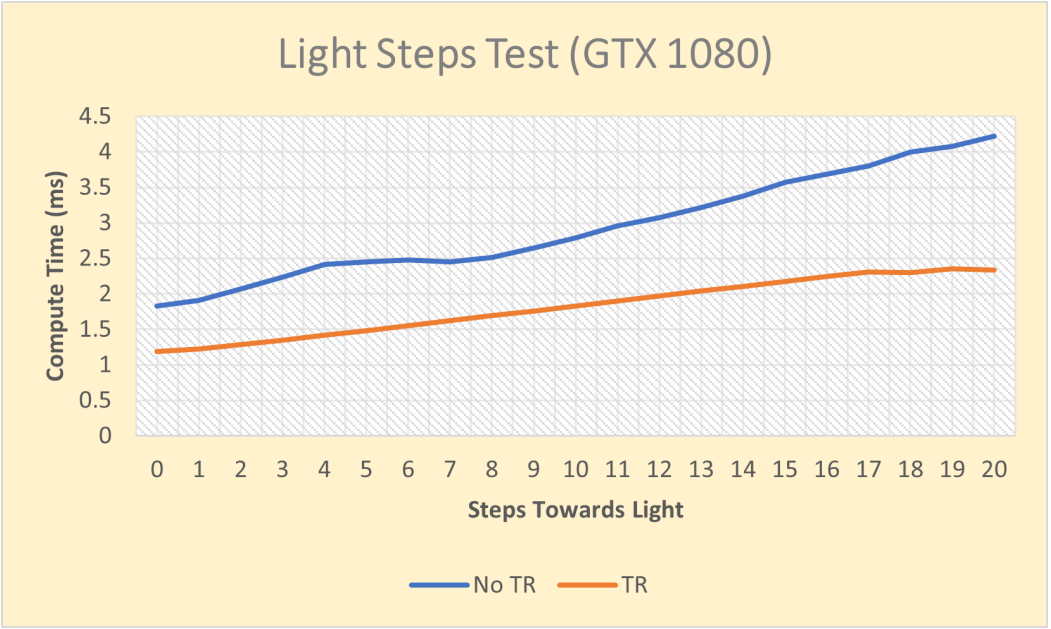

Distance Test (GTX 1080)
Compute Time (ms)
3
2.5
2
1.5
1
0.5
0
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100
Camera Distance From Clouds
No TR
TR

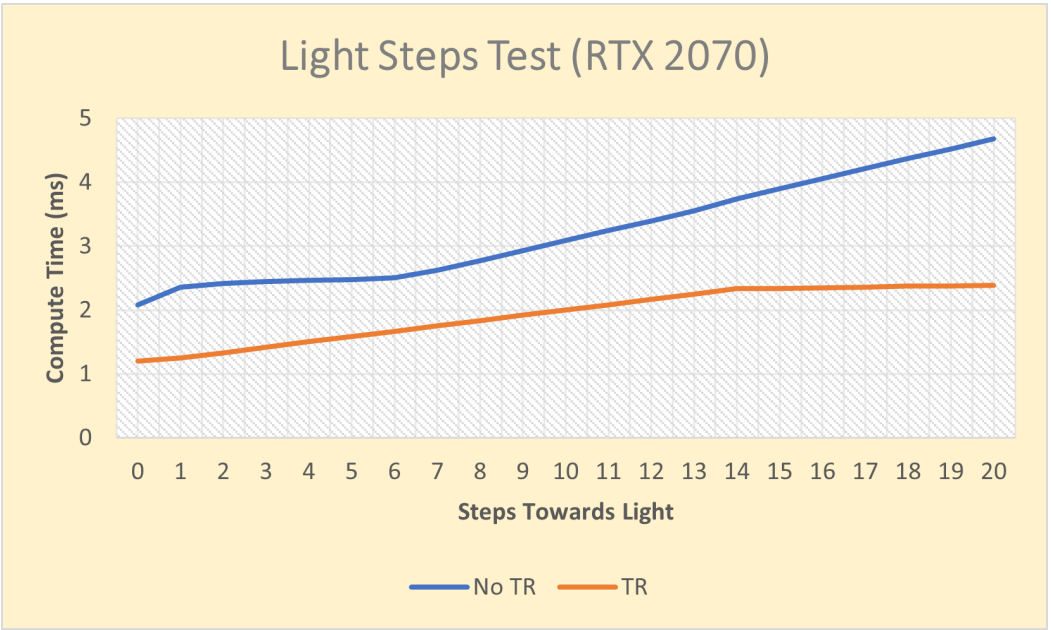

Distance Test (RTX 2070)
Compute TIme (ms)
3
2.5
2
1.5
1
0.5
0
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100
Camera Distance From Clouds
No TR
TR

A.3 Light Steps Test Results:

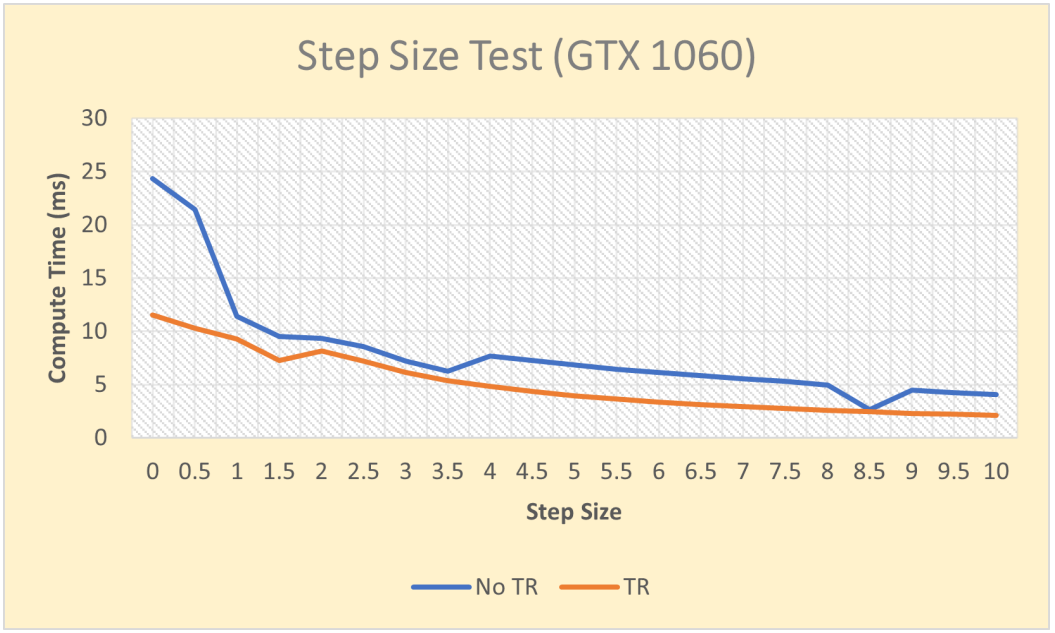


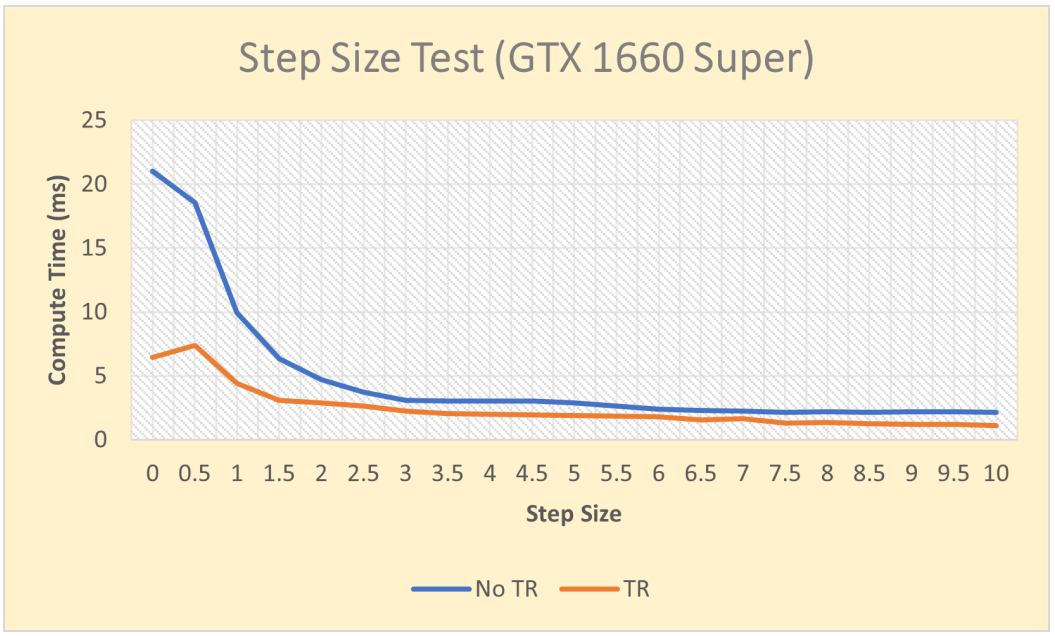


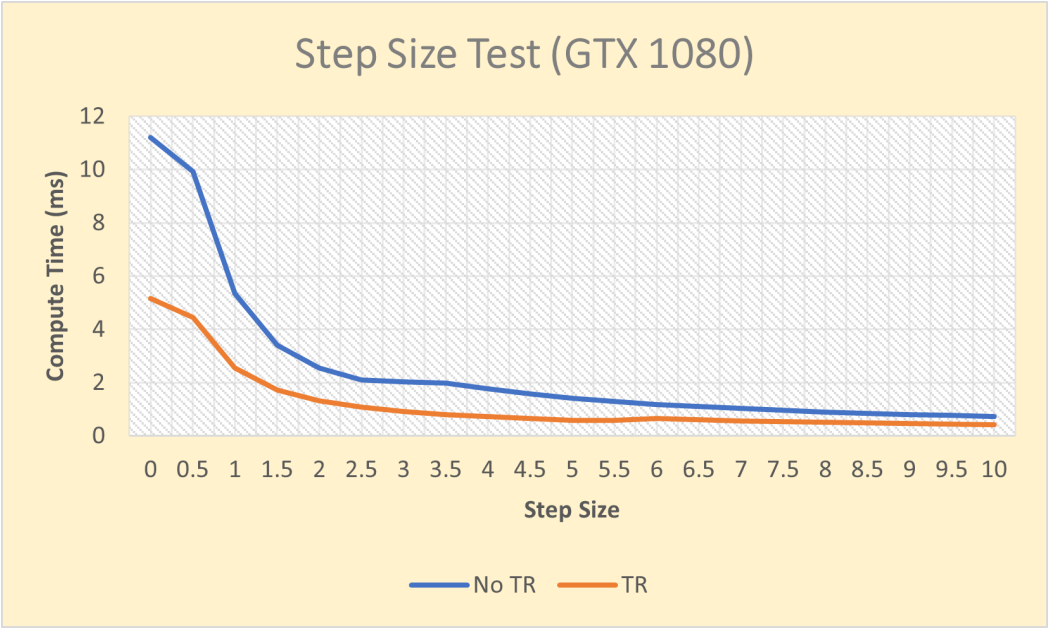

Light Steps Test (GTX 1080)
Compute Time (ms)
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Steps Towards Light
No TR
TR

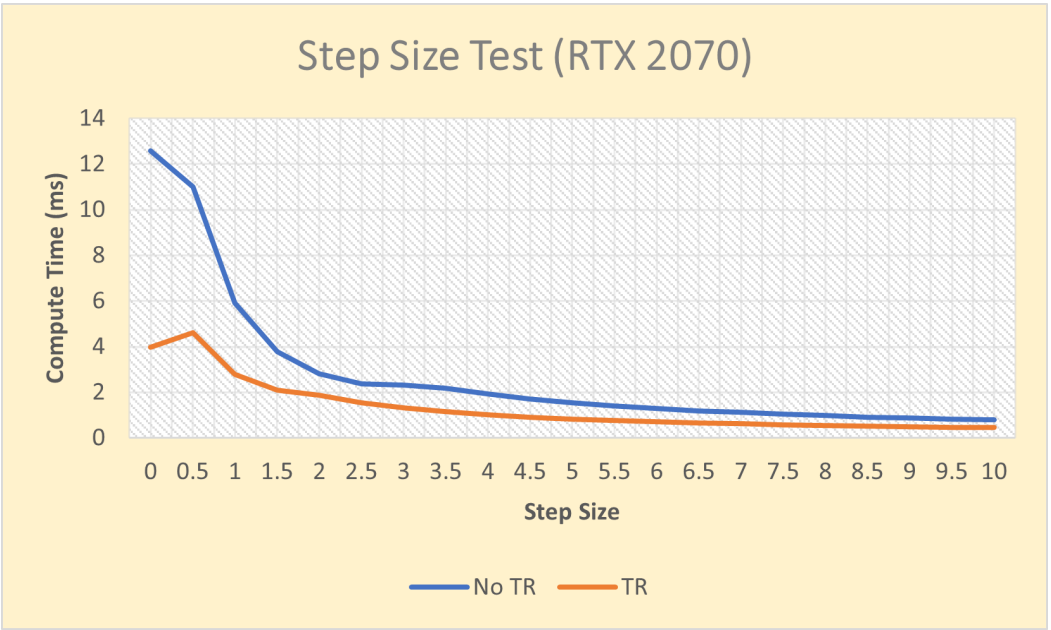

Light Steps Test (RTX 2070)
Compute Time (ms)
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Steps Towards Light
No TR
TR

A.4 Step Size Test Results





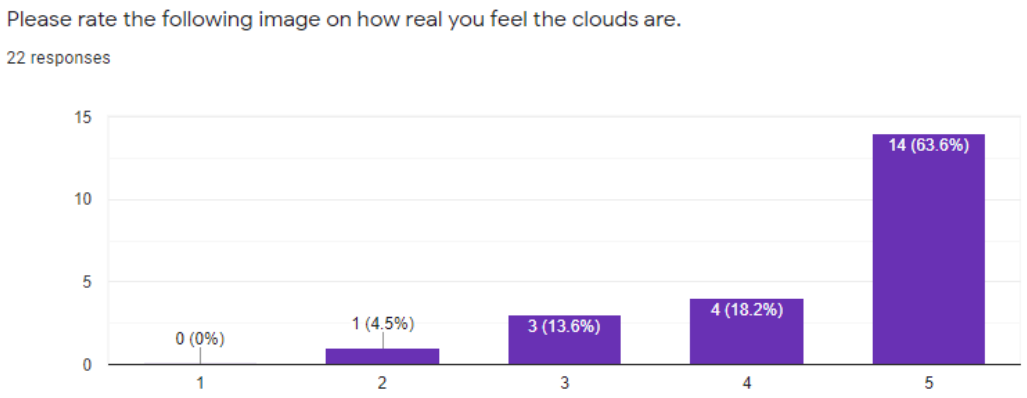

Step Size Test (GTX 1080)
Compute Time (ms)
12
10
8
6
4
2
0
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5 5.5 6 6.5 7 7.5 8 8.5 9 9.5 10
Step Size
No TR
TR


Step Size Test (RTX 2070)
Compute Time (ms)
14
12
10
8
6
4
2
0
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5 5.5 6 6.5 7 7.5 8 8.5 9 9.5 10
Step Size
No TR
TR

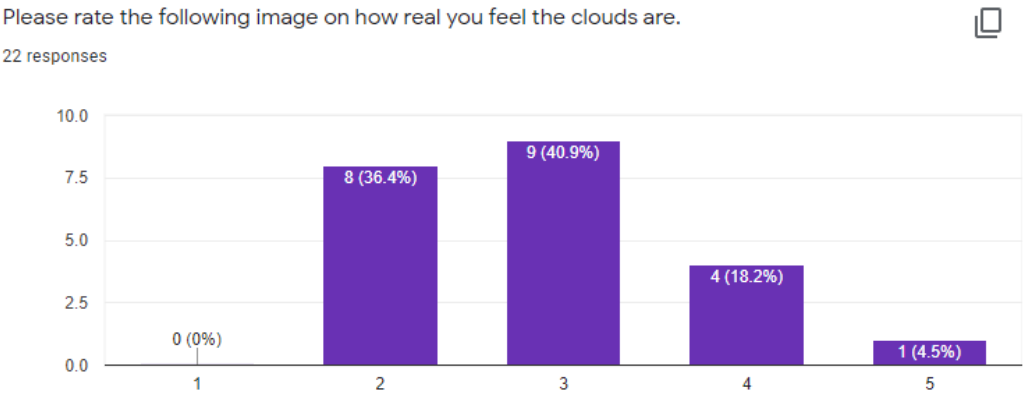
Appendix B - Full Survey Results

B.1 Survey Section 1 Results:

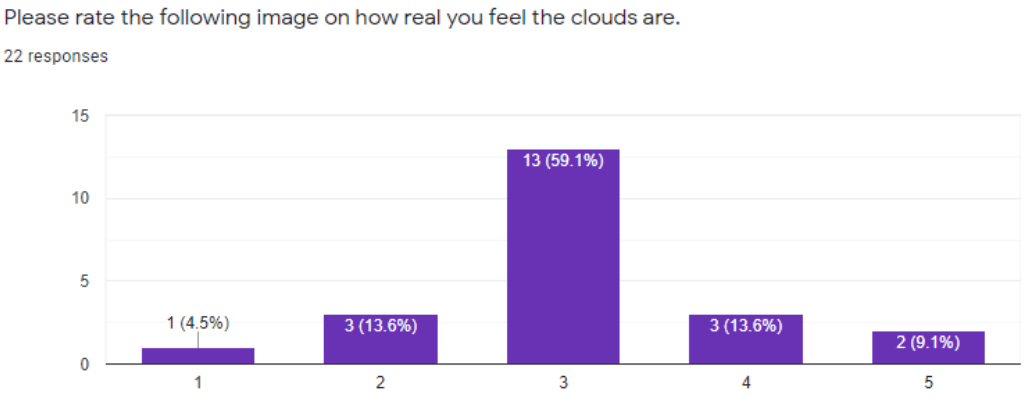
Q1 - Real Clouds



Q2 - This Application's Clouds



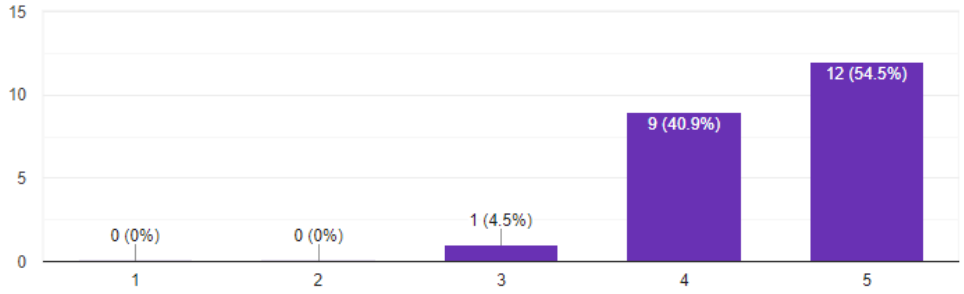
Q3 - This Application's Clouds



Q4 - Real Clouds

Please rate the following image on how real you feel the clouds are.

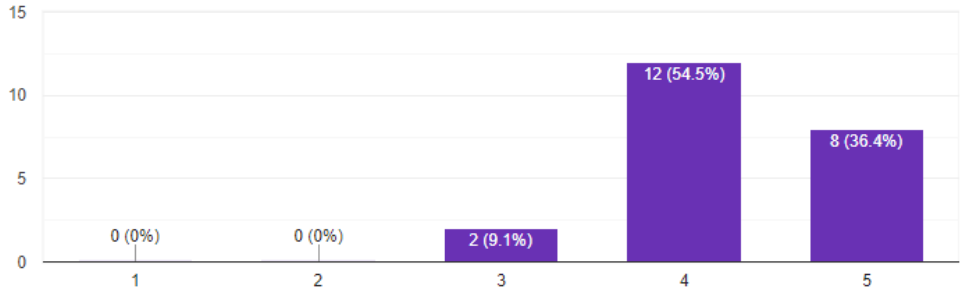
22 responses



Q5 - Haggstrom's Clouds

Please rate the following image on how real you feel the clouds are.

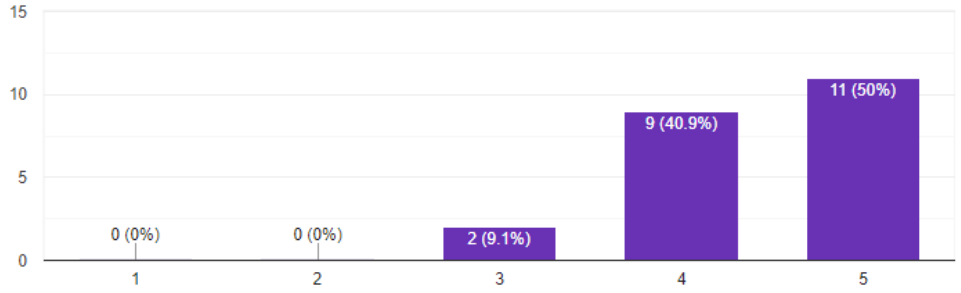
22 responses



Q6 - Schneider's Clouds

Please rate the following image on how real you feel the clouds are.

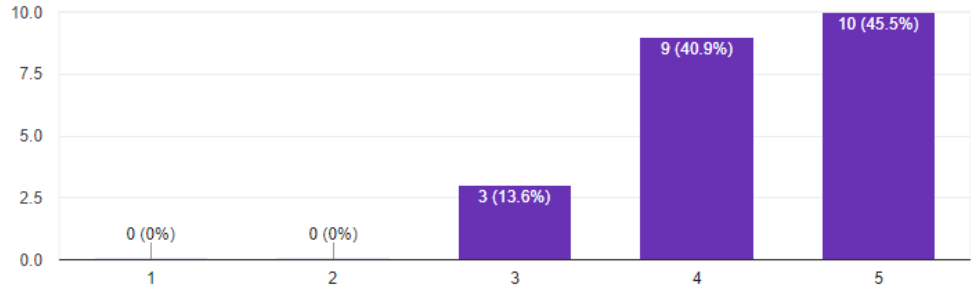
22 responses



Q7 - Schneider's Clouds

Please rate the following image on how real you feel the clouds are.

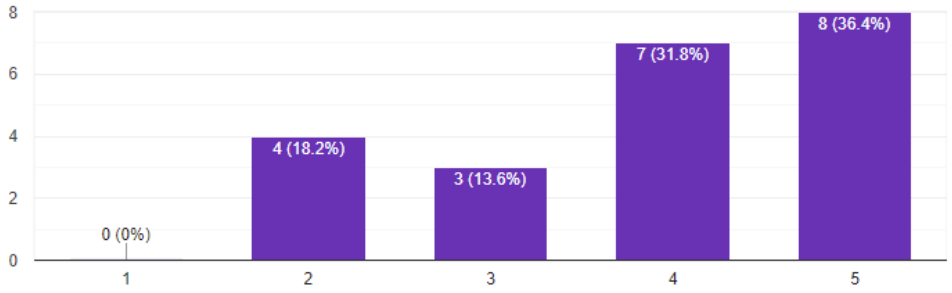
22 responses



Q8 - Haggstrom's Clouds

Please rate the following image on how real you feel the clouds are.

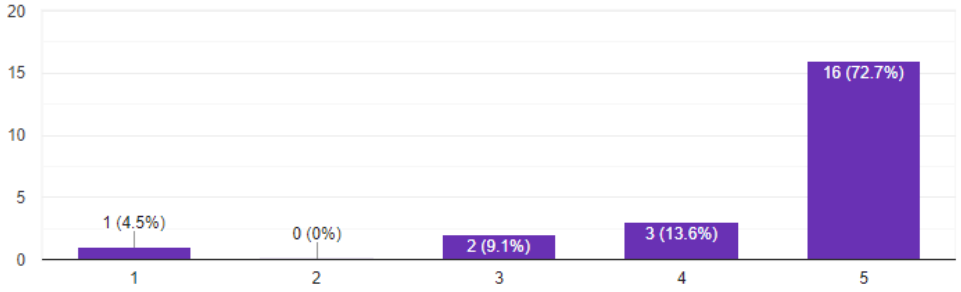
22 responses



Q9 - Real Clouds

Please rate the following image on how real you feel the clouds are.

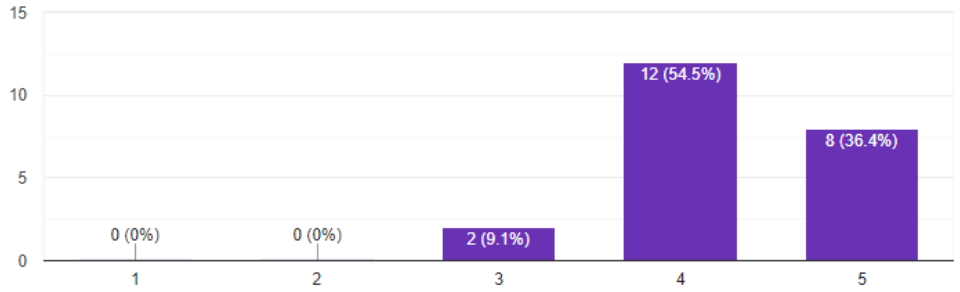
22 responses



Q10 - Schneider's Clouds

Please rate the following image on how real you feel the clouds are.

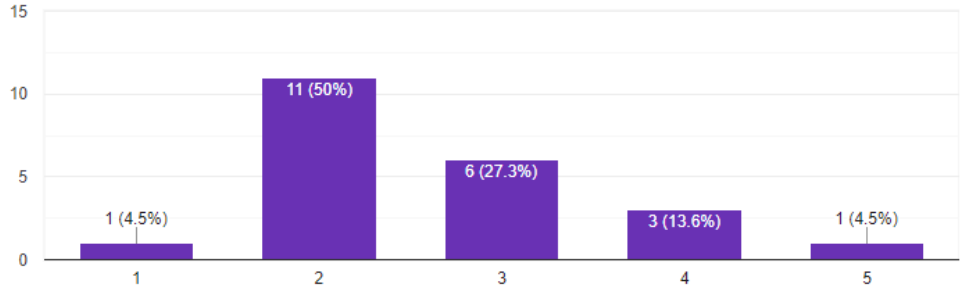
22 responses



Q11 - This Application's Clouds

Please rate the following image on how real you feel the clouds are.

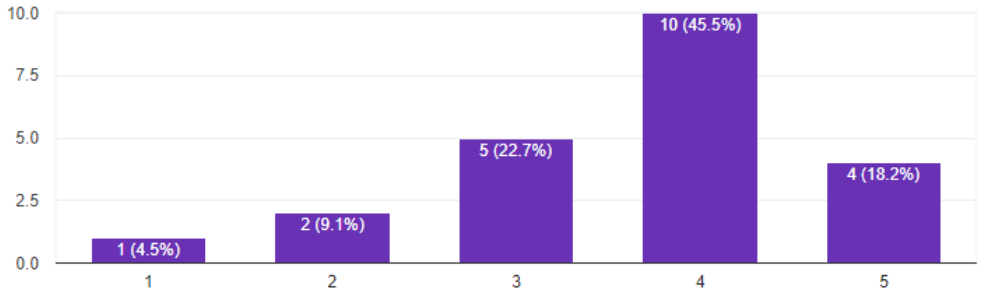
22 responses



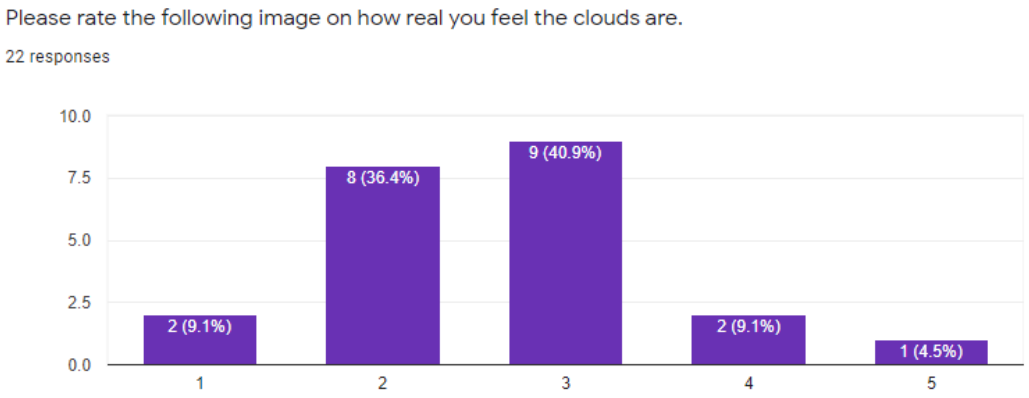
Q12 - Haggstrom's Clouds

Please rate the following image on how real you feel the clouds are.

22 responses

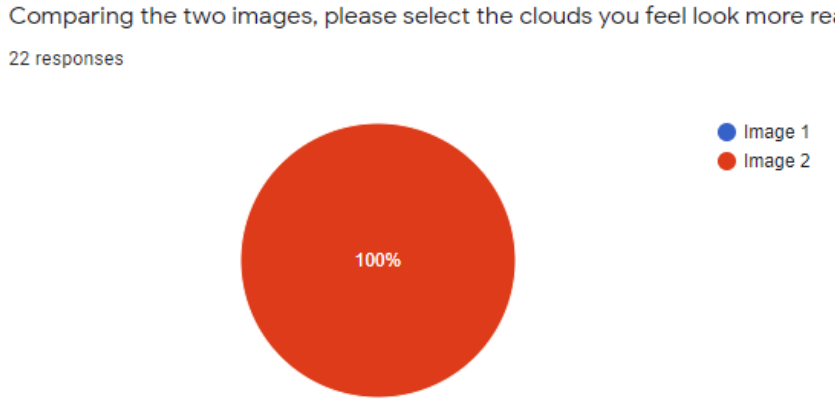


Q13 - This Application's Clouds

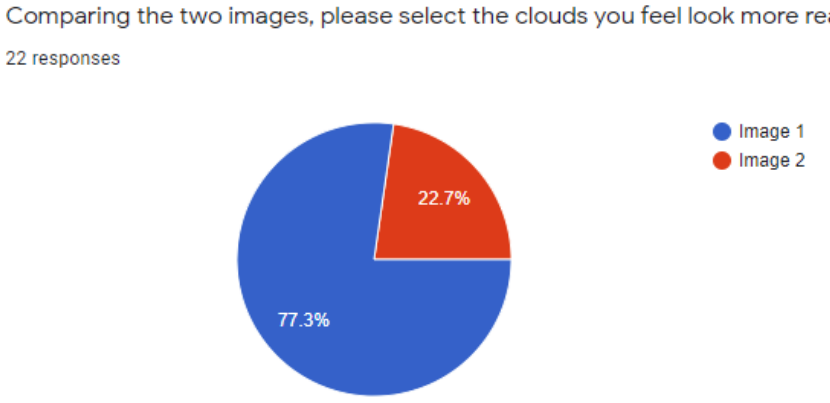


B.2 Survey Section 2 Results:

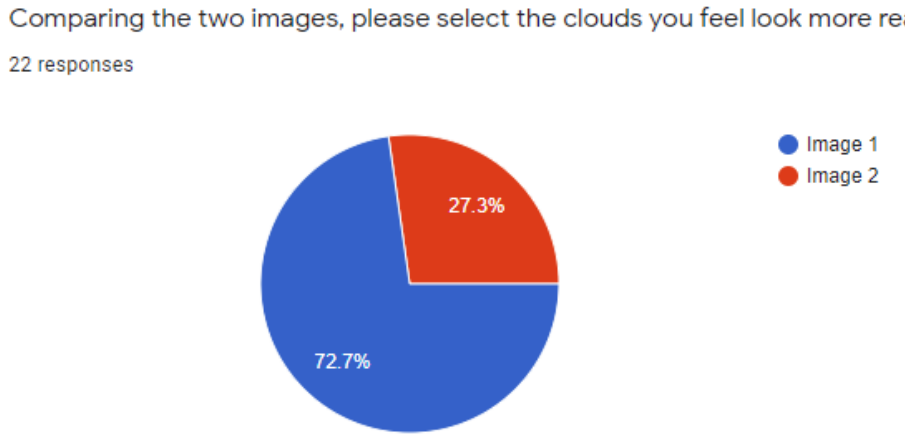
Q1 - Image 1 = The Application's Clouds, Image 2 = Haggstrom's Clouds



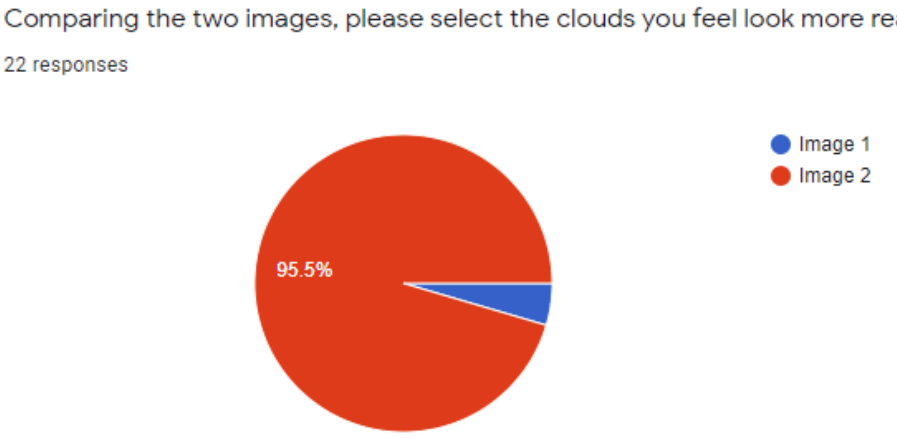
Q2 - Image 1 = Real Clouds, Image 2 = The Application's Clouds



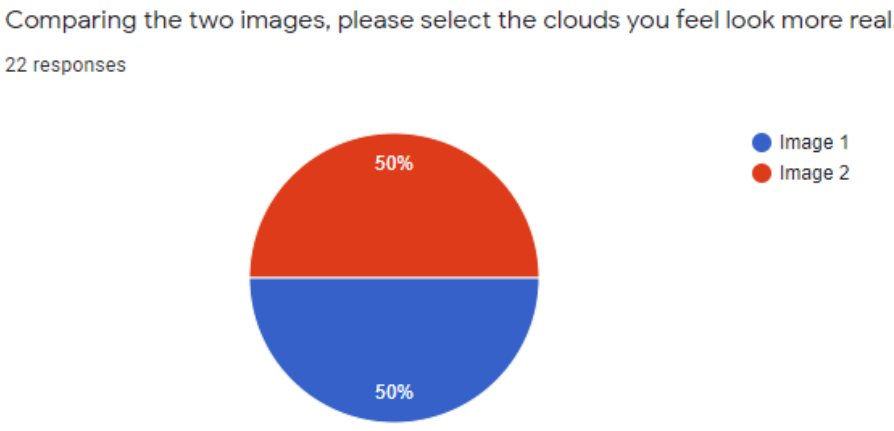
Q3 - Image 1 = Schneider's Clouds, Image 2 = Haggstrom's Clouds



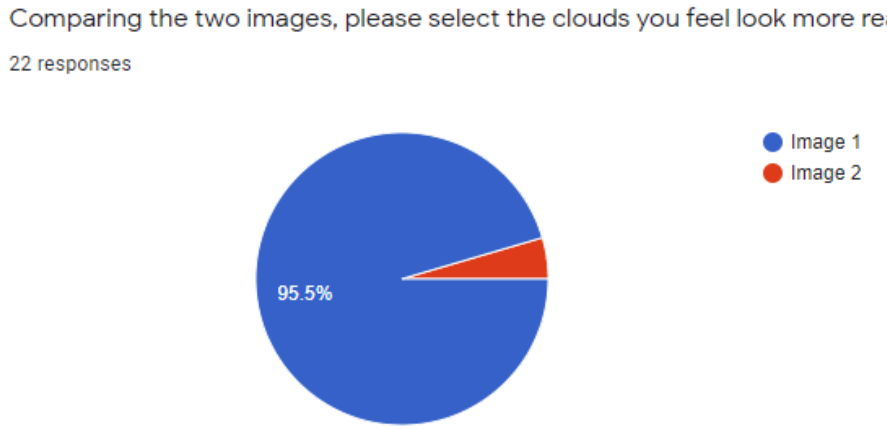
Q4 - Image 1 = The Application's Clouds, Image 2 = Schneider's Clouds



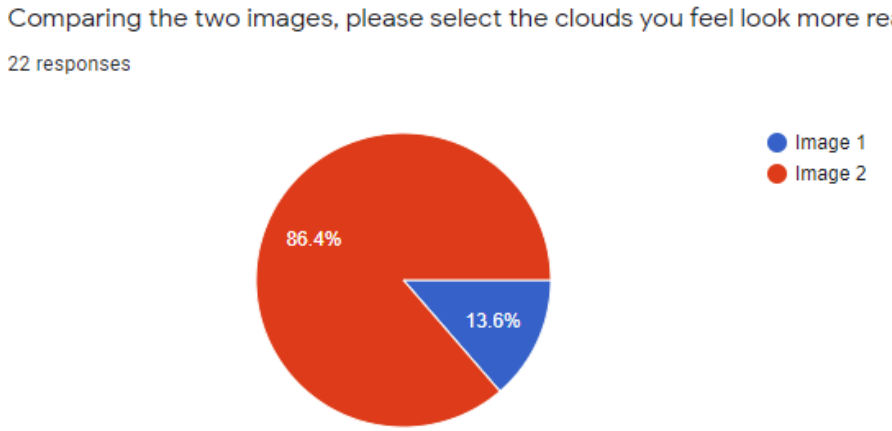
Q5 - Image 1 = Real Clouds, Image 2 = Schneider's Clouds



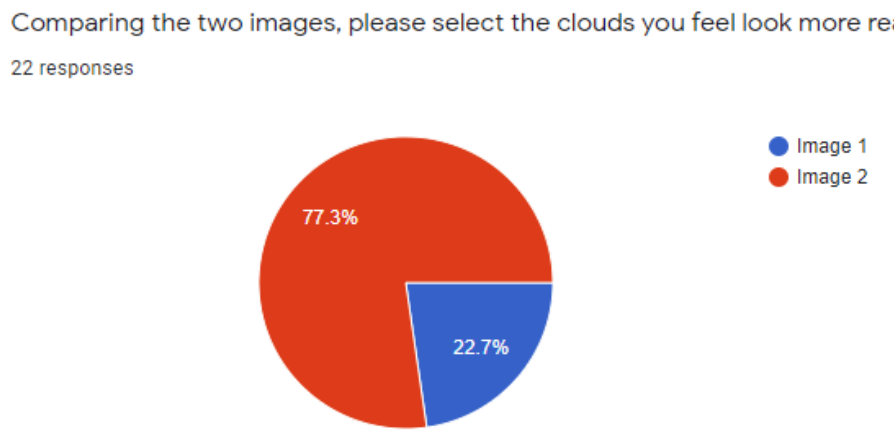
Q6 - Image 1 = Real Clouds, Image 2 = The Application's Clouds



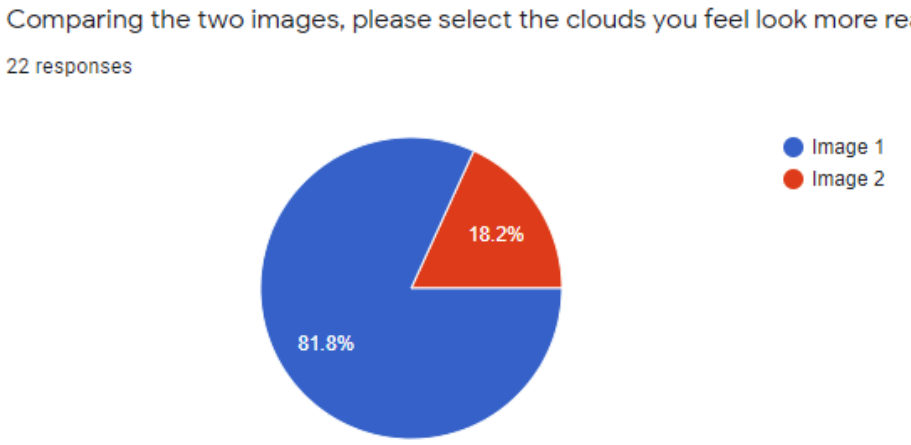
Q7 - Image 1 = The Application's Clouds, Image 2 = Haggstrom's Clouds



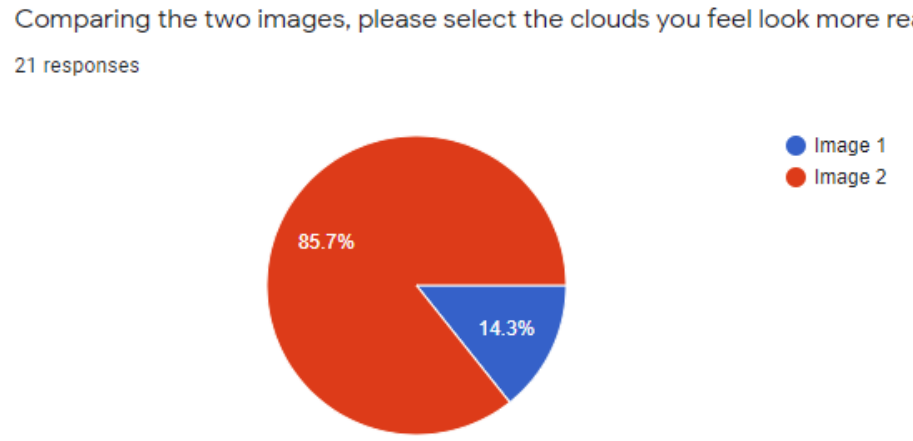
Q8 - Image 1 = Haggstrom's Clouds, Image 2 = Real Clouds



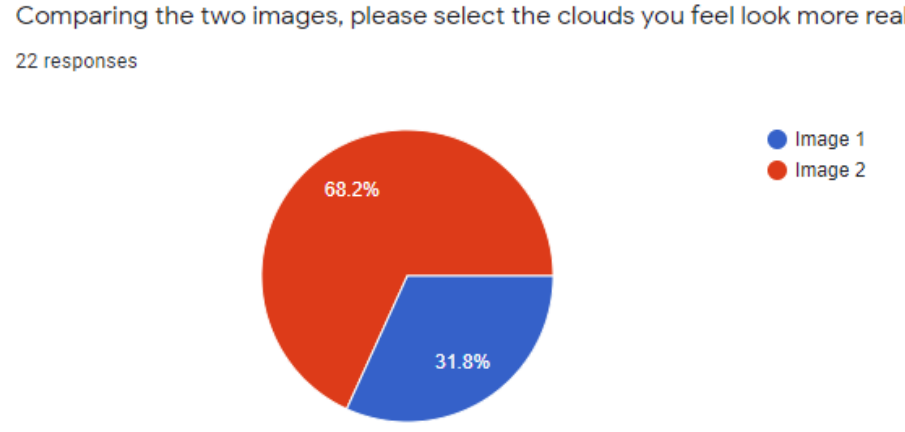
Q9 - Image 1 = Schneider's Clouds, Image 2 = The Application's Clouds



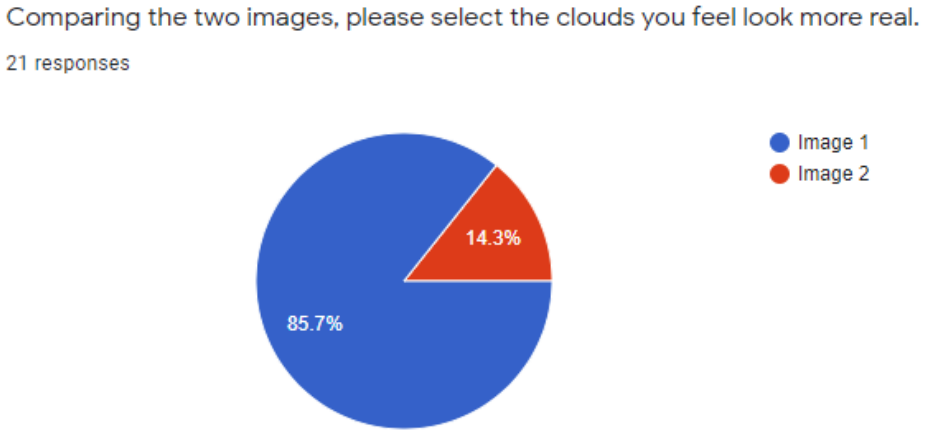
Q10 - Image 1 = Haggstrom's Clouds, Image 2 = Schneider's Clouds



Q11 - Image 1 = Haggstrom's Clouds, Image 2 = Real Clouds



Q12 - Image 1 = Real Clouds, Image 2 = Schneider's Clouds



Appendix C - GDPR Form for Survey

Abertay University

GDPR Research Data Management
Data Sign Off Form

For undergraduate or postgraduate student projects supervised by an Abertay staff member.

This form MUST be included in the student's thesis/dissertation. Note that failure to do this will mean that the student's project cannot be assessed/examined.

Part 1: Supervisors to Complete

By signing this form, you are confirming that you have checked and verified your student's data according to the criteria stated below (e.g., raw data, completed questionnaires, superlab/Eprime output, transcriptions etc.)

Student Name:	Aidan Murray		
Student Number:	1702270		
Lead Supervisor Name:	Paul Robertson		
Lead Supervisor Signature	PRobertson		
Project title:	An Evaluation of Volumetric Cloud Implementations in Video Games to Find a Good Balance Between Visual Reality and Performance		
Study route:	PhD ☐	MbR ☐	MPhil ☐
	Undergraduate ☑	PhD by Publication ☐	

Part 2: Student to Complete

	Initial here to confirm 'Yes'
I confirm that I have handed over all manual records from my research project (e.g., consent forms, transcripts) to my supervisor for archiving/storage	AM
I confirm that I have handed over all digital records from my research project (e.g., recordings, data files) to my supervisor for archiving/storage	AM
I confirm that I no longer hold any digital records from my research project on any device other than the university network and the only data that I may retain is a copy of an anonymised data file(s) from my research	AM
I understand that, for undergraduate projects, my supervisor may delete manual/digital records of data if there is no foreseeable use for that data (with the exception of consent forms, which should be retained for 10 years)	AM

Student signature : AMurray

Date: 11/05/2021