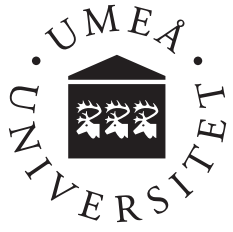



# Real-time rendering of volumetric clouds

*Fredrik Häggström*

**Fredrik Häggström**
Spring 2018
Degree Project in Computing Science Engineering, 30hp
Supervisor: Niclas Börlin
External Supervisor: Anton Stenmark (Arrowhead Game Studios)
Examiner: Henrik Björklund
Degree of Master of Science in Engineering, Computing Science and Engineering 300hp

## Abstract

A common cloud-rendering implementation in computer-games is based on having a library of cloud-images. However, recent development has shown that it is possible to implement 3-dimensional volumetric clouds on current consumer hardware.

This thesis presents and evaluates a volumetric cloud implementation that aims to further improve aspects of the already published solutions. The implementation relies on multiple different textures (both 2-dimensional and 3-dimensional) along with values and functions to create the shapes of the clouds. To visualize the clouds, a ray-marching algorithm was used to incrementally step through the cloud-volume, sample the density, and calculate the lighting.

To optimize the implementation, different experiments were performed to find the best parameter-values that resulted in good performance figures (render-time) while still achieving visually pleasing clouds.

Without the clouds visual fidelity being compromised to much, the experiment results show that the number of steps taken towards the sun (for each step through the cloud-volume) can be lowered to a single digit number, and that the ray-marching step-length can be altered with relation to the ray-marching start distance, the global cloud coverage and the global cloud density.

The resulting volumetric cloud implementation can seamlessly transition from low coverage to overcast, create different cloud types, handle different times of the day, and have clouds move across the sky.

# Acknowledgements

I would like to thank the people at *Arrowhead Game Studios* who gave me the opportunity to do this thesis at a game developer that has, besides for providing help and tools, enabled me to learn a lot about the game development process.

Big thanks to my supervisor at *Arrowhead Game Studios*, Anton Stenmark, who has helped me overcome difficult obstacles and that has been of great help when discussing possible implementation solutions.

For continuous support when writing the thesis I would like to thank my supervisor at Umeå University, Niclas Börlin, who has provided great cloud-knowledge and without whom my thesis would not be as academically correct.

I would also like to thank my family and friends that have proofread the report and provided me with useful feedback.

# Contents

# Glossary

| | |
|---|---|
| **2D** | Short for 2-Dimensional. |
| **3D** | Short for 3-Dimensional. |
| **cloudscape** | The cloud-equivalent to ”landscape”, describing all the clouds visible in the sky. |
| **FBM** | Short for fractal brownian motion. An FBM can be seen as a hierarchy of functions combined together for a more complex curve. Using waves on the ocean as an example, big waves make out the main function while other higher frequency functions creates smaller waves on top of the big waves. |
| **frame-rate** | How many frames (or images) per second are displayed on screen. |
| **km** | Short for kilometer. |
| **ms** | Short for milliseconds. |
| **polygons** | Any 2D shape defined by three or more straight lines (triangle, square, pentagon, ...). Used extensively in computer-graphics to define and render shapes. |
| **quarter resolution** | Dividing the number of pixels by 4 for both the screens x- and y-axis. Reducing the total amount of pixels by $4 \times 4 = 16$. |
| **render-pass** | To create a fully rendered scene, the rendering is usually separated into multiple render-passes, where each render-pass can perform separate tasks. For example, when creating shadows, there are usually at least two render-passes, one that from the light-source's perspective calculates and stores the depth into a depth-map render-target, and one that uses the depth-map to apply the shadows when rendering objects to the screen. |

**render-target** A run-time image used to store rendered information. An example of a render-targets is the shadow-map, a render-target that usually contains the depth values for objects in relation of the sunlights direction.

**render-time** The amount of time that it takes to draw the image to the screen, measured in milliseconds for the entire thesis.

**RGBA** Red, green, blue and alpha. Frequently used to store pixel data for images, where different combinations of these colors can form any other color. However, in this thesis, these color channels are often used individually to store some attribute.

**screen-space** Space used to specify where an object is in relation to the screen. Therefore, it can be used to specify where an object is on the screen.

**viewing-ray** The ray that can be seen as a combination of the cameras direction and the pixel that is currently being rendered.

**volumetric clouds** A way of representing clouds in 3D. The part ”volumetric” does not specify a specific method, just that the clouds have a volume with some density.

# 1 Introduction

Most people can relate to weather having an impact on their general mood. People can usually feel a significant difference if it is a clear sunny day or if a thick wet mist covers the entire sky. Large dark clouds can feel daunting, thick mist can feel depressing and clouds being lit during a sunset can create a sense of wonder.

This also applies to video-games, there are many things in play to create the correct mood for scenes presented to the player. The goal of this thesis is to improve one of the supposedly most important mood-setters and ”scenery-creators” in video-games: the clouds in the sky.

## 1.1 Clouds in games

A common cloud-rendering implementation in computer-games is based on having a library of cloud-images. This is a solution that can be visually pleasing, but that has several drawbacks. If the library only stores one image per cloud, the illusion that the sky contains real clouds falls apart when the player starts moving, with the issue being increased the further the player moves. This problem can be solved by storing multiple images from different angles and presenting them as the player moves. However, due to having to store multiple images (possible high-resolution images), the images quickly start to use a large amount of storage space. The storage problem is further increased if the solution should present many different cloud variations and cloud-types. An image-based solution also stumbles when trying to change the cloud coverage which should lead to new clouds emerging and building up.

Moving from the 2D image based solution into a 3D solution has the potential to solve these problems. One way of visualizing the clouds in 3D is to render them volumetrically. Volumetric rendering usually involves an algorithm that incrementally steps through a volume, samples the density, and calculates the lighting.

Volumetric clouds have for a long time been seen as too computationally heavy to implement into games. However, in the last few years there has been a lot of development that has proven it possible to do with the current consumer hardware (for example, Xbox One and Playstation 4).

## 1.2 Related work

The most pronounced recent work describing a solution for volumetric clouds is the work by Andrew Schneider on an implementation in the game engine Decima (developed by Guerrilla Games) (Schneider, 2016). Schneider has since continued to work on the implementation and the current development is now called Nubis (Schneider, 2017).

TrueSky is an interesting cloud implementation developed by Simul (2013). However, being a commercial product, the author has been unable to locate any detailed documentation about TrueSky.

Branching from Schneider's work, implementations by Högfeldt and Hillaire (2016) and Grenier (2016) have a similar solutions. All three use the same basic techniques with minor variations in implementation. One difference is that Högfeldt and Hillaire (2016) claim to have improved the cloud-lighting compared to Schneider's implementation.

The game Sea of Thieves has a different solution to modeling the clouds in 3D, developed by Rare (2018). The clouds are represented by polygons that make it possible to control the shapes with more detail. The author has been unable to locate any detailed documentation about the clouds in Sea of Thieves.

## 1.3 Aim

Following the work by Schneider, the aim of this thesis is to present and evaluate an improved realistic real-time volumetric clouds implementation for use in video-games. Being implemented mainly for use in video-games, the volumetric cloud implementation must share the limited execution time with many other visual effects. For example, to achieve a minimum of 60 frames per second, the maximum total render-time of all render-passes must be at most $16.\overline{6}$ ms.

### 1.3.1 Performance goals

In order to still have enough time for other visual effects, the goal is that the implementation should execute in under 2 ms on the graphics card NVIDIA GTX 980 Ti.

### 1.3.2 Visual goals

The aim is to create a cloudscape that in prioritized order:

1. Vary in size and coverage.
2. Has physically based lighting.
3. Supports a wide variety of cloudscapes, from light clouds to overcast, day and night.
4. Has seamless transitions between night and day, overcast and clear, and changes in movement speed.
5. Casts soft shadows.

## 1.4 Limitations

Apart from the clouds themselves there are several other factors that contribute to the realism of the cloudscape. Two of the most distinguishable being the atmosphere and the sun. However, this thesis focuses on a volumetric clouds implementation. Other factors that may limit the realism of the solution are outside the scope of this thesis.

# 2 Theory

Most modeling methods for creating clouds do not share any real physics with real clouds. However, it is of utmost importance to understand the real-life counterparts to be able to model something correctly. Cloud characteristics becomes the basis for every problem and its solution when modeling.

## 2.1 Clouds

Unless otherwise stated, the information in this section is derived from MetOffice (2017).

Clouds consist of water droplets or ice crystal that are so small that each cubic meter contains about 100 million droplets. Whether they consist of water or ice depends on altitude and temperature. However, the water droplets are so small that they remain liquid until temperatures fall below about $-30\,^{\circ}\mathrm{C}$. With lower temperatures, high altitude clouds are dominated by ice crystals.

### 2.1.1 Clouds formation

The air constantly contains water vapour that when cooled can condense into droplets or ice crystals. However, for the water vapour to condense, other small particles such as salt or dust, called aerosols, are necessary.

When the water particles collide with aerosols they can stick and start to condense. Bigger and bigger droplets form around the aerosols until they become visible and form into clouds.

With higher temperatures, the air can hold more water vapour before it is saturated. Once saturated, the air cannot hold any more water and clouds form.

According to MetOffice (2017), some of the driving forces behind cloud formation are:

1. **Surface heating** - This happens when the ground is heated by the sun which heats the air in contact with it causing it to rise. The rising columns are often called thermals. Surface heating tends to produce cumulus clouds.
2. **Topography or orographic forcing** - The topography - or shape and features of the area - can cause clouds to be formed. When air is forced to rise over a barrier of mountains or hills it cools as it rises. Layered clouds are often produced this way.
3. **Frontal** - Clouds are formed when a mass of warm air rises up over a mass of cold, dense air over large areas along fronts. A 'front' is the boundary between warm, moist air and cooler, drier air.
4. **Convergence** - Streams of air flowing from different directions are forced to rise where they flow together, or converge. This can cause cumulus cloud and showery conditions.
5. **Turbulence** - A sudden change in wind speed with height creating turbulent eddies in the air.

### 2.1.2 Different cloud types

Unless otherwise stated, the information in this section follows Hamblyn (2008).

Clouds can be categorized into several different species and subspecies. Depending on what altitude the clouds have their base at, the clouds can be grouped into three main species.

- Low clouds (Cumulus, Stratus, Stratocumulus, Cumulonimbus).
- Medium clouds (Altostratus, Nimbostratus, Altocumulus).
- High clouds (Cirrus, Cirrostratus, Cirrocumulus).

#### 2.1.2.1 Low altitude clouds

Low altitude clouds usually have their base below 2 km above sea level.

Cumulus (figure 1a) are possibly the most recognizable cloud-type. They are the regular billowy looking clouds that fill the summer skies.

If the cumulus clouds grow taller they later form Cumulonimbus clouds (figure 1b). These are the ”thunderstorm” cloud giants with cloud peaks usually above 10 km above sea level.

Stratus (figure 1c) form a less dense cloud sheet. Very low altitude stratus may also be defined as mist or fog.

Stratocumulus (figure 1d), perhaps the most common cloud on earth, can form from the spreading out of cumulus clouds, or the lifting or breaking up of sheets of stratus.

Credit: *Brockenhexe @ Pixabay*

**(a)** A Cumulus cloud.

Credit: *NOAA/AOML/Hurricane Research Division. NOAA Photo Library: fly00890 @ flickr*

**(b)** Cumulonimbus cloud with anvil formation.

Credit: *JACLAU-DL @ Pixabay*

**(c)** A layer of stratus clouds.

Credit: *PIX1861 @ Pixabay*

**(d)** Stratocumulus clouds.

**Figure 1:** Low altitude clouds usually have their base below 2 km above sea level.

#### 2.1.2.2 Medium altitude clouds

Medium altitude clouds have their base usually between 2 km and 6 km above sea level.

Altostratus clouds form a thin featureless grey/blue layer that can spread to cover the entire sky.

Nimbostratus clouds form a grey cloud blanket that is usually filled with rain.

Altocumulus clouds (figure 2) can occur as individual rounded clouds in local patches or in vast layers.

Credit: *rkit @ Pixabay*

**Figure 2:** Altocumulus clouds.

#### 2.1.2.3 High altitude clouds

High altitude clouds usually have their base above 6 km above sea level.

Cirrus (figure 3a) take the form of filaments, strands or hooks.

Like other stratus variants, Cirrostratus is a cloud layer that may cover the sky.

Cirrocumulus (figure 3b) are high altitude clouds that look like individual small grainy clouds.

Credit: *strecosa @ Pixabay*

**(a)** Cirrus clouds.

Credit: *jingoba @ Pixabay*

**(b)** Cirrocumulus clouds.

**Figure 3:** High altitude clouds usually have their base above 6 km above sea level.

### 2.1.3 Lighting and clouds

Informally, light-rays enter the clouds, changes angle, bounces around and becomes absorbed when it comes in contact with the water droplets or ice crystals that the clouds consist of (Frost, 1998). This creates some special light characteristics for clouds:

**Absorption**: The light can be absorbed in the cloud.

**In-Scatter**: The light from a light-source behind clouds has the possibility to go through and exit the cloud towards the eye. This is what highlights the edges of the clouds when looking towards the sun, visible in figure 4a. This effect is often refereed to as a silver-lining.

**Out-Scatter**: The light from a light-source can ”bounce back” and exit the clouds in the opposite direction. Visible in figure 4b, the out-scattering for clouds have a tendency to create darker edges highlighting the billowy shapes.

Credit: *taniadelongchamp @ Pixabay*

**(a)** In-scatter creating the silver-lining towards the edges of the cloud.

Credit: *WolfBlur @ Pixabay*

**(b)** Out-scatter reflecting back and highlighting the clouds darker edges.

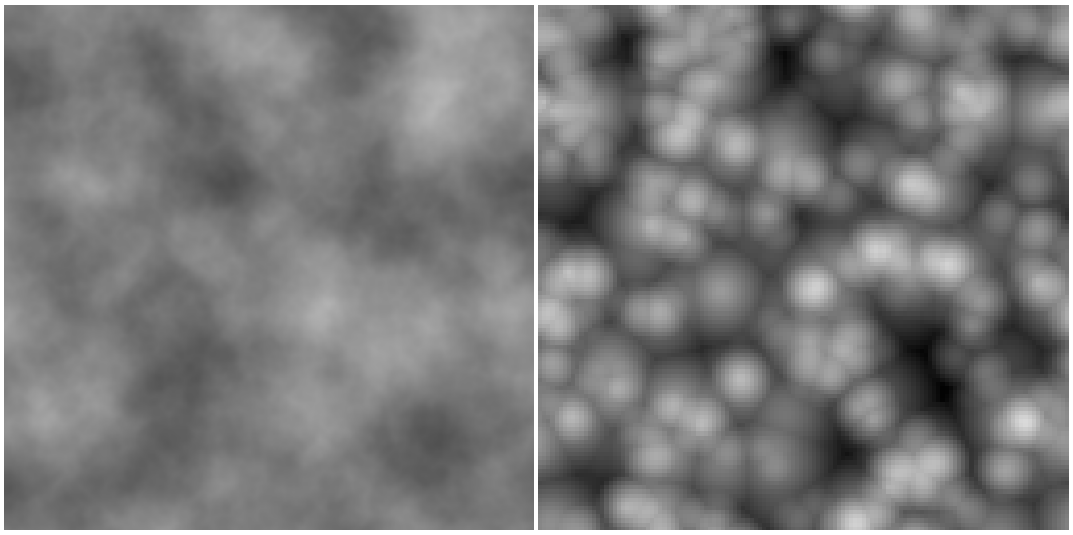
**Figure 4:** In-/out-scatter.

## 2.2 Rendering clouds

The term ”rendering” is often used to describe the rendering of non-transparent objects with shapes defined by a set of points that are grouped into polygons. This method has also been used to render clouds (Rare, 2018). However, while it is possible to render transparent polygons, issues with opacity may arise when trying to have the clouds be of varying density.

In order to have clouds of varying opacity, the cloud could instead be defined by the density at specific points rather than polygons. This requires that the method for rendering has to be altered to instead send rays through the medium. Ray-marching is a common method to sample a volumes density (Muñoz, 2014), a method that involves following the rays from the camera into the world/scene.

A popular method for modeling volumetric clouds is to have a series of different 2D and 3D textures that are combined to form shapes that resemble clouds (Schneider, 2016). The main shape is defined by an FBM (Mandelbrot and Ness, 1968) of Perlin noise (Perlin, 1985) combined with inverted[1] Worley noise (Worley, 1996). Figure 5a visualizes one example of Perlin noise and figure 5b visualizes one example of Worley noise. The Perlin noise creates fog-like structure while Worley noise adds more billowy looking shapes.



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

**Figure 5:** Examples of Perlin and Worley noise.

[1] Invering $(1 - noise)$ worley noise creates ”white blobs” instead of ”black blobs”.

# 3 Methods

All testing has been done on a Windows 10 system with a NVIDIA GTX 980Ti graphics card. The game-engine Stingray was used for this thesis work (AutoDesk, 2014). The shading language used was HLSL (Feinstein, 2013).

Some functions frequently used:

Function 1 converts/remaps a value from one range to another, where $v \in \mathbb{R}$ is the value to be remapped, $l_o \in \mathbb{R}$ is the left endpoint of original range, $h_o \in \mathbb{R}$ is the right endpoint of original range, $l_n \in \mathbb{R}$ is the left endpoint of new range, and $h_n \in \mathbb{R}$ is the right endpoint of new range.

$$R(v, l_o, h_o, l_n, h_n) = l_n + \frac{(v - l_o) \times (h_n - l_n)}{h_o - l_o} \tag{1}$$

Function 2 clamps the value $v \in \mathbb{R}$ to be in the range 0 to 1.

$$SAT(v) = \begin{cases} 0 & \text{if } v < 0 \\ 1 & \text{if } v > 1 \\ v & \text{otherwise} \end{cases} \tag{2}$$

Function 3 linearly interpolates between $v_0 \in \mathbb{R}$ and $v_1 \in \mathbb{R}$ by an amount of $i_{val} \in \mathbb{R}$. With $i_{val} = 0$, the function will return $v_0$, with $i_{val} = 1$, the function will return $v_1$, values in-between are interpolated, and values outside the range $[0, 1]$ are extrapolated.

$$L_i(v_0, v_1, i_{val}) = (1 - i_{val}) \times v_0 + i_{val} \times v_1 \tag{3}$$

Throughout the thesis, function/parameter names defined for one function/equation are not necessary redefined. However, a summary containing most function/parameter names and a short description can be found in appendix A.

Also worth mentioning, to improve readability, functions defined throughout the thesis only define input parameters if they are prone to change. One example is function 1, where all function parameters are defined due to it being called with multiple different input parameters. However, if the function is only called with the same input parameters, the parameters are not defined in the function header. Additionally, uppercase letters are used to specify functions and lowercase letters are used to specify parameters and variables.

## 3.1 Shapes / density

Following the work by Schneider (2016 & 2017), a combination of different values, 2D and 3D textures form the cloud shapes.

Presented here are the three different types of parameters/functions that govern the cloud shapes. Regular scalar probabilities (such as the global coverage), texture probabilities (weather-map, shape and detail textures) and additional functions that alter the cloud shape depending on the sample points altitude (height percentage in the cloud layer). A value is often referred to as a probability because values are multiplied, meaning that all probabilities combined decide whether clouds will form at specific points in 3D-space. Thus, resulting in a cloud-volume layer of varying density.

**Simplified shape pipeline:**

1. The locations where clouds can form in the sky is specified in a texture called the weather-map.
2. To achieve clouds that are rounded towards the bottom and top (not have flat edges), a shape-altering height-function is used.
3. Shape and detail noise is added to make the cloud density volume be cloud-like (have non-uniform density).
4. As the global coverage term increases, clouds can form on darker and darker areas, where ”darker” is defined as the weather-map value times the shape-height function.
5. A density-altering height-function is used to alter the density depending on the sample points height percentage in the cloud-layer.
6. The resulting cloud is multiplied with the global density term.

### 3.1.1 Scalar probabilities

The implemented clouds have their base at 400 meters ($ch_{start}$), the top can stretch up to 1000 meters ($ch_{stop}$). The seemingly low heights are an artistic decision. The clouds are made smaller and moved closer to the ground to allow for more varying clouds when moving around in the world.

The main scalar probabilities are:

- A global coverage term $g_c \in [0,1]$ controls the basic probability for clouds to appear.
- A global density term $g_d \in [0,\infty]$ determines the clouds global opacity.

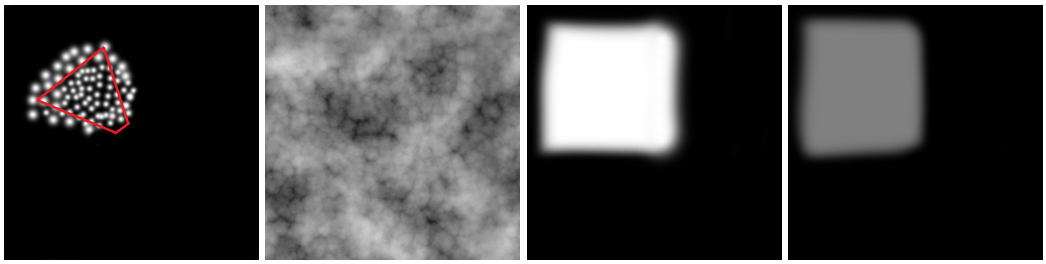
### 3.1.2 Weather-map

A weather-map is used to control where clouds can appear, the cloud height and the cloud density. The weather-map consists of a 2D texture with a resolution of 512x512 pixels that contains information on all four RGBA color channels. However, the color channels are not used the same way as when visualizing an image, rather to individually store data specifying certain cloud-attributes for different regions.

To enable more control, two color channels (red and green) are used to specify where clouds can appear. The probability that clouds will appear is calculated using function 4, where $w_{c0} \in [0,1]$ is the coverage value that corresponds to the weather-map's red color channel, and $w_{c1} \in [0,1]$ is the coverage value that corresponds to the weather-map's green color channel. This allows for more sparse clouds when the global coverage ($g_c$) is low and to be able to fill the entire sky when the global coverage goes towards a value of 1. The blue color channel describes the maximum cloud height ($w_h \in [0,1]$) and the alpha channel describes the cloud density ($w_d \in [0,1]$).

$$WM_c = \max(w_{c0},\ SAT(g_c - 0.5) \times w_{c1} \times 2) \tag{4}$$

Figure 6 visualizes an example weather-map. In the red color channel (figure 6a) some clouds are "hand-drawn" along with an approximate view-frustum[1] (red lines) used for when rendering figure 7. The green color channel (figure 6b) contains noise that is used to fill the skies with a non-uniform cloud-layer. The blue color channel (figure 6c) and the alpha channel (figure 6d) are in this case constant for the area inside the view-frustum. However, they can be altered pixel-wise to for example, have thin low-density clouds in one area and thick storm-clouds in another area. More weather-maps are presented in section 5.



**(a)** Low-coverage map (red channel).

**(b)** High-coverage map (green channel).

**(c)** Cloud peak height (blue channel).

**(d)** Cloud weather-map density (alpha channel).

**Figure 6:** The weather-map contains data in all four color channels (RGBA). The red and green color channels specify where clouds can form. The blue color channel specifies the maximum height of the cloud peaks and the alpha channel specifies the density.

[1] A view-frustum is the space in the world that might appear on the screen, a pyramid shape expanding from the camera in the viewing-direction.

Figure 7 visualizes the weather-map from figure 6 and show roughly where clouds can form.

**Figure 7:** The weather-map from figure 6 rendered to visualize where clouds can form. Whiter round areas towards the bottom hints where clouds are likely to form.

### 3.1.3 Height-dependent functions

The weather-map specifies two dimensions where the clouds can form. Thus, using only the weather-map, will result in clouds with flat edges. To create the third dimension (height), two different height-dependent functions are used, one to alter the shape and one to alter the density.

Term definitions for equations used in and below this section:

$p_h \in [0,1]$, the height percentage of where the currently sampled value is in the cloud.
$w_h \in [0,1]$, the cloud maximum height value from the weather-map's blue color channel.
$w_d \in [0,1]$, the cloud density value from the weather-map's alpha channel.

#### 3.1.3.1 Shape-altering height-function

Without altering the clouds with a shape-altering height-function, the clouds would have sharp edges.

Function 5 is used to round the cloud towards the bottom. This is done with the remapping function ($R$) to make $SR_b$ go from 0 to 1 when $p_h$ goes from 0 to 0.07.

$$SR_b = SAT(R(p_h,\ 0,\ 0.07,\ 0,\ 1)) \tag{5}$$

Function 6 is used to round the cloud towards the top. This is done with the remapping function ($R$) to make $SR_t$ go from 1 to 0 when $p_h$ goes from $w_h \times 0.2$ to $w_h$. Including $w_h$ in the function ensures that clouds are height-altered with the weather-map's blue color channel.

$$SR_t = SAT(R(p_h,\ w_h \times 0.2,\ w_h,\ 1,\ 0)) \tag{6}$$

The resulting shape-altering function 7 is a combination of functions 5 and 6. The combination ensures that the clouds are slightly rounded at the bottom and more towards the top. Together with the weather-map, function 7 creates clouds that are higher in altitude where the weather-map has a higher probability for clouds to form. Therefore, making the weather-map have higher probabilities towards the center of the clouds, will create rounded clouds.

$$SA = SR_b \times SR_t \tag{7}$$

The code for the shape-altering height-function can be found in appendix B.2.

#### 3.1.3.2 Density-altering height-function

In order to make the clouds be more fluffy at the bottom and have more defined shapes towards the top a height-dependent density function is used.

Function 8 is used to reduce the density towards the bottom. The function linearly increases with $p_h$ and further reduces the density in the $p_h$-interval $[0, 0.15]$.

$$DR_b = p_h \times SAT(R(p_h,\ 0,\ 0.15,\ 0,\ 1)) \tag{8}$$

To create a softer transition towards the top, function 9 is used. When $p_h$ goes from 0.9 to 1, $DR_t$ goes from 1 to 0.

$$DR_t = SAT(R(p_h,\ 0.9,\ 1.0,\ 1,\ 0))) \tag{9}$$

Function 10 is the density-altering height-function and is a combination of functions 8, 9, together with the introduction of the weather-map's density channel ($w_d \in [0,1]$). Therefore, function 10 ensures that clouds look more fluffy at the bottom, have more defined shapes towards the top, and that $w_d$ is used to have the weather-map influence the density. Multiplied with 2 so that $w_d > 0.5$ will create higher density clouds.

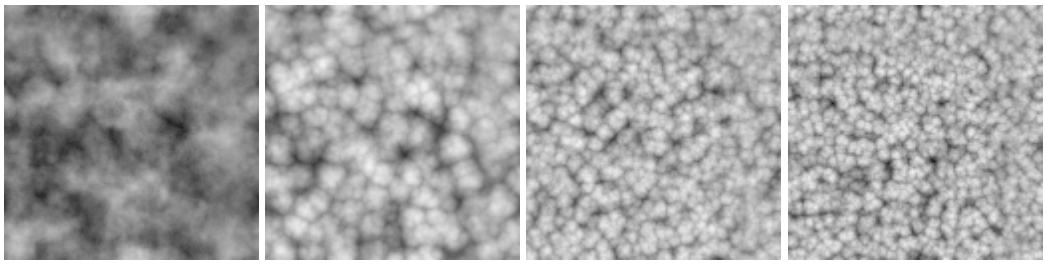
$$DA = g_d \times DR_b \times DR_t \times w_d \times 2 \tag{10}$$

The code for the density-altering height-function can be found in appendix B.3.

### 3.1.4 Shape and detail noise

The weather-map describes the probability of where clouds can form. However, the volumes in figure 7 have a uniform density with no cloud-like features. In order to add cloud-like features the next step is to ”carve” out the clouds shape using 3D noise.

The 3D noises in figure 8 visualizes one of 128 slices[2] that form the basic shape of the clouds. The 3D image used to extract the cloud-shapes has a resolution of $128x128x128$ pixels and contains information in four different color channels (RGBA).



**(a)** Low frequency Perlin-Worley (red channel).

**(b)** Medium frequency Worley noise (green channel).

**(c)** High frequency Worley noise (blue channel).

**(d)** Highest frequency Worley noise (alpha channel).

**Figure 8:** Noise with increasing frequency used to extract cloud shapes.

From figure 8, the noise in the green, blue and alpha color channels are combined using a process called FBM (Mandelbrot and Ness, 1968), the FBM is used to add some detail to the main shape noise in the red color channel. Specifically, function 11 is used for extracting the weighted shape-noise, where $sn_r \in [0,1]$ is the shape noise from the red color channel, $sn_g \in [0,1]$ is the shape noise from the green color channel, $sn_b \in [0,1]$ is the shape noise from the blue color channel, and $sn_a \in [0,1]$ is the shape noise from the alpha channel. Using the remapping-function 1 instead of combining all color channels into an FBM ensures that too much density is not carved away from the center of the clouds (Schneider, 2017). The code equivalent for function 11 can be found in appendix B.4.

$$SN_{sample} = R(sn_r,\ (sn_g \times 0.625 + sn_b \times 0.25 + sn_a \times 0.125) - 1,\ 1,\ 0,\ 1) \tag{11}$$

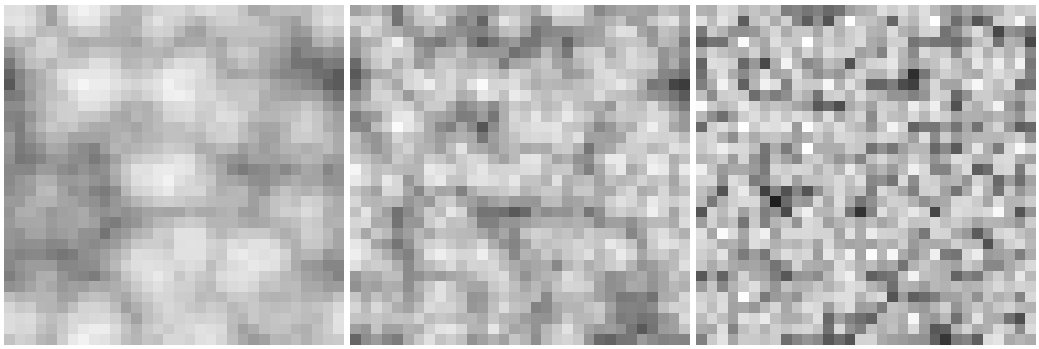
Combining the weather-map from figure 7 (section 3.1.2), the noise from figure 8, and the height-dependent functions (described in sections 3.1.3.1 and 3.1.3.2) now creates the cloud-like formations visualized in figure 9. Specifically, they are combined using function 12. The multiplication of $SN_{sample} \times SA$ is performed before the remapping to alter the shapes and the multiplication with $DA$ is performed after the remapping to alter the density. The remapping function ensures that more and more of $SN_{sample} \times SA$ is used as $g_c \times WM_c \rightarrow 1$.

$$SN = SAT(R(SN_{sample} \times SA,\ 1 - g_c \times WM_c,\ 1,\ 0,\ 1)) \times DA \tag{12}$$

[2] A 3D image can be thought of as multiple stacked 2D images. Thus, a 3D image visualized in 2D only contains one of the multiple different slices that describes the 3D image.

**Figure 9:** The weather-map from figure 6, the shape noise from figure 8 and the height dependent functions combined to create cloud shapes.

Continuing on the result from figure 9, more detail is added with the 3D noises from figure 10 (visualized as one 2D slice). A 3D image at a resolution of $32x32x32$ pixels containing information on three different color channels.



**(a)** Low frequency Worley noise (red channel).

**(b)** Medium frequency Worley noise (green channel).

**(c)** High frequency Worley noise (blue channel).

**Figure 10:** Detail noise with increasing frequency used to refine the cloud-shapes.

The detail noise from figure 10 is sampled with the FBM process using function 13, where $dn_r \in [0,1]$ is the detail noise from the red color channel, $dn_g \in [0,1]$ is the detail noise from the green color channel, and $dn_b \in [0,1]$ is the detail noise from the blue color channel.

$$DN_{fbm} = dn_r \times 0.625 + dn_g \times 0.25 + dn_b \times 0.125 \tag{13}$$

Function 14 is used to modify the detail noise, where $e$ is the natural number. The entire influence of the detail noise is reduced to be maximum 0.35, with $e^{-g_c \times 0.75}$ the influence is reduced with the global coverage, and the linear interpolation ensures that clouds are more fluffy towards the base and more billowy towards the peak.

$$DN_{mod} = 0.35 \times e^{-g_c \times 0.75} \times L_i(DN_{fbm},\ 1 - DN_{fbm},\ SAT(p_h \times 5)) \tag{14}$$

In order to use the shape noise together with the detail noise, function 15 replaces all instances of function 12. This so that the multiplication with $DA$ is performed last (see function 16).

$$SN_{nd} = SAT(R(SN_{sample} \times SA,\ 1 - g_c \times WM_c,\ 1,\ 0,\ 1)) \tag{15}$$

With equation 16, the final cloud density ($d$) is calculated with the normalized detail-noise from function 14 together with the shape noise function 15. The function $SN_{nd}$ is remapped using the modified detail noise $DN_{mod}$. The same way as with function 12, the multiplication with $DA$ is performed after the remapping to alter the density. The code equivalent to function 16 (apart from $DA$) can be found in appendix B.5.

$$d = SAT(R(SN_{nd},\ DN_{mod},\ 1,\ 0,\ 1))) \times DA \tag{16}$$

As visible in figure 11 (visualizing the final density $d$), adding the detail noise results in clouds that look more refined.

**Figure 11:** Continuing on the clouds from figure 9, detail noise is added to carve out the clouds more. Resulting in more refined looking clouds.

### 3.1.5 Anvil-formations

In order to create cumulonimbus clouds (described in section 2.1.2) with their iconic anvil shapes, some modifications to both the shape-altering height-function (section 3.1.3.1) and the density-altering height-function (section 3.1.3.2) were made.

To create anvil formations, all uses of $SA$ are replaced with function 17. How much the anvil shapes are altered depends on the height percentage, the maximum amount of anvil to apply ($a_a$) and the global coverage ($g_c$). If $a_a \times g_c$ equates to 0 the clouds are not modified, but if $a_a$ is set to 1 the amount of anvil applied will increase with the global coverage.

$$SA_{anvil} = (SA)^{SAT(R(p_h,\ 0.65,\ 0.95,\ 1,\ 1-a_a \times c_g))} \tag{17}$$

Figure 12a visualizes an example where $a_a = 0$ and $g_c = 1$, this results in a cloud that lacks resemblance with cumulonimbus clouds. Setting the $a_a = 1$ without modifying the density, results in figure 12b, this visualizes how the ordinary density model (increasing density over height) fails when an anvil is added.

**(a)** Cloud without anvil.

**(b)** Cloud with anvil. Unaltered density model.

**Figure 12:** Anvil comparison images.

To solve the density problem visualized in figure 12b, function 18 is used to modify to the density model when $a_a \to 1$. With $a_a = 1$, the function reduces the density with a multiplier going towards 0.2 as $\sqrt{p_h} \to 0.95$

$$DA_{anvil} = DA \times L_i(1,\ SAT(R(\sqrt{p_h},\ 0.4,\ 0.95,\ 1,\ 0.2)),\ a_a) \tag{18}$$

Replacing all uses of $DA$ with function 18 creates the final resulting "Cumulonimbus cloud" visible in figure 13.

**Figure 13:** Cloud with anvil. Altered density model.

### 3.2 Visualization / Ray-marching

To visualize the clouds, a method called ray-marching (Muñoz, 2014) is used to sample the cloud-volume layer described in section 3.1. Illustrated in figure 14, one use of ray-marching is when rays are sent into the scene from the cameras perspective, usually from each pixel/fragment. In the case of a partially transparent medium like clouds, the sampling points are incremented along the viewing-ray further into the scene by a fixed step-length and the density of the medium is accumulated at each step. At each sample where there is a non-zero cloud density, a similar ray-march towards the sun is initiated to accumulate the density between the sample point and the sun. The final alpha/opacity value for the pixel currently rendered is solely dependent on the density along the viewing-ray while the color (RGB) is further dependent on the color of the sun-light and the density between the sun and the sample point.

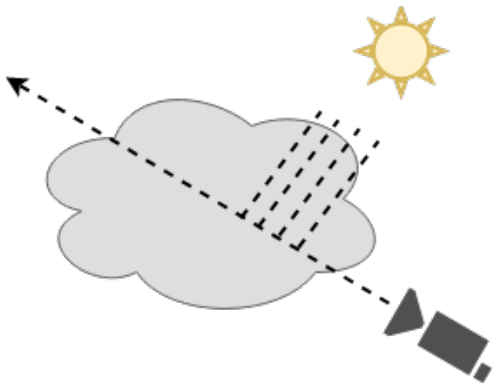


**Figure 14:** Ray-marching clouds. Visualizing one viewing-ray.

The ray-marching algorithm and how many pixels are ray-marched are the most important performance factors. Equation 19 is an approximation of the total execution-time the entire algorithm will take. In equation 19, $t_{tot}$ is the total execution time, $p$ is the number of pixel rendered, $n$ is the number of step taken along the viewing-ray, $s_n$ is the number of steps taken towards the sun, $t_{d0}$ is the time it takes to sample the clouds density at each step and $t_{d1}$ is the time it takes to sample the clouds density for each step towards the sun.

$$t_{tot} = p \times (n \times t_{d0} + n \times s_n \times t_{d1}) \tag{19}$$

### 3.2.1 Ray-march optimizations

The different variables in equation 19 provides good starting points for where optimizations can be applied.

#### 3.2.1.1 Marching interval

With the knowledge at what heights the clouds can appear at (section 3.1), it is possible to extract an interval to ray-march.

Illustrated in figure 15, with the clouds being in the atmosphere, finding the interval to march becomes the problem of finding where the viewing-ray intersects the sphere that determines the intervals close range and where it intersects the sphere that determines the intervals far range.

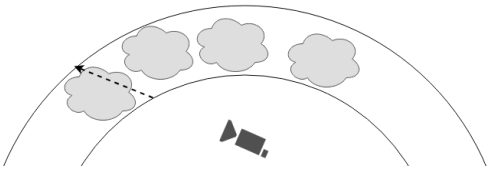


**Figure 15:** The ray-marching interval. The close range is defined by the inner sphere, the far range is defined by outer sphere.

#### 3.2.1.2 Step-lengths and related optimizations

The step-length is not directly related to $n$ from equation 19. However, the length of the interval to march divided by the step-length approximately equates to $n$.

Selecting a shorter step-length can create a finer detailed image but with worse performance. Selecting a bigger step-length can improve performance by reducing the number of steps necessary to march, but may degrade the quality by introducing banding[3] and noise. In the implementation described in this thesis, there are several optimizations done to alter the step-lengths in order to improve performance while still having visually pleasing clouds.

[3]Banding is a common problem in computer graphics that is often caused by using low precision colors. However, in the case of ray-marching, most of the banding is introduced due to taking large steps (see section 3.2.1.5).

Using equation 20, with a constant original step-length ($step_{orig}$) and a constant increase rate ($inc_{rate}$), the step-lengths are increased linearly the further away the camera is from the ray-marching starting distance ($dist_{start}$). Visualized in figure 16, this ensures that close clouds are highly detailed while clouds further away (where the distance to march through becomes bigger) do not take an unnecessarily long amount of time to compute. Proper parameter values for equation 20 are further tested and elaborated on in section 4.2.1.

$$step_{new} = step_{orig} + dist_{start} \times inc_{rate} \tag{20}$$

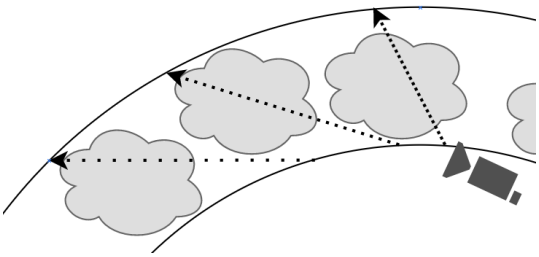


**Figure 16:** Take longer steps towards the horizon.

Furthermore, the implementation takes longer steps until a non-zero cloud density has been sampled along the ray (figure 17). When a cloud is found, the implementation backs up one step and then starts sampling with shorter steps. When the cloud has been passed through the implementation goes back to take longer steps. In order to not end up switching back and forth between these stages, a minimum short-step distance has to be covered before switching back to long steps.

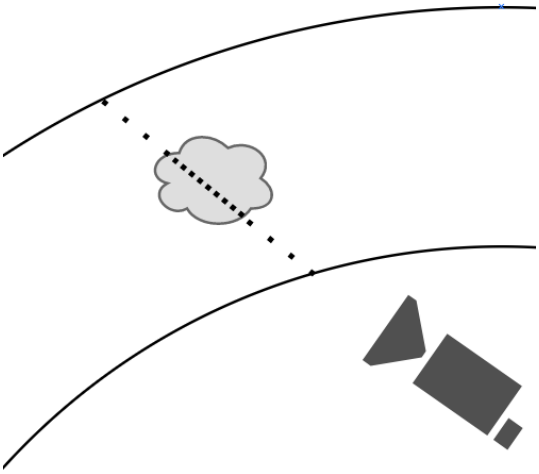


**Figure 17:** Take longer steps when not inside a cloud.

To mitigate the effect of $t_{d0}$ in equation 19, while taking longer steps, the implementation only samples the base shape of the clouds (not the detail noise, see section 3.1.4). Only sampling the basic shape takes less time and as visualized in figure 18, this is possible due to the fact that the final cloud shape will always be enclosed by the basic shape volume.

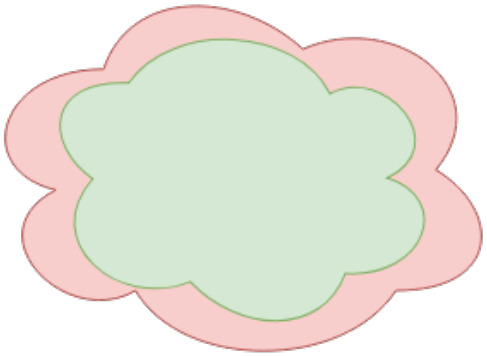


**Figure 18:** The final cloud shape inside the low detailed base shape.

Visualized in figure 19, in order to save performance by not doing unnecessary work, the implementation stops sampling when full opacity has been reached.

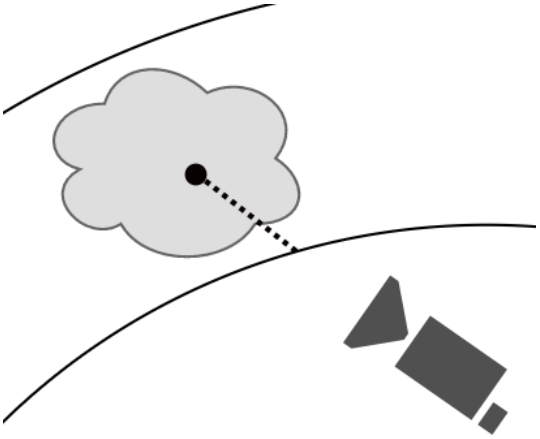


**Figure 19:** Stop ray-marching when enough cloud has been sampled to reach full opacity.

Additionally, there are two more step-altering functions used to make both the performance and the visual presentation more consistent. One function makes the step-length shorter when the global coverage ($g_c$) is low, the other function increases the step-length when the global density ($g_d$) is low. Both these functions are experimental, they are further described and tested in section 4.2.

#### 3.2.1.3 Steps towards the sun

Looking back at equation 19, the regular number of steps ($n$) times the number of sun-steps ($s_n$) can influence the final execution time the most.

For this implementation, the number of sun-steps ($s_n$) are limited to 4 (tested for optimal value in section 4.1). Equation 21 is used to calculate the step-length for the sun-march ($step_{sun}$). The equation ensures that the same distance is marched no matter how $s_n$ is set, the total distance marched will always be half of the height of the cloud layer ($ch_{stop} - ch_{start}$).

$$step_{sun} = \frac{0.5 \times (ch_{stop} - ch_{start})}{S_n} \tag{21}$$

To reduce the impact of $t_{d1}$ from equation 19, another optimization is that the sun-steps are sampled at decreasing fidelity the further away the sun-sample is from the current sample point along the viewing-ray. Fidelity is decreased by sampling all textures (2D and 3D) at increasing mipmap-levels[4]. Equation 22 is used for selecting the mipmap-level, truncated to only take the integer part, the first sun-step ($i$ iterates from 0 to $s_n$) is sampled at mipmap-level 0 while the fourth is sampled at mipmap-level 2.

$$miplevel = Integer(i \times 0.5) \tag{22}$$

Additionally, in order to minimize unnecessary work, no sun-march is initiated if the current sample point along the viewing-ray is not inside any cloud.

#### 3.2.1.4 Re-projection

Extracting the interval to march and balancing the step-lengths make for a significant performance difference. However, in order to reach the necessary performance figures, a drastic optimization had to be implemented. Therefore, having seemingly extracted the most of other optimizations, the only thing left to do is to reduce the number of pixels ($p$ in equation 19) that are rendered each frame. But reducing the resolution usually significantly decreases the picture quality. However, using re-projection, the quality can be preserved while still having the benefits of the lower rendering resolution. The re-projection solution implemented for this thesis follows Schneider (2016). The re-projection is done in two steps:

- Only render 1 pixel of each 4x4 pixel-block each frame. Essentially rendering the entire frame at quarter resolution each frame.
- To avoid ghosting[5], move/re-project the pixels not being ray-marched for the current frame to the appropriate place.

[4] A mipmap contains the same image of decreasing resolution (Williams, 1983).

[5] Ghosting is when objects on the screen leave a trail of previous positions behind them.

For each 4x4 block of the pixels, only one pixel is rendered each frame. A naive order would be to update them in the order visible in figure 20a. However, that is a pattern that can easily stand out and be visible to most people, a better solution is to update the pixels in a cross pattern, figure 20b.

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 |

**(a)** Naive order.

| | | | |
|---|---|---|---|
| 1 | 9 | 3 | 11 |
| 13 | 5 | 15 | 7 |
| 4 | 12 | 2 | 10 |
| 16 | 8 | 14 | 6 |

**(b)** Cross pattern.

**Figure 20:** Naive order versus a cross pattern ordering of rendering pixels.

Updating 1/16th of a frame each frame reduces the render-times by approximately the same amount. While this works exceptionally until something moves, as visible in figure 21, due to ghosting the result looks blurry when either the camera or the clouds move.

**Figure 21:** The introduced blurriness/ghosting when only updating 1/16th of the frame (without proper re-projection).

To visualize what causes the ghosting, a 2$x$2 block example is used instead of the 4$x$4 reality. Figure 22 visualizes one pixel-block on the screen where everything is fine until the camera rotates.

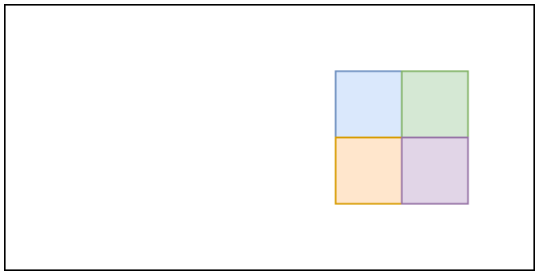


**Figure 22:** A block of 4 pixels on the screen.

However, as visualized in figure 23, when the camera is rotated, ghosting is introduced. From one frame to another, rotating the camera to the right should have all pixels within the screen move to the left by about the same distance as the rotation. However, in figure 23 only the pixel with an *X* on it is ray-marched/updated and the rest are as they where on the previous frame (figure 22). After three more frames, all pixels will be positioned on the left (their correct position). But until then, the ghosting is visually unpleasing.

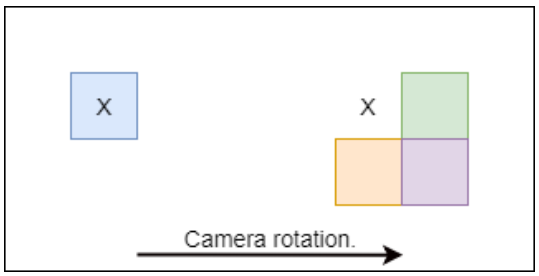



**Figure 23:** When the camera rotates, due to only one pixel (1 of 4) being updated, ghosting is introduced.

Re-projection is used to solve the ghosting issue, how much each pixel have moved between frames (often called motion-vectors) is saved in a render-target. Using the motion-vectors, the pixels that are not ray-marched for the current frame, are instead moved to where they should be in relation to the current and the previous state. How the motion-vectors are saved is visualized in figure 24. Visualized, the colors can be any combination of red and green. The red channel describes the motion for the screens x-axis while the green channel describes the motion for the screens y-axis. When nothing moves (either the clouds or the camera) the render-target is completely black. When movement occurs the render-target stores the movement in screen-space that has occurred for each pixel. As previously stated, the motion-vectors are used while re-projecting to move pixels rendered in previous frames to the approximately correct position for the new frame.

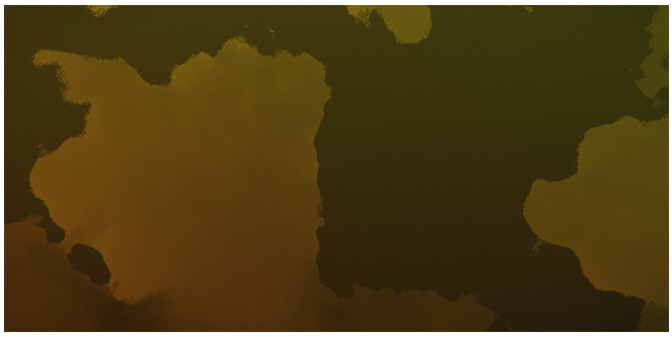
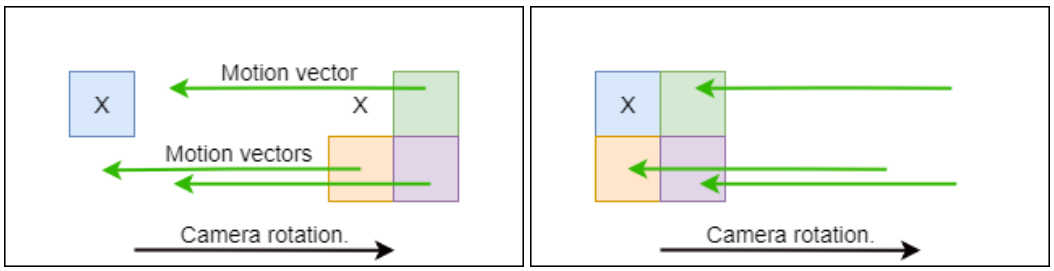


**Figure 24:** Visualized motion-vectors. Describes the movement in screen-space for each pixel (Stored in the red and green channel).

Thus, continuing on the $2x2$ example, figures 25a and 25b visualizes how the motion-vectors are used to move pixels that are not ray-marched/updated for the current frame. Worth noting, is that the implementation uses the motion-vectors the other way around than as they are visualized in figures 25a and 25b. For each pixel that is not ray-marched/updated, the implementation uses the motion-vectors to find the pixel from the previous frame that corresponds to the pixel currently rendered.




**(a)** Before moving the pixels. **(b)** After moving the pixels.

**Figure 25:** Moving pixels with motion-vectors usually creates a good approximation where the non-updated pixels should be. Non-updated, meaning the pixels not ray-marched for the current frame.

With the re-projection optimization, the final render-time is reduced to about 1/10th of the original render-time. Not 1/16th of the time, since some performance is lost to the re-projection algorithm. It must however be stated that some problems do arise with the reduction in rendered pixels.

- When moving through clouds the relative movement is too large and causes a visually unpleasing result. This can be solved by disabling the re-projection and only use the newly ray-marched pixels, which eliminates ghosting but introduces a more pixelated result.
- Even if some object is in front of the clouds, the pixels behind the objects one frame can be presented in the next frame, meaning that the clouds most certainly have to be rendered even while behind objects.
- If the motion-vector points at a position outside of the screen, the corresponding sample from the newly ray-marched pixels is used. Rotating the camera fast will thus create a more pixelated result towards the edges of the screen.

#### 3.2.1.5 Blue-noise

Blue-noise was introduced to deal with banding when using longer step-lengths. Visible in figure 26 is a sample of blue-noise[6]. In this thesis, blue-noise is used to offset starting distance when ray-marching. The blue-noise is placed in screen-space tiled in front of the camera and the pixel value $bn \in [0,1]$ from the noise is used with equation 23 to offset the ray-march start distance ($dist_{start}$) ranging from one step ($step_{new}$) backwards to one step forward.

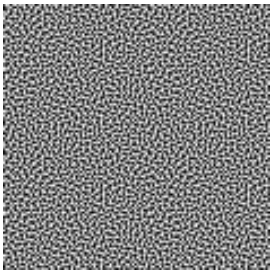
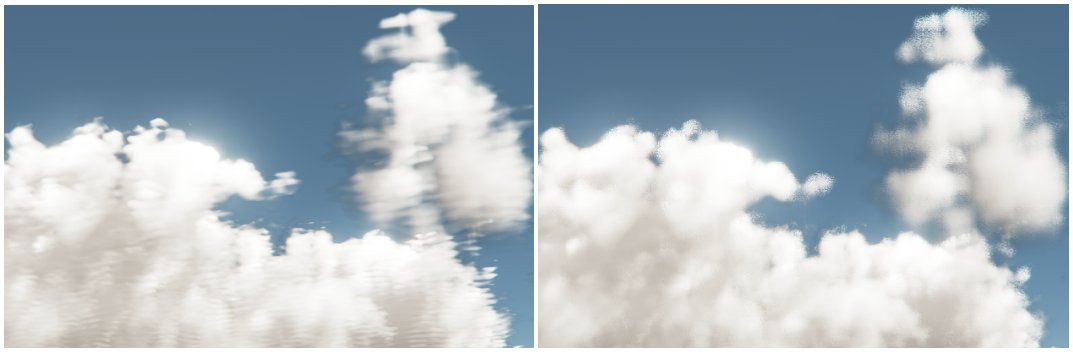


**Figure 26:** Blue-noise

$$dist_{start} += (bn - 0.5) \times 2 \times step_{new} \tag{23}$$

Figure 27a visualizes the banding without blue-noise, figure 27b visualizes the result when blue-noise is added to soften the image. Adding blue-noise gives a much better result, but introducing noise does have some visible drawbacks, the right image is considerably more noisy. However, with some softening/blurring of the image, this noise can be dampened.



**(a)** Without blue-noise. **(b)** With blue-noise.

**Figure 27:** Comparison between without blue noise and with blue noise. Blue-noise is used to offset the ray-march start distance ($dist_{start}$) ranging from one step ($step_{new}$) backwards to one step forward.

[6]Blue-noise is just called ”blue-noise”, not directly related to the color.

## 3.3 Lighting

All implemented code for calculating the lighting can be found in appendix B.6.

During the ray-march (see section 3.2), for each non-zero-density sample, the light is calculated with the density at the sample and the density between the sample and the sun. This is already a heavy approximation of the lighting properties presented in section 2.1.3, the light-scattering around inside of the clouds is lost completely. To approximate the lighting, the main component is a simplified version of Beer's law (1852). Beer's law calculates the attenuation of light in relation to the mediums properties. In this case, Beer's law is simplified to function 24, where $e$ is the natural number, $d_s \in [0,\infty)$ is the accumulated density towards the sun and $b \in [0,\infty)$ determines the amount of light absorption. With $d_s$ being somewhat detached from realistic density, $b$ is an artistic term used to balance the solution.

$$E(b,d_s) = e^{-b \times d_s} \tag{24}$$

Visible in figure 28, with Beer's law the cloud shapes and properties begin to show. But only looking at the clouds give little hints that the sun is in fact in the center of the screen, the silver-lining as a result of in-scatter (described in section 2.1.3) is missing.

**Figure 28:** Using Beer's law (function 24) used to calculate attenuation.

To create the silver-lining effect, Beer's law is complemented with Henyey-Greenstein's phase function 25, that can be used to calculate the in-/out-scattering for the medium. The term $g \in [-1,1]$ ranges from back-scattering to isotropic-scattering to in-scattering. The term $\theta \in [-1,1]$ is the "dot-angle" between the sun's light-direction and the ray-direction from the camera. This function is most importantly used for in-scattering to create the effect where clouds facing the sun has a bright silver lining (see figure 4a, section 2.1.3).

$$HG(\theta,g) = \frac{1}{4\pi} \frac{1-g^2}{[1+g^2-2g\ cos(\theta)]^{3/2}} \tag{25}$$

Adding Henyey-Greenstein's phase function to the lighting calculation creates the silver-linings visible in figure 29. With this phase function, the suns position is more correctly reflected in the clouds.

**Figure 29:** The result after adding Henyey-Greenstein's phase function 25.

To be able to add more centralized intensity around the sun, in addition to the Henyey-Greenstein's phase function, a secondary term can be used to, for example, achieve more dramatic sunsets (Schneider, 2017). The secondary term is calculated using function 26, where $cs_i$ is the amount of extra intensity to add around the sun, and $cs_e$ is the exponent deciding how centralized around the sun the extra intensity should be. The term $\theta$ continues to be the "dot-angle'.

$$IS_{extra}(\theta) = cs_i \times SAT(\theta)^{cs_e} \tag{26}$$

The final in-/out-scattering function 27 is a combination of functions 25 and 26, where $in_s \in [0,1]$ is the amount of in-scatter, and $out_s \in [0,1]$ is the amount of out-scatter. By selecting the maximum intensity from $HG(\theta, in_s)$ or $IS_{extra}(\theta)$, more intensity can be added around the sun. Additionally, the out-scatter term $HG(\theta, -out_s)$ is added together with the term $ivo \in [0,1]$ to be able to linearly interpolate values, allowing for more control of in- versus out-scatter (Högfeldt and Hillaire, 2016).

$$IOS(\theta) = L_i(max(HG(\theta, in_s),\ IS_{extra}(\theta)),\ HG(\theta, -out_s),\ ivo) \tag{27}$$

Using function 27 to combine the in- and out-scattering, figure 30b illustrates how the extra in-scattering function (26) can be used to alter the results.

**(a)** Without secondary term.

**(b)** With secondary term ($cs_i = 10$, $cs_e = 20$).

**Figure 30:** Illustration of the effect when adding the secondary term using function 26 (adding extra silver-lining around the sun).

The code used to calculate the in- and out-scatter can be found in function *InOutScatter*, appendix B.6.

### 3.3.1 Additional non-physical lighting alterations

Taking very large steps towards the sun (see section 3.2.1.3) does however create some issues that functions based in real-life physics seemingly cannot fix. Illustrated in figure 31a is the result when the clouds are rendered from the sun's perspective. Due to a high probability that the first sun-step is outside the cloud, it has almost no contours and can look like a big white spot. This is visually unpleasant since clouds also tend to create darker edges that highlights the billowy looking shapes as visible in figure 4b, see section 2.1.3. This can be solved by adding the completely non-physical ambient absorption function 28, where $d$ is the cloud-density sampled at the current sample point (see equation 16), and $os_a \in [0,1]$ is the amount of out-scattering ambient to apply. Function 28 uses the height-percentage to add more out-scatter ambient towards the top.

$$OS_{ambient} = 1 - SAT(os_a \times d^{R(p_h,0.3,0.9,0.5,1.0)}) \times (SAT(R(p_h,0,0.3,0.8,1.0)^{0.8}))) \quad (28)$$

Visualized in figure 31b, function 28 can be used to create more contours.

**(a)** Without out-scattering ambient applied.

**(b)** With out-scattering ambient applied. ($os_a = 0.9$)

**Figure 31:** Seen from the suns perspective. A comparison without and with the out-scattering ambient function 28.

The code used for the out-scattering ambient can be found in function *OutScatterAmbient*, appendix B.6.

As visualized in figure 32a, another issue that can arise from using Beer's law with long sun-steps is the high contrasts between light and dark areas. This issue can be partially fixed by clamping the maximum attenuation with function 29, where $d_s \in [0,\infty]$ is the density to the sun, and $a_c \in [0,1]$ is the value used to clamp the attenuation. Clamping the attenuation with $b = 12$ and $a_c = 0.2$ is visualized in figure 32b.

$$A_{clamp} = max(E(b,d_s),E(b,a_c))) \tag{29}$$

**(a)** Beer's law without attenuation clamping.

**(b)** Beer's law with attenuation clamping. ($b = 12$, $a_c = 0.2$)

**Figure 32:** The difference between no clamping and clamping the maximum attenuation from Beer's law 24 using function 29.

However, clamping the maximum attenuation results in another issue arising. As visualized in figure 33a, the grayer parts of the underside of a cloud look featureless due to clamping. To solve this issue, the cloud-density can be used to alter the attenuation, visible in figure 33b. Specifically, this is done with function 30, where $a_{min} \in [0,1]$ is the amount the influence the cloud-density ($d$) should have on the result.

$$A_{alter} = max(d \times a_{min},\ A_{clamp}) \tag{30}$$

**(a)** Without density dependent ambient applied.

**(b)** With density dependent ambient applied. ($a_{min} = 0.2$)

**Figure 33:** The difference between not addressing the dark areas and addressing by adding a density dependent ambient to the clamped result.

### 3.3.2 Lighting summary

There are several lighting values that can be changed to alter the clouds appearance:

- **Beer term** $\in [0,\infty]$, ($b$ in function 24).
  Determines the amount of light attenuation.
- **In-scatter** $\in [0,1]$, ($in_s$ in function 27).
  Determines the amount of light that is scattered trough the clouds.
- **Out-scatter** $\in [0,1]$, ($out_s$ in function 27).
  Determines the amount of light that is scattered back the direction the light entered the clouds.
- **In- versus out-scatter** $\in [0,1]$, ($ivo$ in function 27).
  Linear interpolation between **In-scatter** and **Out-scatter**.
- **Extra silver-lining intensity** $\in [0,1]$, ($cs_i$ in function 26).
  Determines how extra bright the surrounding area around the sun should be.
- **Extra silver-lining exponent** $\in [0,1]$, ($cs_e$ in function 26).
  Determines how centralized around the sun the extra brightness should be.
- **Ambient out-scatter** $\in [0,1]$, ($os_a$ in function 28).
  Adds detail to very bright areas.
- **Attenuation clamp value** $\in [0,1]$, ($a_c$ in function 29).
  Determines the how dark the clouds can be (clamps to a maximum attenuation).
- **Minimum attenuation ambient** $\in [0,1]$, ($a_{min}$ in function 30).
  Adds detail to dark areas.

The final lighting function 31 is a combination of all three lighting functions (attenuation, in-/out-scatter, and out-scatter ambient).

$$L_{final}(\theta) = A_{alter} \times IOS(\theta) \times OS_{ambient} \tag{31}$$

## 3.4 Color blending

To create believable scenery, it is also important to take the atmosphere into account. Figure 34a looks rather flat and without depth. To solve this the clouds color is blended with the background color more and more towards the horizon, resulting in figure 34b. Furthest down in the horizon, only the color from the background is used.

**(a)** Without atmospheric blending. **(b)** With atmospheric blending.

**Figure 34:** Comparison of without versus with color blending. Used to add more depth to far away clouds.

## 3.5 Rendering pipeline

Three rendering steps are used to accumulate and present the pixel-data that becomes the clouds.

1. Ray-march the cloud-layer to get the cloud color (dependent on lighting and density) stored in a quarter-resolution[7] render-target[8]. While marching the depth is fetched and weighted with the alpha value accumulated at that step, that depth is then used as the depth in space that the motion-vectors are calculated from. The motion-vectors are calculated and stored using the current and the previous camera are to determine where the previous camera had the correct pixel to use when re-projecting. The depth is also stored to be used as depth-check in rendering step 3.
2. Re-project the previous frame using the motion-vectors and update 1/16th of the frame with the new cloud color data.
3. Use the re-projected frame together with the depth-map[9] to render the clouds to the screen.

[7] In quarter-resolution both the x and y axis are divided by 4, resulting in a frame containing 1/16th of the pixels of the original.

[8] A run-time ”image” in which rendered data is stored in pixels.

[9] The depth-map is used to have objects appear in-front or behind the clouds depending on the relative depth between the cloud depth-map and the ”regular” depth-map.

The shadows are rendered in a different render-pass. For each fragment (piece of surface) in the scene, a ray-march for accumulating density is initialized from that fragment towards the sun. This is done at quarter-resolution and with large step-lengths to increase performance.

### 3.6 Movement

All movements are handled by offsetting the different textures that are used to create the cloud shapes. Offsetting the weather-map moves the entire clouds, offsetting the shape-noise makes the clouds change shape and offsetting the detail-noise can introduce some turbulence without the shapes being altered much. Offsetting all three in approximately the same direction can create clouds that move and change shape over time.

# 4 Experiments and corresponding results

To find the optimal trade-off between performance and visual quality can be a very difficult task. This section highlights some of the experiments performed to optimize the parameters in equation 32. As previously mentioned in section 3.2, $t_{tot}$ is the total execution time, $p$ is the number of pixel rendered, $n$ is the number of steps taken along the viewing-ray, $s_n$ is the number of steps taken towards the sun, $t_{d0}$ is the time it takes to sample the clouds density at each step and $t_{d1}$ is the time it takes to sample the clouds density for each step towards the sun.

$$t_{tot} = p \times (n \times t_{d0} + n \times s_n \times t_{d1}) \tag{32}$$

All render-times presented in this section are the time it takes to render the entire screen. The author of this thesis was the one comparing the results. For each experiment, the results are presented one by one and later summarized into a table.

## 4.1 Sun-step experiment

Due to $(n \times s_n)$ being the only per-pixel O($n \times m$) in equation 32, $s_n$ can have a huge performance impact if not greatly reduced.

The purpose of this experiment is to analyze the effect of different values for $s_n$. Specifically, to find the optimal trade-off between performance and visual quality. Measuring the visual aspect is done by comparing the resulting image against a reference image where $s_n = 100$. The experiment will start at $s_n = 1$, and be incremented by 1 until the desired fidelity is achieved, compared to the reference image.

As described in section 3.2.1.3, the value of $s_n$ does not affect the total length marched towards the sun.

Worth noting, when taking 100 sun-steps the decreasing mipmap-levels described in section 3.2.1.3 is disabled.

### 4.1.1 Reference image

Starting of with a large number ($s_n = 100$), the reference image has smooth lighting edges (going from dark to light). However, it takes about 60 ms to render.

**Figure 35:** Reference image. The visual result when taking 100 steps towards the sun for each sample (takes 60 ms).

### 4.1.2 Experiment results

Starting with $s_n = 1$, the result (figure 36b) is quite prominent. Taking 1 step gives good performance (down from 60 ms to 1.9 ms) but a rough visual result with sharp lighting transitions from dark to light. However, it is still possible to approximately determine where the sun is located.

**(a)** Reference image, $s_n = 100$.

**(b)** $s_n = 1$, (takes 1.9 ms).

**Figure 36:** Comparing the 100 sun-steps result to a 1 sun-step result.

By increasing the number of steps to 2 the resulting image in figure 37b has more different nuances visible, but fine detail is still somewhat missing. The render-time is increased to 2.8 ms.

**(a)** Reference image, $s_n = 100$. **(b)** $s_n = 2$, (takes 2.8 ms).

**Figure 37:** Comparing the 100 sun-steps result to a 2 sun-steps result.

Figure 38b visualizes the result when taking 3 steps towards the sun, even more different nuances are now visible. The render-time is further increased to 3.35 ms.

**(a)** Reference image, $s_n = 100$. **(b)** $s_n = 3$, (takes 3.35 ms).

**Figure 38:** Comparing the 100 sun-steps result to a 3 sun-steps result.

Taking 4 steps towards the sun, results in figure 39b. The resulting image starts to look more and more like the reference image (figure 39a), while diminishing visual returns start to become apparent. Renders in 3.85 ms.

**(a)** Reference image, $s_n = 100$. **(b)** $s_n = 4$, (takes 3.85ms).

**Figure 39:** Comparing the 100 sun-steps result to a 4 sun-steps result.

Finally, taking 5 steps towards the sun, results in figure 40b. With small visual improvements over 39b and a render-time increased to 4.4 ms.

**(a)** Reference image, $s_n = 100$. **(b)** $s_n = 5$, (takes 4.4 ms).

**Figure 40:** Comparing the 100 sun-steps result to a 5 sun-steps result.

#### 4.1.2.1 Experiment result summary

Table 1 summarizes the result with render-times and comments.

**Table 1** The render-times and comments when varying $s_n$ from equation 32

| Value of $s_n$ | Render-time | Visual comment |
|---|---|---|
| **100** | 60.00 ms | (reference) High lighting fidelity with smooth edges. |
| **1** | 1.90 ms | Rough visual result. |
| **2** | 2.80 ms | Much better than $s_n = 1$. However, fine detail is somewhat missing. |
| **3** | 3.35 ms | Adding upon $s_n = 2$, more fine detail starting to show. |
| **4** | 3.85 ms | Some improvements over $s_n = 3$ can be seen. However, the result show signs of diminishing visual returns. |
| **5** | 4.40 ms | Very small improvements over $s_n = 4$ can be seen. Stopping due to the result showing significant signs of diminishing visual returns. |

### 4.1.3 Experiment conclusion and discussion

The results shows that the number of steps taken towards the sun can be lowered to a single digit number while still retaining a good visual result. Taking less than 4 steps may give a rough result but going from 4 to 5 steps results in only small visual improvements. Thus, one can argue that taking 4 steps results in the best trade-off between performance and visual quality.

## 4.2 Step-length experiments

Finding the best marching step-length to get good performance while still retaining pleasing visuals can be difficult. Thus, to find the best step-length, some experiments where designed to find the best performance versus visual parameters for some of the different step-length altering functions highlighted in section 3.2.1.2.

More specifically the parameters to compare are the render-time versus the amount of noise and banding introduced as a side-effect of taking longer steps.

For all experiments, the interval to march is extracted (described in section 3.2.1.1) and the number of steps towards the sun will stay at a constant of 5 steps ($s_n = 5$).

To test the performance versus visual impact of different scenarios, three different experiments where designed.

- **Experiment 1**: The impact of longer step-lengths towards horizon.
- **Experiment 2**: The impact of shorter steps when at low global cloud coverage. Additionally, experiment 2 uses equation 33 from experiment 1 to the increasing step-lengths towards the horizon, specifically with $inc_{rate} = 0.04$.
- **Experiment 3**: The impact of longer steps when at low global cloud density.

### 4.2.1 Experiment: The impact of longer step-lengths towards horizon

The purpose of this experiment is to test both the visual and performance impact when increasing the step-lengths over distance. Specifically, the purpose is to reduce the number of steps necessary to march the interval ($n$ from equation 32) while still retaining a pleasing visual result.

Function 33 is used to increase the step-lengths over distance (towards the horizon). A simple function that increases linearly with the distance to the sphere ($dist_{start}$, the starting distance in the interval to march). The term $step_{orig}$ is the original step length and is always set to 14 during this experiment. The variable that will be tested is the increase rate ($inc_{rate}$), starting at a relative high value of 0.1 and reducing it until reaching pleasing visual fidelity. With $inc_{rate} = 0.1$, the step-lengths ($step_{new}$) are increased by 1 meter for every 10 meters in between the camera and the sphere.

$$step_{new} = step_{orig} + dist_{start} \times inc_{rate} \tag{33}$$

#### 4.2.1.1 Reference image

Figure 41 visualizes a reference image that does not use function 33 to scale the step-lengths. The figure provides a good looking visual result that takes 15 ms to render. Used in section 4.2.1.2 as a comparison for each test.

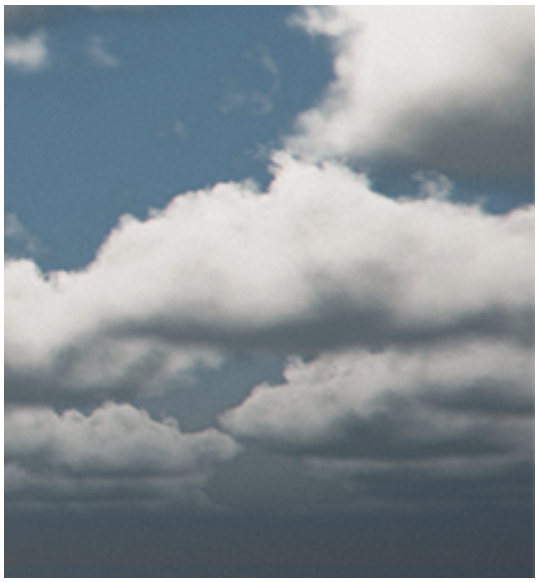


**Figure 41:** Reference image. Initial result, when using a constant step-length (takes 15 ms)

#### 4.2.1.2 Experiment results

Starting with $inc_{rate} = 0.1$. Figure 42b takes 1.3 ms to render, but it has clearly visible noise and banding compared to 42a.

**(a)** Reference image (15ms). **(b)** $inc_{rate} = 0.1$, (1.3 ms).

**Figure 42:** Comparing the reference (constant step-length) image to an image with distance scaling step-length.

Reducing the $inc_{rate}$ to 0.0667 increases the render-time up to 1.7 ms but a lot of the visible noise and banding has been reduced (figure 43b). However, the noise/banding is still very visible when scaling the image up to full-screen.

**(a)** Reference image (15 ms). **(b)** $inc_{rate} = 0.0667$, (1.7 ms).

**Figure 43:** Comparing the reference (constant step-length) image to an image with distance scaling step-length.

Further reducing the $inc_{rate}$ to 0.05 takes the render-time up to 2.1 ms and now most of the noise and banding has been reduced (figure 44b). However, the noise/banding is still somewhat visible when scaling the image up to full-screen.

**(a)** Reference image (15 ms). **(b)** $inc_{rate} = 0.05$, (2.1 ms).

**Figure 44:** Comparing the reference (constant step-length) image to an image with distance scaling step-length.

Finally, reducing the $inc_{rate}$ to 0.04 makes the render-time go up to 2.45 ms and even more noise and banding has been removed (figure 45b). Now the noise is not as clearly visible even when at full screen. However, looking at the furthest clouds in both figures, a still visible drawback is that in figure 45b, the clouds lighting is brighter and with less variation.

**(a)** Reference image (15 ms). **(b)** $inc_{rate} = 0.04$, (2.45 ms).

**Figure 45:** Comparing the reference (constant step-length) image to an image with distance scaling step-length.

**Experiment result summary**

Table 2 summarizes the result with render-times and comments.

**Table 2** The render-times and comments when varying $inc_{rate}$ from equation 33

| Value of $inc_{rate}$ | Render-time | Visual comment |
|---|---|---|
| **unused** | 15.00 ms | (reference) A good looking visual result that takes 15 ms to render. |
| **0.1** | 1.30 ms | Clearly visible noise and banding compared to reference image. |
| **0.0667** | 1.70 ms | A lot of the visible noise and banding has been reduced compared to $inc_{rate} = 0.1$, but it is still visible though. |
| **0.05** | 2.10 ms | Most of the visible noise and banding has been removed, but it is still visible when scaling the image up to full-screen. |
| **0.04** | 2.45 ms | Noise and banding reduced so that it is not visible that much even when scaling the image up to full-screen. |

#### 4.2.1.3 Experiment conclusion and discussion

As evident by the results it is possible to take longer steps towards the horizon and still get a visual result not that far from having a constant step-length. While there are reductions in image quality, it is not nearly as prominent as the increase in performance. The experiments also show that selecting the best $inc_{rate}$ can be hard, increasing the step-lengths too much over distance will create clearly visible noise and banding. However, comparing the result from the first test (figure 42b) and the last test (figure 45b), there is a noticeable visual difference just by reducing $inc_{rate}$ from 0.1 to 0.04.

The lighting difference for further clouds is quite prominent looking at figure 45b. However, the atmospheric blending is not activated (described in section 3.4). Activating the blending reduces the visibility of far away clouds, resulting in less prominent lighting, noise and banding drawbacks (figure 46).

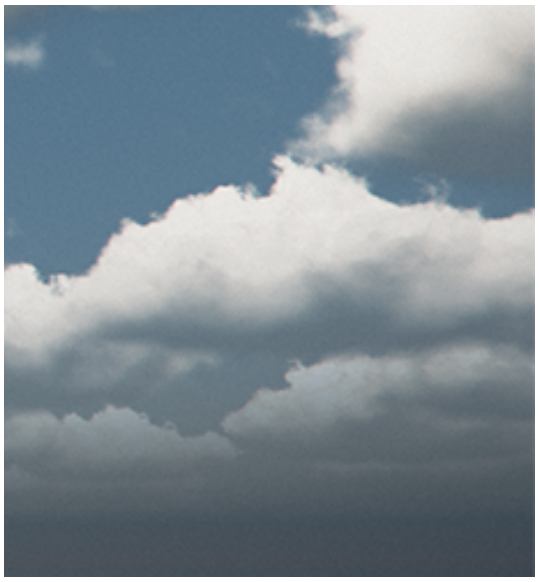


**Figure 46:** Figure 45b but with atmospheric blending activated.

Decreasing the $inc_{rate}$ even further will create a better visible result. But this experiment show that an $inc_{rate}$ of 0.04 is sufficient to create pleasing visuals.

### 4.2.2 Experiment: Shorter steps when low coverage

Increasing the step-length towards the horizon can result in a problem where small clouds far away look noticeable more noisy than big clouds. This issue comes from the steps being nearly as big or bigger than the clouds radius, making for example some pixels skip the cloud completely.

To mitigate this issue, function 34 can be used to decrease the step-length from $step_{orig}$ to $step_{new}$ when the global coverage ($g_c$) goes down.

$$step_{new} = step_{orig} \times max(mult_{min},\ min(1,\ R(g_c,\ 0.2,\ 0.7,\ mult_{min},\ 1))) \tag{34}$$

Since the term $g_c$ will stay at a constant 0.2 throughout the experiment, equation 34 can be simplified to equation 35.

$$step_{new} = step_{orig} \times mult_{min} \tag{35}$$

Also worth noting is that without this step-length altering, the render-times are quite a significant amount lower when reducing $g_c$ towards a value of 0. Thus, making for a bit of performance headroom when trying to reduce the noise.

The starting value used for the $mult_{min}$ will be a value of 0.1.

#### 4.2.2.1 Initial performance and visuals

Without altering the step-lengths, figure 47 visualizes the noisy issue when the coverage is low. Renders in 2 ms.

**Figure 47:** No coverage altering step-lengths (2ms).

Side note: looking rather rough, figure 47 visualizes a very small and far away area without use of the atmospheric blending described in section 3.4. Furthermore, neither the weather-map or the lighting parameter are altered properly to accommodate for small clouds.

#### 4.2.2.2 Experiment results

Starting with $mult_{min} = 0.1$ the step-lengths are decreased tenfold. While all the noise visible in figure 48a is mostly removed in figure 48b, the performance impact goes from 2 ms up to 8 ms. Figure 48b might take too long to execute, but it becomes a good reference image for the following tests.

**(a)** No coverage altering step-lengths (2 ms) **(b)** $mult_{min} = 0.1$, (8 ms).

**Figure 48:** Comparing the original (no coverage altering step-length) image to an image with coverage altering step-length.

Comparing figure 49a with figure 49b, increasing $mult_{min}$ to 0.3 gives no visible impact but significantly reduces the performance impact from 8 ms down to 3.3 ms.

**(a)** $mult_{min} = 0.1$, (8 ms). **(b)** $mult_{min} = 0.3$, (3.3 ms).

**Figure 49:** Comparing the reference image to an image with coverage altering step-length.

As visible in figure 50b, further increasing $mult_{min}$ to 0.5 gives a clear visible impact. The render-time goes down to 2.3 ms making it almost as low as the original of 2 ms (figure 47).

**(a)** $mult_{min} = 0.1$, (8 ms). **(b)** $mult_{min} = 0.5$, (2.3 ms).

**Figure 50:** Comparing the reference image to an image with coverage altering step-length.

Further testing, a value of $mult_{min} = 0.4$ (undocumented) resulted in the most consistent performance numbers when increasing $g_c$ from 0 to 1. Making it increase visual clarity while not increasing the worst case performance.

**Experiment result summary**

Table 3 summarizes this experiments result with render-times and comments.

**Table 3** The render-times and comments when varying $mult_{min}$ from equation 35

| Value of $mult_{min}$ | Render-time | Visual comment |
|---|---|---|
| **unused** | 2.00 ms | (initial result) The noisy issue when the coverage is low. |
| **0.1** | 8.00 ms | (reference) No visible noise or banding, but with a long render-time. |
| **0.3** | 3.30 ms | No visible impact compare to the reference image, but significantly reduces the performance impact from 8 ms down to 3.3 ms. |
| **0.5** | 2.30 ms | Results in a clear visible impact. The render-time goes down to 2.3 ms making it almost as low as the initial result of 2 ms. |

#### 4.2.2.3 Experiment conclusion and discussion

The result shows that introducing the $mult_{min}$ term (comparing figures 49a and 49b) can significantly increase visually clarity. While $mult_{min} = 0.5$ might be going to far, the best value seems to be somewhere in between 0.3 and 0.5. With $mult_{min} = 0.4$ resulting in the most consistent performance, it is chosen as the best value.

Altering the step-length with only the global coverage ($g_c$) does however not cover all cases, different weather situations such as a weather-map with very low coverage overall, might still result in small clouds being noisy.

### 4.2.3 Experiment: Longer steps when low density

Lowering the density significantly increases the render-times due to more cloud-volume having to be sampled before full opacity is reached. Having little to no density causes the implementation to march the entire interval.

The purpose of this experiment is to improve the render-times when at low density. To solve this it might be possible to take longer step-lengths due to the fact that each point sampled now has less influence contributing to the final color.

Equation 36 can be used to increase the step-lengths when the density is lowered. Decreasing $div_{min}$ will make the largest possible steps larger. The global cloud density ($g_d$) will stay at a constant 0.1.

$$step_{new} = step_{orig} \times \frac{1}{max(g_d, div_{min})^{0.8}} \tag{36}$$

#### 4.2.3.1 Reference image

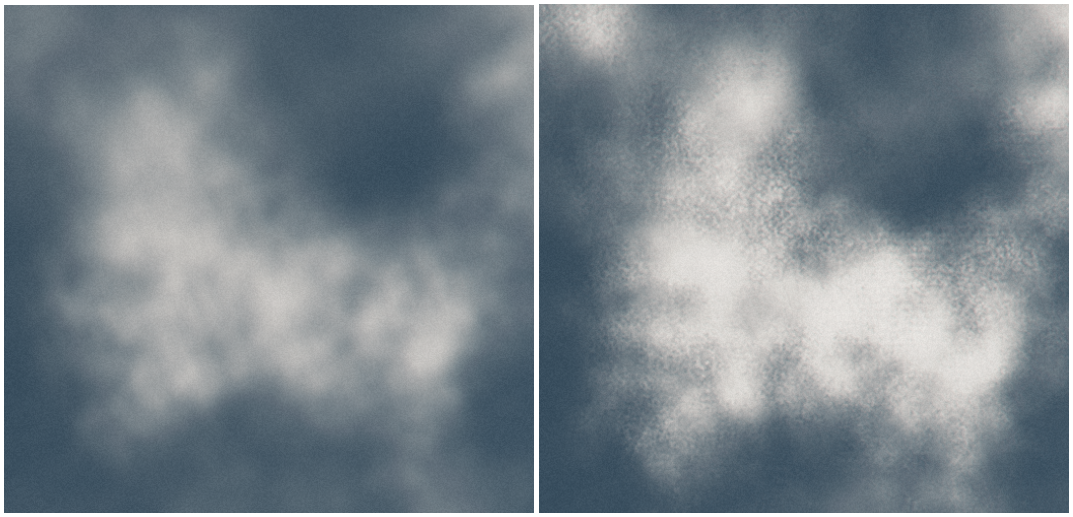
Not altering the step-lengths results in figure 51 being rendered in 5 ms, about double the render-time than looking the same direction but with the global cloud density ($g_d = 1$).

**Figure 51:** Reference image. No density altering step-lengths (5ms).

Worth noting is that figure 51 only shows a small part of the full rendered image, making possible noise more visibly prominent than it might be.

#### 4.2.3.2 Experiment results

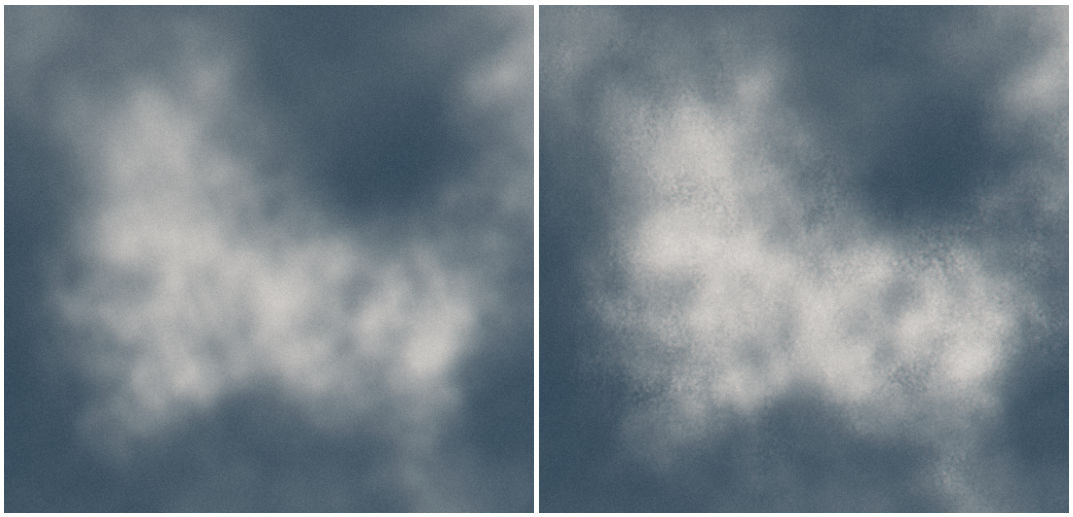
Starting with $div_{min} = 0.1$ the step-lengths are increased about tenfold, resulting in the render-time going from 5 ms down to 0.9 ms. However, it heavily increases the visible noise in figure 52b.



**(a)** Reference image (5 ms). **(b)** $div_{min} = 0.1$, (0.9 ms).

**Figure 52:** Comparing the reference (no density altering step-length) image to an image with density altering step-length.

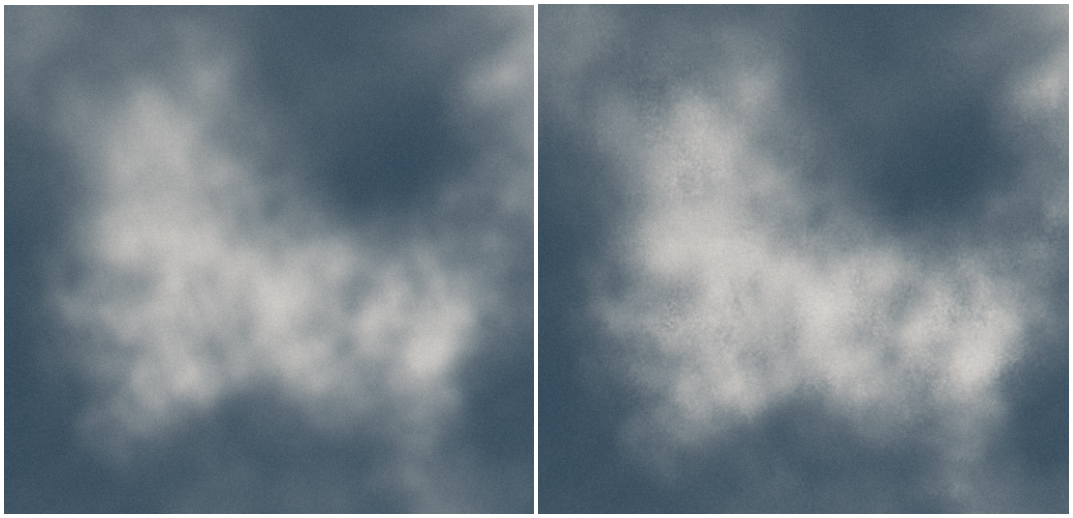
Increasing $div_{min}$ to 0.2 gives a less noisy result (figure 53b), but still very much apparent. It takes 1.5 ms to render.



**(a)** Reference image (5 ms). **(b)** $div_{min} = 0.2$, (1.5 ms).

**Figure 53:** Comparing the reference (no density altering step-length) image to an image with density altering step-length.

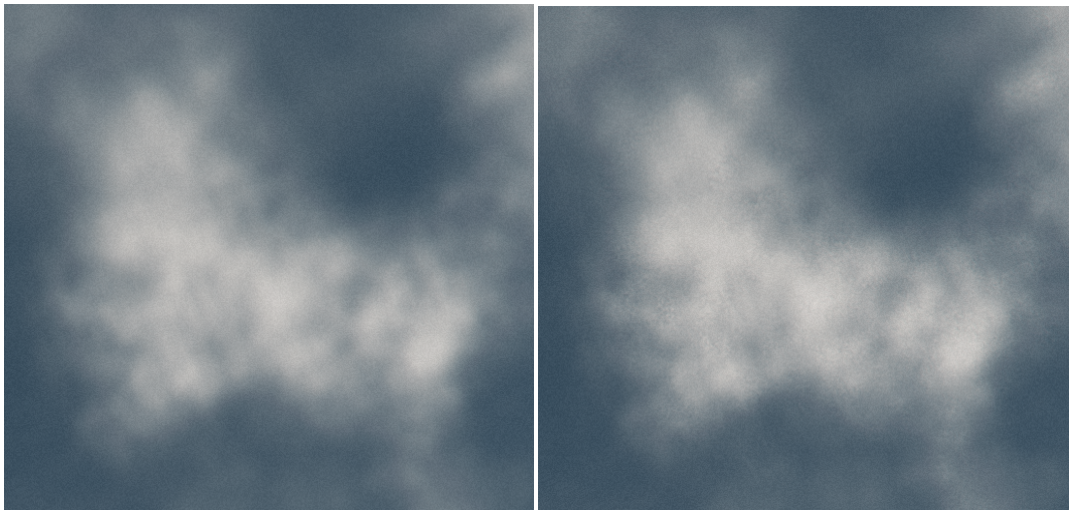
Further increasing $div_{min}$ to 0.3 gives a even less noisy result (figure 54b), but it is still quite apparent. It takes 2.0 ms to render.



**(a)** Reference image (5 ms). **(b)** $div_{min} = 0.3$, (2 ms).

**Figure 54:** Comparing the reference (no density altering step-length) image to an image with density altering step-length.

Finally, increasing $div_{min}$ to 0.4 removes most of the visible noise (figure 55b), this while taking 2.5 ms to render, half of the original result visible in figure 55a.



**(a)** Reference image (5 ms). **(b)** $div_{min} = 0.4$, (2.5ms).

**Figure 55:** Comparing the reference (no density altering step-length) image to an image with density altering step-length.

**Experiment result summary**

Table 4 summarizes this experiments result with render-times and comments.

**Table 4** The render-times and comments when varying $div_{min}$ from equation 36

| Value of $div_{min}$ | Render-time | Visual comment |
|---|---|---|
| **unused** | 5.00 ms | (reference) It takes about double the render-time than when looking at the same patch but with the global cloud density at a value of 1 . |
| **0.1** | 0.90 ms | Heavily increased the visible noise compared to the reference image. |
| **0.2** | 1.50 ms | Less noisy than when $div_{min} = 0.1$, but still very much visible. |
| **0.3** | 2.00 ms | Less noisy than when $div_{min} = 0.2$, but still somewhat visible. |
| **0.4** | 2.50 ms | Most of the visible noise removed. |

#### 4.2.3.3 Experiment conclusion and discussion

By increasing $div_{min}$ higher than 0.4, a less noisy result can be achieved. However, $div_{min} = 0.4$ is the highest tested value that result in a render-time close to when having the global density ($g_d$) at a value of 1. Thus, favoring consistent performance, the value of $div_{min} = 0.4$ gives a pleasing visual result while still keeping the render-times consistent.

# 5 Results

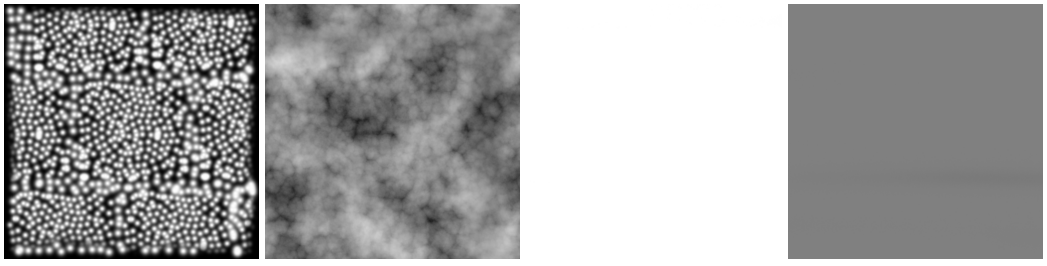
In this section the visual and performance results will be presented. The visual result is presented as different cloud types and different times of the day. With additional visualizations of cloud movement and varying cloud coverage. The performance result is presented separately with render-times for the different cloud-types.

## 5.1 Visual results

To create the perfect weather-map and set lighting parameters correctly for different cloud-types and scenarios can be a difficult task. However, it can be easy to create clouds that are visually pleasing enough, but perhaps not as good as they could be. To highlight the possibilities of the implementation presented in this thesis, this section will visualize the results of some weather-maps and lighting settings used to create different scenarios.

### 5.1.1 Cloud types

Starting with the weather-map from figure 56 and altering the height (figure 56c) and density (figure 56d) color channels can allow for different cloud types. However, in order to create some cloud-types (when for example creating stratus clouds), further tweaks should also be done to both the coverage color channels (figures 56a and 56b).



**(a)** Low-coverage map ("red" channel)

**(b)** High-coverage map ("green" channel)

**(c)** Cloud height ("blue" channel) (white = full height)

**(d)** Cloud density ("alpha" channel)

**Figure 56:** The weather-map contains data in all four color channels (RGBA). The red and green channels specify where clouds can form. The blue channel specifies the maximum height of the cloud peaks and the alpha channel specifies the density.

#### 5.1.1.1 Cumulus

To create Cumulus clouds no alterations has to be done to the weather-map from figure 56. Figure 57 visualizes the resulting clouds when the global coverage ($g_c$) is set to 0.6 and the global density ($g_d$) is set to 1.

**Figure 57:** Cumulus clouds created with the weather-map from figure 56 ($g_c = 0.6$, $g_d = 1$).

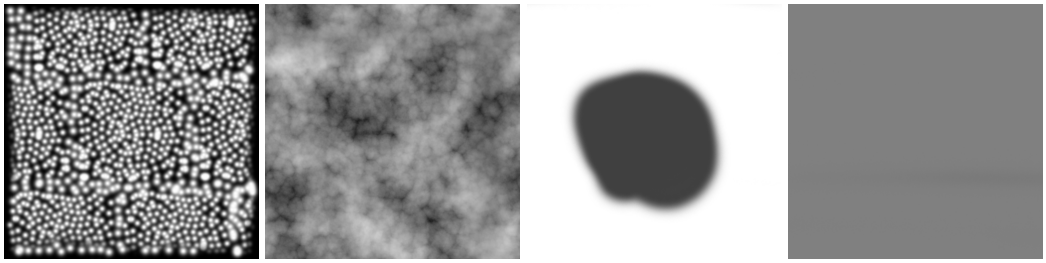
#### 5.1.1.2 Stratocumulus

In order to alter the clouds to look more like Stratocumulus, the global coverage ($g_c$) is increased to 0.8 and the global density ($g_d$) is reduced to 0.5. This makes the clouds larger but with lower density.

**Figure 58:** Stratocumulus clouds created with the weather-map from figure 56 ($g_c = 0.8$, $g_d = 0.5$)

#### 5.1.1.3 Stratus

To create Stratus clouds the weather-map is altered to be as in figure 59, the only change is that the height channel (figure 59c) is altered in the center to create thinner clouds. The global coverage ($g_c$) is increased to 1.0 to cover the entire sky and the global density ($g_d$) is reduced further to 0.35. This makes for a continuous clouds sheet with very low density.

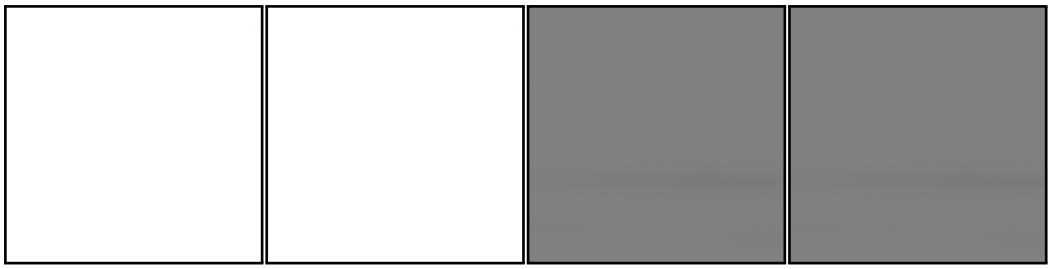


**(a)** Low-coverage map (red channel) **(b)** High-coverage map (green channel) **(c)** Cloud height (blue channel) **(d)** Cloud density (alpha channel)

**Figure 59:** The weather-map contains data in all four color channels (RGBA). The red and green channels specify where clouds can form. The blue channel specifies the maximum height of the cloud peaks and the alpha channel specifies the density.

**Figure 60:** Thin Stratus sheet created with the weather-map from figure 59, ($g_c = 1$, $g_d = 0.35$).

Even more continuous Stratus can be achieved by using a continuous weather-map as the one visible in figure 61. Here both coverage channels are maximized to have equal coverage across the sky. The weather-map from figure 61, results in the Stratus sheet visible in figure 62.



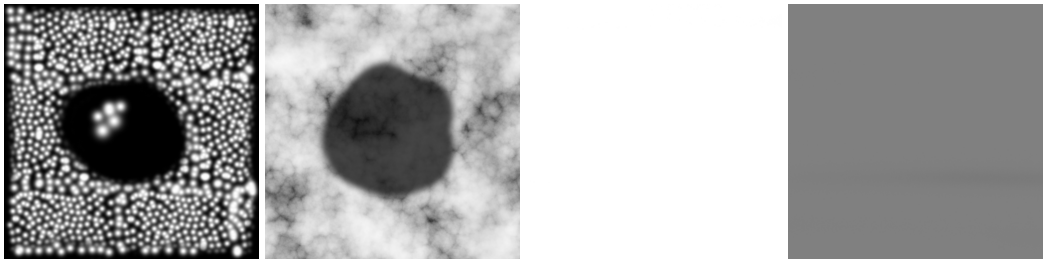
**(a)** Low-coverage map (red channel) **(b)** High-coverage map (green channel) **(c)** Cloud height (blue channel) **(d)** Cloud density (alpha channel)

**Figure 61:** The weather-map contains data in all four color channels (RGBA). The red and green channels specify where clouds can form. The blue channel specifies the maximum height of the cloud peaks and the alpha channel specifies the density.

**Figure 62:** Thicker and more continuous Stratus sheet. Created with the weather-map from figure 61, ($g_c = 1$, $g_d = 0.2$)

#### 5.1.1.4 Cumulonimbus

For Cumulonimbus some weather-map alterations are needed. Looking at the weather-map in figure 63, the larger dots in the center of figure 63a form the Cumulonimbus clouds visible in figure 64. To avoid the result being a uniform cloud-layer, the high-coverage maps probability is reduced around the Cumulonimbus clouds to not influence the result to much when $g_c = 1$, $g_d = 1$.



**(a)** Low-coverage map (red channel)

**(b)** High-coverage map (green channel)

**(c)** Cloud height (blue channel) (white = full height)

**(d)** Cloud density (alpha channel)

**Figure 63:** The weather-map contains data in all four color channels (RGBA). The red and green channels specify where clouds can form. The blue channel specifies the maximum height of the cloud peaks and the alpha channel specifies the density.

**Figure 64:** Cumulonimbus clouds created with the weather-map from figure 63 ($g_c = 1$, $g_d = 1$)

### 5.1.2 Lighting and different times of the day

To achieve pleasing lighting results for most cases the following settings (described in section 3.3, summarized in section 3.3.2) are used:

- **Beer term**, $b = 6$
- **In-scatter**, $in_s = 0.2$
- **Out-scatter**, $out_s = 0.1$
- **In- vs out-scatter**, $ivo = 0.5$
- **Extra silver-lining intensity**, $cs_i = 2.5$
- **Extra silver-lining exponent**, $cs_e = 2$
- **Ambient out-scatter**, $os_a = 0.9$
- **Attenuation clamp value**, $a_c = 0.2$
- **Minimum attenuation ambient**, $a_{min} = 0.2$

#### 5.1.2.1 Midday

To achieve the result in figure 65 no alterations to the lighting settings (listed in section 5.1.2) are done. The sun sits high on the sky and with a high intensity.

**Figure 65:** Midday, settings identical to settings listed in section 5.1.2

#### 5.1.2.2 Sunset/Sunrise

The color of the clouds is connected to the color of the light. Positioning the sun close to the horizon and altering the color to be more orange, creates the effect of a sunset/sunrise. For the result in figure 66 most of the lighting parameters are the same as the regular settings (listed in section 5.1.2), apart from lowering the suns intensity, the only part altered is the in-scattering to allow for a more defined silver-lining around the sun. The *in-scatter* term is increased from 0.2 to 0.5, the *extra silver-lining intensity* is increased from 2.5 to 8, and the *extra silver-lining exponent* is increased from 2 to 15.

**Figure 66:** Sunset, altering settings ($in_s = 0.5$, $cs_i = 8$, $cs_e = 15$)

#### 5.1.2.3 Night

During nighttime the light-source that illuminates the clouds is changed to be the moon. Making the light more blue and reducing the light-sources intensity results in figure 67. No alterations to the lighting setting from regular settings (listed in section 5.1.2) are done.

**Figure 67:** Nighttime, moon as the light-source, settings identical to settings listed in section 5.1.2

### 5.1.3 Cloud movement

It can be difficult to visualize cloud movement without a video, but figures 68a to 68c give some notion how the clouds can move across the sky. The movement of the clouds is performed by offsetting the textures that form the cloud shapes (see sections 3.1 and 3.6). Figures 68a to 68c visualizes the movement when offsetting both the weather-map and the shape noise incrementally 200 meters to the right (as seen from the current perspective).

**(a)** Clouds with no offset.

**(b)** Clouds with 200 meter offset to the right.

**(c)** Clouds with 400 meter offset to the right

**Figure 68:** Visualizing movement. Aside from the clouds moving to the right, small alterations can be observed for the cloud shapes.

### 5.1.4 Varying cloud coverage

As partly visualized in figures 69a to 69c, when the global coverage goes from 0 to 1, clouds emerge and build-up.

**(a)** Global cloud coverage equal to 0.3. **(b)** Global cloud coverage equal to 0.65. **(c)** Global cloud coverage equal to 1.

**Figure 69:** Clouds building up as the global coverage ($g_c$) goes towards a value of 1.

## 5.2 Performance results

As mentioned, all testing was done on a Windows 10 system with a NVIDIA GTX 980Ti graphics card. Two different screen-resolutions where tested, 1920$x$1080 and 960$x$540 pixels, both with re-projection (described in section 3.2.1.4) to essentially render at quarter resolution (down from the corresponding tested resolution). For example, this means that a resolution 960$x$540 has 240$x$135 pixels being ray-marched each frame.

All performance results (render-times) presented in table 5 are from the corresponding different rendered cloud-types presented in section 5.1.1. Due to small fluctuations, the render-times are rounded up to the nearest 0.05 ms step. When rendering at a resolution of 1920$x$1080 pixels, each of the different cloud types in table 5 have render-times close to the initial performance goal of 2 ms. To further reduce the render-times and for every case achieve less than the goal of 2 ms, the resolution can be lowered to 960$x$540 pixels.

**Table 5** The render-times for the different cloud-types presented in section 5.1.1

| **Cloud-type** | **Render-time** (1920x1080) | **Render-time** (960x540) | **Rendered image** |
|---|---|---|---|
| **Cumulus** | 2.05 ms | 0.85 ms | Figure 57 |
| **Stratocumulus** | 2.20 ms | 0.90 ms | Figure 58 |
| **Stratus** | 1.90 ms | 0.80 ms | Figure 60 |
| **Cumulunimbus** | 2.00 ms | 0.85 ms | Figure 64 |

## 5.3 Result summary

The implementation presented and evaluated in this thesis can handle multiple different cloud types and times of the day. Furthermore, the clouds can move across the sky and emerge from nothing to eventually fill the entire sky. This while keeping the render-times below the goal of 2 ms (with a resolution of 960$x$540 pixels).

# 6 Analysis and discussion

Rendering volumetric clouds is no easy task to get right. While it can be somewhat easy to get an implementation working in a couple of days, after that, the eternal ”fight” between performance and visual fidelity begins. Furthermore, especially for visual fidelity, the line between clear and noisy is extremely narrow and extending the step-lengths by even a small amount can reduce the visual quality immensely. However, since the entire solution is performance dependent on how many steps are taken, it is also important to find that perfect step-length for each and every one of the seemingly infinite cases.

## 6.1 Differences from previous work

Since the previous work done in this area does not mention all small solutions, it can be difficult to fully analyze what parts are better or worse. However, it can be argued that minor improvement have been achieved. Specifically, this thesis further highlights some of the issues with respective possible solutions when implementing volumetric clouds.

To further alter the shape of the clouds, previous work has also relied on adding curl-noise to to create turbulence (Schneider, 2016), while curl-noise can add visual improvements, it was not used in this implementation in favor of performance. A reason why curl-noise is less important for this implementation, is that Schneider focused of having the clouds move in-place and thus wanted to add something extra to ”move” the clouds, while the implementation described in this thesis moves the clouds around in a wind-direction.

## 6.2 Limitations

Using different types of textures (2D, 3D, see section 3.1) to create volumetric clouds have some inherent limitations. One of the clearest limitation being the inability to have different parts of the clouds moving with different speeds. For example, this results in swirling storms being difficult to mimic due to their rotation. Furthermore, the clouds are bound to a layer in the sky (see section 3.1.1), meaning that it can be difficult to, for example, have tornadoes evolving and reaching down to the ground, at least while keeping the render-times low.

### 6.2.1 Human inspection

The different banding and noise related ”performance versus visuals” variables presented throughout this thesis are subject to human inspection, meaning that the variables can somewhat be subject to human preference as well. However, while it can be argued that these variables can further be optimized, it can also be argued that through experiments, the final implementation presented in this thesis get many variables optimized to a very acceptable level.

The different cloud types and lighting settings presented in section 5.1 show how different parameters can be altered to create arguably very beautiful clouds. However, as a result of this thesis ultimately focusing more on the performance part of the issue, these purely visual results can be subjective in accordance to what the author, and sometimes people at *Arrowhead Game Studios*, thought was correctly looking clouds.

### 6.2.2 Medium and high altitude clouds

The initial goal for the thesis was to implement all three main altitude species of clouds (see section 2.1.2). However, only low altitude clouds have been implemented and presented throughout the thesis. While it can be possible to add more cloud layers for medium- or high altitude clouds the main problem becomes performance. Specifically, adding more layers of volumetric clouds will increase the render-times.

It can also be argued that high altitude clouds are so far away that they do not require a volumetric implementation to look convincing. Thus, one solution for high altitude clouds can be to add an image based implementation above the low altitude volumetric clouds.

### 6.2.3 Day / Night cycle

The volumetric-cloud implementation presented in this thesis can with small changes be integrated to support a full day and night cycle. However, due to other parts of the sky (for example, the sun and the atmosphere) not supporting that, it is difficult to visualize.

### 6.2.4 Traveling through clouds

Due to the extreme re-projection optimization implemented (see section 3.2.1.4), major visual quality issues arise when traveling though the clouds. This is caused by the pixels being updated slower than the frame-rate together with the re-projection not being sufficient when, for example, the cloud that the pixel represents is behind the camera in the next frame. While it is possible to solve this by only picking the latest rendered pixels, the quality will suffer due to the implementation rendering the clouds in quarter resolution.

### 6.2.5 Performance

It can be possible to create more realistic clouds with the same kind of implementation as the one presented in this thesis, but performance most probably will become a hindrance. If the only requirement was real-time, the render-time requirement would arguably instead be at $41.66ms$ (to achieve 24 frames per second with nothing else being rendered) and more realistic clouds would be able to be rendered.

While it must be possible to at least do minor optimization, the performance factors seemingly relies most on the hardware that renders the clouds. Thus, without more powerful hardware, it can be argued that only small improvements can be done to render-times.

## 6.3 Future work

Currently, the re-projection algorithm (see section 3.2.1.4) implemented is quite static (on or off, 1 pixel or 16 pixels). However, it can be possible to further improve it by dividing it into steps. For example, when the movement is a tiny bit too large to use all of the 16 latest pixels, instead of using only the latest pixel, use the 4 latest pixels[1]. Furthermore, it may be possible to get even better performance by applying the re-projection more aggressively (for example, 1/32 pixels instead of 1/16 pixels). However, reducing the number of pixels is prone to reduce the visual quality when using only the latest (newly ray-marched) pixels. To mitigate the reduction in visual quality, the dividing of the re-projection into steps might help, specifically by using the latest 2 pixels instead of the latest pixel for most of the cases where a large movement has occurred.

As visible in the different weather-maps presented (for example in section 5), they all are completely hand-drawn. A future improvement would be to create a weather-map generator. Allowing for easier ways to create different cloud-types.

Somewhat outside the scope of this thesis, another interesting future improvement would be to connect the volumetric clouds system to a separate weather-system. Thus allowing rain and thunder to roll in with respective rain- and thunder-clouds.

[1] This is assuming a pattern when rendering the pixels that every 4 frames updates one of the pixels in each $2x2$ corner-block inside the $4x4$ block.

# References


AutoDesk (2014). Stingray. https://www.autodesk.com/products/stingray/overview. [Online; accessed 2018-05-02].

Beer, A. (1852). Bestimmung der Absorption des rothen Lichts in farbigen Flüssigkeiten (Determination of the Absorption of Red Light in Colored Liquids). *Annalen der Physik und Chemie 86*, page 87–94.

Feinstein, D. (2013). *HLSL Development Cookbook*. Packt Publishing.

Frost, H. (1998). Light scattering by small particles. 47, Issue 2:87–94.

Grenier, J.-P. (2016). Volumetric Clouds - Stingray. https://area.autodesk.com/blogs/game-dev-blog/volumetric-clouds/. [Online; accessed 2018-01-21].

Hamblyn, R. (2008). *The Cloud Book, How to understand the skies*. David and Charles.

Högfeldt, R. and Hillaire, S. (2016). EA: Physically Based Sky, Atmosphere and Cloud Rendering. https://www.ea.com/frostbite/news/physically-based-sky-atmosphere-and-cloud-rendering. [Online; accessed 2018-01-21].

Mandelbrot, B. B. and Ness, J. W. V. (1968). Fractional brownian motions, fractional noises and applications. 10, No 4.:422–437.

MetOffice (2017). What are clouds? https://www.metoffice.gov.uk/learning/clouds/what-are-clouds. [Online; accessed 2018-01-18].

Muñoz, A. (2014). Higher order ray marching. 33, No 8.:167–176.

Perlin, K. (1985). An image synthesizer. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 287–296. ACM New York.

Rare (2018). Dynamic clouds in sea of thieves. https://www.seaofthieves.com/it/news/short-haul-3. [Online; accessed 2018-04-01].

Schneider, A. (2016). *GPU Pro 7, Real-time Volumetric Cloudscapes*, chapter 4, pages 97–127. CRC press.

Schneider, A. (2017). Nubis, Authoring Real-Time Volumetric Cloudscapes with the Decima Engine. http://advances.realtimerendering.com/s2017/Nubis%20-%20Authoring%20Realtime%20Volumetric%20Cloudscapes%20with%20the%20Decima%20Engine%20-%20Final%20.pdf. [Online; accessed 2018-02-06].

Simul (2013). TrueSky. https://simul.co/truesky/. [Online; accessed 2018-03-13].

Williams, L. (1983). Pyramidal parametrics. In *Proceedings of the 10th annual conference on Computer graphics and interactive techniques*, pages 1–11. ACM New York.

Worley, S. (1996). A cellular texture basis function. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 291–294. ACM New York.

# Appendices

# A Function/parameter names

This appendix can be used to look-up function-/parameter-names and short descriptions.

## A.1 Global parameters

$g_c \in [0,1]$, global coverage, controls the basic probability for clouds to appear.
$g_c \in [0,\infty]$, global density, determents the clouds opacity.
$p_h \in [0,1]$ is the height percentage of where (in altitude) in the cloud the current sample point is.

## A.2 The remapping function

Function 37 converts/remaps a value from one range to another.

$$R(v,l_o,h_o,l_n,h_n) = l_n + \frac{(v-l_o)\times(h_n-l_n)}{h_o-l_o} \tag{37}$$

$v_r \in \mathbb{R}$ is the value to be remapped.
$l_o \in \mathbb{R}$ is the left endpoint of original range.
$h_o \in \mathbb{R}$ is the right endpoint of original range.
$l_n \in \mathbb{R}$ is the left endpoint of new range.
$h_n \in \mathbb{R}$ is the right endpoint of new range.

## A.3 The saturate function

Function 38 clamps the value $v \in \mathbb{R}$ to be in the range 0 to 1.

$$SAT(v) = \begin{cases} 0 & \text{if } v \leq 0 \\ 1 & \text{if } v \geq 1 \\ v & \text{otherwise} \end{cases} \tag{38}$$

$v \in \mathbb{R}$ is the value to be clamped between 0 and 1.

## A.4 The linear interpolation function

Function 39 linearly interpolates between $v_0$ and $v_1$ by an amount of $i_{val}$.

$$L_i(v_0, v_1, i_{val}) = (1 - i_{val}) \times v_0 + i_{val} \times v_1; \tag{39}$$

$i_{val} \in \mathbb{R}$ is the value used to linearly interpolate between $v_0$ and $v_1$.
$v_0 \in \mathbb{R}$ is the value returned if $i_{val} = 0$.
$v_1 \in \mathbb{R}$ is the value returned if $i_{val} = 1$.

## A.5 Weather-map sampling function

Function 40 samples the weather-maps coverage.

$$WM_c = \max(w_{c0}, SAT(g_c - 0.5) \times w_{c1} \times 2) \tag{40}$$

$w_{c0} \in [0,1]$ is the coverage value that corresponds to the weather-maps red channel.
$w_{c1} \in [0,1]$ is the coverage value that corresponds to the weather-maps green channel.
$g_c$, see section A.1.

## A.6 Height-dependent functions

Term definitions for equations used in this section:

$w_h \in [0,1]$ is the cloud maximum height value from the weather-map.
$w_d \in [0,1]$ is the cloud density value from the weather-map.

### A.6.1 Shape-altering height-function

Function 41 rounds the cloud shapes towards the bottom.

$$SR_b = SAT(R(p_h,\ 0,\ 0.07,\ 0,\ 1)) \tag{41}$$

Function 42 rounds the cloud shapes towards the top. Also, alters the cloud height with $w_h$.

$$SR_t = SAT(R(p_h,\ w_h \times 0.2,\ w_h,\ 1,\ 0)) \tag{42}$$

Function 43 combines functions 41 and 42 to round the cloud shapes.

$$SA = SR_b \times SR_t \tag{43}$$

### A.6.2 Density-altering height-function

Function 44 reduced the cloud density towards the bottom.

$$DR_b = p_h \times SAT(R(p_h,\ 0,\ 0.15,\ 0,\ 1)) \tag{44}$$

Function 45 reduced the cloud density towards the top.

$$DR_t = SAT(R(p_h,\ 0.9,\ 1.0,\ 1,\ 0))) \tag{45}$$

Function 46 combines functions 44 and 45, together with the global density $g_d$ and the weather-maps density value $w_d$ to alter the cloud density.

$$DA_{regular} = g_d \times DR_b \times DR_t \times w_d \times 2 \tag{46}$$

## A.7 Shape noise functions

$sn_r \in [0,1]$ is the shape noise from the red color channel.
$sn_g \in [0,1]$ is the shape noise from the green color channel.
$sn_b \in [0,1]$ is the shape noise from the blue color channel.
$sn_a \in [0,1]$ is the shape noise from the alpha channel.

Function 47 is used to sample the shape noise into a scalar.

$$SN_{sample} = R(sn_r,\ (sn_g \times 0.625 + sn_b \times 0.25 + sn_a \times 0.125) - 1,\ 1,\ 0,\ 1) \tag{47}$$

Function 48 is used to combine the weather-map (function 40), the height altering functions 43 & 46, and the shape noise (function 47)

$$SN_{shape} = R(SN_{sample} \times SA,\ 1 - g_c \times WM_c,\ 1,\ 0,\ 1) \times D_{regular} \tag{48}$$

## A.8 Detail noise functions

$dn_r \in [0,1]$ is the detail noise from the red color channel.
$dn_g \in [0,1]$ is the detail noise from the green color channel.
$dn_b \in [0,1]$ is the detail noise from the blue color channel.

Function 49 is used to sample the detail noise into a scalar.

$$DN_{fbm} = dn_r \times 0.625 + dn_g \times 0.25 + dn_b \times 0.125 \tag{49}$$

Using function 50, the noise from function 49 is modified.

$$DN_{mod} = 0.35 \times e^{-g_c \times 0.75} \times L_i(DN_{fbm},\ 1 - DN_{fbm},\ SAT(p_h \times 5)) \tag{50}$$

## A.9 Combining detail and shape noise

Equation 51 is used to combine the shape from function 48 with the detail noise from function 50.

$$d = SAT(R(SN_{shape},\ DN_{mod},\ 1,\ 0,\ 1))) \tag{51}$$

$d$, the final cloud density.

## A.10 Anvil formations

To create anvil shapes, function 52 replaces the shape-altering height-function 43. The term $a_a$ is the maximum amount of anvil to apply.

$$SA_{anvil} = (SA)^{SAT(R(p_h,\ 0.65,\ 0.95,\ 1,\ 1-a_a \times c_g))} \tag{52}$$

To reduce the anvil shapes density, function 53 replaces the density-altering height-function 46.

$$DA_{anvil} = DA \times L_i(1,\ SAT(R(\sqrt{p_h},\ 0.4,\ 0.95,\ 1,\ 0.2)),\ a_a) \tag{53}$$

## A.11 Lighting functions

### A.11.1 Beer's law

Function 54 is a simplified version of Beer's law and is used to calculate light attenuation.

$$E(b, d_s) = e^{-b \times d_s} \tag{54}$$

$d_s \in [0, \infty)$ is the accumulated density towards the sun.
$b \in [0, \infty)$ determines the amount of light absorption.

### A.11.2 Henyey-Greenstein's phase function

Henyey-Greenstein's phase function 55 can be used to calculate the in/out-scattering for a medium.

$$HG(\theta, g) = \frac{1}{4\pi} \frac{1 - g^2}{[1 + g^2 - 2g\ cos(\theta)]^{3/2}} \tag{55}$$

$g \in [-1, 1]$ ranges from back-scattering to isotropic-scattering to in-scattering.
$\theta \in [-1, 1]$ is the "dot-angle" between the sun's light-direction and the ray-direction from the camera.

### A.11.3 Extra in-scattering function

Function 56 can be used to create more dramatic sunsets.

$$IS_{extra}(\theta) = cs_i \times SAT(\theta)^{cs_e} \tag{56}$$

$cs_i$ is the amount of extra intensity to add around the sun.
$cs_e$ is the exponent deciding how centralized around the sun the extra intensity should be.

### A.11.4 Combined in-/out-scattering function

Function 57 combines function 55 and 56 to create the in-/out-scattering function.

$$IOS(\theta) = L_i(max(HG(\theta, in),\ IS_{extra}(\theta)),\ HG(\theta, -out),\ ivo) \tag{57}$$

$in \in [0,1]$ is the amount of in-scatter.
$out \in [0,1]$ is the amount of out-scatter.
$ivo \in [0,1]$ linearly interpolates between $in$ and $out$.

### A.11.5 Extra out-scattering ambient

Function 58 can be used used to create darker edges, specifically for clouds facing the sun.

$$OS_{ambient} = 1 - SAT(os_a \times d^{R(p_h,0.3,0.9,0.5,1.0)}) \times (SAT(R(p_h,0,0.3,0.8,1.0)^{0.8}))) \tag{58}$$

$os_a \in [0,1]$ is the amount of out-scattering ambient to apply. $d$, the cloud density.

### A.11.6 Attenuation clamping equation

Equation 59 can be used to clamp the values from function 54.

$$A_{clamp} = max(E(b,d_s),E(b,a_c))) \tag{59}$$

$a_c \in [0,1]$ is the value used to clamp the attenuation.

### A.11.7 Density minimum ambient

Equation 60 can be used to add nuances to the result from equation 59.

$$A_{alter} = max(d \times a_{min},\ A_{clamp}) \tag{60}$$

$a_{min} \in [0,1]$ is the amount the influence the cloud-density ($d$) should have on the result.
$d$, the cloud density.

### A.11.8 Lighting combined

The final lighting function 61 is a combination of functions 60, 57 and 58).

$$L_{final}(\theta) = A_{alter} \times IOS(\theta) \times OS_{ambient} \tag{61}$$

# B Code

To understand the code presented in this appendix there are some basic functions that might need an initial explanation.

The **saturate** function is a built-in function in the HLSL shader language that clamps the inputed value to be in the range $[0, 1]$. Meaning, every number under 0 will be returned as 0, every number above 1 will be returned as 1. Numbers in between 0 and 1 are returned the same as inputed.

The **lerp** function (also a built-in function) linearly interpolates between two values by a specified amount.

## B.1 ReMap code

The ReMap function converts a value from the one range to another.

ReMap example: Converting a value of 0.66 from the range $(0, 1)$ to the range $(0, 2)$ results in 1.32 being returned.

**Listing B.1:** The ReMap function converts a value from the one range to another.

```
/ \times  Converts value from one range to another
 \times /
float ReMap(float value, float old_low, float old_high, float new_low,
    float new_high){

  float ret_val = new_low + (value - old_low)  \times  (new_high -
    new_low) / (old_high - old_low);

  return ret_val;
}
```

## B.2 Height-dependent shape altering function

**Listing B.2:** Height-dependent function that makes the clouds slightly rounded towards the bottom and quite much towards the top.

```
float HeightAlter(float percent_height,float4 weather_map){

  //Round bottom a bit
  float ret_val =  saturate(ReMap(percent_height, 0.0,0.07,0.0,1.0));

  //Round top a lot
  float stop_height = saturate(weather_map.b+0.12);
  ret_val  \times = saturate(ReMap(percent_height, stop_height \times
    0.2,stop_height,1.0,0.0));

  //Apply anvil (cumulonimbus/"giant storm" clouds)
  ret_val = pow(ret_val, saturate(ReMap(percent_height,0.65,0.95,1.0,(1 -
     cloud_anvil_amount  \times  global_coverage))));

  return ret_val;
}
```

## B.3 Height-dependent density altering function

**Listing B.3:** Height dependent function that alters the cloud density in regards to the 0 to 1 height percentage (the percentage of on what altitude in the 400-1000 meters the sample point is).

```

float DensityAlter(float percent_height, float4 weather_map){

  //Have density be generally increasing over height
  float ret_val = percent_height;

  //Reduce density at base
  ret_val  \times = saturate(ReMap(percent_height,0.0,0.2,0.0,1.0));

  //Apply weather_map density
  ret_val  \times = weather_map.a  \times  2;

  //Reduce density for the anvil (cumulonimbus clouds)
  ret_val  \times = lerp(1,saturate(ReMap(pow(percent_height,0.5)
    ,0.4,0.95,1.0,0.2)),cloud_anvil_amount);

  //Reduce density at top to make better transition
  ret_val  \times = saturate(ReMap(percent_height,0.9,1.0,1.0,0.0));

  return ret_val;
}
```

## B.4 Basic noise function

**Listing B.4:** Function used to sample the low frequency shape-noise.

```
float shape_noise = shape_sample.g  \times  0.625 + shape_sample.b  \
    times  0.25 + shape_sample.a  \times  0.125;
shape_noise = -(1 - shape_noise);
shape_noise = ReMap(shape_sample.r,shape_noise,1.0,0.0,1.0);
```

## B.5 Detail noise function

**Listing B.5:** Functions used to sample and apply the high frequency detail_noise to the previously sampled shape_noise.

```
float detail = (detail_noise.r)  \times  0.625 + (detail_noise.g)  \times
     0.25 + (detail_noise.b)  \times  0.125; //FBM

//For low altitude regions the detail noise is used (inverted)
//to instead of creating round shapes,
//create more wispy shapes. Transitions to round shapes over altitude.
float detail_modifier = lerp(detail,1-detail,saturate(percent_height  \
    times  5.0));

//Reduce the amount of detail noise is being "subtracted"
//with the global_coverage.
detail_modifier  \times = 0.35 \times exp(-global_coverage \times 0.75);

//Carve away more from the shape_noise using detail_noise
final_density = saturate(ReMap(shape_noise,detail_modifier,1.0,0.0,1.0));
```

## B.6 Lighting calculations

**Listing B.6:** All functions used to calculate the lighting (having previously sampled the density and the density to the sun). Continued in next listing.

```
float HG( float cos_angle, float g){

  float g2 = g  \times  g;
  float val = ((1.0 - g2) / pow( 1.0 + g2 - 2.0  \times  g  \times
    cos_angle, 1.5)) / 4  \times  3.1415;

  return val;
}

float InOutScatter(float cos_angle){
  float first_hg = HG( cos_angle, cloud_inscatter);
  float second_hg =  cloud_silver_intensity  \times  pow( saturate(
    cos_angle), cloud_silver_exponent);

  float in_scatter_hg = max(first_hg, second_hg);
  float out_scatter_hg = HG( cos_angle, -cloud_outscatter);

  return lerp(in_scatter_hg,out_scatter_hg,cloud_in_vs_outscatter);
}

float Attenuation( float density_to_sun,float cos_angle){

  float prim = exp(-cloud_beer  \times  density_to_sun);
  float scnd = exp(-cloud_beer  \times  cloud_attuention_clampval)  \
    times  0.7;

  //reduce clamping while facing the sun
  float checkval = ReMap(cos_angle,0.0,1.0,scnd,scnd \times 0.5);

  return max(checkval,prim);
}

float OutScatterAmbient(float density,float percent_height){

  float depth = cloud_outscatter_ambient  \times  pow(density,ReMap(
    percent_height,0.3,0.9,0.5,1.0));
  float vertical = pow(saturate(ReMap(percent_height,0.0,0.3,0.8,1.0))
    ,0.8);
  float out_scatter = depth  \times  vertical;

  out_scatter = 1.0 - saturate(out_scatter);

  return out_scatter;
}
```

**Listing B.7:** (Cont) All functions used to calculate the lighting (having previously sampled the density and the density to the sun).

```
float3 CalculateLight(float density, float density_to_sun, float
    cos_angle, float percent_height, float bluenoise, float dist_along_ray
    ){

  float attenuation_prob = Attenuation(density_to_sun,cos_angle);

  float ambient_out_scatter = OutScatterAmbient(density,percent_height);

  //Can be calculated once for each march but gave no/tiny perf
    improvements.
  const float sun_highlight = InOutScatter(cos_angle);

  float attenuation = attenuation_prob \times sun_highlight \times
    ambient_out_scatter;

  //Ambient min (dist_along_ray used so that far away regions (huge steps
    ) arent calculated (wrongly))
  attenuation = max(density \times cloud_ambient_minimum \times (1 -
    pow(saturate(dist_along_ray/4000),2)),attenuation);

  //combat banding a bit more
  attenuation += bluenoise \times 0.003;

  float3 ret_color = attenuation \times sun_color;

  return ret_color;
}
```