# Single Scattering in Scenes containing Semi-Transparent Objects and Participating Media

## Researching, solving and evaluating single scattering and transparency as a single problem

Kevin J.M. van Nes

Computer Graphics and Visualization
Data Science and Technology, EEMCS
Delft University of Technology

# Single Scattering in Scenes containing Semi-Transparent Objects and Participating Media

Researching, solving and evaluating

single scattering and transparency as a single problem

by

Kevin J.M. van Nes

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday April 25, 2018 at 02:00 PM.

| | | |
|---|---|---|
| Student number: | 4020871 | |
| Thesis committee: | Prof. dr. E. Eisemann, | TU Delft, supervisor |
| | Dr. Z. Erkin, | TU Delft |
| | Drs. P. R. van Nieuwenhuizen, | TU Delft |

TU Delft

# Abstract

Volumetric light scattering caused by participating media has been and still is a complicated, but visually pleasing lighting effect that increases the sense of realism of digital scenes tremendously. Various solutions to rendering these effects in real-time exist, basing themselves upon the single scattering approximation. Transparency is another difficult subject within the field of computer graphics, due to the constraint of having to somehow find the order of semi-transparent objects, or to approximate them independently of order. However, no prior research has focused on scenes in which both participating media and semi-transparent objects are present. While a technique such as Depth Peeling (Everitt, 2001, [9]) can be used to solve the problem, this method has some major downsides and often does not obtain real-time results. In this thesis we attempt to obtain faster results, while trying to pertain as much quality as possible. To this end, we take the Depth Peeling method as our reference.

By partitioning scenes into what we call 'depth zones' as seen from the light, we can obtain necessary color and depth information to then approximately render the required result using ray-marching. Partitioning scenes is done in various ways (uniformly, non-uniformly) and special data structures are used to store all necessary information efficiently. The combination of partitioning and data storage led to two new methods being implemented: Multisample Texture Depth Zones (MSTDZ) and histogram-based MSTDZ, of which the last sorts fragments of a scene into bins to determine where to partition depth zones in a less naive way than standard MSTDZ. Both of these methods have the ability to approximate Depth Peeling's results very well, at higher speeds and, depending on the approximation, using lower amounts of memory. A final addition was implemented which groups texels of the histogram texture used in histogram-based MSTDZ as to use smaller textures. In low complexity scenes, this gives the possibility to lower the amount of consumed memory, at the cost of time and a small amount of quality, compared to the reference method.

# Preface

Approximately two years ago, when I was close to starting my work on this thesis, I expected it to be a larger, more complex and challenging version of any other Computer Science project that I'd worked on in the years before. It would become the pinnacle of scientific and technical trials. I expected to learn a lot about the subject that I'd specialize myself in. I had quite some expectations and was prepared to face the challenges, but I learned that not everything can always be smooth sailing like that. The past months of my thesis have led me to learn an immense amount of things, on a scientific level, but also on a personal level, and I am deeply thankful to the people who were there, my family, closest friends, colleagues and supervisors, either as sources of inspiration or as any other kind of support. I want to express my thanks to everyone for being so patient.

Leonardo, thank you for the many hours that you invested in aiding me through all of the phases of my thesis. Your tight involvement helped me incredibly much in getting my thesis to the end result that I am proud of.
I'd like to thank Elmar for supervising me and for supporting me in many ways, despite the process being so lengthy.
Eric, I'm very grateful to you for offering me an engaging place to work, where I was able to develop myself in ways that were not taught during my studies, using incredible technology, all while also being given time to focus on my studies.

Last of all, Sanne, your tremendous amount of strength has inspired me and your support, patience and love have motivated me time and time again to finish this thesis. Thank you so, so much for always being there for me.

*Kevin J.M. van Nes*
*Delft, April 2018*

# Contents

# 1

# Introduction

Volumetric light scattering has been and still is a complicated, but visually pleasing lighting effect that increases the sense of realism of digital scenes tremendously. Correctly rendering these scattering effects is crucial in various areas of the digital industry, such as simulations, computer games, (animated) movies and digital special effects. Scattering effects are caused by light interacting with particles in the air. These particles form a medium, which participates in the propagation of light throughout an environment, giving the particles their more commonly used name: participating media.

However, there is one major obstacle that withholds today's digital industry of extensively using participating media in real-time applications: rendering these effects is computationally very costly. One reason for this is that light rays may scatter many times in multiple directions. Furthermore, participating media are almost ubiquitous. These two facts make it necessary to do extremely high amounts of computations if one wants to render them properly, the likes of which can not be done in real-time using today's hardware.

Take the following example: when talking about animated movies, multiple hours are often invested to compute one frame of the movie. In this case multiple scattering methods, such as Monte Carlo methods or photon mapping, may be used to solve the problem. However, in real-time applications such methods would simply take too much time, seeing as only up to $0.166ms$ can be spent on calculating each frame, as opposed to the hours needed by multiple scattering methods.

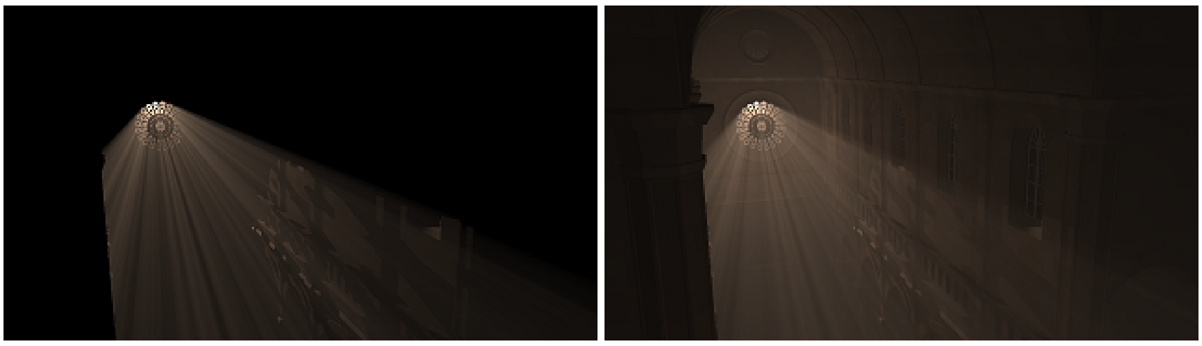


*Figure 1.* Left: A scene with only single scattering. Right: A scene with both single and multiple scattering (Billeter, 2012, [3])

To resolve this issue, it is common use to model light rays as if they only scatter once within a participating medium: the single scattering model. This takes away a large chunk of computational time, but it comes at a cost in terms of quality (see Figure 1). However, in frequent cases, single scattering methods make scattering effects look good enough to give a proper feeling of realism and their computational efficiency which is why single scattering is still used as the base of research into the real-time rendering of scattering effects.

Some very efficient, high quality methods were proposed by (Klehm, Seidel, & Eisemann, 2014, [12]) and (Peters et al. 2016, [18]). However, whereas the results of these methods sketch a bright future for real-time scattering effects, both of these methods have one missing component in common: the possible presence of semi-transparent objects.

Semi-transparent objects affect scenes in such a way that when light propagates through these objects, it is altered, both in terms of color and visibility. Nonetheless, transparency is another complicated phenomenon to render properly. This is due to the necessity of having to know the depth order of semi-transparent fragments, since fragments are not automatically passed to the fragment shader in a sorted fashion. An example of why it is important to know the depth order of semi-transparent fragments is as follows: imagine two semi-transparent objects, a red one and a green one. The red object floats in front of the green one. The color propagated through both spheres will look slightly different than when the green sphere would be in front of the red sphere, as can be seen in Figure 2.

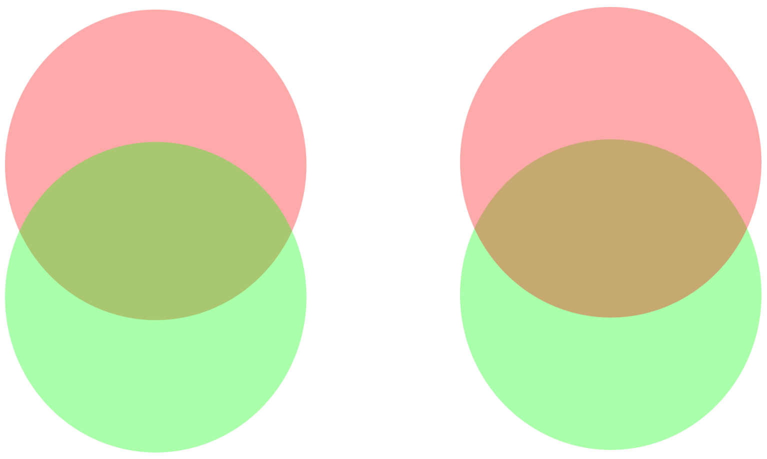


*Figure 2.* Schematic representation of two semi-transparent spheres (both 33% alpha). Left: Green sphere in front of red sphere. Right: Red sphere in front of green sphere.

One obvious way to solve this problem is to sort semi-transparent fragments or objects by depth as a pre-process before rendering a scene, after which colors can be blended. However, color blending techniques come with their own difficulties and inaccuracies. Nowadays, a lot of research focuses on Order Independent Transparency methods, or OIT methods. These OIT methods focus on solving the transparency problem without having to know the actual order of any of the semi-transparent fragments in a scene.

Correctly rendering both single scattering and transparency effects is complex and computationally costly. This becomes even more complex considering the presence of participating media in scenes with semi-transparent objects will also reveal colored crepuscular rays (or 'god rays'). The goal of this thesis is to render these colored crepuscular rays and other colored scattering effects that arise from the presence of semi-transparent objects within participating media. An attempt is made to find similarities between single scattering and transparency methods, as to combine them and overcome their respective computational hurdles, while also obtaining high quality results.

A brute force way of solving the issue described above would be to solve the Radiative Transfer Equation, or RTE (Chandrasekhar, 1960, [5]). However, only multiple scattering methods are able to solve this equation numerically, and, as noted before, multiple scattering solutions are currently not fast enough for real-time rendering. Luckily, we can work around this issue by approximating the RTE using a single scattering model. Using such a model and applying it correctly, we will be able to draw participating media correctly. However, we will still need to know where semi-transparent objects are located in the scene and in which cases light rays are affected by them and when they are not (see Figure 3).

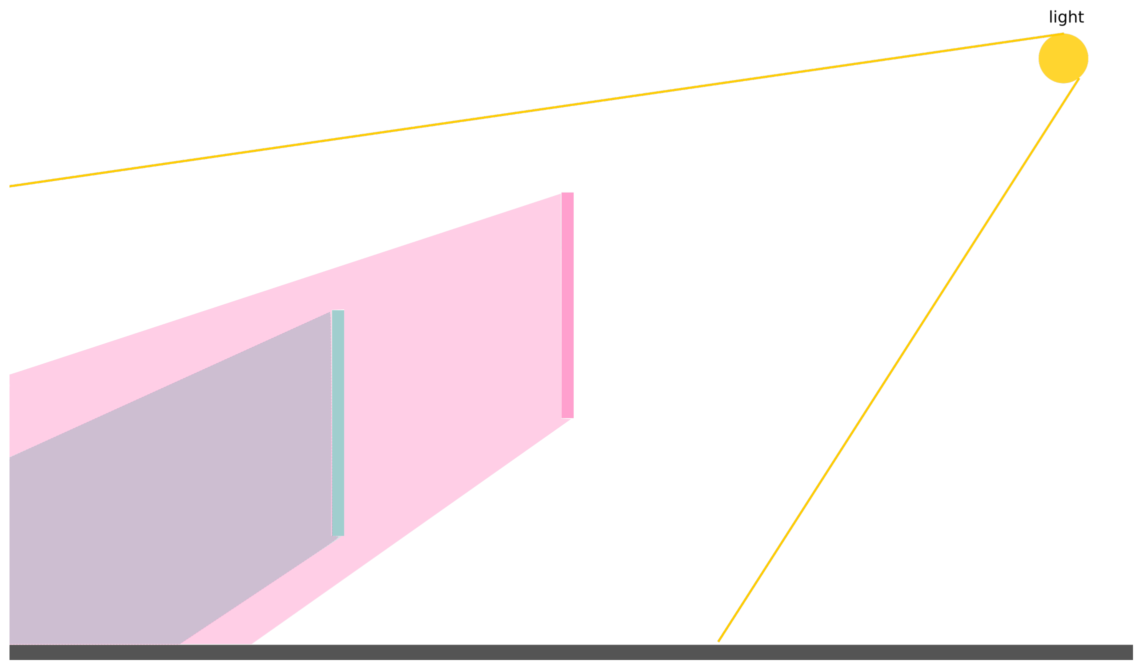


*Figure 3.* In a scene with participating media and two semi-transparent objects, light rays that intersect these objects interact with them, leading to colored participating media. Light rays that do not intersect semi-transparent objects behave like normal light rays would do within a participating medium and retain their original color.

Indeed, both single scattering solutions for lighting effects in participating media, as well as transparency are highly complex. Even today the best performing methods have drawbacks making them not completely generally applicable for all scenes, and many attempts are constantly being made to solve these drawbacks. This thesis does not aim to take the state-of-the-art methods to forcefully combine them. Instead, an attempt is made to find similarities between the two problems to see if a single solution can be found that solves the issues of having semi-transparent objects within participating media in real-time.

To solve the effect caused by the combination of scattering and transparency, a first combination of solutions was developed by combining ray marching, which is often used to render single scattering, and depth peeling (for order-independent transparency). This method is computationally slow, but the results are of reference quality, which made it possible to compare later developed methods both in terms of quality and speed.

Having taken this reference method as a starting point, we were able to come up with a new method, for which the goal is to approximate the results of the reference method. This is done by 'slicing up' a scene into different zones, called depth zones, and saving a single color value per zone (per pixel). Now if multiple semi-transparent objects fall into one zone, only one color will be saved. The new method's performance differs based on whether some of the various proposed optimizations and additions are used. The basic version of this method, called Multisample Texture Depth Zones (MSTDZ) improves upon the reference method in terms of speed, but does not work in all specific cases. An improved version, based on histograms, distributes depth zones based on the density of semi-transparent objects in a scene. This addition enhances the accuracy and thus the quality of the standard method, but adds another render pass, trading off quality for a slight amount of time. A final addition trades off memory and quality, making it possible to mould the performance of the method based on the specific properties of a scene.

The main contributions of this thesis are:

- A method, called MSTDZ, that solves the issue of having semi-transparent objects in a scene containing participating media, using a single algorithm, which is faster than the brute-force method or the reference method used in this paper (Depth Peeling). Uses only four render passes, independent of the scene complexity (as opposed to Depth Peeling) and often retains a high amount of quality.
- An adapted method that solves the same issue, obtaining an even higher amount of quality in specific scenes, but computationally slower than MSTDZ. Also uses a limited amount of render passes (five). Offers promising possible future research directions.
- An explored alternative: the possibility of doing alpha-based discarding of fragments, made possible by the low frequency nature of (colored) single scattering effects within participating media. This addition can be switched on and off and does not cost additional render passes. Sadly, no significant speed-up is gained with this yet, but it is expected that with some changes, results will be very promising. See Appendix B.
- A way to trade speed and quality for memory and vice versa, allowing for different kinds of optimization, depending on the complexity of the geometry of a scene.

# 2
# Related Work

Both single scattering and transparency have a long history of research. Despite the fact that a method for rendering the effects caused by the combination of the two has not been researched yet, it is interesting to take a look at what has been achieved in these two fields so far. A deeper understanding of the two subjects will be gained and insight is given into what has been achieved in terms of the topics' research. Furthermore, one might find inspiration to dig deeper into the state-of-the-art methods to apply them to this research in the future.

## 2.1. Participating Media and Single Scattering

Proper rendering of light scattering effects is a research topic that has been actively researched since the early 1980s, when Blinn (1982, [4] TODO) first attempted to create a light model that would be able to incorporate the presence of participating media in computer graphics scenes. However, whereas recently very good results have been obtained in creating realistic scenes with participating media by using off-line methods (e.g. Monte Carlo methods, photon mapping, virtual light methods), until this day there is still not a method that perfectly solves the same issue in real-time.

As was noted before, to be able to render participating media effects in real-time, certain simplifying assumptions need to be made. An often-made assumption is one that is called the single scattering model, which assumes that light rays bounce only once within a participating medium. Normally a light ray would instead bounce around between particles multiple times before leaving the medium.

Many of these single scattering methods base themselves upon a simplified version of what is called the Radiative Transfer Equation (Chandrasekhar, 1960, [5]) and its computer graphics derived version, the Rendering Equation (Kajiya & von Herzen, 1986, [11]).

While good attempts have been made to render scattering effects using an explicit analytical solution (Sun, 2005, [20]), (Pegoraro, 2009 and 2010, [15], [16]), the more interesting solutions - and also today's state-of-the-art methods - all propose the use of ray-marching. In the meanwhile, these methods strive to reduce the (normally constant) amount of sampling steps during the ray-marching phase, e.g. by using information about the scene or by modifying the shadow map.

In 2008, Wyman & Ramsey (2008, [21]) proposed a method that uses shadow volumes to determine sampling locations, effectively reducing the standard amount of ray-marching sampling steps. However, calculating shadow volumes becomes costly in scenes containing a high amount of shadows. Futhermore, the method becomes very slow in complex scenes, in which it becomes difficult to reduce the amount of sampling steps.

Engelhardt & Dachsbacher (2010, [8]) developed a method which samples at depth discontinuities along epipolar lines in screen space. Their main observation was that crepuscular rays are epipolar lines on screen. They combined this observation with the assumption that pixels on these epipolar lines vary smoothly, except at depth discontinuities. In this case, ray-marching samples only need to be computed at these depth discontinuities. Once again, however, this technique suffers severly in complex scenes, in which many depth discontinuities could be present or no significant changes occur at these locations. This might cause the technique to need a high amount of sample points, nullifying the speed-up that it is intended to achieve.

A more interesting line of real-time scattering methods, which has lead to today's state-of-the-art technique, started with the introduction of the method proposed by Chen et al. (2011, [6]). They adapted the epipolar shadow map rectification technique Baran et al. (2010, [1]) came up with and combined it with a 1D min-max mipmap structure that allows for an accelerated procedure and a reduction in the amount of ray-marching steps. The latter is achieved as follows: each row of the shadow map can be seen as a 1D heightfield, along which the view direction walks to intersect with the heights, allowing for the determination of the visibility. For each of these shadow map rows, a 1D min-max mipmap is computed, which opens up the possibility of computing all of the view rays in parallel making this method a very effective and efficient one. However, shadow maps used during this technique need to be of very high resolution as to avoid aliasing artifacts. On top of that, the epipolar shadow map rectification leads to other small aliasing artifacts.

In 2014, Klehm et al. [12] proposed a method that improved upon the one proposed by Chen et al. (2011, [6]). This technique was called Prefiltered Single Scattering. Using an adapted form of rectifying the shadow map, by representing visibility functions as a linear combination of Fourier series, and using some other optimizations, Prefiltered Single Scattering fully removes the necessity of doing ray-marching, reducing the activity of stepping through a 3D scene to the calculation of a single dot product per pixel. As a result, the method only requires an almost constant amount of computations per pixel.
Still, this method has some drawbacks, one of which is the usage of Fourier series, which, in turn, lead to ringing artifacts. These are partially resolved at the cost of having slightly darker images, but the issue still remains and can be quite obstructive.

Peters et al. (2016, [18]) found a way to replace the convolution shadow maps of Prefiltered Single Scattering by moment shadow maps, which were proposed by Peters & Klein (2015, [17]). The main proposal was to interchange the Fourier basis functions with the first four moments of depth values and store these in a shadow map. Using a weighted combination of these values, visibility functions can be approximated.
Even though the results of this method are very visually pleasing and efficient, some minor artifacts still remain to be solved in future research.

## 2.2. Transparency

Rendering scenes that include transparent objects is an often underestimated endeavor. Transparency poses issues in OpenGL due to the fact that the order of fragments needs to be known to correctly blend colors of the respective transparent objects. The rasterizer does not automatically order fragments by depth, so if a user does not control this order, results may become faulty. In a scene containing only opaque objects, depth testing will make sure to cull those objects that are occluded by others. This way, the color of the front-most objects are written to the respective pixels.
On the contrary, when transparency comes into play, this does not hold anymore, as colors of underlying objects need to somehow be blended with those in front, before being written to a screen.
One classic way to deal with this problem is the use of alpha blending, which works as follows:

1. Firstly, draw opaque objects first, so that the depth buffer can discard potential hidden transparent triangles (e.g. those behind opaque objects).
2. Secondly, sort transparent triangles from farthest to closest, based on distance to the camera or viewpoint.
3. Finally, Draw transparent triangles using the following blending function: $C_d = \alpha_s * C_s + (1 - \alpha_s) * C_d$, where $C_d$ is the to-be-written content of the framebuffer (the destination color), $C_s$ is the output of the shader (the source color) and $\alpha_s$ is the alpha value of the source color.

The use of this specific type of blending gives a result that looks very well in almost all cases, but there is a major drawback to this, which occurs in the second step: the sorting of the transparent triangles. This sorting step has to happen, because of the non-commutative nature of the blending function, i.e. subtraction is non-commutative. Consequently, having a different order of transparent fragments than farthest to nearest would lead to obtaining faulty results.

Sorting comes at a major computational cost. In fact, even if we do sort everything, the results aren't always perfect, for two reasons.
First, a major difficulty lies in the question of what property to sort the transparent triangles on. Should we take a triangle's center? Or the nearest or farthest vertex of a triangle? All of these sorting properties come with their own disadvantages in terms of visual results. Aside of the sorting property problem, concave geometry leads to cases in which certain triangles may cover the same pixel, and in case there is intersecting geometry, some triangles might intersect in separate draws. These types of geometry will make sorting obsolete, no matter the sorting property that is chosen.
The other reason for non-perfect results, is the computational performance. In complex scenes, fragments might be written to multiple times, which in turn leads to memory issues, and, as stated before, sorting is a computationally slow process.

Over the years, researchers have come up with a global way of solving this issue of needing this specific order in the topic that is called Order Independent Transparency, or OIT. In present day, this specific topic of research can be subdivided into three main categories (Wyman, 2016, [22]): depth peeling approaches, stochastic approaches, and visibility approximation approaches. The main contributions for each of these categories will be treated shortly as to give an idea in which ways the reader could use research that was done until now for the purpose of potential future research on this thesis' topic.

Depth peeling solutions are those that 'peel' layers from a scene to find out where transparent fragments are and in which order they should be blended. The original Depth Peeling method (Everitt, 2001, [9]) essentially works as follows: for all non-occluded semi-transparent objects, write the corresponding colors of these objects as seen from the light into a shadow map and also store the respective depths of these fragments. In subsequent render passes, discard all fragments less than or equal to depths of fragments whose colors were already written to a texture. As a result, all fragments in the scene are peeled in the order that they appear from the light. This method deals with all fragments in the scene, leading to results of theoretically perfect quality. In spite of these results, depth peeling is slow in that it takes one render pass per peel, which can lead to extremely high amounts of render passes in complex scenes. On top of this, in each render pass, two textures are required (one for color and one for depth), which in turn leads to high memory usage in complex scenes. This makes the method highly unsuitable for real-time approaches. Although extensions and improvements to the original method have been proposed by Bavoil & Myers (2008, [2]) and (Lin et al., 2009, [13]), depth peeling approaches remain inefficient in terms of speed and memory when compared to other approaches.

Another class of OIT methods is the stochastic approaches, of which Enderton et al. (2010, [7]) were the first to attempt such an approach. This approach will be discussed slightly more in-depth, as an extension that was implemented (see Appendix B) is largely based on this idea. Stochastic Transparency sets, for each fragment, a ratio of samples of semi-transparent objects to their respective color. This ratio is based on the alpha value of said semi-transparent object, leading to correctly alpha-blended colors. These sampling patterns differ for each consequent semi-transparent object, which ensures that semi-transparent objects that lie behind others are still taken into account. An example of how this works is given in Figure 4. In this example we have three semi-transparent objects: a red, blue and pink one. The red object has the lowest depth value and the pink one the highest. We use a multisample texture with 8 subsamples. If the red object (with $\alpha = 0.25$) is sampled, red will be written to a random selection of 25% (i.e. 2) of the subsamples. As soon as the blue object ($\alpha = 0.5$) is sampled, blue will be written to another 4 subsamples. The pink object is next, but it is occluded by the red and blue objects. 2 random subsamples will be written to, but one of them already contains a value of a fragment with lower depth value and only 1 subsample will obtain the pink color. Finally, the colors within all texels are averaged.

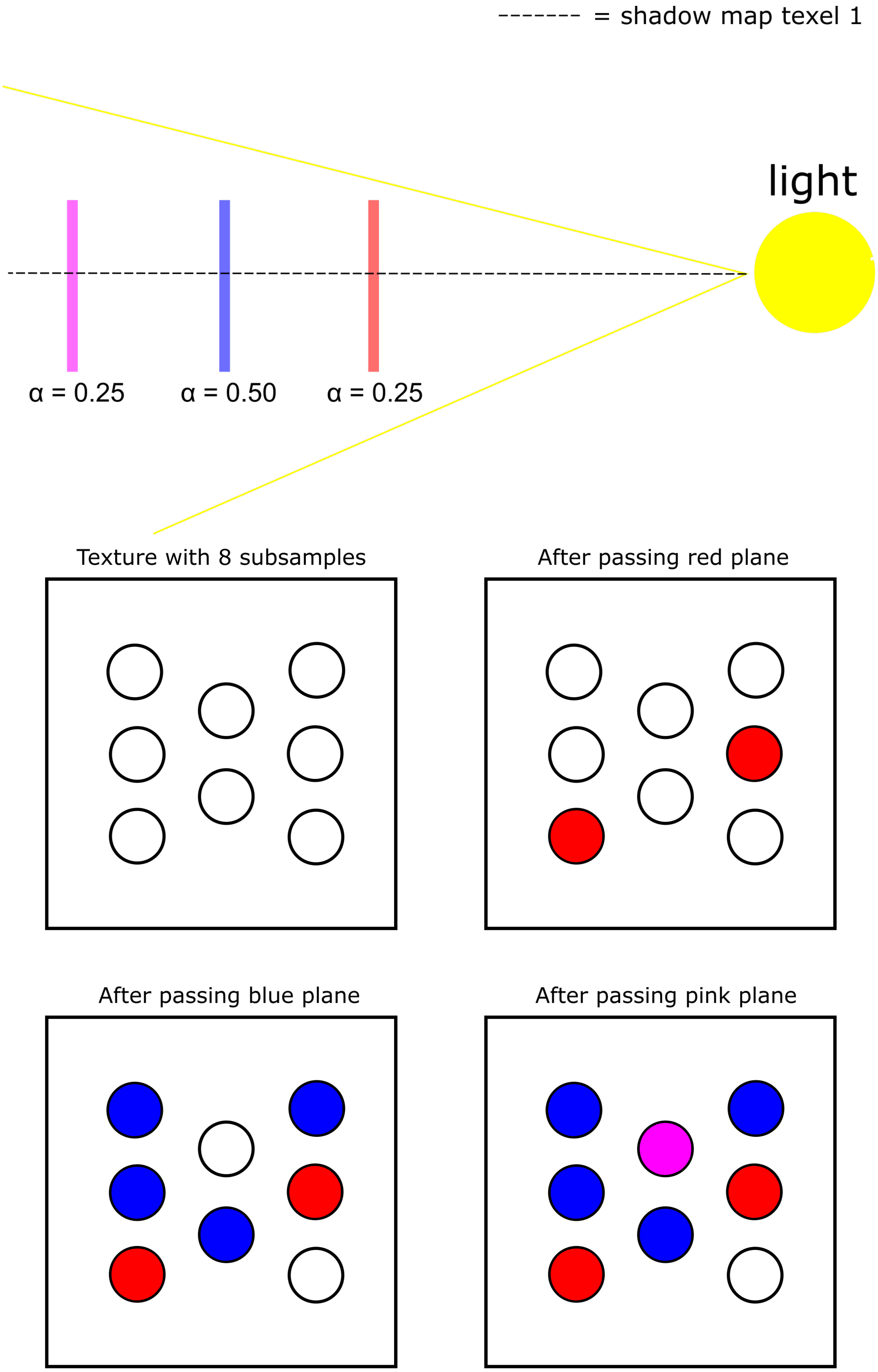


*Figure 4.* Example of how Stochastic Transparency gathers colors from semi-transparent objects.

Of course, semi-transparent geometry is not necessarily traversed in order, but for explanatory purposes this is done in the example of Figure 4. Although this solution works for any kind of transparent geometry and scenes, noise turns out to be a big problem, since each texel will have its own random color averaging. Even with correcting passes, one still needs a very high amount of samples to counteract the noise issue. This direction of research is interesting, but still needs some improvements.

Finally, there is a group of OIT methods that aim to approximate the visibility function for each pixel. We can model the contribution of a semi-transparent fragment $f_i$ as $c_i * \alpha_i * vis(z_i)$. In this formula, $c_i$ is the color of the fragment, $\alpha_i$ its transparency, $z_i$ its depth and $vis(z_i)$ a visibility function that gives the total transmittance between $f_i$ and the camera. The total contribution of overlapping semi-transparent fragments is given by $\sum_{i=0}^{n} c_i * \alpha_i * vis(z_i)$. The visibility function $vis(z)$ is defined as the product of the contributions of every semi-transparent fragment that intersects a ray from the viewer to $z$. This can be represented as follows: $vis(z) = \prod_{0<z_i<z}(1-\alpha_i)$. This is always a piecewise decreasing function, and it contains all of the depth and visibility information about a fragment. It is, however, very inefficient to calculate all visibility functions for each fragment in a scene. Certain methods therefore attempt to approximate them for efficiency.

Some visibility approximation proposals are those that attempt to present a blending function that is fully commutative, in contrast to the original non-commutative one. Having a commutative blending function of which the results approach reference quality would imply that semi-transparent fragments could simply be blended without having to worry about their order. OpenGL functions using additive or multiplicative blending do not lead to correct results (although in extremely simple scenes they might be sufficient). Multiple proposals were made over the past decade, but the most notable one is the one by McGuire & Bavoil (2013, [14]). This method extends the blending function by Bavoil & Myers (2008, [2]) by computing the exact coverage of the background by semi-transparent fragments. In addition, they add weights to the function which depend on the depths of each semi-transparent fragment. While results can come close to reference quality, the weights differ for each scene, and obtaining good results is a matter of trial-and-error, which can turn out to be a lengthy task in many cases.

Another noteworthy method within this category was proposed by Salvi et al. (2011, [19]). Salvi et al. propose to represent visibility functions as a product of Heaviside basis functions, which turns visibility functions into pairs of depth and transmittance $(z_i, 1-\alpha_i)$. Subsequently, they compress these pairs by using an optimization function that eliminates certain pairs based on the effect of the removal of each pair. The algorithm performs faster than many earlier methods while maintaining high quality. On the other hand, memory requirements are high and improvements can still be gained.

Despite the existence of the state-of-the-art scattering and transparency techniques that were described above, there has been no specific research towards rendering the effects that are found in scenes containing both participating media and translucent objects. Seeing as this area of research is a completely new endeavor, we hope to spark others' interest to dive into this topic, as we presume that various improvements could be made to the final product proposed in this thesis.

# 3
# Method

As was mentioned before, the main focus of this thesis is the rendering of effects that occur in scenes in which both scattering and transparency effects present themselves. This effect can be described as *colored participating media*. The final method should be real-time, or at least very close to it. At the same time an attempt will be made to stay close to reference quality.

It is important to note that all methods essentially boil down to two main phases. The first phase is gathering color and depth values of semi-transparent objects. The second phase is the ray-marching phase, in which this information is used to render colored single scattering effects as efficiently as possible, while trying to maintain a high amount of quality. It was chosen to lay focus on improving this first phase, as gathering this information in more efficient ways will allow for a good, solid basis. This opens up the possibility of improving the ray-marching phase in future research, for which a number of ideas are given in Chapter 7.

## 3.1. Setting up a Reference Method

When work on this thesis had just begun, the following question was asked: what would be the most straightforward way of solving the scattering problem? Or in other words: how could we solve this problem in a brute-force way?

To answer this question, we must take into account something that many rendering algorithms omit on purpose: the environment which is to be rendered is filled with many microscopic particles, with which the light in a scene interacts. In other words, these particles, forming a medium, participate in the light's propagation from surface to surface, giving rise to their name: participating media. There are four types of scattering events that occur within participating media: absorption, out-scattering, in-scattering and emission.

It would be intractable to calculate the propagated amount of light at each particle, especially considering the scattering events that can take place at each particle. In this thesis, it is assumed that all scenes are entirely filled with homogeneous participating media. The goal is to compute the radiance, which is the amount of incoming light, at each point in this volume.

The change in radiance at a certain position can be calculated by using the integro-differential form of the Radiative Transfer Equation (RTE) (Chandrasekhar, 1960, [5]), which was originally used in heat transfer calculations (see Equation 3.1, notation by Jarosz (2008, [10])). By integrating both sides of the RTE, a form is obtained in which the radiance at a certain point, incoming from a specific direction, is described as a sum of three terms: the reduced surface radiance, the accumulated emitted radiance and the accumulated in-scattered radiance.

The Radiative Transfer Equation is expressed as

$$(\omega \cdot \nabla)L(x \to \omega) = -\sigma_a(x)L(x \to \omega) - \sigma_s(x)L(x \to \omega) + \sigma_a(x)L_e(x \to \omega) + \sigma_s(x)L_i(x \to \omega) \tag{3.1}$$

In this Equation, $(\omega \cdot \nabla)L(x \to \omega)$ is the total change in radiance along a light ray at a position $x$ in the direction $\omega$. The first term on the right represents the absorption, where $\sigma_a$ is the absorption coefficient. The second term is the out-scattering term, where $\sigma_s$. The third term represents the emission factor and the final term represents the in-scattering factor. As one can see, all possible interaction events within a participating medium are represented in the RTE. Emission is simple to solve for and can thus be omitted from the equation. The integrated equation then boils down to two terms, resulting in what is often called the Volume Rendering Equation (Jarosz, 2008, [10]), which is expressed as

$$L(x \leftarrow \omega) = T_r(x \leftrightarrow x_s)L(x_s \to -\omega) + \int_0^s T_r(x \leftrightarrow x_t)\sigma_s(x_t)L_i(x_t \to -\omega)dt \tag{3.2}$$

Here, $L(x \leftarrow \omega)$ denotes the total radiance that reaches a position $x$ from direction $\omega$. $s$ represents the depth of the participating medium from $x$ in direction $\omega$, $x_t = x + t\omega$ with $t \in (0, s)$, i.e. a point within the participating medium $t$ steps into the direction of $x$. Lastly, $x_s = x + s\omega$ is a point on a surface behind the participating medium (i.e. the background). The transmittance $T_r$ gives the reduction of radiance as the ray travels through the medium (Jarosz, 2008, [10]).

Solving the Volume Rendering Equation is quintessential to correctly render scenes with participating media. However, it is highly complex and extremely costly to solve. The equation can be numerically approximated by multiple scattering methods. However, doing so is intractable for real-time applications. To decrease the amount of computations drastically, the assumption is made that light scatters only once, which is called 'single scattering'. This makes it much easier to compute the effects in real-time.

As with many single scattering methods, ray-marching can be used for the computation of single scattering. Ray-marching is a technique that, for each pixel, 'shoots' a ray into the scene, after which samples are taken at equidistant points in the scene. These samples are then summed up. A sample at some point X will denote how much of a light ray that was 'shot' into this pixel propagates up until point X.

To determine the color of a sample point X, a shadow map is used. This works well in scenes in which semi-transparent objects are absent, where shadow maps would simply consist of a single layer. In this case, a point X can either be in light or in shadow and the point's color is easily determined. However, when there are multiple subsequent semi-transparent objects to be found in a scene, each object having its own color and depth values, a single shadow map will not suffice. Either the shadow map should consist of more than a single layer, or multiple shadow maps need to be used to contain all of these values.

A method that collects and stores the colors of semi-transparent objects as seen from the light is depth peeling (Everitt, 2001, [9], see Figure 5). However, if during ray-marching we want to determine behind how many layers a point X (situated within a participating medium) is positioned, we need to also capture the depth values corresponding to the color values. Using depth peeling, we are able to collect both the necessary color and depth values in the following pre-process: we create, for each layer that needs to be peeled, two shadow maps: one that contains the color values and one that contains the corresponding depth values. During ray-marching, by projecting the respective sample point to the shadow map's view, we will know whether a sampled point is behind one or more semi-transparent objects. The shadow maps can thus be used to check at each sample point $i$ whether this point's depth is behind any transparent or opaque object. If point $i$ is behind an opaque object, no light will reach point $i$, so this point will not have to be rendered. More importantly though, if point $i$ is behind one or multiple semi-transparent volumes, the shadow maps that were created during depth peeling are iterated over from nearest to furthest, as to blend the color of the light up until point $i$.

The combination of depth peeling and ray-marching with enough precision results in obtaining images of perfect quality. The results of this depth peeling method are therefore used as a reference to compare the results of the methods proposed in this thesis.

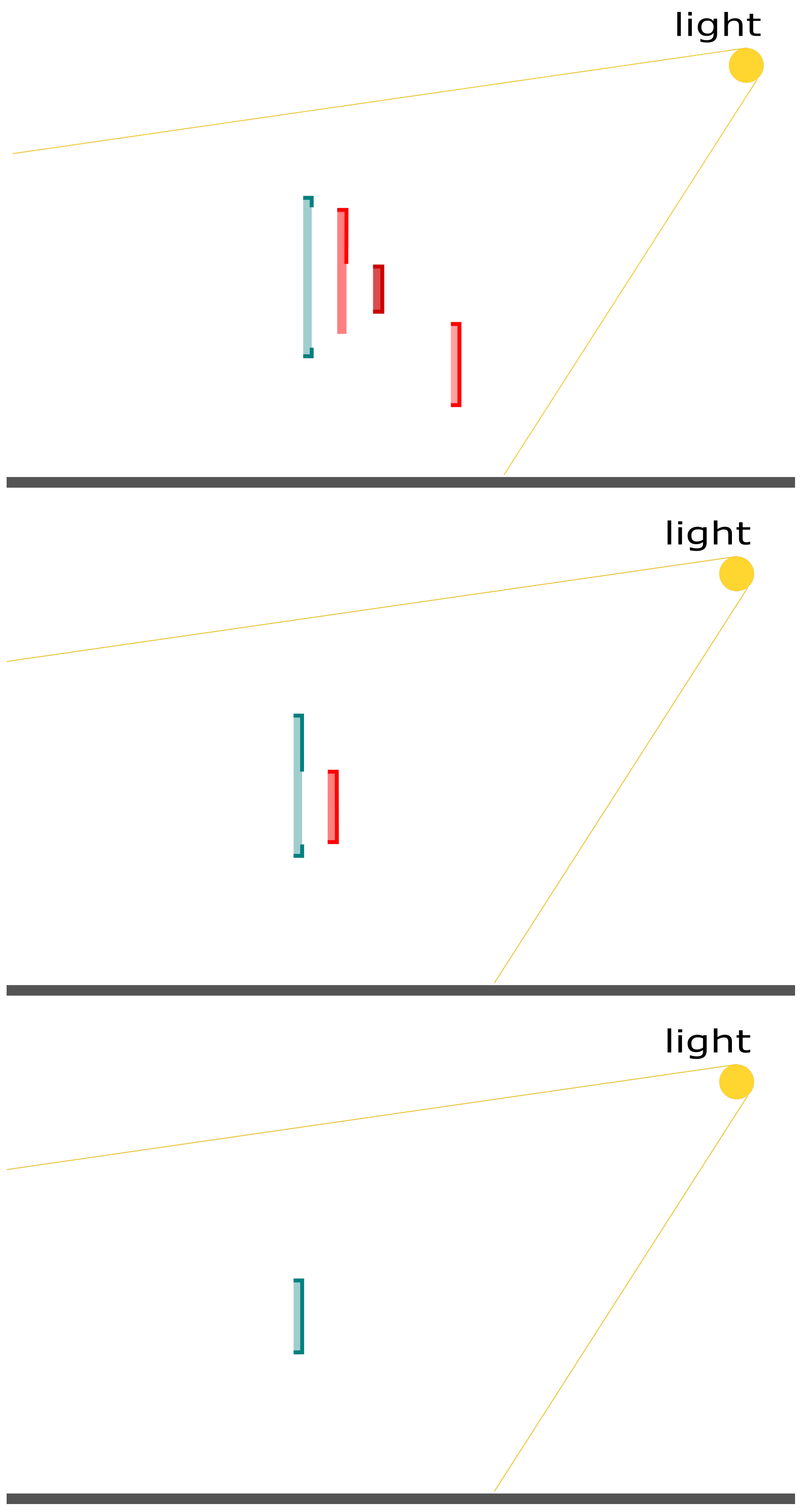


*Figure 5.* Schematic side-view of how depth peeling goes to work. From top to bottom: First, second and third peel done by depth peeling in a scene with 4 colored, translucent planes. Thick lines are the fragments that are stored in color textures during normal depth peeling. The shadow maps resulting from this scene are found in Figure 6.

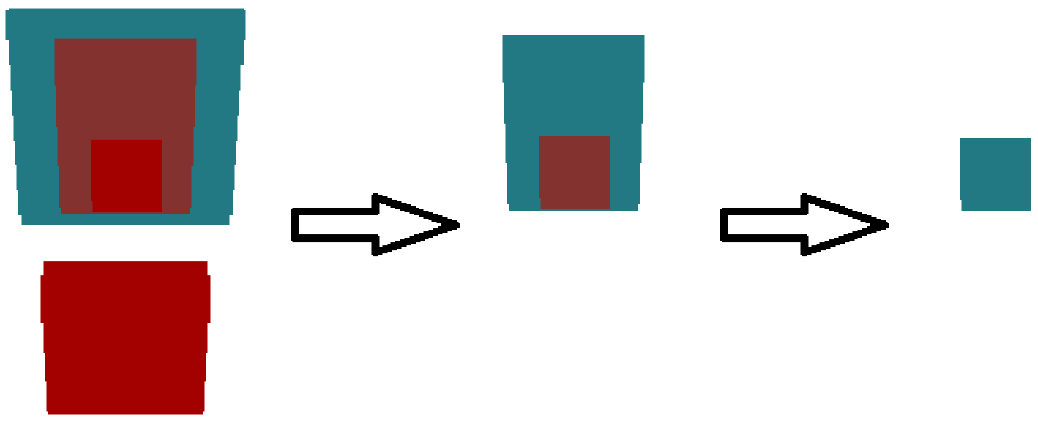


*Figure 6.* The contents of the first three color textures after depth peeling the scene in Figure 5.

Interestingly, one could view the results of this method as if a scene is partitioned as seen from the light, where each view ray contains a perfect partitioning of the scene's semi-transparent objects. This could be translated to a shadow map or shadow maps that contain colors and depths of semi-transparent objects based on this partitioning. In the case of depth peeling, this partitioning is perfect and covers all semi-transparent fragments in a scene.

The general idea behind this partitioning became the inspiration for all methods that are proposed in this thesis. In general terms it is as follows:

**Color/Depth gathering phase**

1. Partition a scene (as seen from the light source) into several 'depth zones' based on the depth values. (Depth Peeling does this automatically resulting in what could be seen as the perfect partitioning.)

2. Create a shadow map data structure that is able to contain color and depth values for semi-transparent objects in each partition of the scene.

3. Now, for each fragment (still as seen from the light source), if this fragment falls inside a certain partition, write its color and depth values to the corresponding color and depth shadow map.

**Ray-marching phase**

1. From the camera view, step through every screen pixel until a certain threshold (either an opaque background or the far plane of the frustum is reached).

2. For each fragment, project its position to the shadow map and determine behind which semi-transparent objects it is located. Do this in ascending order, starting from the data structure corresponding to the lowest depth value all the way up to the one that corresponds to the highest depth value.

3. Sum the colors of the fragments that were stepped over and blend with the background color to obtain the correct color of participating media that result from semi-transparent objects.

The 'partitioning' that results from the peels of depth peeling is an interesting property, but it comes with an inherent issue: for every single peel leading to this perfect partitioning an entire render pass needs to be done. The obvious side-effect of this is the high computation times that are involved in this process. This becomes especially evident in complex scenes, where tens or maybe even hundreds of peels might have to be computed and stored, making this technique not only inefficient in terms of time, but also in terms of memory. It would be interesting if we could approximate the results of depth peeling, while at the same time taking off a large chunk of time from this information gathering phase.

As was previously stated, there is no simple way to obtain color and depth information about semi-transparent objects, except when using inefficient methods such as depth peeling. The described issues with depth peeling ask for a different approach. In this thesis, the focus was laid on finding ways of obtaining color and depth information about semi-transparent objects in an efficient manner. One way to do so, is to bound the amount of render passes to be constant, preferably even a single one. However, to do so, some approximations would have to be made. By partitioning a scene into uniform zones, based on depth, as opposed to depth peeling's perfect, non-uniform partitioning, we can pass over the entire scene and save one color per 'depth zone'. This partitioning happens as seen from the scene's light source. Such an approximation would be much faster.
Conversely, a big issue with partitioning is that, especially in scenes containing many semi-transparent objects close to each other, not all information will be obtained. To perfectly reconstruct all of the information in scenes, we would need, at worst, an infinite amount of partitions, as depth peeling could theoretically need. However, using a finite number of partitions could be enough to pertain a high percentage of the reference quality and the corresponding speed-up could be well worth it.

The idea of different ways of partitioning was deeply explored and the newly found methods that were developed based on this exploration, will be discussed in the upcoming sections. First of all, uniform partitioning methods are discussed in Section 3.2. Uniform partitioning is expected to be very naive. For example, certain zones or even entire texels could be empty (i.e. not containing any semi-transparent objects). A look will be taken at how to restrict depth zones to be more accurate and save computational time, concurrently, in Section 3.3. Non-uniform partitioning methods have the potentital of taking away the naivety from which uniform partitioning methods suffer. Such methods are discussed in Section 3.4. Finally, a way of conserving memory is proposed in Section 3.5.

## 3.2. Uniform Depth Partitioning

A first attempt to gather all information about semi-transparent objects in a single pass was done by making use of uniform partitioning.
For example, a scene could be partitioned into a number of zones (once again, as seen from the light source). These zones span a certain segment of depth values. For each 'depth zone' a texture is then created, to which, for each pixel, a color belonging to that depth zone can be saved into its respective texture. For the general case, if the scene is partitioned into $n$ equally-sized zones, zone $i$ will have a lower depth boundary at $\frac{i-1}{n}$ and an upper depth boundary at $\frac{i}{n}$. Each depth zone spans a segment of depth of size $\frac{1}{n}$, which will from here on be called the size of a depth zone.

An example of uniform partitioning would be to have four equally-sized zones, one for depths between 0 and 0.25, a second zone for depths between 0.25 and 0.5, etc. If a semi-transparent pixel is now found at depth 0.48, its corresponding color is stored inside the texture belonging to the second zone. A schematic example is given in Figure 7, in which a scene is partitioned into four depth zones. The color information is written to the corresponding two (out of four) textures (see Figure 8). During ray-marching, this information will be used to color fragments correspondingly. If, for example, a fragment now falls between the pink and blue objects, its light color will be pink. Consequently, if a fragment falls behind both of these objects, the light color will become a blended pink and blue, depending on the alpha values of both objects.

Two different methods of uniform partitioning were devised: one using multiple render targets (MRT) and one making use of the structure of multisample textures. Both of these methods will be discussed in-depth in Chapter 4.

As was mentioned, uniform partitioning of zones could result in visual artifacts if multiple semi-transparent objects fall within the same depth zone. However, this does not necessarily have to be an issue. What could be improved however, is what was also stated before: certain empty depth zones or even entire empty texels could be skipped during ray-marching to speed up the process.

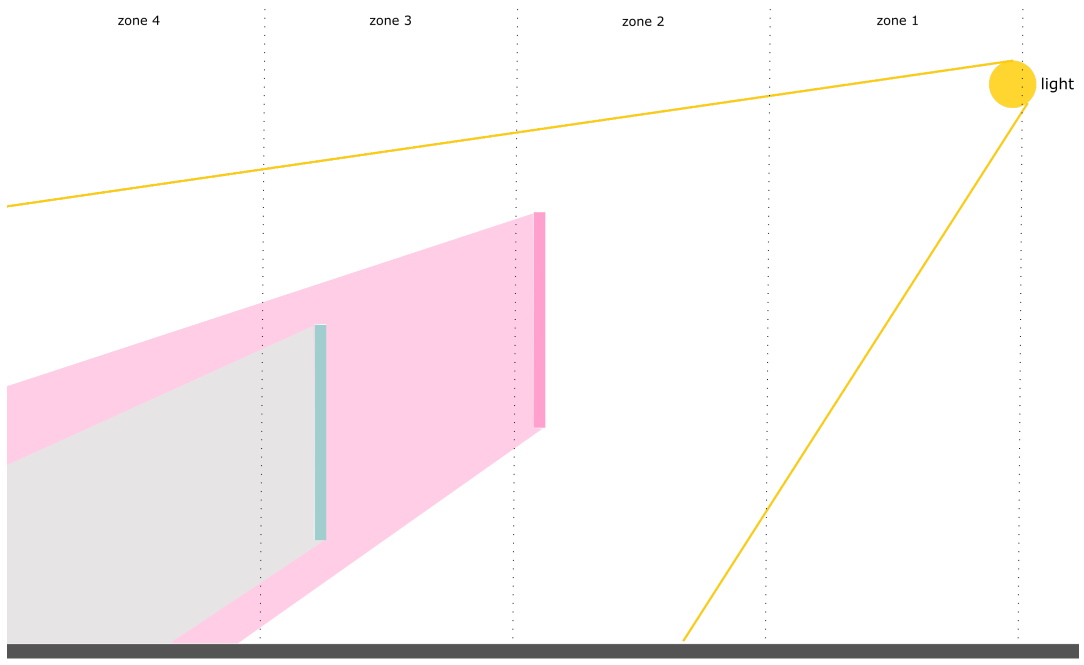


*Figure 7.* Schematic side-view of a scene being partitioned into 4 zones.

*Figure 8.* The contents of the second (left) and third (right) shadow map textures after storing the colors of the semi-transparent geometry found in Figure 7.

## 3.3. Restricting the Depth Zones

An issue with uniform partitioning methods is the fact that they do not take into account whether certain zones will actually contain semi-transparent objects. Furthermore, the distribution of zones will be equal for all shadow map textures that are used. Therefore, it could easily occur that entire shadow map texels along the same view ray will be empty. The issue with this is that, during ray-marching, $n$ texels have to be passed through, with $n$ being the amount of depth zones the scene was partitioned into. However, it would be a waste of time to iterate over these empty texels. It would be computationally more efficient if we could skip passing over these texels during the ray-marching phase. On top of this, we could save more computational time and even memory by restricting the values between which depth zones are partitioned in shadow map texels that do contain semi-transparent objects.

Firstly, we solved the latter issue. By attaining information about the depth of the semi-transparent fragments that are nearest to the light source (the '*first transparent depth*' value, or *ftd* value), and the depth of the first opaque object (the '*first opaque depth*' value, or *fod* value) as seen from the light source, we can put a lower and upper bound, respectively, on the depth values between which depth zones should be distributed. This can be done for each texel and this will result in tighter, and therefore more accurate, depth zone distributions. These *ftd* and *fod* values are saved in their own respective *ftd* texture and *fod* texture. Depth zones can now only be partitioned between depth values *ftd* and *fod*, instead of between 0 and 1. The size of each depth zone will now be $\frac{ftd-fod}{n}$, where $n$ is the amount of depth zones that are used.

As a result of this restriction, each texel of every shadow map will cover its own segment of depth values, as opposed to each texel having identical ones, and each distribution will more properly fit the locations of the semi-transparent objects found in these texels. Depth zone distributions will still be uniform, but no longer identical between texels. Furthermore, by having tighter bounds, it could become possible to lower the amount of depth zones that need to be used. This saves memory, as fewer shadow map textures will have to be used, and at the same time, fewer computations will have to be done during the ray-marching phase, as fewer texels will have to be iterated over. Figures 9 and 10 showcase the effect of using the *ftd* and *fod* values.

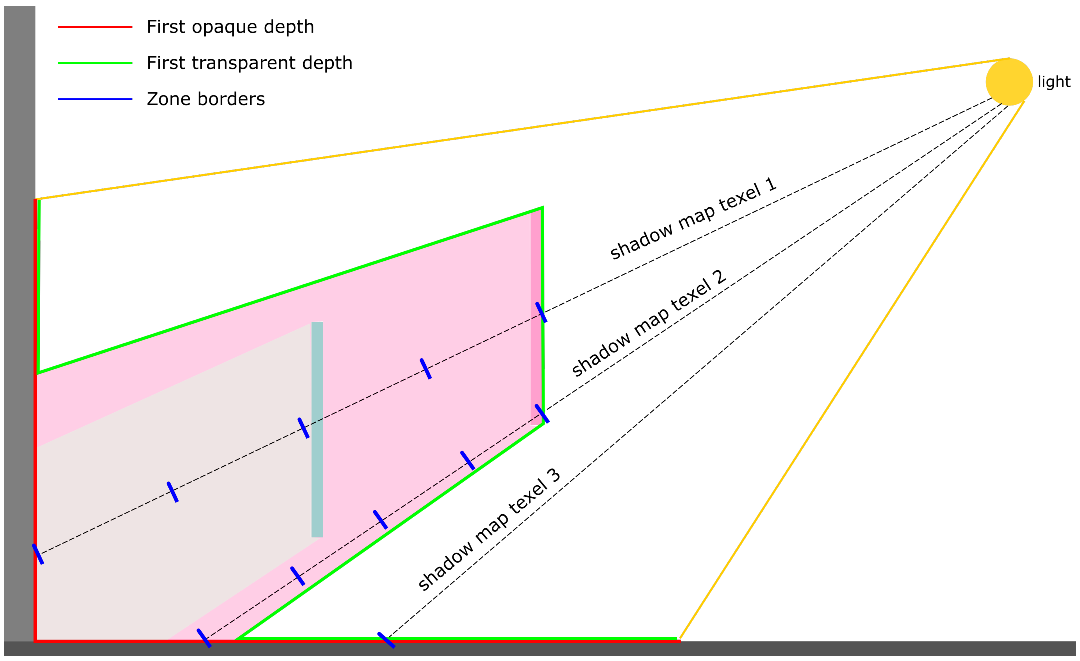


*Figure 9.* Schematic overview of using the first opaque depth and first tranparent depth values. From this picture we can see that each texel will obtain its own zone distribution and zones are only partitioned within the boundaries of these two depth values. Shadow map texel 3 sees no semi-transparent objects, so no zones are created.

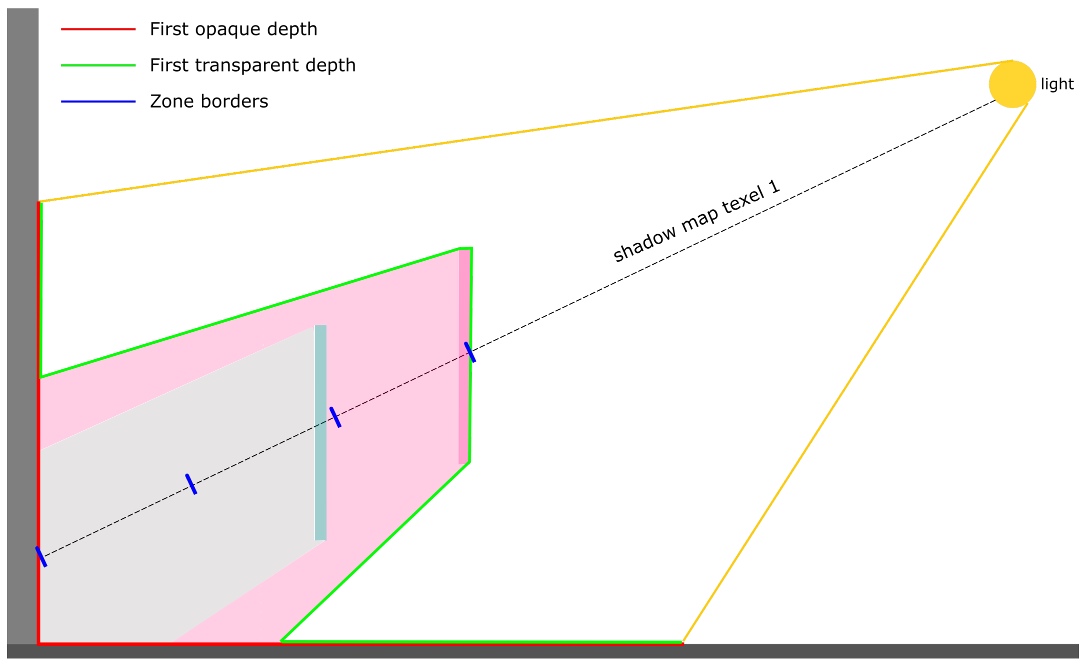


*Figure 10.* Showcase of first transparent depth and first opaque depth optimizations leading to a more efficient use of space. This allows for the use of a lower amount of zones to obtain good results, whereas more zones were required before these restrictions were applied.

During a later stage of research it was found that it would be possible to use ’last transparent depth’ values (or *ltd* values) and a corresponding texture, instead of the opaque depth values and texture, to make the bounds even tighter. See Figure 11 for an example of how this works. During the last phase of the thesis, it was found that the to-be-partitioned area could be restricted even more by taking the camera’s frustum as boundaries. However, there was no more time to implement this and to re-evaluate using this optimization.

After restricting the depth zones, we solved the issue of having to iterate over empty texels using the aforementioned ’first transparent depth’ value. As was stated before, certain shadow map texels along the same view ray could remain empty if no semi-transparent objects are found along this ray. This means that no colors are written to these texels, and consequently we would like to skip these texels during ray-marching. This is because each time such a texel is fetched during ray-marching, $n$ unnecessary computations will be done, with $n$ being the amount of zones that are used, even though all of the texels are empty. It is highly important to omit these computations, as a large amount of time can be saved by doing so, especially for scenes containing large areas with no semi-transparent geometry.

The following solution was found: by clearing the aforementioned ’first transparent depth’ texture to a value of 1.0, all depth values are initially set to 1.0. During ray-marching, before iterating over the color shadow maps to see whether a fragment lies behind semi-transparent geometry, we will first look at whether a fragment’s corresponding *ftd* texel is equal to 1.0. If so, this signifies that no new depth value was written to the corresponding shadow map texels along that view ray during the color gathering phase (see Figure 9, shadow map texel 3). Consequently, no semi-transparent geometry is to be found in any of those texels. Therefore, we no longer have to check whether the current fragment is behind any semi-transparent geometry, as we know there is none at all between this fragment and the light source. We can now start looking at the next fragment, which saves potentially large amount of computational time, especially if semi-transparent geometry is sparse in a scene.

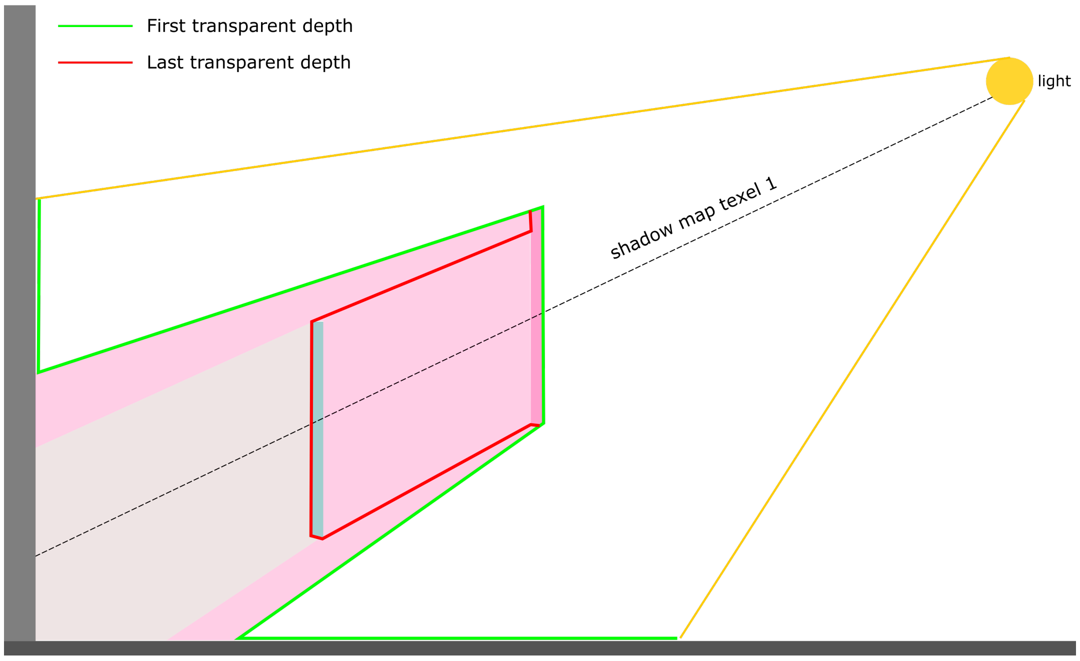


*Figure 11.* Schematic example of bounds when using a combination of first and last transparent depths. When compared to Figure 10, the bounds are even tighter, and even fewer zones (in this case two) need to be distributed and used to obtain the same result.

## 3.4. Non-uniform Depth Partitioning

Up until now depth zones were always divided uniformly. Researching different ways of partitioning depth zones is important, as results of the proposed methods will highly depend on this distribution in two different ways. First of all, if fewer depth zones need to be used, while at the same time being able to maintain equivalent quality, computational speed will rise and memory can be saved. In addition, partitioning depth zones at more meaningful locations (e.g. complex semi-transparent geometry close to each other) will lead to higher quality, since less information will be lost and colors will be more accurately conserved.

The expectation is that there is no global solution for the division of zones. Instead, the number and locations of zones are highly dependent on the scene. However, certain distributions can give results that could be subjectively seen as 'better' than others. One example of this is the following situation: imagine you are viewing a scene from a viewpoint slightly to the right of the light source. This scene contains a number of translucent objects at various low and high depth values. The scene is segmented into five zones, with more zones located a lower depth values (closer to the light source). If, such as in this case, many depth zones are created nearby the light source, while only few are created far away from the light source the colors of the semi-transparent objects that are nearby will now be more accurately preserved (see Figure 12). On the other hand, colors of far-away objects and participating media will be poorly preserved. If we were only looking at the semi-transparent objects themselves, this would not necessarily be an issue. However, we are focusing on the colored participating media that are found in this scene. The poor conservation of colors and the resulting artifacts are much more clear when looking at the colored participating media.

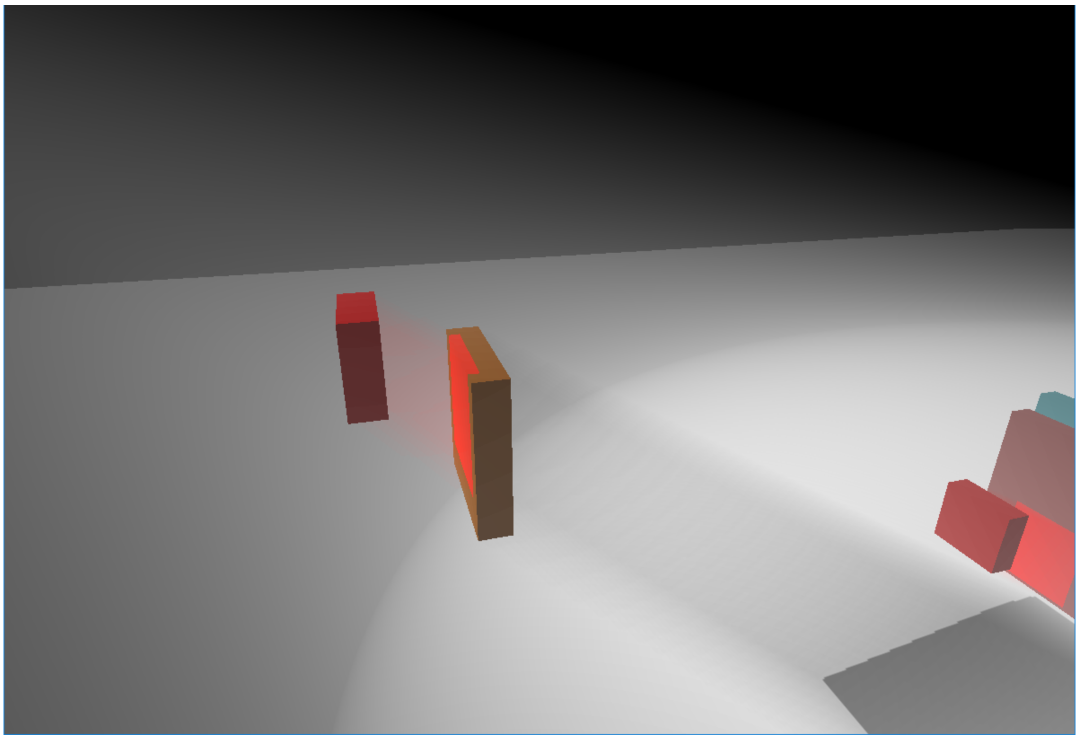


*Figure 12.* From this viewpoint, visual artifacts that occur using the 'many nearby, few far away' zone distribution are not visible.

A major disadvantage is the following: such manually set non-uniform depth zone distributions would lose their value as soon as the camera viewpoint is changed. If, for example, the same example scene with the same partitioning of depth zones would be viewed from the side, color information about the semi-transparent objects that are close to the light source is still preserved well. However, the error in colors far away from the light source, that were not as visible from the viewpoint in Figure 12, will now be much more pronounced and will leave noticeable visual artifacts, as can be seen in Figure 13.

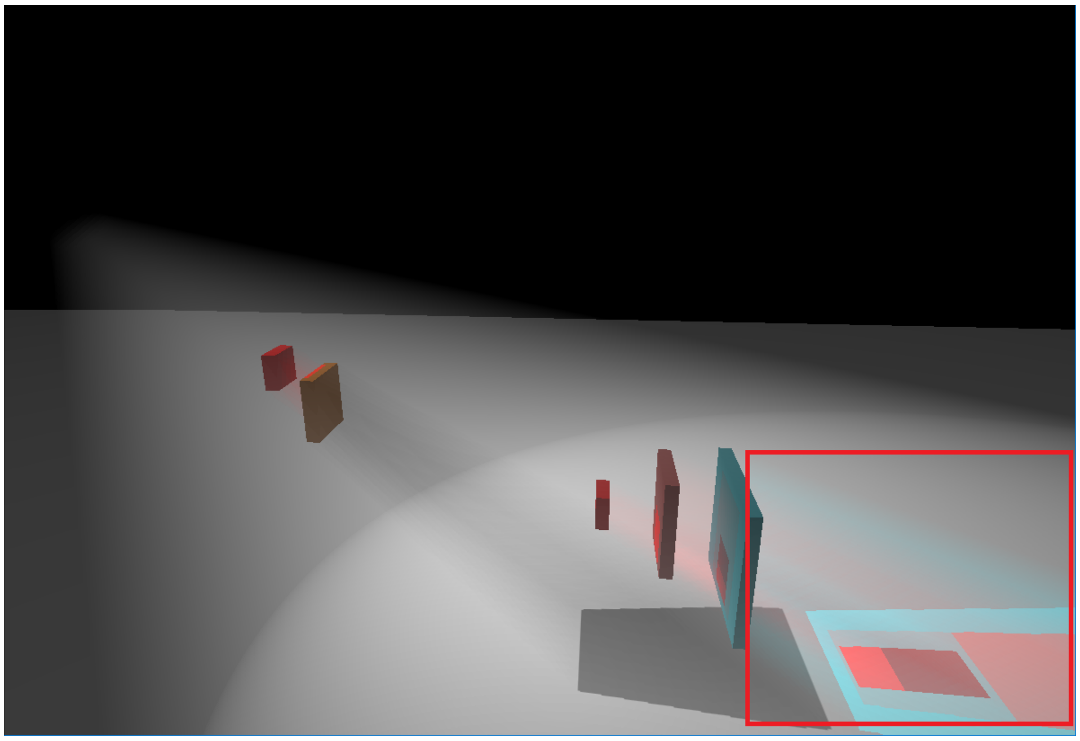


*Figure 13.* From another viewpoint, the same scene with the same 'many nearby, few far away' zone distribution shows visual artifacts: red light is fully propagating through the light-blue surface, which should not happen. Colors are slightly intensified for clarity.

Manually determined depth zone distributions like the one described above can be useful in specific cases, but obtaining the optimal depth values is a matter of guessing and trial-and error. A much more optimal approach lies in what was already mentioned before: to be able to know where to place the depth zones based on the locations of semi-transparent geometry. This idea is explored in Section 3.4.1.

Another important question is how many zones are needed to approximate reference quality to a certain degree, without having to resort to a brute-force solution such as depth peeling. In other words, how few depth zones can we use so that the quality of the approximation looks well enough, without having to employ a trial-and-error approach. This amount of zones will depend on the structure of the scene, but dynamically finding the perfect amount of depth zones to dynamically approximate reference quality in any scene requires extensive in-depth research. This problem is not solved in this thesis and will be seen as future work (see Chapter 7).

### 3.4.1. Histogram-based Partitioning

A big issue that was investigated and partially solved in the previous two Sections, was how to restrict uniformly partitioned depth zones and how to distribute depth zones and where to place them. While the first transparent depth and first opaque depth values will expectedly lead to improvements in terms of quality and efficiency, there are many situations in which having even a high number of zones would not result in perfect quality. An example of a troublesome scene is shown in Figure 14. In this Figure, an arbitrarily high amount of zones (32) is used to try to capture two semi-transparent objects within different zones, as to conserve the colors of both. However, the two objects fall into the same zone (zone 16), even though such a fine partitioning was used. Because of the close proximity of these objects, the first transparent and first opaque depth restriction will not have a large effect on zone distribution for shadow map texel 1. Therefore, the yellow and pink planes in zone 16 will lead to inaccurate color information, which will in turn lead to erroneous results during the ray-marching phase.

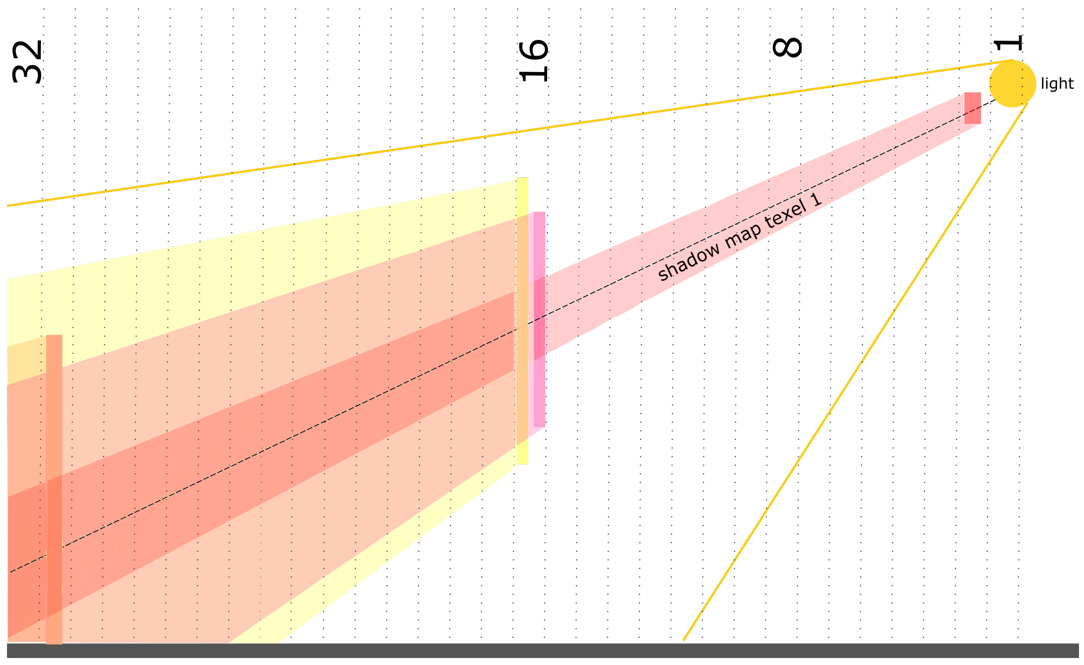


*Figure 14.* Example of a scene in which uniform partitioning methods struggle to capture all color information correctly.

A more ideal way to partition the depth zones would be to have a method that dynamically determines where to distribute zones based on the amount of semi-transparent geometry at each location in a scene.

One possible way of dynamically knowing where to place zones in a scene would be to count the amount of semi-transparent fragments at each depth in the scene. This has to be done for each shadow map texel, as the distribution of semi-transparent geometry across various shadow map view rays can vary between texels. By allocating a higher number of depth zones to areas with a high amount of semi-transparent fragments, and vice versa, it will become possible to distribute zones even more accurately than before. The idea behind this is that the presence of more fragments implies a high amount of geometry and geometrical complexity, requiring more depth zones to more accurately capture color information, whilst the opposite goes for areas with low amounts of fragments. This addition will, expectedly, lead to an increase in quality compared to prior methods.

To this end, before distributing zones over a scene, we divide the scene into a number of bins, called 'histogram zones'. Along each shadow map texel's view ray, we count fragments and add them to these bins based on their depths. By creating such a histogram for each pixel, we can get a more detailed representation of the whereabouts of semi-transparent geometry across the entire scene. By utilizing both the 'first opaque depth' ($fod$) and the 'first transparent depth' ($ftd$) values, we can determine the sizes of the histogram zones. The size of each histogram zone is equal to $\frac{fod-ftd}{|histogramZones|}$. For clarity purposes, during the explanation of this method, the first opaque depth and first transparent depth optimizations are left out.

In Figure 15, we re-use the scene from Figure 14, but this time we firstly divide the scene into four segments, corresponding to four histogram zones. Assume that for shadow map texel 1, the yellow and pink objects contain a high amount of fragments, while the other objects do not. This means that a high amount of depth zones will be assigned to histogram bin 2.

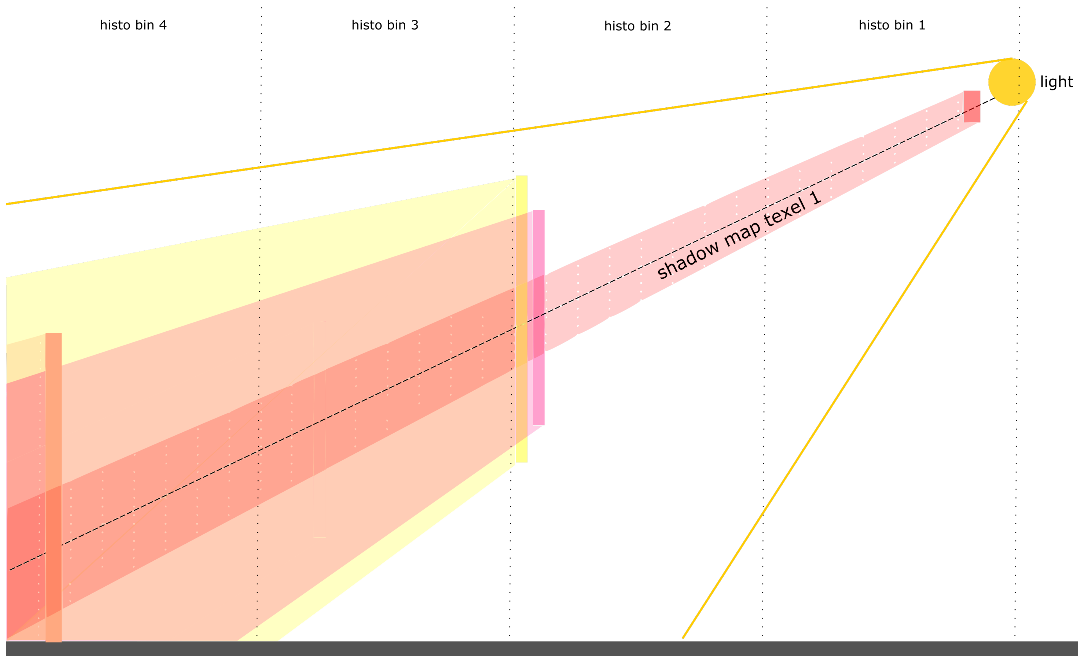


*Figure 15.* The problematic scene is divided into four histogram zones. Histogram bin 2 contains a relatively high amount of fragments compared to the other bins, implying that the geometrical complexity in this area is higher.

After having gathered each texel's amount of fragments, it is time to assign depth zones to each of the histogram zones. There are three possibilities that determine the amount of depth zones to allocate to each histogram zone:

1. If there are zero fragments inside a histogram bin, then allot zero depth zones to this area of the scene, as there is no semi-transparent geometry to be found here.

2. If $0 < \frac{frags_i}{frags_{total}} < \frac{1}{|totalDepthZones|}$, assign a single depth zone to that area of the scene, as there are semi-transparent fragments, but relatively few. Therefore, one depth zone should be able to catch all information in this part of the scene. Here, $frags_i$ is the amount of fragments in the current histogram zone, $frags_{total}$ is the total amount of fragments over all four of this texel's histogram bins, and $|totalDepthZones|$ is the amount of depth zones that the scene should be partitioned into.

3. Otherwise, take the ratio of the current amount of fragments over the total amount of fragments and multiply it by the total to be distributed amount of depth zones. Then round off to the nearest integer. Thus: $depthZones_i = \lfloor \frac{frags_i}{frags_{total}} * |totalDepthZones| \rceil$. Any errors caused by the rounding of this value are compensated later.

The above algorithm will distribute over the scene an amount of depth zones equal to either $totalDepthZones$, or, due to rounding errors, equal to $(totalDepthZones - 1)$ or $(totalDepthZones + 1)$. If too few or too many depth zones are assigned, a single depth zone will be either added to or removed from the histogram zone with the highest amount of complexity (i.e. the histogram zone with the highest amount of translucent fragments), as this is probably the most complex zone, asking for higher amounts of accuracy.

Take the following example: by using the above algorithm, we can find the distribution of depth zones for shadow map texel 1 in Figure 15. Along this texel's view ray, a total of 112 fragments are found. For the sake of this example, we arbitrarily estimate that using 16 depth zones should be able to capture all color information within this scene. A schematic example is given in Figure 16. The resulting depth zone distribution is found in Figure 17. The depth zone distribution is as follows:

- Histogram bin 1 contains 4 fragments. $0 < \frac{4}{112} < \frac{1}{16}$, so we find ourselves in scenario 2 of the algorithm, and assign a single depth zone to histogram bin 1.
- Histogram bin 2 contains 100 fragments. $\frac{100}{112} > \frac{1}{16}$, so we find ourselves in scenario 3 of the algorithm, and assign $\lfloor \frac{100}{112} * 16 \rceil = 14$ depth zones to histogram bin 2.
- Histogram bin 3 contains 0 fragments. We find ourselves in scenario 1 of the algorithm, and assign zero depth zones to histogram bin 3.
- Histogram bin 4 contains 8 fragments $\frac{8}{112} > \frac{1}{16}$, so we find ourselves in scenario 3 of the algorithm, and assign $\lfloor \frac{8}{112} * 16 \rceil = 1$ depth zone to histogram bin 4.

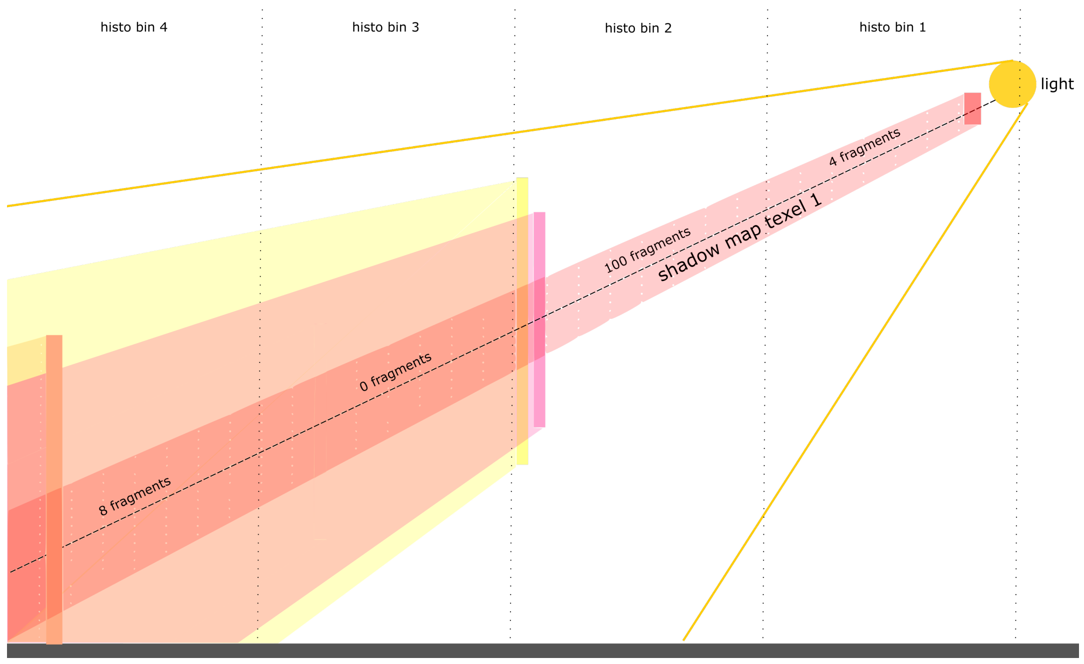


*Figure 16.* The fragments along shadow map texel 1 are counted.

To further improve accuracy, it is important to avoid situations in which one or multiple histogram zones are found to contain zero translucent fragments. If this is the case, we will want to ignore the entire range of depth values corresponding to this histogram zone. This will make sure not a single depth zone is assigned to these depth values, since this would be a waste of speed, memory and precision. To this end we will define zone borders as a start and end pair of depth values. This is as opposed to earlier methods, where the end-value of zone $i-1$ is equal to the start-value of zone $i$, which will not necessarily be the case anymore.
After having assigned a number of depth zones to each non-empty histogram bin, the depth zones are uniformly distributed over these bins. The total distribution, however, is (in most cases) non-uniform. Figure 22 shows how this new distribution of zones solves the problem we had before and once again all color information is perfectly captured. This non-uniform way of partitioning was combined with a data structure that was also used for the uniform partitioning method. More details on this can be found in Chapter 4

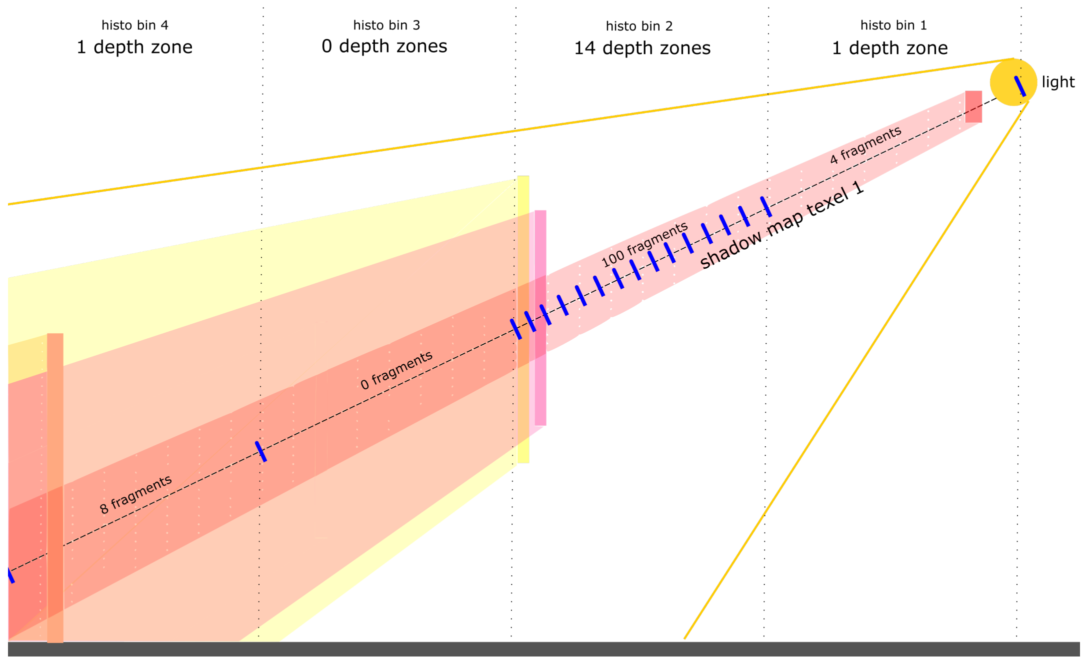


*Figure 17.* Using the histogram binning algorithm, we distribute an amount of depth zones to each histogram zone. The depth values in histogram bin 2 are now highly partitioned. In turn, this allows the yellow and pink colors in the second histogram bin to be captured correctly, which will now give correct results.

## 3.5. A Trade-off between Resolutions: Texel Groups

As was mentioned in the previous Section, the distribution of semi-transparent geometry can vary across various depths between different texels. However, this is not always the case. For certain neighboring texels, translucent geometry could be very similarly distributed. As an effect, the amounts of fragments in the corresponding histogram zones could be equal for these texels. A schematic example of such a situation is shown in Figure 18.

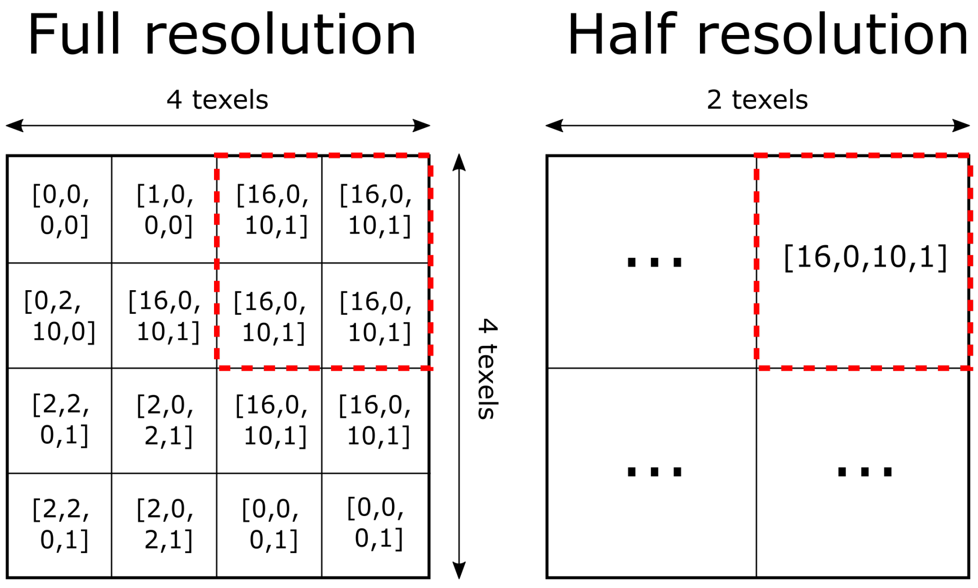


*Figure 18.* If a group of texels has similar histogram distribution, we can use a lower-resolution texture to denote this group of texels as a smaller group of texels. In this example, a 2-by-2 group of texels (indicated by the red dotted lines) is stored as a single texel in a half-resolution texture. The numbers between the square brackets are the amounts of fragments in each histogram zone, from the lowest histogram depth segment to the highest. In this case four histogram zones are used.

Now, if multiple neighboring texels have equal histogram zones, and thus equal depth zone distributions, we can assume that the corresponding geometry, and therefore the resulting colored scattering effects will also be similar. This allows us to create a trade-off between spatial resolution (i.e. the amount of depth zones) and the shadow map resolution. How this works, is that we can group texels with similar histograms together, and save the zone distributions in shadow maps of lower resolution (e.g. half, or quarter). Take the following example: we have an original shadow map texture of 1024x1024 texels. Alongside this texture, we create a 512x512 texture and a 256x256 texture. If we now find a group of 4-by-4 texels that have equal histogram zone distributions (i.e. same amounts of fragments in each histogram zone for all texels in that group), then we could just as well save the information about that group inside a single texel of a smaller texture, i.e. a texture that is 16 (=4x4) times smaller: a 256x256 texture. A 2-by-2 group of similar texels can be saved as a single texel in a half-resolution texture, i.e. a 512x512 texture. Conversely, if neighboring texels are not similar at all, they are likely located in a more complex part of the scene, or along edges of geometry. In these cases we will want to continue using the full resolution texture.

Now, the trade-off comes with the fact that we can set the amount of depth zones, for these lower resolution shadow maps separately from the original shadow map's amount of depth zones. For example, using $TexelGroups_{i,j,k}$ would mean that the full-resolution texture uses $i$ depth zones, the half-resolution texture uses $j$ depth zones, and the quarter-resolution texture uses $k$ depth zones. The reason we want to be able to do this is that we can set depth zone values for the lower resolution textures to a lower amount. This comes at the cost of accuracy, but since the accuracy is only lost in a scene's detailed areas, the visual difference will be minimal. At the same time, memory usage will be lower compared to previous methods. When applying this method, during ray-marching, we can now check for each fragment whether it projects back to a texel that belongs to a group of texels (i.e. written to a lower resolution texture) or whether it is still an individual texel (i.e. written to the full resolution texture). Each of the cases will be individually handled during ray-marching in a mutually exclusive way. Now take this example: in case we used lower amounts of depth zones for the lower resolution textures, fewer shadow maps will have to be created. Consequently, this allows us to conserve memory. Memory is saved as fewer shadow maps will have to be used.

In general, one could choose to use multiple different levels of lower resolution textures, depending on the complexity of the geometry of a scene. Furthermore, the grouping condition could be based on other conditions. For example, zone distributions might not have to be exactly equal, but instead very similar, as a grouping condition.

This way of grouping texels to trade off depth and shadow map resolutions will henceforth be called ‘Texel Groups’. It was implemented on top of the histogram partitioning method for reasons that will be made clear in Sections 4 and 5.

# 4
# Implementation

Various textures and data structure manipulations were used in a variety of ways throughout the methods that were discussed in this thesis. This section serves to clarify some of the specifics, as to make the reproduction of the discussed methods more convenient. Pseudocode, which can be found in Appendix A, was also created to facilitate the process of reproduction. Lastly, a TUDelft engine, called MxEngine, was used to ease the process of implementing and evaluating the various methods. A short description about this piece of software is given in the final paragraph of this Chapter.

## 4.1. General

A very important observation that was made, was that depths within the first phase of each method, the information gathering phase, should be linearized to make the uniform division of zones more logical. If depths are not linearized, and for example four zones are used, then the first three zones would visually cover only approximately $\frac{1}{4}$ to $\frac{1}{3}$ of the scene. Especially in scenes containing many translucent objects this would lead to a big loss of information, from which multiple artifacts would ensue. The choice to linearize depth values in the information gathering phase was applied to all methods but depth peeling, since depth peeling creates its own zones and no confusion can occur.

Furthermore all methods, besides depth peeling, use a firstDepth and a lastDepth texture. Depth peeling only uses a lastDepth texture as to know when no more peels can be made. Each of these textures costs one render pass to set up. Both textures are regular 32-bit depth textures and are used to restrict the values between which zones are partitioned. As was mentioned before, it was later found that the to-be-partitioned area could be restricted even more by utilizing the borders of the camera frustum. Sadly, due to time limitations, this was not implemented.

Many of the textures that are created in the various methods are written to during individual (render) passes. More details about each method's amount of passes is given in Chapter 5.

## 4.2. Partitioning Methods

Uniform partitioning was implemented using two different methods. Both of these methods use a lower (and constant) amount of render passes compared to depth peeling. The first method, which was called the Multiple Render Target method (or MRT), had its shortcomings, which will be discussed below. These shortcomings led to the implementation of the second method: the Multisample Texture Depth Zones method (or MSTDZ).

### 4.2.1. Depth Peeling

Since depth peeling is fairly straightforward to implement there are not a lot of implementation specifics to give. For each peel/pass, one 8-bit RGB texture and one 32-bit depth texture are used.

### 4.2.2. Multiple Render Target method

Using multiple render targets, it is possible to save information about multiple parts of a scene in multiple textures by writing to multiple color attachments at the same time during a single render pass. This implies that we could come up with a method that saves information about certain parts of a scene to one texture, while writing information about another part to another texture.

As was mentioned in Chapter 3, a scene can be partitioned into a set amount of depth zones. Using multiple render targets, we can assign one shadow map texture to each depth zone. This works for either uniform or non-uniform partitioning. For example, when uniformly partitioning a scene into four depth zones, the first texture is assigned to depth values between 0 and 0.25, the second one to depth values between 0.25 and 0.5, etc. Now if a semi-transparent fragment falls within such a region, its color is written to the texture corresponding to that fragment's depth value. If multiple colors are written to the same texel, depth testing will make sure that the front-most object's color is retained. This possibly leads to color artifacts in the final result, as colors could be skipped if partitioning is not done accurately enough.

Take for example a scene with multiple transparent objects located behind each other in a single depth zone (see Figure 19). Figure 20 shows the output of the shadow map corresponding to zone 3, which, due to depth testing, only contains pink. This will lead to wrong final results during ray-marching, as any fragment behind this object will only show as a pink color, instead of a mixed pink/blue color.

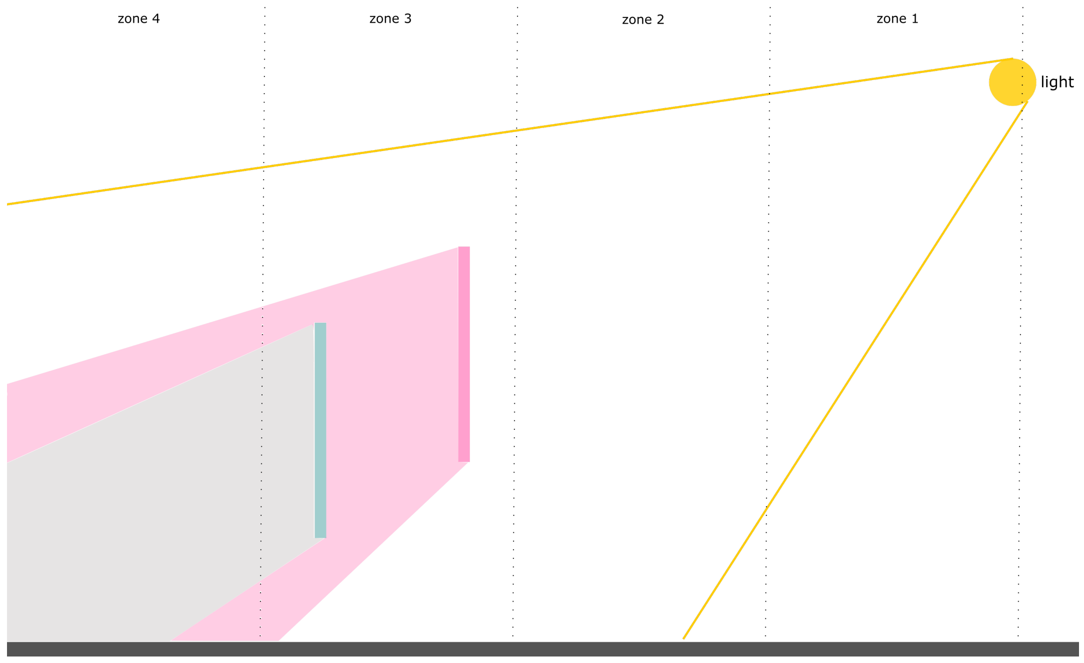


*Figure 19.* Schematic side-view of the MRT method in a scene in which it fails. In this case, two translucent objects are in the same depth zone (the third) and (due to depth testing) the pink color will be written to the third texture. As a consequence, the results become erroneous. Figure 20 shows the third texture (i.e. the shadow map belonging to the third zone).

*Figure 20.* The third texture will look like the above image after applying MRT to the scene in Figure 19, giving erroneous results.

One issue is that it is not possible to write to multiple depth attachments at the same time. This means another solution had to be found to be able to store the depth values of written fragments. These values are necessary during the ray-marching phase for determining behind which semi-transparent objects a fragment is found. For this purpose, we chose to use a 32-bit RGB color texture. Within the 32 bits of the texture's Red channel, RGB values of respective translucent geometry are stored, taking up 24 out of 32 bits of memory per texel. This leaves 8 empty unused bits. The Green channel contains the corresponding 32-bit alpha value. In the texture's Blue channel, we store the 32-bit depth values corresponding to the color values in the Red channel. We need to store the depth value like this (instead of in depth textures), because it is not possible to write to several depth textures at the same time. A schematic overview of what is written to every single texel is given in Figure 21. Per depth zone we need one of these RGB textures.

On one hand, the MRT method has a very positive feature, in that it provides a method that solves this thesis' problem in a single render pass. On the other hand, if multiple semi-transparent objects are situated behind each other in the same zone, then only one of their colors will be written to that depth zone's respective texture. This will obviously lead to a discrepancy when compared to the reference quality. Combined with the fact that most current graphics cards can only write to a maximum of 8 render targets at the same time, visual artifacts in complex scenes are inevitable. The consequence of this limitation is that a scene can be partitioned into a maximum of only 8 zones, with each zone spanning an average of $\frac{1}{8}$ of a scene's depth. Especially in complex scenes, this could lead to highly inaccurate gathering of color information of semi-transparent objects, especially if many such objects are present in a scene. These shortcomings occur for both uniform and non-uniform partitioning. One could of course always use more render passes, but this beats the purpose of this method and costs a lot of computational time. This issue is overcome by the newest graphics cards, which support read-write functionality. However, even if the limitation of having only 8 zones is lifted, memory is still consumed by all of the corresponding textures.

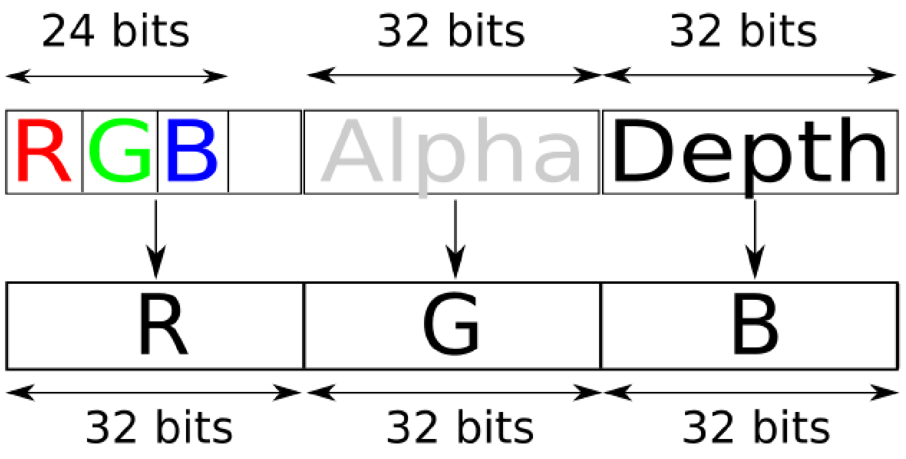


*Figure 21.* The distribution of information over a texel within an MRT texture. 88 out of 96 bits are occupied within each texel.

### 4.2.3. Multisample Texture Depth Zones

A solution was needed that would make it possible to create more partitions than the MRT method could, or to find a solution that did the same as MRT, but faster, as to make it worth the loss in quality. A possibility would be to write all of the color information into a single texture, while still needing only a single render pass. The question, however, was how to save all of the color information across multiple depth zones per texel in a single texture. The answer: multisample textures.

Multisample textures are textures that possess more than a single sample per texel. The original use of this type of texture was to use it for anti-aliasing purposes, of which multisample anti-aliasing (MSAA) is a widely used example, especially in computer games. However, the data structure behind these textures allows for a different kind of use. Each sample inside a pixel of a multisample texture can contain a color value. This means that having an 8-sample multisample texture, comes down to having 8 textures in one. On today's graphic cards, it is common to be able to have up to 16 or 32 samples per pixel. Using extremely high-end graphics cards this number can even go up to 64. This opens up the possibility of saving a much higher amount of color values per pixel in a single pass than the MRT method could do. This is exactly what was needed to improve upon it. In summary, multisample textures come with three benefits compared to MRT:

- Only one texture is needed, saving memory compared to MRT.
- Writing only happens to one texture at a time.
- The possibility to partition a scene into more than 8 depth zones (on most modern graphics cards), allowing for much more qualitative accuracy.

The main idea behind the resulting method, which was dubbed Multisample Texture Depth Zones (MSTDZ), is as follows:

1. Instead of creating a texture for each depth zone, create a single multisample texture and set the amount of samples per pixel equal to the required amount of depth zones.
2. Once again, partition the scene into depth zones, as seen from the light source. This time, many more zones can be used than if the MRT method was used.
3. For each fragment along each texel's view ray, write the fragment's color value to the texture's corresponding subsample of the current texel.
4. Finally, during ray-marching, for each sample that is behind one or more translucent objects, we once again step through the involved depth zones in ascending order up until the current fragment's depth, blending colors along the way.

A schematic example is given in Figure 22. As with MRT, the distribution of zones can be as desired, either uniformly or non-uniformly distributed.

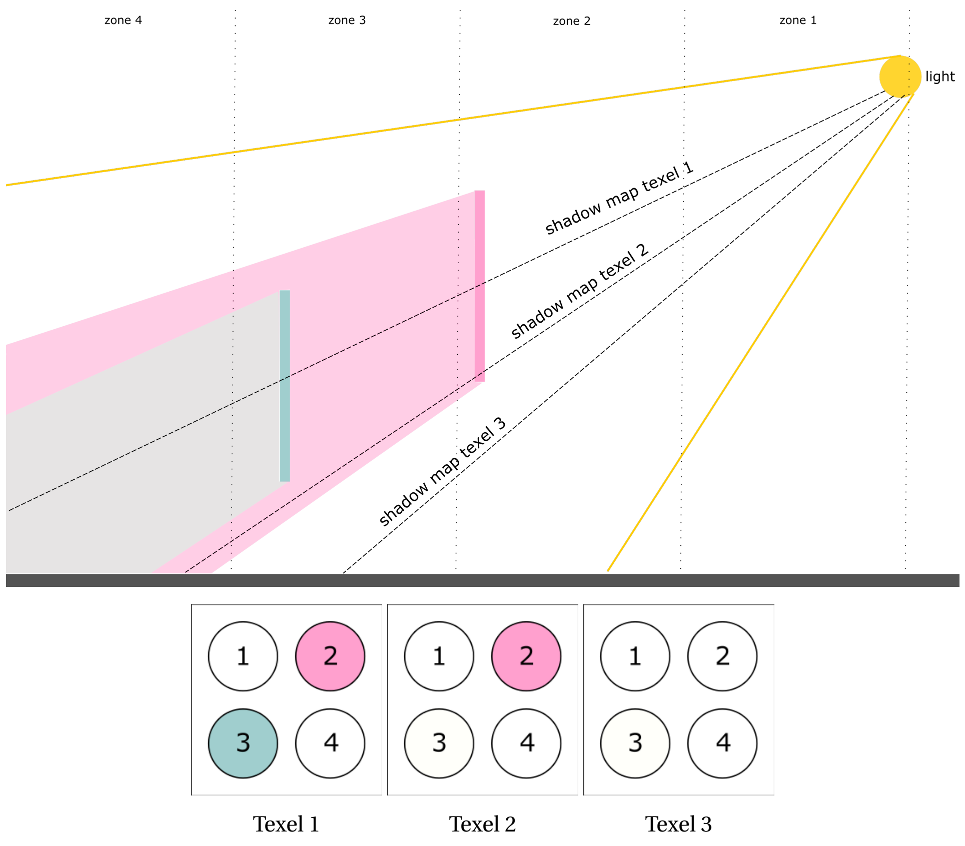


*Figure 22.* Schematic example of how MSTDZ works. Various shadow map texels are shown to give an idea of how colors are written to subsamples of the multisample texture.

MSTDZ can suffer from the same inaccuracies as MRT, where multiple translucent objects are located in the same zone and thus only one of the colors getting written to the multisample texture (see Figure 23). However, one of the positive features of the MSTDZ method, is that multisample textures can have up to 32 subsamples, and thus 32 zones can be used, in contrast to MRT's 8 zones. What this implies, is that a lot more acccuracy can be gained at only little computational cost (albeit higher memory cost). Increasing the amount of zones will cause even semi-transparent objects that are relatively close to each other to be written to different subsamples, thus taking away the artifacts that would occur if a lower amount of zones with MRT was used. In Figure 24, this is showcased properly, where the scene that was first cut up into 4 zones is now cut up into 32 zones. The corresponding 'shadow map texel 1' is now properly filled (see Figure 24), whereas this was not the case when using MRT with 4 zones.

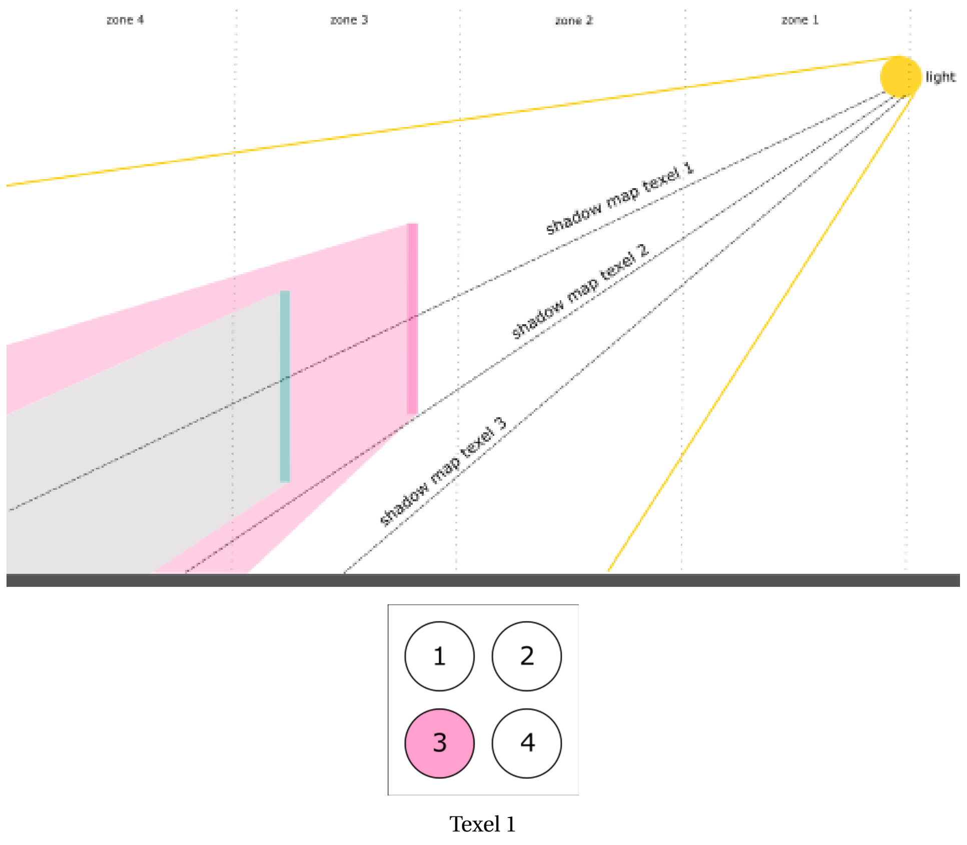


*Figure 23.* MSTDZ suffers from the same inaccuracies as MRT does when using a low amount of subsamples/zones. Depth testing will see that the pink semi-transparent object is closer to the light source and will therefore write pink to this texel. The blue semi-transparent object will be completely ignored along this texel's view ray, so final results after ray-marching will be erroneous.

Regardless of the amount of zones that are used, only one 8-bit RGBA multisample color texture and one 32-bit multisample depth texture are used in this method. Depending on the amount of samples that one wishes to utilize, the amount of used memory will increase or decrease. Due to the existence of multisample depth textures it is no longer necessary to write depth values to the color texture as was done in the MRT method.

To our best knowledge, there are no dedicated OpenGL functions or extensions that allow users to write to exactly one subsample at a time. However, this is exactly what we needed to do for this method to work. The solution we came up with is as follows: there is a way to mask samples using a sample mask. This sample mask has the same amount of bits as there are zones, where a 0 bit at position $i$ masks sample $i$ and a 1 bit at position $i$ leaves sample $i$ unmasked. This makes it possible to simply mask all samples and unmask the exact sample that needs to be written to for each fragment, which solves the problem.

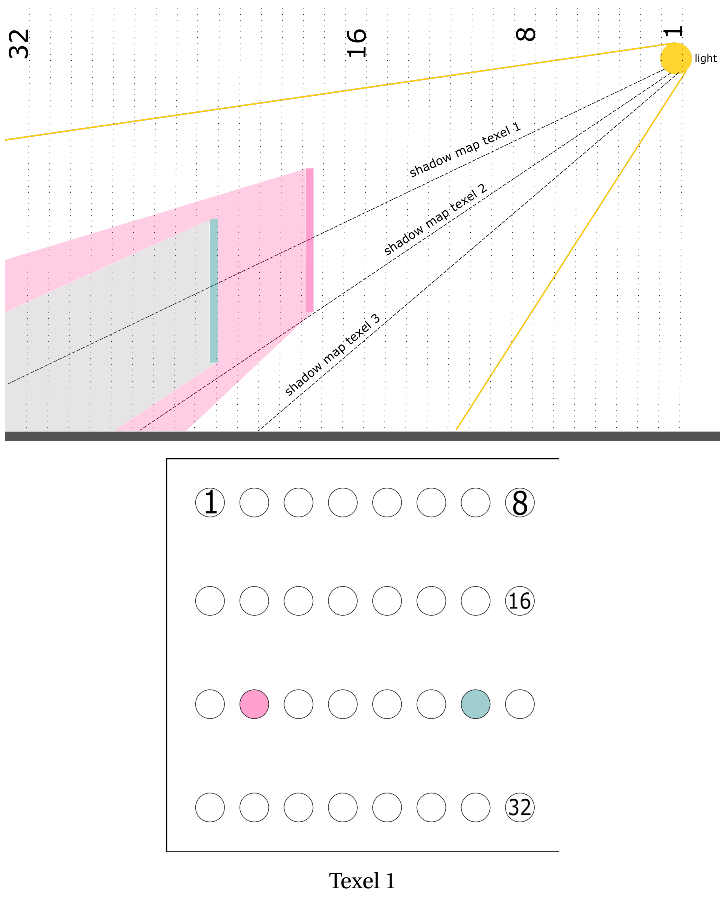


*Figure 24.* Top: Increasing the amount of zones will take away a large amount of the inaccuracies that MRT has. In this case, both translucent objects' colors are written correctly to individual samples (18 and 23, for pink and blue respectively). Bottom: The corresponding 'shadow map texel 1' is shown with 32 subsamples, of which two are filled with the corresponding zone colors.

### 4.2.4. Histogram-based MSTDZ

The histogram-based partitioning can be attached as a pre-pass for both MRT and MSTDZ. Setting up the corresponding histogram texture (which contains the amounts of fragments per histogram zone as seen from the light) costs one render pass. It is possible to lower the resolution of the histogram texture to save computational time. This might go at the cost of quality, however, as certain depth zone distributions will be based on others in this case.

In our implementation the histogram texture is an 8-bit RGBA texture. This allowed us to write amounts of fragments to each of the four color channels, up to a number of $2^8 - 1$ fragments per histogram bin per texel. Fragments are counted using a separate fragment shader which adds $\frac{1}{255}$ to the fragment's respective texel channel (based on its depth), e.g. a fragment with a depth value between 0 and 0.25 would lead a value of $vec4(\frac{1}{255}, 0, 0, 0)$ to be written and a fragment between 0.5 and 0.75 would have a $vec4(0, 0, \frac{1}{255}, 0)$ written to the histogram texture.

By utilizing additive blending, the final value within each channel of a texel will simply be the sum of all fragments within the corresponding histogram zone over 255, which in the color gathering pass is simply multiplied by 255 to obtain the actual amounts of fragments.

## 4.3. Texel Groups

The implementation of the texel groups (or, the trade-off between resolutions) involved the addition of multiple extra passes and textures. We chose to use a half-resolution texture and a quarter-resolution texture to represent the texels that are in groups of 2-by-2 and those that are in groups of 4-by-4, respectively. These textures are used on top of the standard full resolution texture, which will represent the texels that remain single texels. The three textures combined are called 'texel group textures'. We also need a texture that determines for each texel in what group size they are located (i.e. 1, 2 or 4, for single, 2-by-2 and 4-by-4, respectively).

First of all, some way was required to determine which texels should be in groups of 2-by-2, which ones should be in groups of 4-by-4 and which texels should remain as they were originally. To determine this, a compute shader was set up which takes a texture as input and outputs the half-resolution texture. The compute shader will check every top-left pixel of groups of 4 pixels (grouped 2-by-2). The half-resolution texture is then filled in the following way:

- If a (top-left) pixel in the input texture has the same histogram distribution as all three of its neighbors (directly right, directly below, and diagonally right below, see Figure 25), then this pixel will be grouped with these neighbors and the histogram distribution is written to the half-resolution output texture at the corresponding half-resolution texel.
- If a (top-left) pixel in the input texture does not have the same histogram distribution as all three of its neighbors, then write a vector of zeroes to the half-resolution output texture at the corresponding half-resolution texel.

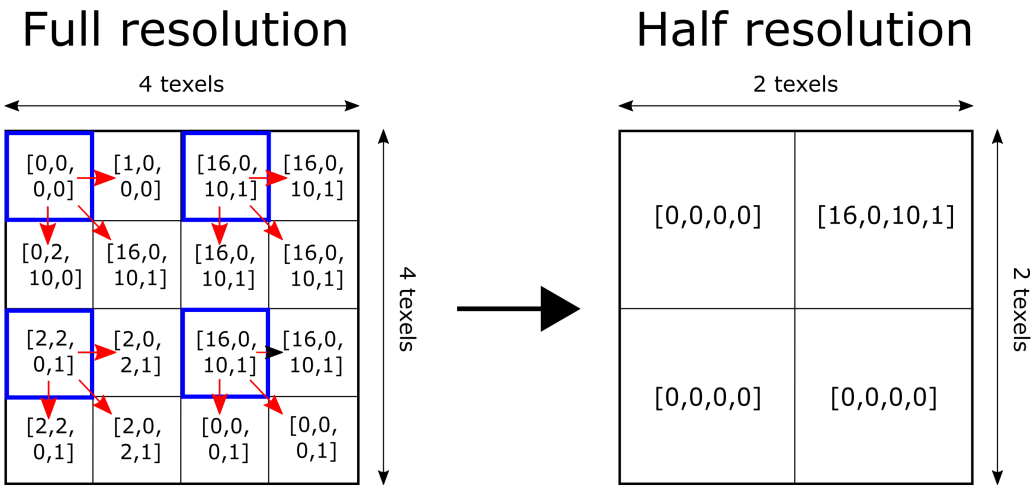


*Figure 25.* The compute shader steps over every top-left texel of groups of 2-by-2, denoted by the blue squares. Only the top-right blue texel actually has exactly equal neighbors, so the content of the histogram distribution is written to the corresponding half-resolution texel. Vectors of zeroes are written to other pixels. These zeroes are later skipped during ray-marching.

This process can be repeated by using the half-resolution texture as input, which will result in a quarter-resolution texture as output.

Using these half- and quarter-resolution textures, we then create another texture of original, full resolution, which stores the earlier mentioned group sizes of each texel. This 8-bit R texture (containing only a red channel) will be filled entirely with ones, twos and fours, based on the contents of all three currently obtained textures (see Figure 26). This 1,2,4-texture will be used during ray-marching, to determine from which of the three texel group textures colors will need to be read.

After going through the four passes of creating the histogram texture, the half- and quarter-resolution textures and the 1,2,4-texture, we go through three more render passes to set up a full resolution multisample texture (which was also set up within the other methods), along with two new half-resolution and quarter-resolution multisample textures. These three textures are the ones that are going to contain the actual color

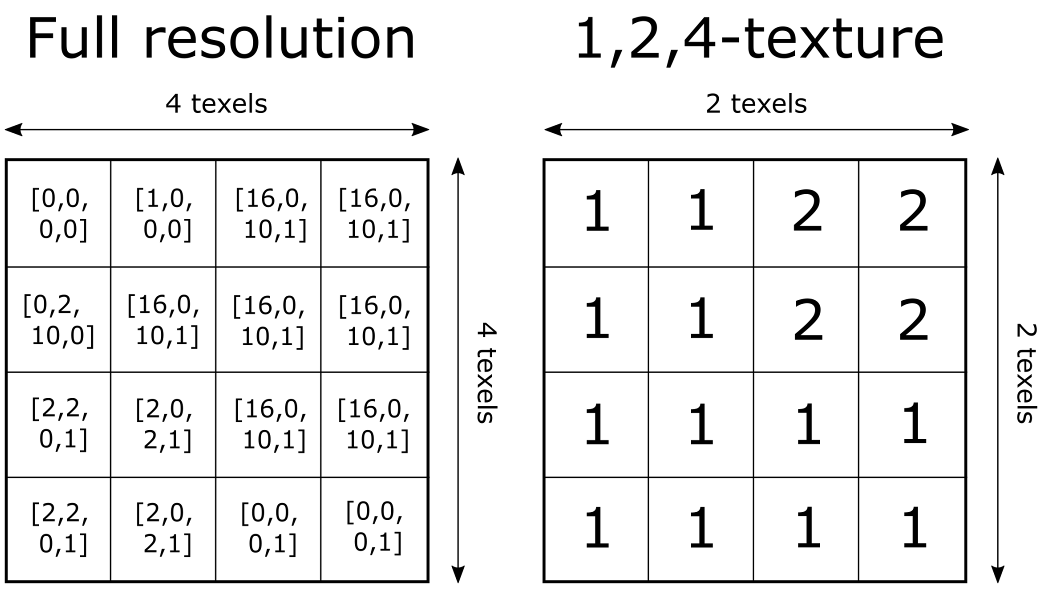


*Figure 26.* Example of how the 1,2,4-texture is filled. In reality, it uses the half- and quarter-resolution textures too, but for visualization purposes only the full resolution texture is shown. Since there are no groups of 4-by-4, no 4s are written to the 1,2,4-texture in this example.

information and they are also the ones that each have their own amount of samples.

During ray-marching, we will check for each sampled fragment whether it projects to a 1, 2 or 4 within the 1,2,4-texture. From this value we determine which multisample texture to sample our colors from as follows: if a pixel contains a 1, the original resolution texture should be used during ray-marching sampling. Consequently, a 2 denotes the half-resolution texture ought to be used and a 4 signifies the quarter-resolution texture needs to be utilized.

The texel groups addition is built on top of the Histogram MSTDZ method and adds a total of 5 render passes to the previous amount of 5, for a total of 10 render passes.

### MxEngine

To increase productivity during the implementation of the methods proposed in this thesis, a program called MxEngine was used. MxEngine was created by TU Delft and is able to contain bits of code into what are called ’filters’ in a convenient GUI. These filters can be interconnected to form a graph, which will turn into an entire graphics pipeline; starting from reading in a mesh, all the way to rendering it onto the user’s screen. A major benefit of this graph structure is that each new method could be captured into a single filter that fits right between all the other steps, taking away the necessity to multiply implement many of the trivial steps of the graphics pipeline. Potential future researchers who happen to be associated to Delft University of Technology can look to use MxEngine to optimize their workflow.

# 5 Results and Evaluation

In this Chapter, all of the developed methods are evaluated extensively. Comparisons will be made between memory usage, computational efficiency, visual quality, geometrical complexity, and differences in shadow map resolutions. All methods were implemented, timed and evaluated on an i7 system with a GeForce GTX 1070. Furthermore, the optimizations that were added to the MSTDZ method (see Section 3.4) were also added to the MRT method to make the evaluation as fair as possible. All colored objects in the scenes below are semi-transparent. There is one exception: if an object is gray, it is opaque and meant to have shadows reflected upon it.

All methods are abbreviated: depth peeling (DP), Multi Render Target method (MRT) and Multisample Texture Depth Zones (MSTDZ).
For reasons that are made clear in this Chapter, it was chosen to only combine the histogram method and the texel groups addition with the MSTDZ method, and not to combine these with the MRT method.

All images are available in a larger, more easily viewable resolution in Appendix C.
Lastly, it is important to realize that we must also take into account the fact that many other visual effects have not been added to the scene. If a method performs in real-time, we should also add those other effects to obtain a more full-fledged result and then determine whether a method is still appropriate for real-time use. Of course, obtaining better results than the reference method is still a welcome improvement.

## Basic Characteristics

Two of the basic characteristics of the methods that can be compared are the number of passes each method requires and the approximate memory consumption when using textures of various sizes. The number of passes should give us a good estimation to the method's general computational efficiency. 1024x1024 is the standard size of textures used throughout the evaluation, but, for reference, memory consumption using other shadow map sizes is also given.

| **Method** | **# of passes** | **Memory consumption 1024x1024** | **Memory consumption 512x512** | **Memory consumption 2048x2048** |
|---|---|---|---|---|
| $DP_2$ | 3 | 22 MB | 5.5 MB | 88 MB |
| $DP_8$ | 9 | 64 MB | 16 MB | 256 MB |
| $DP_{32}$ | 33 | 232 MB | 58 MB | 928 MB |
| $MRT_2$ | 4 | 36 MB | 9 MB | 144 MB |
| $MRT_8$ | 4 | 108 MB | 27 MB | 432 MB |
| $MRT_{32}$ | 4 | 400 MB | 99 MB | 1584 MB |
| $MSTDZ_2$ | 4 | 28 MB | 7 MB | 112 MB |
| $MSTDZ_{32}$ | 4 | 268 MB | 67 MB | 1072 MB |
| Histogram $MSTDZ_2$ | 5 | 32 MB | 8 MB | 128 MB |
| Histogram $MSTDZ_{32}$ | 5 | 272 MB | 68 MB | 1088 MB |
| Histogram MSTDZ + Texel $groups_{2,2,2}$ | 10 | 43.25 MB | 10.8125 MB | 173 MB |
| Histogram MSTDZ + Texel $groups_{32,32,32}$ | 10 | 358.25 MB | 89.5625 MB | 1433 MB |

*Table 1.* Basic characteristics of all methods used throughout the thesis. Subscript numbers represent the amount of peels (for Depth Peeling) or zones (for all other methods). Texel groups have three subscript numbers, representing the amount of zones for the full, half and quarter resolution textures, respectively.

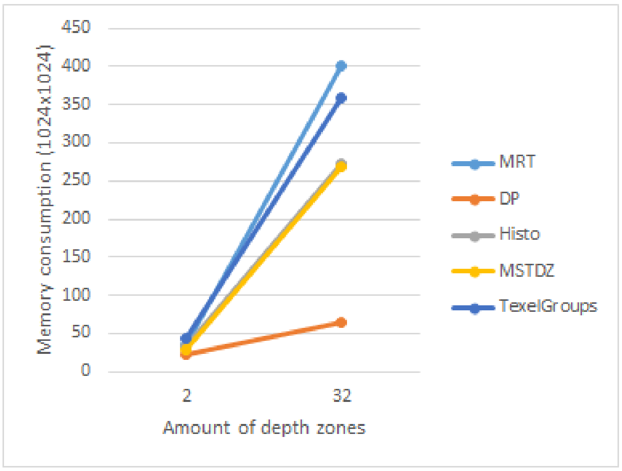


*Figure 27.* Visualization of memory consumption at 1024x1024 resolution using 2 or 32 depth zones.

As is easily observable from Table 1, Depth Peeling is the only method that uses a non-constant amount of render passes. All other newly proposed methods require a constant amount of passes, regardless of the amount of zones that are used, the complexity of the geometry, or any other properties. As the amount of zones rises, memory consumption increases for these methods, but the amount of render passes never does.

Since Histogram MSTDZ utilizes only a single extra render pass and texture to determine the distribution of depth zones, it is not surprising that the memory consumption rises only slightly compared to its basic predecessor, MSTDZ. Conversely, if applied well, the Histogram addition often uses fewer zones than regular MSTDZ, so that memory usage will actually be lower when used in a real scene.
The complexity of the Texel Groups addition brings with it a big rise in memory consumption. Yet, if properly used, one or two of the method's multisample textures (full-resolution and half-resolution, respectively) will only contain a very low amount of subsamples when compared to the quarter-resolution texture. In this case, the memory consumption of MSTDZ with the Texel Groups addition may even be lower than that of the prior methods.

Moreover, Table 1 shows that Depth Peeling is most efficient in terms of memory usage. After Depth Peeling, MSTDZ is most efficient in terms of memory consumption. However, the theoretically infinite amount of render passes that Depth Peeling might have to compute, makes the method computationally slow and unpredictable. Furthermore, the goal of methods such as MRT and MSTDZ is to approximate the results of a scene in a faster manner than depth peeling would. This means that a result obtained with depth peeling with a high amount of peels (costing a large amount of memory), could be approximated by MRT or MSTDZ using a lower amount of memory (and possibly lower computational time).

The final important observation is that Figure 27 shows well how MRT does not scale properly into higher amounts of depth zones (i.e. higher than 8), due to the extreme amounts of memory consumption.

## Evaluation Scenes Characteristics

Below is a table (Table 2) containing information about the geometrical complexity of the scenes that were used throughout the evaluation. Any specific properties about a scene that affect the results will be reported in the specific parts of the evaluation that use these scenes. Figures 28 through 32 show Scenes 1 through 5, respectively.

| Scene name | # of vertices | # of faces | # of triangles | Figure # |
|---|---|---|---|---|
| Scene 1: Five Primary Planes | 48 | 72 | 72 | 27 |
| Scene 2: Nine Planes | 88 | 132 | 132 | 28 |
| Scene 3: Far-away Plane | 56 | 84 | 84 | 29 |
| Scene 4: Yellow Planes | 56 | 84 | 84 | 30 |
| Scene 5: Foliage | 1,717,027 | 2,034,876 | 2,034,876 | 31 |

*Table 2.* Geometrical complexity of evaluated scenes.

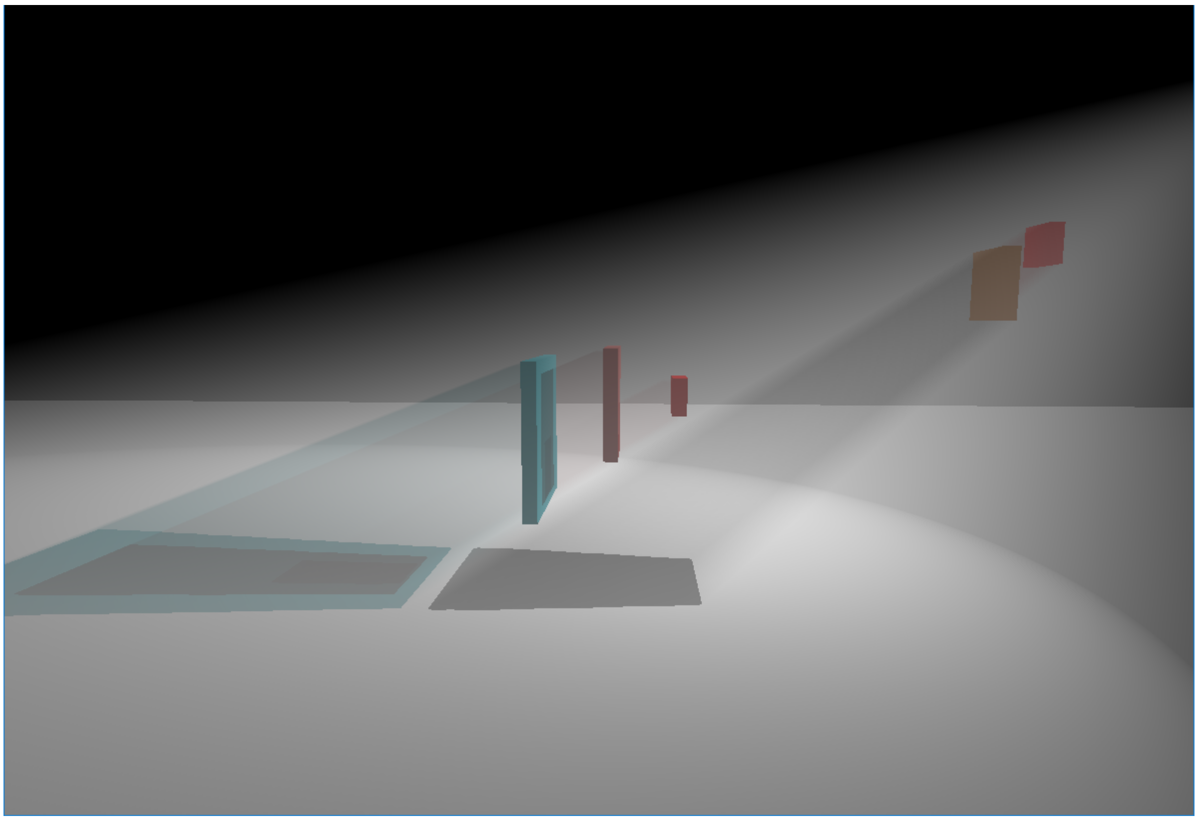


*Figure 28.* The Five Primary Planes scene.

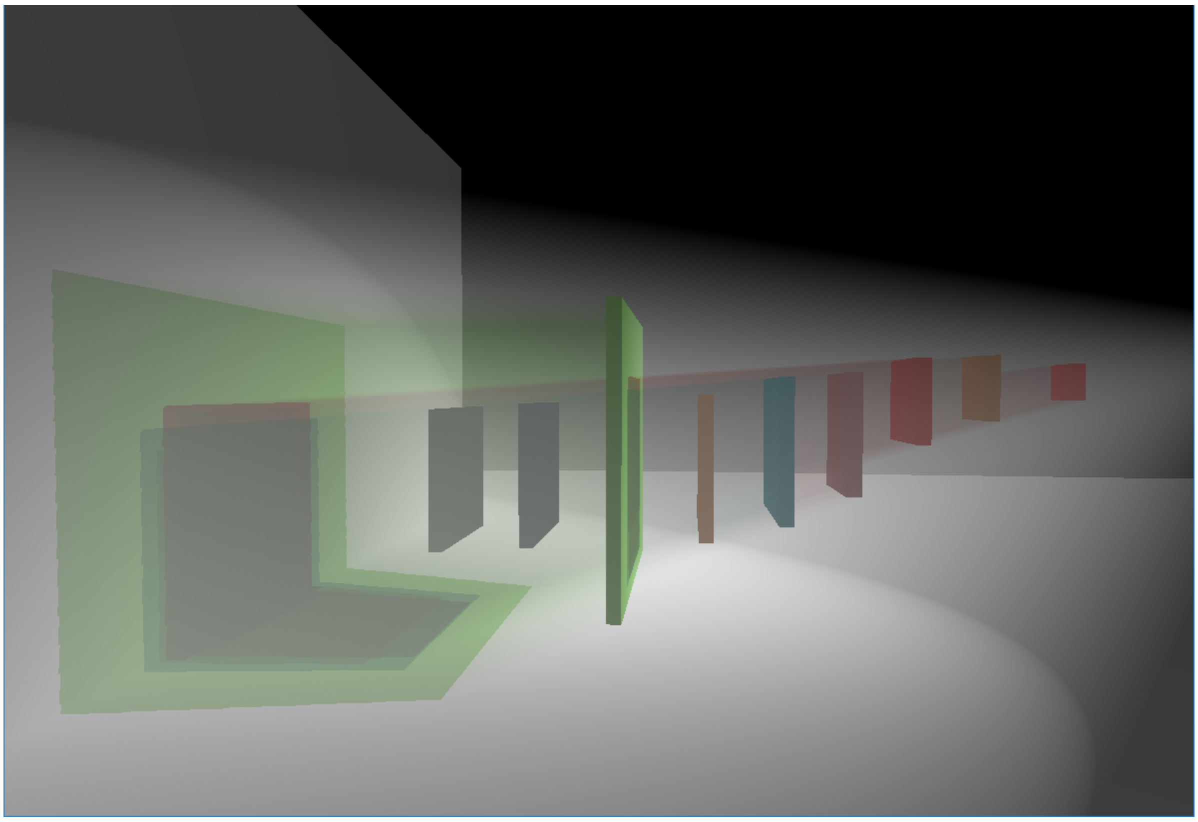


*Figure 29.* The Nine Planes scene.

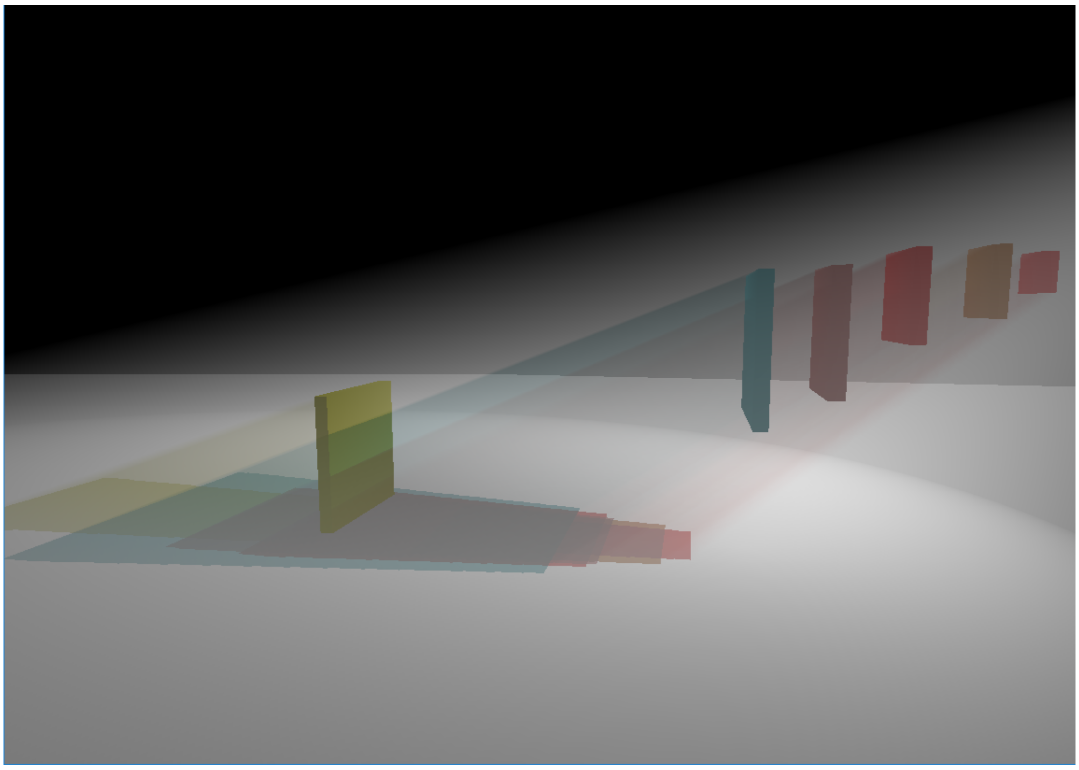


*Figure 30.* The Far-away Plane scene.

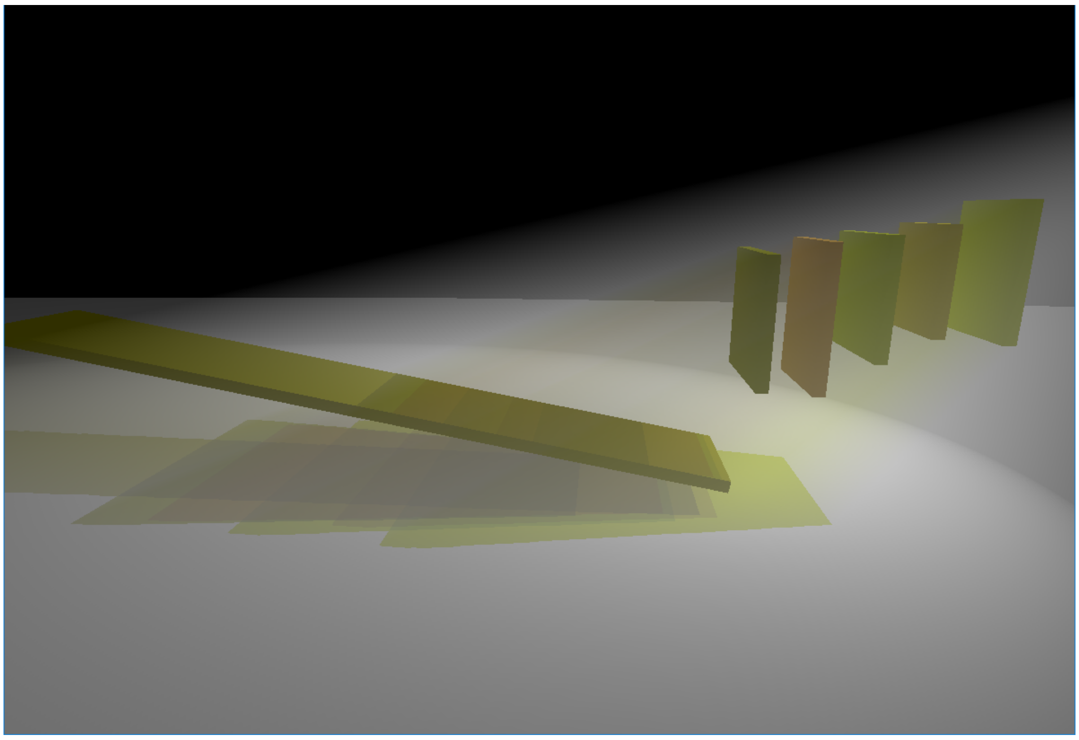


*Figure 31.* The Yellow Planes scene.

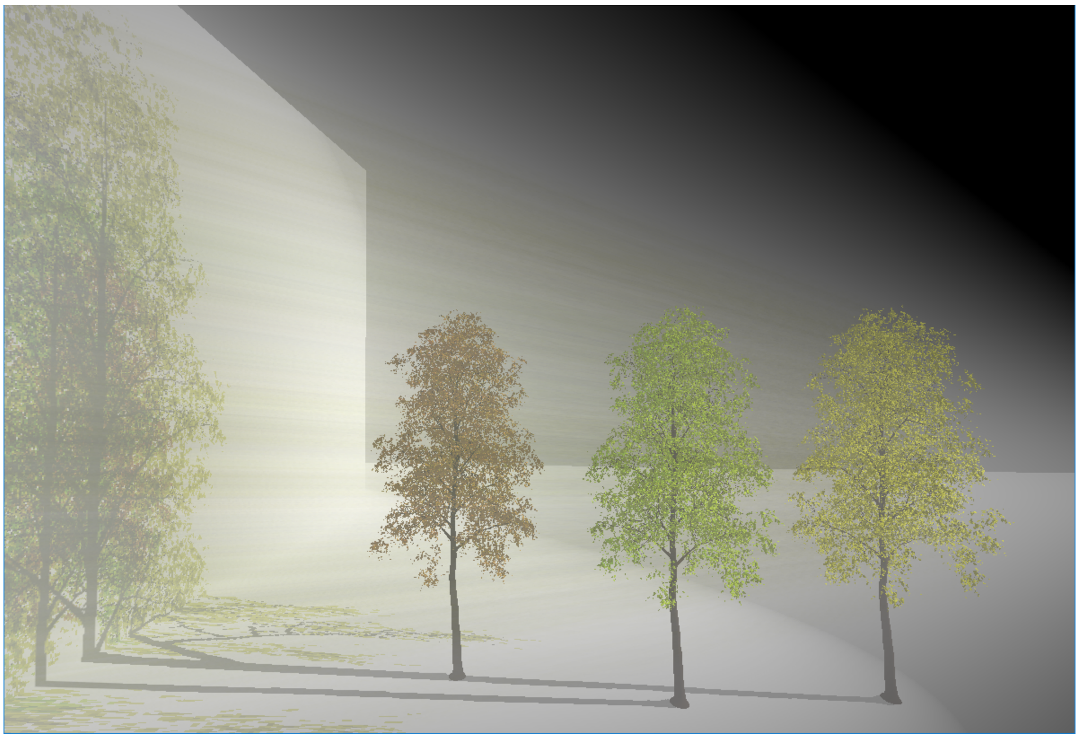


*Figure 32.* The Foliage scene.

## Evaluation Motivation

Firstly, each individual method will be tested and evaluated in a very simple, but distinguishing scene (Scene 1: Five Primary Planes) to showcase their basic workings and differences. This scene is set up in such a way that we can show these differences properly. It also allows us to do some initial timings to start comparing the performance of each method.

Then, the effect of the depth zone restrictions (discussed in Section 3.3) are made clear, also using the Five Primary Planes scene.

After this, using a different scene (Scene 2: Nine Planes), the difference between the storage methods MRT and MSTDZ are evaluated and discussed. The Nine Planes scene contains, as the name suggests, nine semi-transparent planes. Since MRT can only partition a scene into up to eight zones (without resorting to doing more render passes, costing extra time), MRT will not be able to capture all information in this scene without obtaining visual artifacts. MSTDZ does not have this limitation and can therefore solve this scene perfectly.

Both MRT and MSTDZ are then compared to 'Histogram MSTDZ' (or Histo MSTDZ, i.e. MSTDZ with the histogram method addition) to show the strengths and weaknesses of this addition compared to its more basic, naively partitioning versions. This is done in Scene 3: Far-away Plane, which is a scene that is bound to give difficulties to naive uniform partitioning methods (i.e. basic MRT or MSTDZ). The expectation is that Histogram MSTDZ will intelligently partition the scene and that it will be able to give good-looking results using fewer depth zones than its basic counterparts.

The reference method, depth peeling, is then compared to basic MSTDZ, Histogram MSTDZ and 'histogram MSTDZ with Texel Groups' (simply called Texel Groups, for short) to compare time consumption and quality for each of these methods and to see where improvement is needed for the newly proposed methods. This last evaluation is done in two different scenes, a simple one (Scene 4: Yellow Planes) and a complex one (Scene 5: Foliage), to give a sense of the general performance of each of the new methods, while being compared to the reference method. It is expected that either MSTDZ or Histogram MSTDZ will outperform depth peeling in both scenes, but especially in the more complex one. In the Foliage scene, depth peeling needs a high amount of peels to give a result that looks visually pleasing and uniform colors of the leaves of the trees lead us to expect that our proposed methods can easily approximate the results using way fewer depth zones. Texel Groups might not computationally outperform MSTDZ or Histogram MSTDZ, but it should outperform depth peeling. On top of that, we expect that Texel Groups gives the choice to lower memory consumption by trading off for only slight amounts of quality.

Lastly, a look will be taken into the effect of shadow map resolution on performance of each method and some individual render pass timings are done to see where the bottlenecks of each method lie.

## Basic Performance of Individual Methods

To give a solid idea of how each method works, and how results are affected by utilizing different amounts of peels/zones, we start by evaluating each method in a very simple scene: Scene 1: Five Primary Planes. This scene is set up in such a way that it is effective in showing the differences between the various methods. It also allows us to set up a table with some initial timings. Furthermore, it allows us to showcase the immense difference that the depth zone restrictions (discussed in Section 3.3) can make. In the concluding paragraph of this Section, a summarized table containing all timings is given (see Table 8), along with graphs that visualize the data (see Figure 39).

In Figure 33, we see Five Primary Planes rendered using Depth Peeling to get an optimal quality image. Three peels were used to reach this result. Figure 33 also shows what happens if too few peels are used, i.e. one and two peels, respectively.

In Figure 33 (middle), where only one peel is used, we can clearly see artifacts. The red semi-transparent planes seem to be propagating their red light entirely through the blue plane. This is because in the first peel, mostly red fragments were peeled away, and no blue fragments were written (except for the top part of the blue plane, for which there are no occluding red planes).

In Figure 33 (right), where two peels are used, the artifacts are less noticeable, yet they still remain, during the first peel the smaller red plane is peeled, which leaves a few red fragments for the second peel, so that a small portion of the blue fragments are still not peeled yet, leaving a slightly less noticeable red ray that still propagates through the blue plane.

To obtain an idea of the computational speed of Depth Peeling in a simple scene like this, see Table 3.

| **Method** | **Time per frame (in ms) Color gathering** | **Time per frame (in ms) Ray-marching** | **Total time** |
|---|---|---|---|
| $DP_1$ | 0.032 | 9.25 | 9.282 |
| $DP_2$ | 0.088 | 9.39 | 9.478 |
| $DP_3$ | 0.210 | 9.45 | 9.66 |

*Table 3.* Depth Peeling's performance in the Five Primary Planes scene with various amounts of peels up until optimal quality is reached. Lower case numbers denote the amount of depth peels that are used.

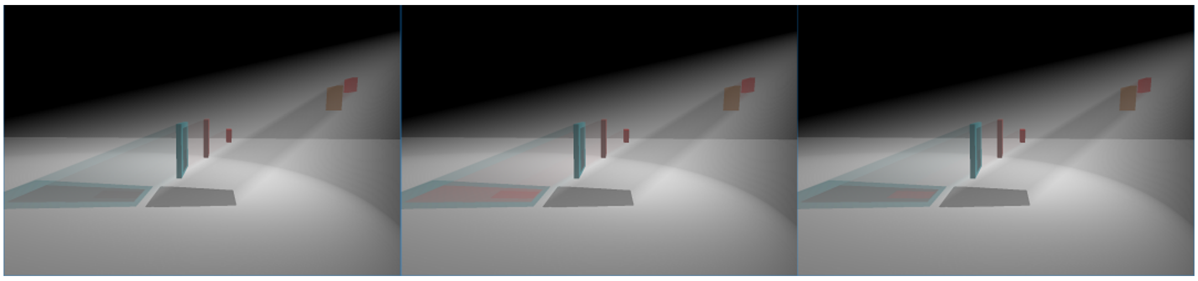


*Figure 33.* The Five Primary Planes scene rendered with Depth Peeling. Left: 3 peels. Middle: 1 peel. Right: 2 peels.

The Multi Render Target method (as well as MSTDZ, as it will turn out) work well for this specific scene due to the choices of color. Figure 34 shows the Five Primary Planes scene rendered using the Multi Render Target Method. To reach an optimal result, 3 zones were needed, just as with Depth Peeling in this same scene. It must be noted, that reaching optimal quality with either MRT or any of the versions of MSTDZ, will always cost at least the same amount of zones as Depth Peeling, or more.

Interestingly, using only two zones seems to be giving exactly the same result as when three zones are used. This can be seen in Figure 34 (middle). Looking more closely (see Figure 35), we can see that the shadow casted by the smaller plane has a slightly more red tint when we use two zones, meaning that some of the red light still passed through the blue plane. Although this is an artifact, it is very barely noticeable to the naked eye, meaning that, in this specific case we, can approximate the result of Depth Peeling using only two zones, instead of three. This is an important observation, as even fewer depth zones could be used in specific scenes like this one where colors of semi-transparent objects are similar.

Additionally, when a single zone is used, the same problem happens as with Depth Peeling (see Figure 34 (right)), where colors are stored incorrectly and the red rays are passing through the blue plane again.

Table 4 shows MRT's computational performance in The Five Primary Planes scene.

| **Method** | **Time per frame (in ms) Color gathering** | **Time per frame (in ms) Ray-marching** | **Total time** |
|---|---|---|---|
| $MRT_1$ | 0.045 | 8.72 | 8.765 |
| $MRT_2$ | 0.049 | 8.88 | 8.929 |
| $MRT_3$ | 0.057 | 8.95 | 9.007 |

*Table 4.* The Multi Render Target method's performance in the Five Primary Planes scene with various amounts of depth zones, up until optimal quality is reached. Lower case numbers denote the amount of depth zones that are used.

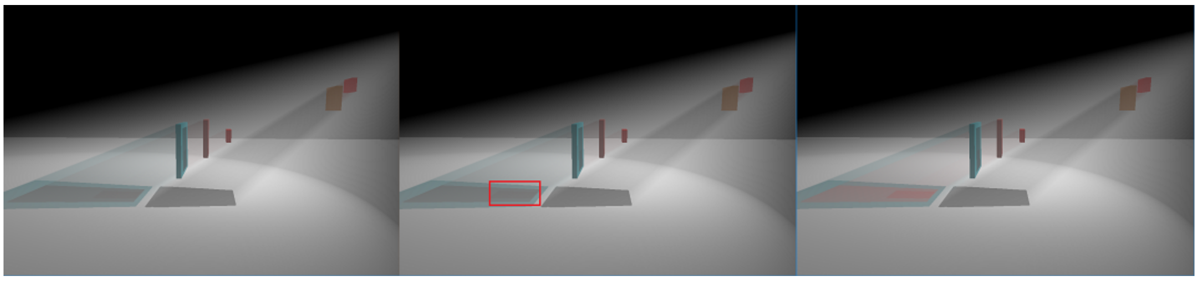


*Figure 34.* The Five Primary Planes scene rendered with the Multi Render Target method. Left: 3 zones. Middle: 2 zones. Right: 1 zone.

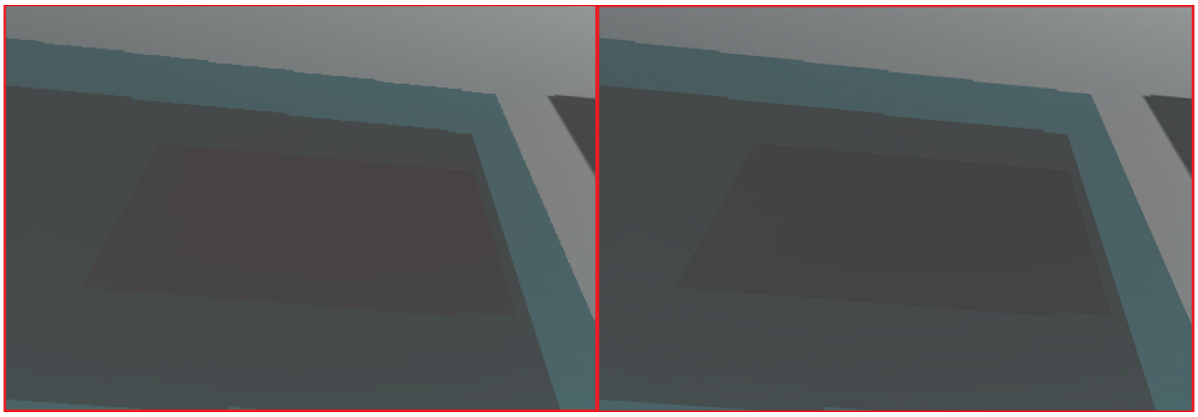


*Figure 35.* Example of barely visible artifact using MRT in the Five Primary Planes scene. Left: 2 zones. Right: 3 zones.

For scenes in which low amounts of zones are used, the basic MSTDZ method's results are almost equivalent to the ones produced by the MRT method. Using basic MSTDZ, Figure 36 shows that, once again, 3 zones are required to obtain an optimal result.

In Figure 36 (middle, right), where the results for using MSDTZ with 2 and 1 depth zones are shown, respectively, we can see a slight error within the blue light rays. It was suspected that these errors arose from using depth bounds that are too tight, but the depth bounds optimization (first and last transparent depth values) was also applied to MRT and this same artifact does not appear in the same scene. Even by adding a small bias to the first and last transparent depth values to stretch the bounds a little bit, the artifact is still not solved. Our first suspicion was that it is caused by the specific angle between the top and front fragments of the blue semi-transparent object. However, even in that case the artifact should be visible when using MRT, too. This artifact was not found in other scenes during further evaluation.

Furthermore, the same 'artifact' that happened with MRT (see Figure 36, middle) also happens for MSTDZ in this scene.

Table 5 shows MSTDZ's computational performance in the Five Primary Planes scene.

| **Method** | **Time per frame (in ms)** **Color gathering** | **Time per frame (in ms)** **Ray-marching** | **Total time** |
|---|---|---|---|
| $MSTDZ_1$ | 0.05 | 8.81 | 8.86 |
| $MSTDZ_2$ | 0.051 | 8.89 | 8.941 |
| $MSTDZ_3$ | 0.055 | 8.96 | 9.015 |

*Table 5.* The MSTDZ method's performance in the Five Primary Planes scene with various amounts of depth zones, up until optimal quality is reached. Lower case numbers denote the amount of depth zones that are used.

In Table 5, we see that the differences in computational performance are extremely minimal. In fact, they are so small that they can be neglected. More interestingly, we can compare the results in these two Tables with the one in Table 3 (DP). This comparison can more easily be made using Table 8 or Figure 39. We can see that for a simple scene like the Five Primary Planes scene, the newly proposed methods perform better, not only during the color gathering phase (which is the main objective), but also during the ray-marching phase.

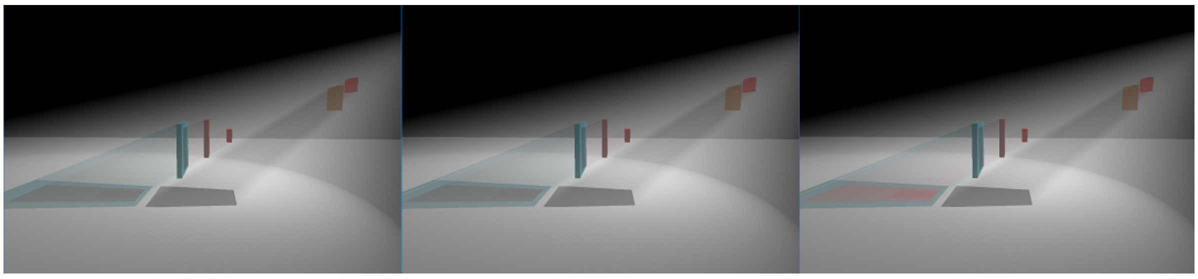


*Figure 36.* The Five Primary Planes scene rendered with MSTDZ. Left: 3 zones. Middle: 2 zones. Right: 1 zone.

The histogram addition to MSTDZ (i.e. Histogram MSTDZ) should theoretically entail a slight increase in computational effort, but when used correctly, it should be able to retain more quality than its basic predecessor, at lower computational costs. Figure 37 shows us the result of Histogram MSTDZ in the Five Primary Planes scene. Three depth zones were distributed to end up at this result, just as with MRT and MSTDZ. This implies that Histogram MSTDZ is not necessarily useful in a simple scene like this, but this is expected. Figure 37 (middle, right) shows that colors are captured in the same (wrong) way as MRT and MSTDZ do when using too few zones.

The expected decrease in computational performance for the color gathering phase is evident from Table 6.

| **Method** | **Time per frame (in ms) Color gathering** | **Time per frame (in ms) Ray-marching** | **Total time** |
|---|---|---|---|
| $Histo_1$ | 0.14 | 8.98 | 9.12 |
| $Histo_2$ | 0.146 | 9.02 | 9.166 |
| $Histo_3$ | 0.155 | 9.10 | 9.255 |

*Table 6.* Histogram MSTDZ's performance in the Five Primary Planes scene. A relatively significant increase in time can be seen during the color gathering phase compared to basic MRT and MSTDZ. Lower case numbers denote the amount of depth zones that are used.

The change of first opaque depth value to last transparent depth value also has its influence on the Histogram MSTDZ method, but in this scene it does not lower the amount of zones any further. This is a logical observation, as Histogram MSTDZ already needs the (for this scene) minimum of three zones to capture all colors correctly. A tighter bound does not lead to any increase in performance in this case.

As expected, Histogram MSTDZ performs slightly slower during the color gathering phase than basic MSTDZ. However, as was mentioned before, in a later part of the evaluation we will see that Histogram MSTDZ can obtain similar results to MSTDZ at much lower amounts of depth zones (and thus lower computational time as well as lower memory consumption).

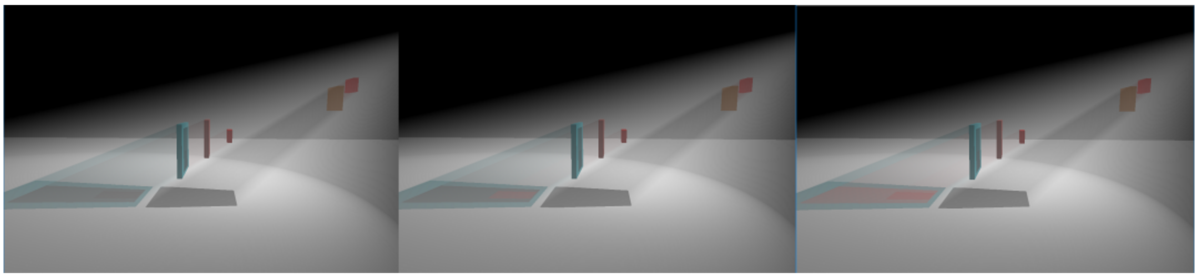


*Figure 37.* The Five Primary Planes scene rendered with Histogram MSTDZ. Left: 3 zones. Middle: 2 zones. Right: 1 zone.

As was stated earlier, Histogram-based MSTDZ with Texel Groups (from now on: Texel Groups) is expected to do well in scenes with little detail. Since areas with little detail (i.e. containing texels with similar histogram distributions) will be stored in the quarter-resolution texture, a lot of calculations will be spared. If a scene is very detailed, most of the texels will have different histogram distributions and will all be stored in the full-resolution texture. Having half- and quarter-resolution textures will then be more of an obstacle than an improvement.

As was mentioned before, Texel groups notation works as follows: TexelGroups$_{i,j,k}$ means that the full-resolution texture has $i$ subsamples per texel, the half-resolution texture has $j$ subsamples per texel and the quarter-resolution texture has $k$ subsamples per texel.

To obtain optimal quality in a scene, Texel Groups should use the same amount of zones (and thus samples) for each of its textures. For example, if Histogram MSTDZ needs 3 zones to render a scene perfectly, Texel Groups needs to be TexelGroups$_{3,3,3}$. Figure 38 (top-left) shows us the result for the Five Primary Planes scene using these values. For reference, Figure 38 (top-middle, top-right) shows the results using TexelGroups$_{1,1,1}$ and TexelGroups$_{2,2,2}$, respectively.

The Five Primary Planes scene is a scene that has many uniformly colored areas and little detail. Therefore, it is expected that the quarter-resolution texture will be most dense and that this one should receive the same amount of zones as Histogram MSTDZ, i.e. 3. Figure 38 (bottom-left) shows what happens if we set the number of depth zones for the most detailed texture (full resolution) to the minimum (i.e. 1). Additionally, Figure 38 (bottom-middle, bottom-right) shows the effects of setting the half-resolution and full-resolution textures' number of depth zones to 1.

From Table 7 we can observe that the method's color gathering timings are fairly constant throughput the various scenarios. Ray-marching sees a little more effect from changing the numbers of depth zones. The addition of five more render passes probably also has its effect on the timing, which contributes to the fact that this method seems to be slower than its predecessor. What must be noted, however, is that the use of Texel Groups does have a good effect on memory consumption. A result like the one in Figure 38 (bottom-left, TexelGroups$_{1,3,3}$) uses a full-resolution texture with only one subsample, which (at 1024x1024 shadow map resolution) consumes 4MB of memory, in contrast to 12MB for a 3-subsample texture. In this case, a total of 7.75MB is used by Texel Groups, as opposed to 12MB when using regular Histogram MSTDZ. At higher amounts of samples this will have an even more significant effect. Op top of this, taking away accuracy at the few locations with detail does not take away much from the visual results. Especially for the naked eye, it is hard to see a difference between the optimal result (TexelGroups$_{3,3,3}$) and this approximation.

| **Method** | **Time per frame (in ms) Color gathering** | **Time per frame (in ms) Ray-marching** | **Total time** |
|---|---|---|---|
| $TexelGroups_{1,1,1}$ | 0.19 | 9.76 | 9.95 |
| $TexelGroups_{2,2,2}$ | 0.21 | 9.84 | 10.05 |
| $TexelGroups_{3,3,3}$ | 0.22 | 9.95 | 10.17 |
| $TexelGroups_{1,3,3}$ | 0.21 | 9.92 | 10.13 |
| $TexelGroups_{3,1,3}$ | 0.21 | 9.94 | 10.15 |
| $TexelGroups_{3,3,1}$ | 0.21 | 9.75 | 9.96 |

*Table 7.* Texel Groups' performance in the Five Primary Planes scene.

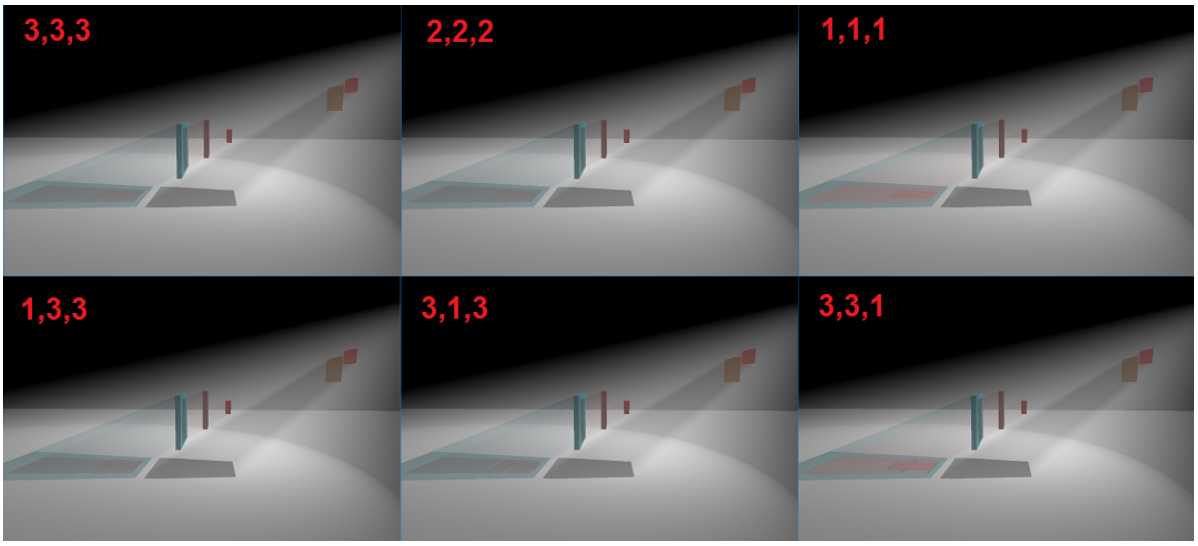


*Figure 38.* The Five Primary Planes scene rendered with Histogram MSTDZ with Texel Groups. Numbers in the pictures represent the amounts of zones used for the three different textures (full/half/quarter resolution, resp.).

## Conclusions on Basic Performance

| Method | Time per frame (in ms) Color gathering | Time per frame (in ms) Ray-marching | Total time |
|---|---|---|---|
| $DP_1$ | 0.032 | 9.25 | 9.282 |
| $DP_2$ | 0.088 | 9.39 | 9.478 |
| $DP_3$ | 0.210 | 9.45 | 9.66 |
| $MRT_1$ | 0.045 | 8.72 | 8.765 |
| $MRT_2$ | 0.049 | 8.88 | 8.929 |
| $MRT_3$ | 0.057 | 8.95 | 9.007 |
| $MSTDZ_1$ | 0.05 | 8.81 | 8.86 |
| $MSTDZ_2$ | 0.051 | 8.89 | 8.941 |
| $MSTDZ_3$ | 0.055 | 8.96 | 9.015 |
| $Histo_1$ | 0.14 | 8.98 | 9.12 |
| $Histo_2$ | 0.146 | 9.02 | 9.166 |
| $Histo_3$ | 0.155 | 9.10 | 9.255 |
| $TexelGroups_{1,1,1}$ | 0.19 | 9.76 | 9.95 |
| $TexelGroups_{2,2,2}$ | 0.21 | 9.84 | 10.05 |
| $TexelGroups_{3,3,3}$ | 0.22 | 9.95 | 10.17 |
| $TexelGroups_{1,3,3}$ | 0.21 | 9.92 | 10.13 |
| $TexelGroups_{3,1,3}$ | 0.21 | 9.94 | 10.15 |
| $TexelGroups_{3,3,1}$ | 0.21 | 9.75 | 9.96 |

*Table 8.* Performance of all methods in the Five Primary Planes scene with various amounts of peels up until optimal quality is reached. Lower case numbers denote the amount of depth peels or depth zones that are used.

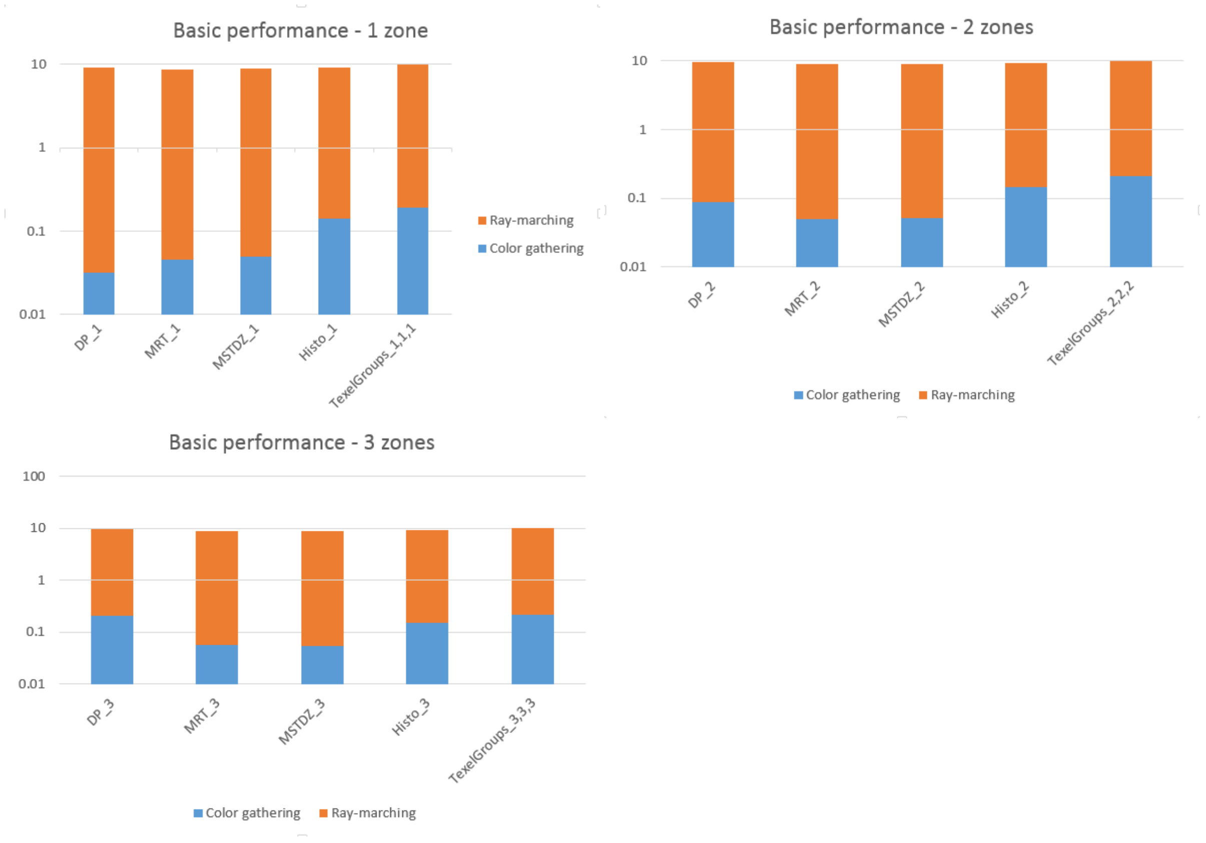


*Figure 39.* Logarithmic comparison of all methods using 1, 2 or 3 depth zones.

We can conclude that the optimal result for a simple scene like the Five Primary Planes scene is obtained faster by MRT and MSTDZ than by Depth Peeling. This goes both for the color gathering phase as well as for the ray-marching phase. Table 8 looks to promise that when Depth Peeling needs more than two or three peels, the time taken for color gathering will rise drastically. It is expected that our proposed methods will perform better in these cases, and in simple scenes overall. Even if Depth Peeling would be faster in a scenario, Depth Peeling always has to go through all fragments, using a theoretically infinite amount of peels. Simply leaving out peels would lead to very erroneous results. Our proposed methods do have this approximation capability: by using lower amounts of zones than would be needed for an optimal result, we can approximate optimal results, while at the same time obtaining these results in a faster manner.

Secondly, the addition of the first and last transparent depth values can make a very large difference for certain distributions of semi-transparent geometry, such as the one in the Five Primary Planes scene.

Last of all, while Texel Groups' computational performance does not yet give reason for optimism, its memory-saving capabilities can be interesting for applications where memory matters (e.g. video game developers who want to offer their game at lower minimum requirements). Using lower amounts of depth zones for the full-resolution texture in scenes without a lot of detail, will lead to results that are slightly worse than, but comparable to Histo MSTDZ, while using lower amounts of texture memory.

## Depth Zone Restrictions

As was stated before, some restrictions were applied to the depth zone partitioning that is used by the MRT and MSTDZ methods. Most notably, the first and last transparent depth values were added to put lower and upper bounds on the depth values between which zones should be distributed. However, during an earlier phase, the last transparent depth value was not yet used, and instead, the first opaque depth value was utilized. It is interesting to shortly look at the impact that this change from opaque depth value restriction to last transparent depth value restriction had.

In Figure 40 (top-left), we can once again see the result of MRT with 3 zones while using the last transparent depth value, which is what we already saw in Figure 34 (left). In this scene, the first opaque depth value is relatively far away, so we can expect a major shift in the results if we interchange the more accurate last transparent depth value for the first opaque depth value. Figure 40 (top-middle) clearly shows that this expectation was not unjustified, as the depth zones are obviously wrongly distributed, leading to displeasing results.

Figure 40 (top-right) shows us the result that we get by distributing 8 zones over the span between the first transparent and the first opaque depth value. Not even with MRT's maximum amount of (easily obtainable) depth zones are we able to flawlessly capture all semi-transparent colors in this scene. This result distinctly shows why MSTDZ was designed in the first place: sometimes it is necessary to be able to partition a scene into more than 8 depth zones.

Using three depth zones, MSTDZ runs into exactly the same kind of problem, which can be concluded from Figure 40 (bottom-left). And even with 8 depth zones (Figure 40, bottom-middle) there are still small artifacts present in the scene. Figure 40 (bottom-right), however, shows that MSTDZ indeed possesses the ability to overcome MRT's shortcoming: by using 9 depth zones, we are finally able to perfectly capture all colors and render the optimal result as is normally given by Depth Peeling with 3 zones.

Noteworthy are the increases in computational cost that come with the usage of a higher number of depth zones, see Table 9.

| **Method** | **Time per frame (in ms) Color gathering** | **Time per frame (in ms) Ray-marching** | **Total time** |
|---|---|---|---|
| $MRT_8$ | 0.111 | 9.39 | 9.501 |
| $MSTDZ_8$ | 0.061 | 9.34 | 9.401 |
| $MSTDZ_9$ | 0.097 | 10.10 | 10.197 |

*Table 9.* The MRT and MSTDZ method's performances in the Five Primary Planes scene when using the first opaque depth value instead of the last transparent depth value. For reference, $MSTDZ_8$ is also shown, as to compare it with the performance of $MRT_8$. Lower case numbers denote the amount of depth zones that are used.

Now, the interchanging of the previously used first opaque depth value by the last transparent depth value did not increase performance by itself. However, the possibility of using a significant smaller amount of zones does lead to a computational speed-up. From this short interlude, we can conclude that the addition of an even tighter upper bound, has brought only good to the accuracy and speed of the proposed methods.

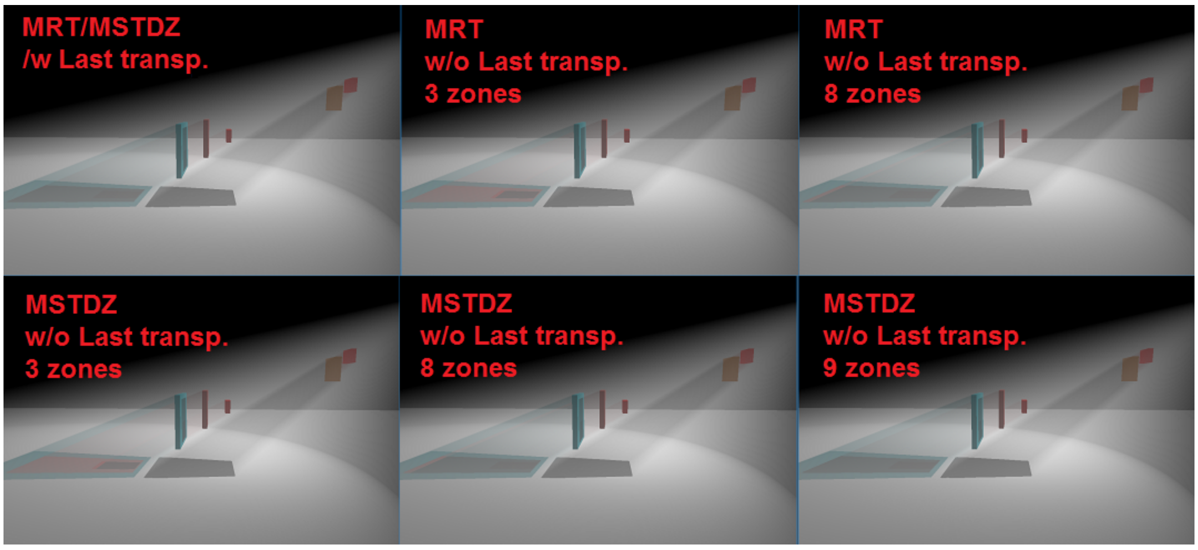


*Figure 40.* The Five Primary Planes scene rendered with the MRT and MSTDZ method, showcasing that the last transparency restriction is key in increasing performance for certain scenes.

## MRT versus MSTDZ

In this part of the evaluation, a short look will be given into the reason why MRT was abandoned early during this thesis: its limitation of only being able to use 8 render targets at the same time without having to resort to utilizing more render passes. The forthcoming limit to the amount of zones is insurmountable and an example will be given as to why this is the case. To this end, the Nine Planes scene is used, which consists of 9 semi-transparent objects in line. This setup is chosen as at least 9 depth zones would be needed to perfectly capture the colors of these objects.

The expectation is that MRT is unable to capture this scene optimally and this is indeed the fact (see Figure 41, middle). The corresponding artifacts are outlined in red. Due to the uniform spread of the planes across the scene, MSTDZ is, expectedly, able to perfectly capture all 9 semi-transparent objects with exactly 9 depth zones, which is observable in Figure 41 (right).

Figure 41 (left) shows the reference image, which is obtained by using Depth Peeling with 9 peels. It is interesting to look at corresponding timings, which are found in Table 10. Table 10 shows that MRT with 8 zones is relatively much slower than MSTDZ with the same amount of zones, both in terms of color gathering as well as ray-marching. Now, this approximation using 8 zones is visually good enough. However, if we look at the amount that we needed for optimal quality, 9 zones, MSTDZ becomes slower in terms of ray-marching (although still much faster during color gathering, and still real-time). This is to be expected as for all semi-transparent texels in the shadow map an extra computational step needs to be taken during ray-marching.

Furthermore, Depth Peeling with the same amount of peels as MSTDZ is significantly faster during ray-marching (which is as expected, since Depth Peeling will never go through more computational steps than necessary), but the color gathering phase is extremely slow (close to 1 ms). This shows that MSTDZ is indeed more efficient at gathering colors than Depth Peeling, which is probably caused by the higher amount of render passes and the overhead that comes with this. However, improvements need to be made to the ray-marching phase to make it worth using MSTDZ over Depth Peeling when going for the same quality.

| **Method** | **Time per frame (in ms)** | **Time per frame (in ms)** | **Total** |
|---|---|---|---|
| | **Color gathering** | **Ray-marching** | **time** |
| $MRT_8$ | 0.287 | 11.21 | 11.497 |
| $MSTDZ_8$ | 0.08 | 10.09 | 10.17 |
| $MSTDZ_9$ | 0.156 | 13.94 | 14.096 |
| $DP_9$ | 0.71 | 9.15 | 9.86 |

*Table 10.* Computational performance of MRT, MSTDZ and Depth Peeling in the Nine Planes scene.

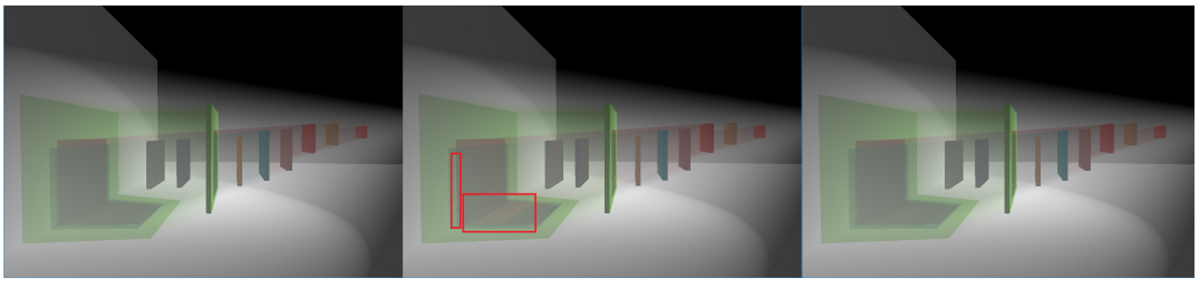


*Figure 41.* The Nine Planes scene. Left: for reference rendered using Depth Peeling, taking 9 peels. Middle: MRT can only easily go up to 8 depth zones, falling short in being able to compute the correct result. Right: MSTDZ can partition more zones than MRT without additional effort and is able to compute the correct result at 9 depth zones.

### MRT and MSTDZ versus Histogram MSTDZ

Since the implementation of the last transparent depth value optimization, MSTDZ has become quite good at capturing where semi-transparent objects are. However, if semi-transparent objects are divided over a scene in clusters that are found both at low and high depth values, both MRT and MSTDZ should fail to capture these details and Histogram-based MSTDZ should outperform MSTDZ, at least in terms of quality. This section is devoted to evaluating the differences in performance between MRT, basic MSTDZ and its histogram extension. For this purpose, the Far-away Plane scene is used. This scene challenges MRT, as more than its easily used 8 layers are required. It also challenges both MRT and MSTDZ for the reason that the distribution of semi-transparent objects makes uniform partitioning fail.

Figure 42 (top) shows us the reference result, created by using Depth Peeling with 6 peels. If we use 6 depth zones with MRT, major artifacts arise (see Figure 42, second row, left). The artifact on the right shows us that some fragments are missed entirely using said distribution. Using the maximum (8) number of zones for MRT (without having to resort to more render passes), we obtain the result in Figure 42 (second row, right). Notice that the artifact on the right is now rectified, but the artifact within the light beams on the left remains, albeit smaller.

In Figure 42 (third row) we see that, as is expected, MSTDZ also fails to capture the scene properly using only 6 or 8 zones, respectively. The artifacts are, expectedly, the same again. Using 9 zones we are able to capture the scene perfectly with regular MSTDZ, as we can see in Figure 42 (bottom row, left).

Using Histogram-based MSTDZ, we only need 6 zones to perfectly capture all zones (see Figure 42, bottom row, right). The method finds a way to distribute more zones in the right cluster of zones and distributes only one zone at the left, yellow object.

We will now look at Table 11 to see the performances of the different methods. Once again, MRT seems to be slower than MSTDZ at equal depth zone numbers. Histogram MSTDZ performs worse than its predecessor in terms of of color gathering, but to reach the same amount of quality, MSTDZ has to sacrifice a large amount of time ( 1.6ms) in ray-marching, where Histogram MSTDZ needs 9.52ms versus regular MSTDZ's 11.1ms. Furthermore, Histogram MSTDZ uses fewer zones, and thus less memory, to get to the same result.

Finally, Depth Peeling is slower than any of the methods in terms of color gathering, and, interestingly, also performs almost equally in terms of ray-marching. This implies that many of the shadow map texels in this scene are filled, leading to almost equal amounts of computations during the ray-marching phase for both Depth Peeling and Histogram MSTDZ. The Far-away Plane scene is therefore perfect for using Histogram MSTDZ, as it is faster or equally fast in both rendering phases.

| **Method** | **Time per frame (in ms)**<br>**Color gathering** | **Time per frame (in ms)**<br>**Ray-marching** | **Total**<br>**time** |
|---|---|---|---|
| $MRT_6$ | 0.122 | 9.64 | 9.762 |
| $MRT_8$ | 0.160 | 10.32 | 10.48 |
| $MSTDZ_6$ | 0.07 | 9.59 | 9.66 |
| $MSTDZ_8$ | 0.078 | 9.79 | 9.868 |
| $MSTDZ_9$ | 0.117 | 11.10 | 11.217 |
| $HistoMSTDZ_6$ | 0.325 | 9.52 | 9.845 |
| $DP_6$ | 0.45 | 9.51 | 9.96 |

*Table 11.* Computational performance of MRT, MSTDZ, Histogram MSTDZ and Depth Peeling in the Far-away Plane scene.

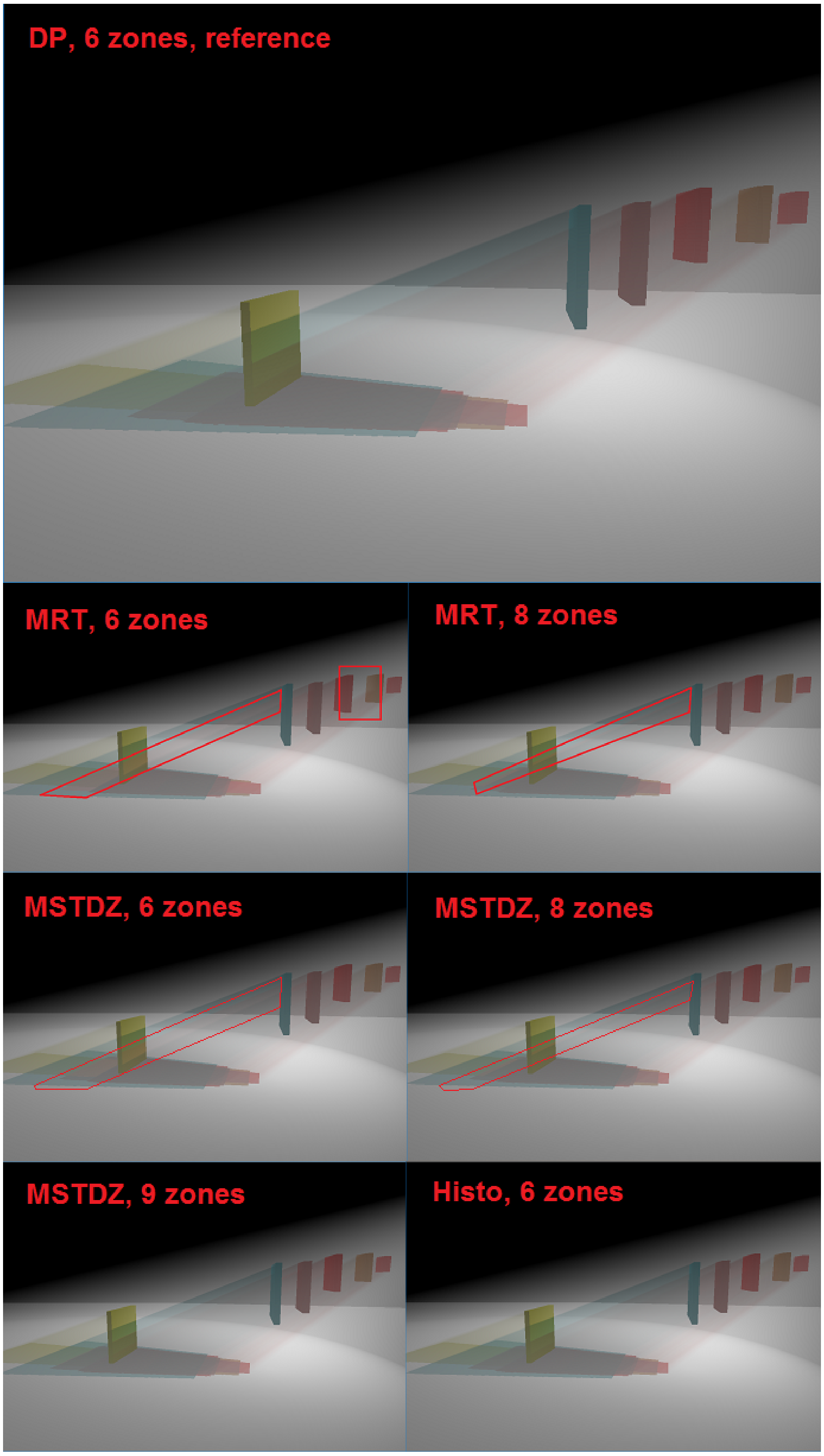


*Figure 42.* The Far-away Plane scene rendered using various methods. The reference image can be seen at the top. All other results are described within the image.

Another noteworthy property of Histogram MSTDZ is that one should try to avoid using too many depth zones at the same time. A small bias is added to the first and last transparent depth values to make sure that fragments are captured properly. Now, if zones are distributed too closely to each other, this bias might lead to fragments being skipped or being written multiple times to different texels in different shadow maps. This, in turn, leads to various kinds of artifacts. Examples of such artifacts are shown in Figures 43 and 44, in which 14 and 32 depth zones are used, respectively. These issues could also happen if too many regular MSTDZ zones are used, but it is more likely to happen with the Histogram addition, as zones are often even more closely distributed thanks to the Histogram pass. No fix to this issue was found yet.

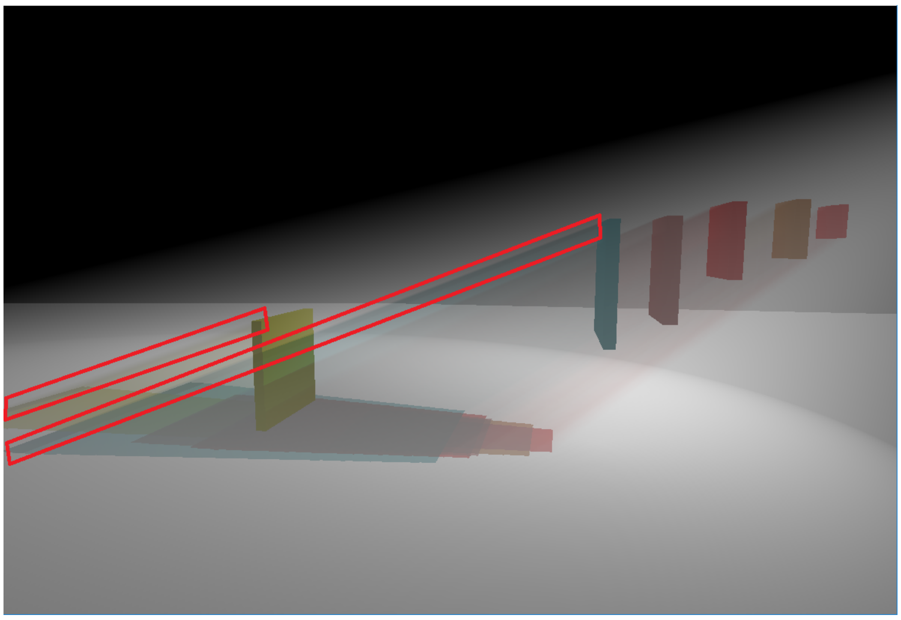


*Figure 43*. The Far-away Plane scene rendered using Histogram MSTDZ, 14 zones. Very obvious artifacts arise.

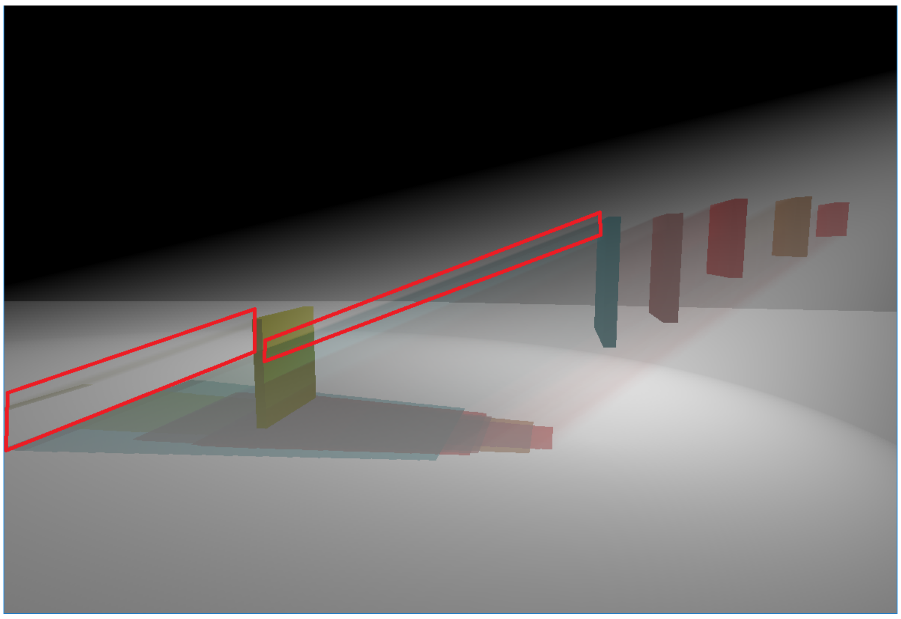


*Figure 44*. The Far-away Plane scene rendered using Histogram MSTDZ, 32 zones. Even more artifacts arise than in Figure 67.

### Comparing Depth Peeling to MSTDZ, Histogram MSTDZ and Texel Groups

While evaluating the methods' basic workings, we have been assuming that certain methods should always reach optimal quality. However, this is most definitely not the case. In fact, as was mentioned multiple times before, the entire idea behind the proposed methods is to approximate the result given by Depth Peeling, as to obtain a more efficient method. In this part we will evaluate a number of scenes by rendering them at optimal quality using Depth Peeling, and then rendering them with our proposed methods, attempting to approximate the optimal result with differing amounts of depth zones, attempting to achieve good results in a shorter amount of time. First of all, we will use the Yellow Planes scene, which is a relatively simple scene, for which Depth Peeling needs 6 peels to gain the optimal result. The reference image is incorporated into all other result figures. We will evaluate the scene for a number of depth zones between 3 and 9. We start with 3 to show what happens if incorrect amounts of layers are used and go all the way up to 9, which for all methods (finally) gives the reference result.

Figure 47 shows the results obtained with MSTDZ using 3 to 9 depth zones, respectively. What is easily observable is that the result images are of varying quality. Two of these are interesting to evaluate in more detail. For MSTDZ with 9 zones, a major artifact arises, as the entire top side of the flat yellow plane is missed and no colored scattering effects are drawn anymore. This is observable on the left side of the image. As can be seen in Table 12, increasing from 8 to 9 depth zones does not only spawn the observable artifact, but it also comes with a drastic computational performance decrease. It looks like too many zones might have been distributed at the top of the stretched plane. In that case, first and last transparent depth values are extremely close to each other. In turn, the bias might mess up. What we can conclude from this is that it might be necessary to add a function in the future that dynamically determines bias based on how close zones are distributed inside each shadow map texel. Nevertheless, color gathering is always done faster than Depth Peeling does it, and ray-marching performance stays close to the reference's.
Additionally, MSTDZ with 8 zones seems to come closest to the reference image. This is confirmed by the PSNR values found in Table 12. These values were added as it is hard to distinguish quality between the different images with the naked eye. At the same time the PSNR values quantify the measurement of quality, so we can draw conclusions from them. Table 12 also shows that the quality rises with each depth zone that is added.

Histogram MSTDZ is expected to not necessarily outperform MSTDZ in terms of quality in this scene, as all semi-transparent objects are relatively close to each other. Figure 48 shows the result obtained with Histogram MSTDZ using 3 to 9 depth zones, respectively. Once again, quality varies between these images. Table 12 shows that, once again, quality rises with each zone that is added. Histogram MSTDZ with 8 zones has the highest PSNR value (see Table 12), which implies that it has the highest quality compared to the reference image, which indeed looks to be the case when looking at the result image. It also seems to outperform regular MSTDZ in this scene, at least in terms of quality.

In terms of computational performance, Histogram MSTDZ using 8 zones is also best. We see another drastic drop in performance between 8 and 9 zones, although this time no serious artifacts occur. However, none of the timings outperform Depth Peeling. By far the largest amount of the Histogram method's speed is determined by the last render pass, in which the calculation of the distribution of zones is done for each texel. Specifically, 85 to 99% is spent on this render pass (e.g. 0.45 of 0.509ms for $Histo_3$ and 2.52 of 2.59ms for $Histo_9$). It seems like calculating the distribution of zones for each texel might be heavy to such an amount that Histogram will not be able to outperform Depth Peeling, unless ray-marching performance is increased in any way. This once again suggests that future work should be aiming towards said improvements. On top of this, using a lower resolution for the histogram texture and/or by only computing the partitioning once every few frames might also improve its performance.

The Yellow Planes scene is a scene in which Texel Groups could prove to be useful, since large uniform areas of semi-transparent geometry are found in this scene. This implies that the quarter-resolution texture will play a prominent role, which is good for performance in terms of time and memory. Figure 49 shows the result of Texel Groups in the Yellow Planes scene, using (3,3,3) to (9,9,9) depth zones. Texel Groups using (8,8,8) zones is more deeply evaluated below.

In terms of computational time for color gathering, we see that Texel Groups outperforms Histogram MSTDZ, although this gain is compensated by the slightly worse performance during ray-marching. Once more, Texel Groups should not be used for its computational performance, but rather for the fact that it offers a possibility of consuming less memory. Since $\text{Histo}_8$ was the method with best PSNR results, we will take a deeper look at $\text{TexelGroups}_{8,8,8}$ and some of its derivatives (found in Figure 50. For example, we can see that the results of $\text{TexelGroups}_{1,1,8}$ and $\text{TexelGroups}_{1,8,8}$ both look very similar. Their PSNR values are not bad either, especially for $\text{TexelGroups}_{1,8,8}$. This implies that not a lot of detail is found in this scene, and that we can take away the 8-subsample full resolution texture (memory cost: 32MB) and replace it by a 1-subsample full resolution texture (memory cost: 4MB), saving a good 28MB of memory. Furthermore, during ray-marching only 1 calculation step per texel will have to be made, instead of 8 per texel. By sacrificing quality, we are able to save time and memory.

Another observation that can be made from the PSNR values in Table 12, is the fact that PSNR values see a large shift when low depth zone values are used for the quarter-resolution texture. This again is implied by the high amount of texels with similar histogram distributions. The quarter-resolution texture will therefore contain a lot of information about the scene. Chipping away at this information will have larger consequences than doing the same for the more sparse full-resolution or half-resolution textures.

| **Method** | **Time per frame (ms) Color gathering** | **Time per frame (ms) Ray-marching** | **Total time** | **PSNR** | **Figure #** |
|---|---|---|---|---|---|
| $DP_6$ | 0.492 | 8.83 | 9.322 | $\infty$ | 31 |
| $MSTDZ_3$ | 0.066 | 8.72 | 8.786 | 38.78 | 47 |
| $MSTDZ_4$ | 0.079 | 8.78 | 8.859 | 43.11 | 47 |
| $MSTDZ_5$ | 0.081 | 9.03 | 9.111 | 44.53 | 47 |
| $MSTDZ_6$ | 0.082 | 9.16 | 9.242 | 47.37 | 47 |
| $MSTDZ_7$ | 0.084 | 9.30 | 9.384 | 49.85 | 47 |
| $MSTDZ_8$ | 0.084 | 9.46 | 9.544 | 55.59 | 47 |
| $MSTDZ_9$ | 0.146 | 11.55 | 11.696 | 37.56 | 47 |
| $HistoMSTDZ_3$ | 0.509 | 8.83 | 9.339 | 40.71 | 48 |
| $HistoMSTDZ_4$ | 0.514 | 8.84 | 9.354 | 43.97 | 48 |
| $HistoMSTDZ_5$ | 0.580 | 9.04 | 9.62 | 44.65 | 48 |
| $HistoMSTDZ_6$ | 0.641 | 9.30 | 9.941 | 47.12 | 48 |
| $HistoMSTDZ_7$ | 0.710 | 9.45 | 10.16 | 50.48 | 48 |
| $HistoMSTDZ_8$ | 0.736 | 9.59 | 10.326 | 69.34 | 48 |
| $HistoMSTDZ_9$ | 2.632 | 11.52 | 14.152 | 56.44 | 48 |
| $TexelGroups_{3,3,3}$ | 0.46 | 9.90 | 10.36 | 39.13 | 49 |
| $TexelGroups_{4,4,4}$ | 0.49 | 10.05 | 10.54 | 40.87 | 49 |
| $TexelGroups_{5,5,5}$ | 0.52 | 10.44 | 10.96 | 44.11 | 49 |
| $TexelGroups_{6,6,6}$ | 0.56 | 10.73 | 11.29 | 46.39 | 49 |
| $TexelGroups_{7,7,7}$ | 0.60 | 11.03 | 11.63 | 51.03 | 49 |
| $TexelGroups_{9,9,9}$ | 1.27 | 11.59 | 12.86 | 57.05 | 49 |
| $TexelGroups_{8,8,8}$ | 0.62 | 11.27 | 11.89 | 60.13 | 50 |
| $TexelGroups_{1,1,8}$ | 0.49 | 10.61 | 11.10 | 41.66 | 50 |
| $TexelGroups_{1,8,8}$ | 0.53 | 11.03 | 11.56 | 47.14 | 50 |
| $TexelGroups_{8,8,1}$ | 0.58 | 9.96 | 10.54 | 32.55 | 50 |

*Table 12.* Computational performance of MSTDZ, Histogram MSTDZ and TexelGroups as compared to Depth Peeling in a simple scene, the Yellow Planes scene. Peak signal-to-noise ratio values are as compared to the reference image created with Depth Peeling.

Above, in Table 12, an overview of the peak signal-to-noise ratio (PSNR) values for this scene is given. The reference image is the one rendered using Depth Peeling. The PSNR values are calculated using the following function:

$$PSNR = 10\log_{10}(\frac{peakval^2}{MSE}),$$

where $peakval$ is the maximum value of an image's pixel (i.e. 255) and $MSE$ is simply the mean square error between the reference image and the evaluated image. Since the MSE between the reference image and itself is 0, the PSNR for this comparison is $\infty$. This formula implies that two images that are more equal will have a higher PSNR value. PSNR values should not be taken as a deciding factor of quality, but they serve a directory purpose, which is how they will be used them during this evaluation.

The results from Table 12 are visualized in Figures 45 and 46.

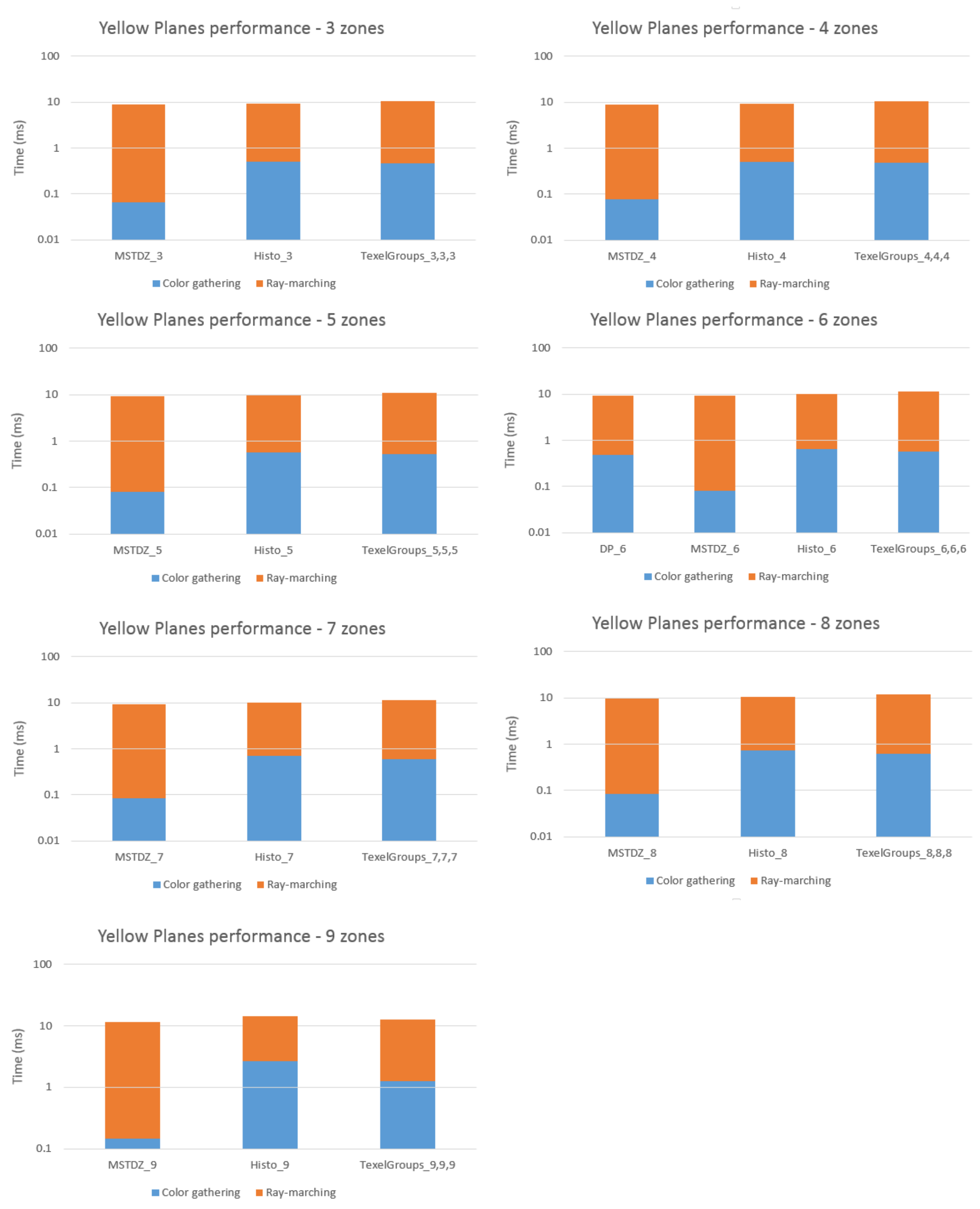
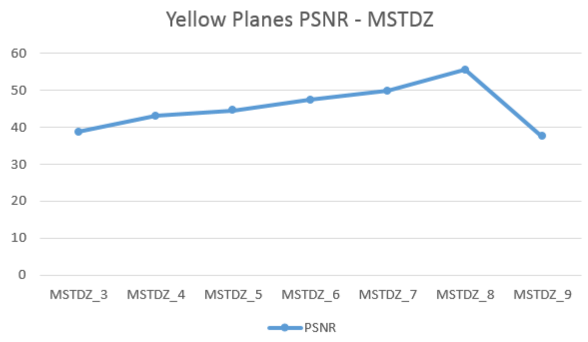
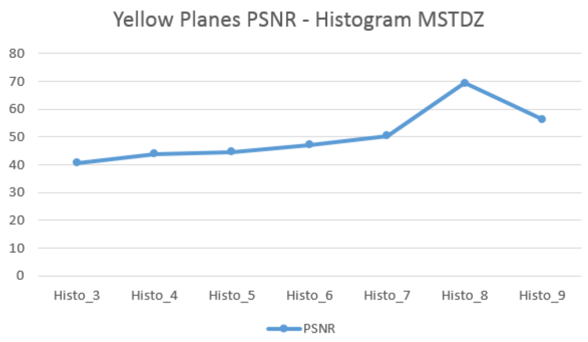


*Figure 45.* Logarithmic comparison of MSTDZ, Histo MSTDZ and Texel Groups performance in the Yellow Planes scene. Depth peeling is shown in the graph for 6 depth zones, as this was the amount that was required to obtain the reference image.





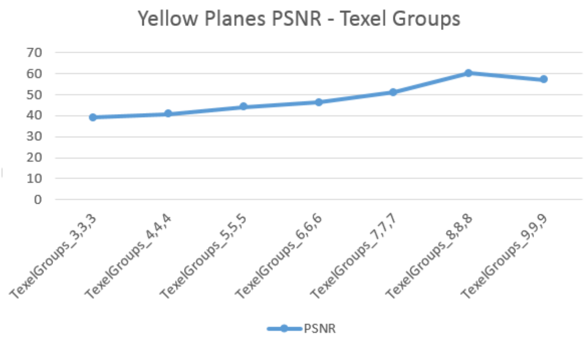


*Figure 46.* Comparison of PSNR values for MSTDZ, Histo MSTDZ and Texel Groups in the Yellow Planes scene.

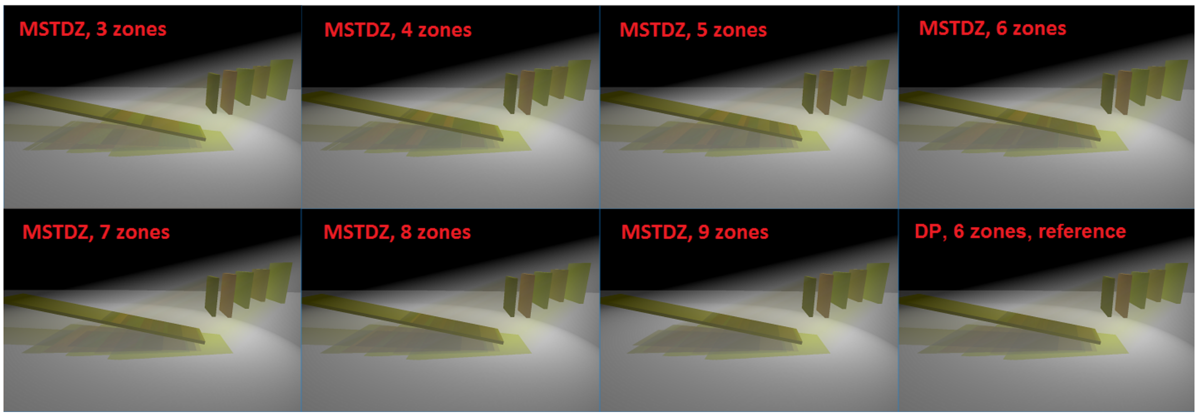


*Figure 47.* The Yellow Planes scene rendered using MSTDZ with various amounts of zones. The reference image can be found at the bottom-right.

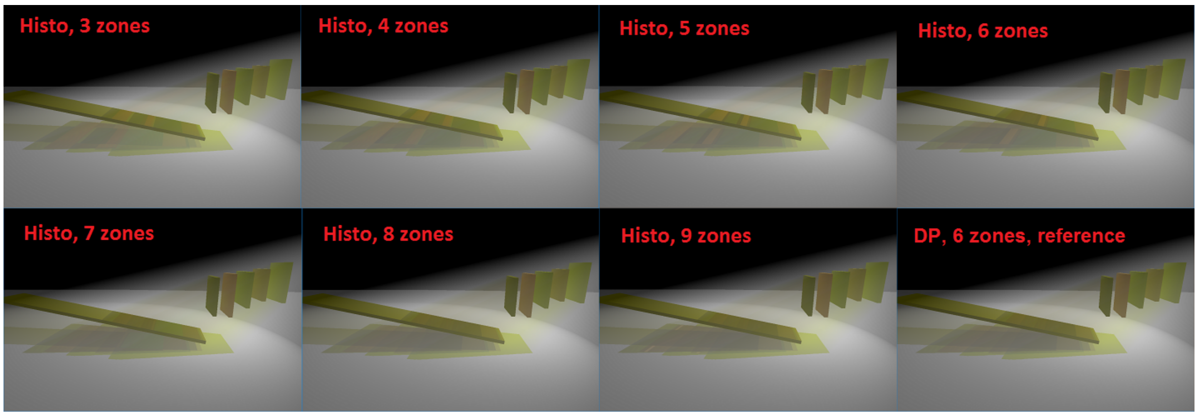


*Figure 48.* The Yellow Planes scene rendered using Histogram MSTDZ with various amounts of zones. The reference image can be found at the bottom-right.

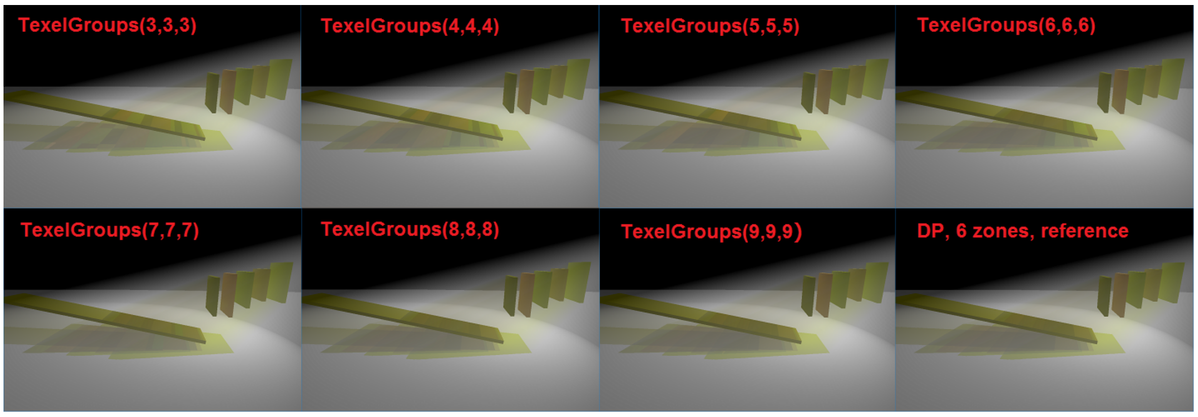


*Figure 49.* The Yellow Planes scene rendered using Texel Groups with various amounts of zones. The reference image can be found at the bottom-right.

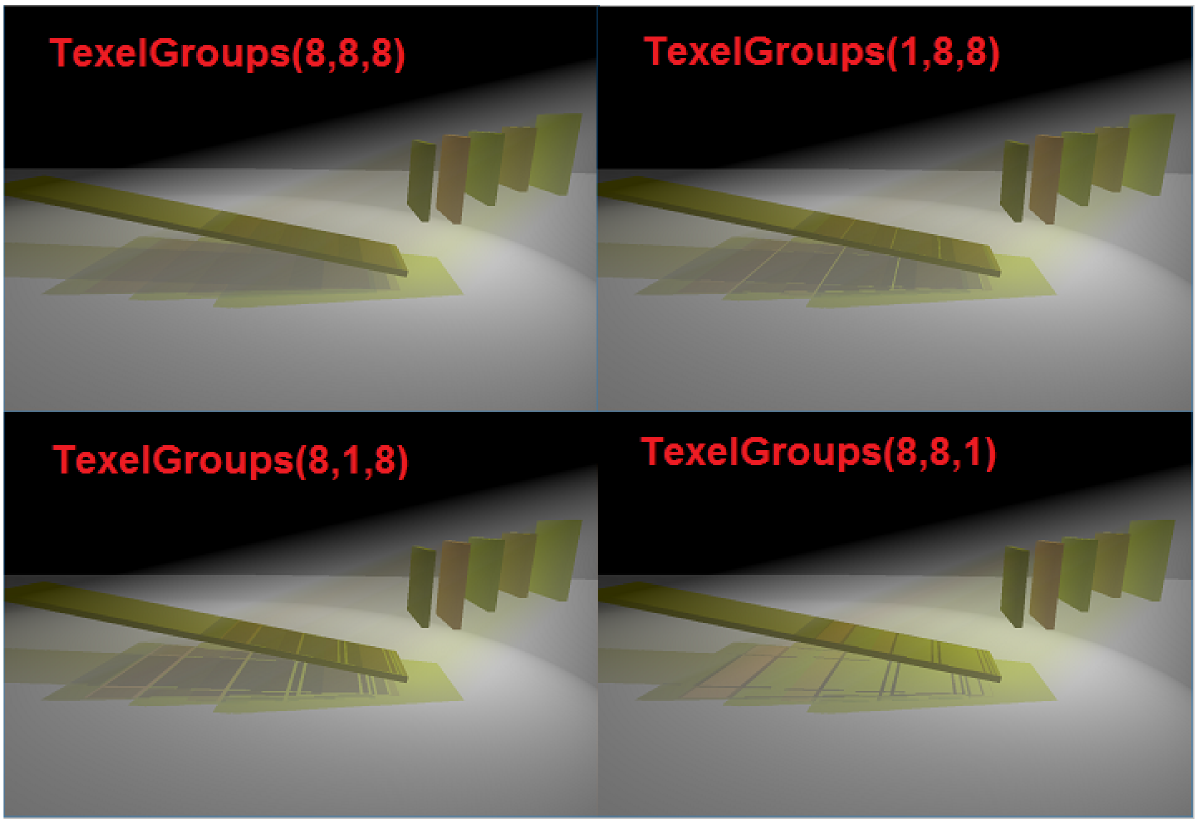


*Figure 50.* The Yellow Planes scene rendered using variations of Texel Groups$_{8,8,8}$.

Besides evaluating the methods in a simple scene, we do an evaluation in a much more complex scene, one in which Depth Peeling is expected to perform poorly: the Foliage scene. We want to evaluate whether MSTDZ or Histogram MSTDZ could outperform Depth Peeling in such a scenario, either by approximation or by going for optimal results. It is expected that the Foliage scene lends itself well for rough approximations, meaning that where Depth Peeling would need a high number of peels for good quality (and obtaining very poor quality for lower amounts of peels), all of our proposed methods should give well-looking results with only a small number of depth zones. We will evaluate with amounts of depth zones from 3 up to 12. It is our expectation that 3 zones would already lead to plausible results for our proposed methods, in which each zone captures the colors of a single tree each. We evaluate the methods up to 12 depth zones to see whether there is much visual difference in quality if we use more depth zones. To obtain optimal quality in the Foliage scene, Depth Peeling needs to peel a maximum of 12 times. The reference result image has been incorporated into all other result images. It is important to notice that the image found on the front page of this thesis is of comparable complexity and results are similar.

Figure 53 shows MSTDZ's results using 3 to 12 zones, respectively. We can see that MSTDZ's performance up until 8 zones is almost constant, after which the times per frame increase very slightly. Depth Peeling does really poorly in this scene, as the time to peel off all 12 layers to gather all color information is incredibly high (11.14ms). Even when combining the color gathering and ray-marching times, Depth Peeling is still outperformed by all of the different results from MSTDZ. At the same time, MSTDZ with 12 zones does not necessarily perform best: more zones are not always a guarantee for more quality. If we look at Table 13, we can see that MSTDZ tends to perform best at 7 zones or around that value. This is very promising, as $\text{MSTDZ}_6$ performs extremely well compared to Depth Peeling and can even be considered real-time. $\text{MSTDZ}_7$ is also extremely close to being real-time.

Histogram MSTDZ's results can be viewed in Figure 54, in which 3 to 12 zones are used, respectively. We once again see that the histogram addition adds a significant portion of time. However, up until 8 depth zones the algorithm remains competitive with Depth Peeling in terms of computational performance. This is very interesting indeed, as the PSNR values in Table 13 also show the best results are gained when 6, 7 or 8 depth zones are used. Because of the uniform spread of the semi-transparent geometry (the leaves), Histogram MSTDZ was not expected to perform better computationally than MSTDZ in this scene, although its PSNR values do imply that Histogram MSTDZ performs better than regular MSTDZ.
A significant increase in computational effort occurs when using more than 8 depth zones and the PSNR values drop significantly (see Table 13, visualized in Figures 51 and 52). What we can conclude by looking at the shadows, is that fewer leaves are being drawn when using 9 or more depth zones. We once again suspect this has something to do with the bias values. However, it also looks like 8 seems to be a border value, as this same kind of behavior happened with both MSTDZ and Histogram MSTDZ in the Yellow Planes scene (see Table 12). One suspected cause of the problem is that multisample textures might take longer to 'texel fetch' subsamples from as soon as more than 8 subsamples are used. However, no documentation about this specific error could be found. The error might also be caused by a coding error, but significant debugging efforts have not helped to get rid of the issue.

The results for Texel Groups are shown in Figure 55. As expected, Texel Groups does not do well in a very complex scene like the Foliage scene. Due to its poor performance, we have chosen to evaluate only the edge cases and the best-performing case. With lots of detail, there are very few neighboring texels with similar histogram distributions. From Table 13 we can observe that computational performance for both color gathering and ray-marching is poor. Regular Histogram MSTDZ performs better in all cases. Furthermore, we can observe that the PSNR values are relatively low. The best result is obtained by using $\text{TexelGroups}_{7,7,7}$. Interestingly, $\text{TexelGroups}_{7,1,1}$ and $\text{TexelGroups}_{7,7,1}$ have similar (or even identical) PSNR values and indeed also look to be similar (see Figure 56). This implies that removing accuracy from less-detailed areas has almost no effect. In turn, we can conclude that almost all of the color information is stored within the full-resolution texture. The scene is indeed very detailed and similar histogram distributions are indeed unlikely to be found. Not even the memory-saving property of Texel Groups does work in this scene. The full resolution texture seemingly needs a high amount of depth zones, so we can not save memory by lowering this amount. The half- and quarter-resolution textures are almost dead-weight in this specific scene.

| Method | Time per frame (in ms) Color gathering | Time per frame (in ms) Ray-marching | Total time | PSNR | Figure # |
|---|---|---|---|---|---|
| $DP_{12}$ | 11.14 | 12.68 | 23.82 | $\infty$ | 32 |
| $MSTDZ_3$ | 2.16 | 12.46 | 14.62 | 31.51 | 53 |
| $MSTDZ_4$ | 2.16 | 12.95 | 15.11 | 31.98 | 53 |
| $MSTDZ_5$ | 2.16 | 13.62 | 15.78 | 33.10 | 53 |
| $MSTDZ_6$ | 2.18 | 14.07 | 16.25 | 33.21 | 53 |
| $MSTDZ_7$ | 2.17 | 14.71 | 16.88 | 33.50 | 53 |
| $MSTDZ_8$ | 2.18 | 15.20 | 17.38 | 33.33 | 53 |
| $MSTDZ_9$ | 2.23 | 16.37 | 18.60 | 30.59 | 53 |
| $MSTDZ_{10}$ | 2.25 | 16.91 | 19.16 | 31.17 | 53 |
| $MSTDZ_{11}$ | 2.27 | 18.39 | 20.66 | 30.87 | 53 |
| $MSTDZ_{12}$ | 2.29 | 18.87 | 21.16 | 31.13 | 53 |
| $HistoMSTDZ_3$ | 5.02 | 12.40 | 17.42 | 35.32 | 54 |
| $HistoMSTDZ_4$ | 5.22 | 13.04 | 18.26 | 36.07 | 54 |
| $HistoMSTDZ_5$ | 5.65 | 13.72 | 19.37 | 36.58 | 54 |
| $HistoMSTDZ_6$ | 6.11 | 14.34 | 20.45 | 37.12 | 54 |
| $HistoMSTDZ_7$ | 6.46 | 14.82 | 21.28 | 37.38 | 54 |
| $HistoMSTDZ_8$ | 6.69 | 15.28 | 21.97 | 37.58 | 54 |
| $HistoMSTDZ_9$ | 15.37 | 17.80 | 33.17 | 32.52 | 54 |
| $HistoMSTDZ_{10}$ | 15.61 | 18.05 | 33.66 | 32.27 | 54 |
| $HistoMSTDZ_{11}$ | 16.10 | 18.65 | 34.75 | 32.24 | 54 |
| $HistoMSTDZ_{12}$ | 17.43 | 19.11 | 36.54 | 32.46 | 54 |
| $TexelGroups_{3,3,3}$ | 6.27 | 15.00 | 21.27 | 34.80 | 55 |
| $TexelGroups_{12,12,12}$ | 17.94 | 24.05 | 41.99 | 32.76 | 55 |
| $TexelGroups_{7,7,7}$ | 7.55 | 18.84 | 26.39 | 32.91 | 56 |
| $TexelGroups_{1,1,7}$ | 7.45 | 18.57 | 26.02 | 29.24 | 56 |
| $TexelGroups_{1,7,7}$ | 5.85 | 14.34 | 20.19 | 29.27 | 56 |
| $TexelGroups_{7,1,1}$ | 5.71 | 13.14 | 18.85 | 32.87 | 56 |
| $TexelGroups_{7,7,1}$ | 8.12 | 18.63 | 26.75 | 32.91 | 56 |

*Table 13.* Computational performance of MSTDZ, Histo MSTDZ and Texel Groups as compared to Depth Peeling in a complex scene, the Foliage scene. Peak signal-to-noise ratio values are as compared to the reference image created with Depth Peeling.

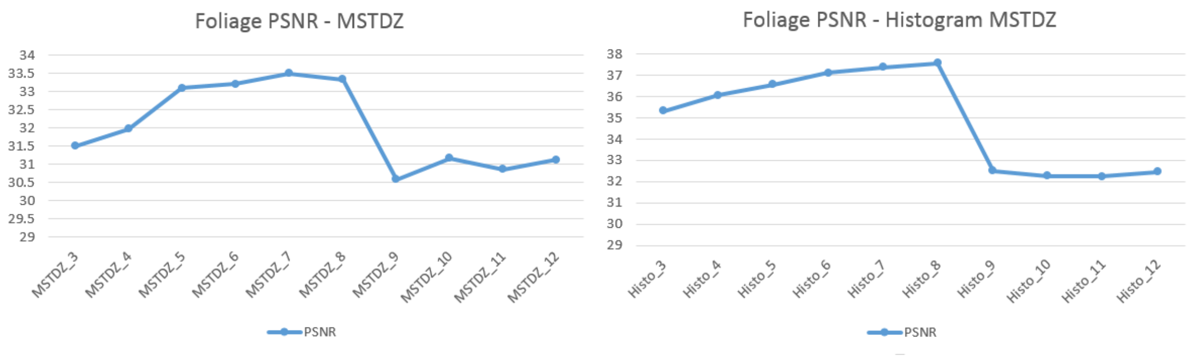


*Figure 51.* Comparison of PSNR values for MSTDZ, Histo MSTDZ and Texel Groups in the Foliage scene.

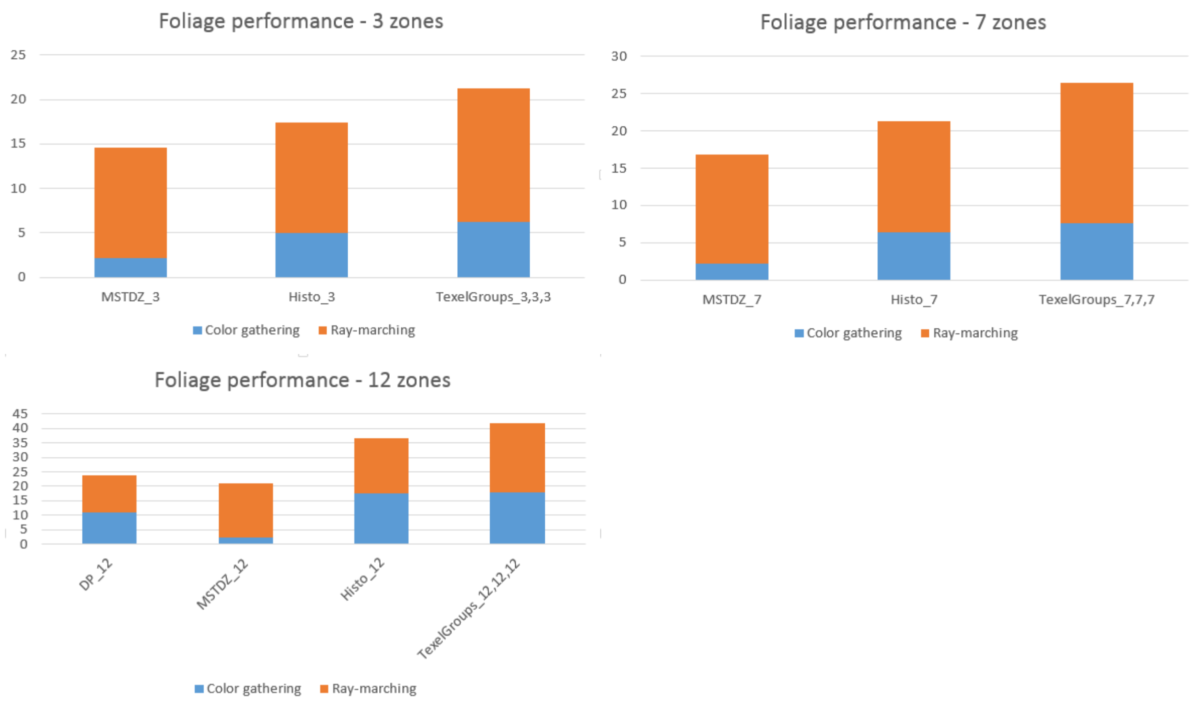


*Figure 52.* Comparison of MSTDZ, Histo MSTDZ and Texel Groups performance in the Foliage scene.

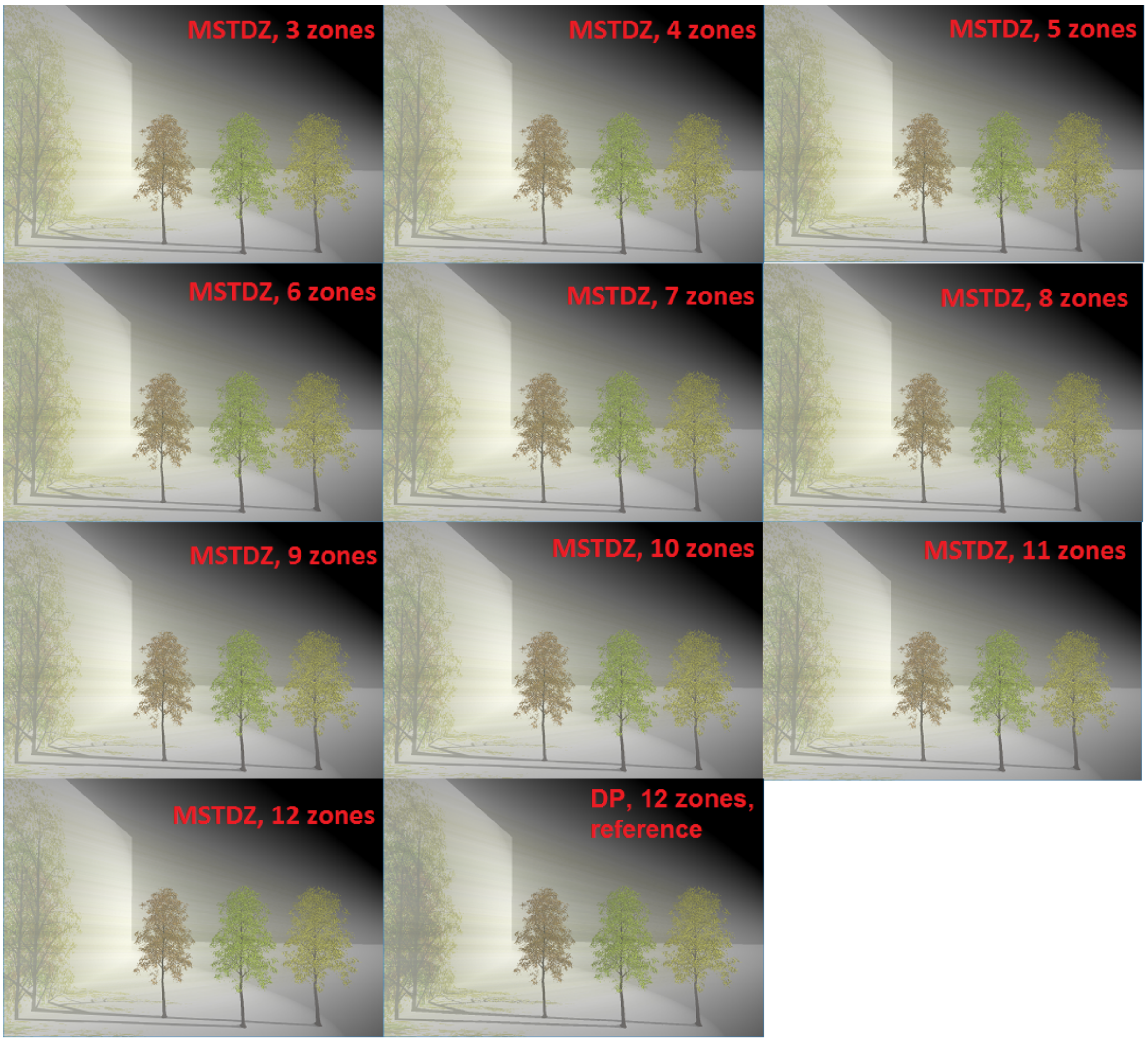


*Figure 53.* The Foliage scene rendered using MSTDZ with various amounts of zones. The reference result can be found in the middle at the bottom of the image.

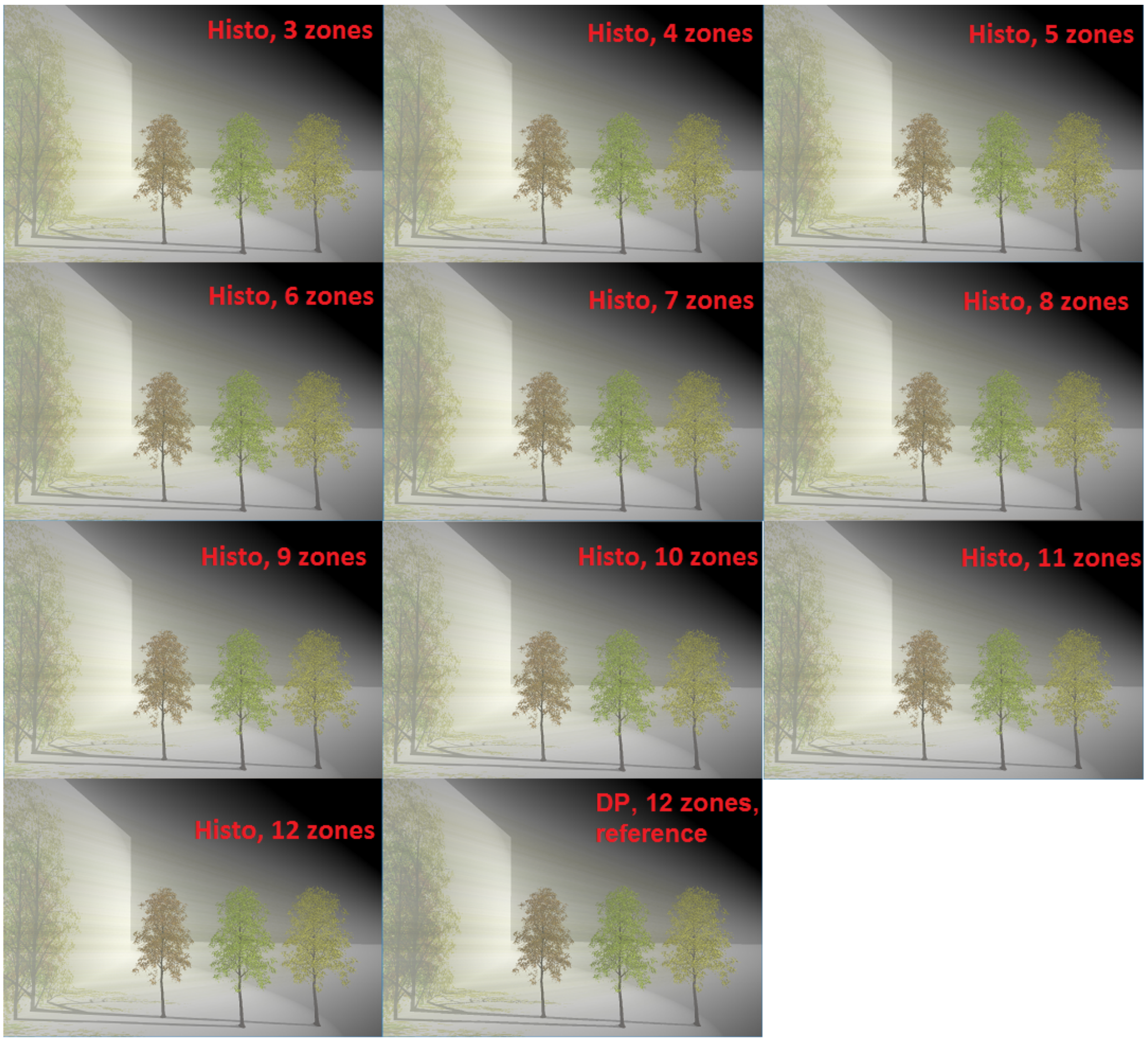


*Figure 54.* The Foliage scene rendered using MSTDZ with various amounts of zones. The reference result can be found in the middle at the bottom of the image.

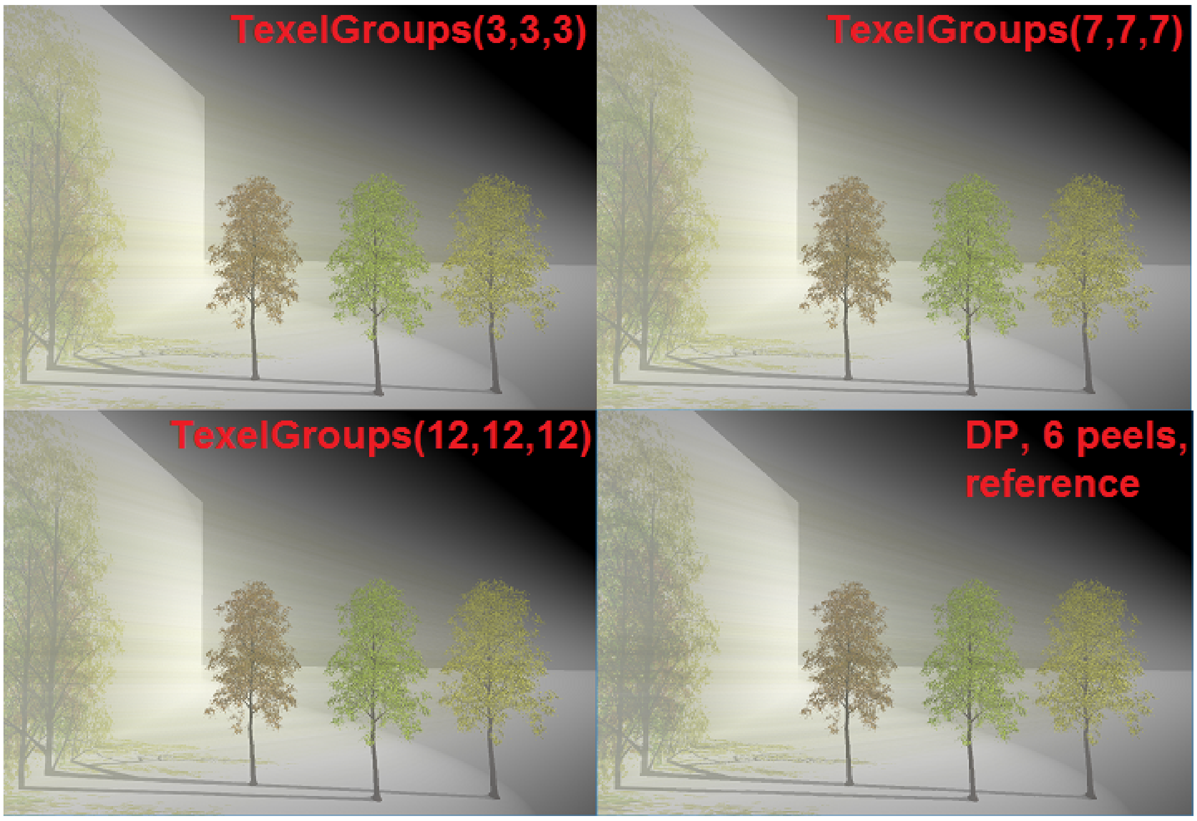


*Figure 55.* The Foliage scene rendered using MSTDZ with various amounts of zones. The reference result can be found in the middle at the bottom of the image.

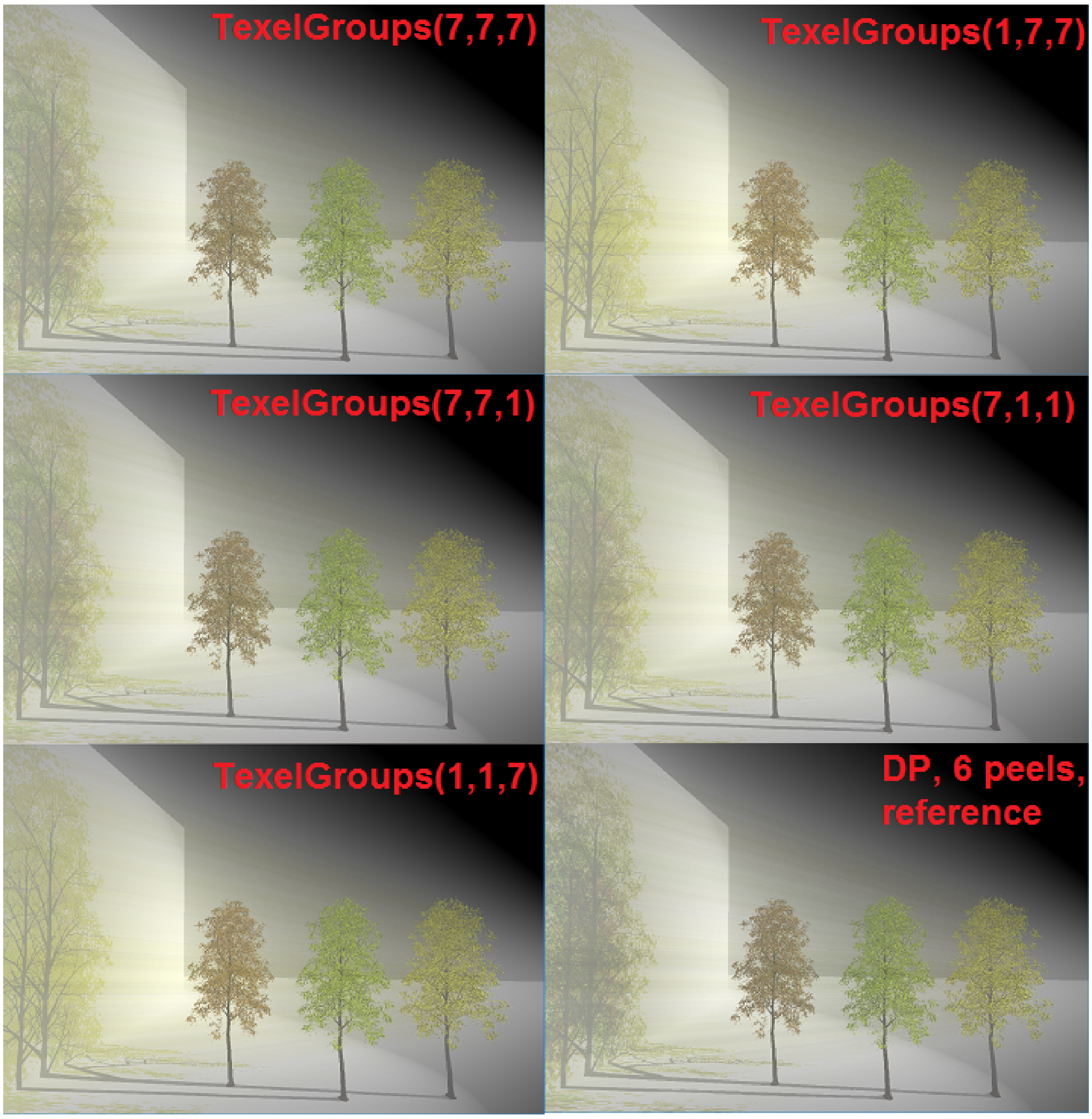


*Figure 56.* The Yellow Planes scene rendered using variations of Texel Groups$_{7,7,7}$.

## The impact of shadow map resolution

All of the optimizations as proposed in Section 3.4 were implemented for one reason: to reduce the amount of unnecessary computations that would have to be done during ray-marching. Tighter bounds, actualized by the addition of first and last transparent depth values, make sure that we never step through zones in which we can be 100% sure we will not find any semi-transparent objects. Furthermore, texels that have no semi-transparent geometry at all are fully ignored. Nevertheless, shadow map resolution has a large effect on computational performance, both for the reference method as well as for our proposed methods. With higher shadow map resolution naturally comes higher amounts of texels. Each of these texels needs to be stepped through during ray-marching. Furthermore, any unnecessary calculations that happen when stepping through these texels will become exponentially problematic when increasing shadow map sizes by factors of 2 (i.e. 4 times more texels have to be handled).

The low-frequency property of single scattering effects brings one advantage: higher shadow map resolutions do not necessarily increase the visual quality of the colored scattering effects. Only shadows are improved when we do so. Conversely, lowering the shadow map resolution also does not have a strong negative effect (up until a certain threshold). This implies that we could even look to use separate, low-resolution shadow map to render colored single scattering effects, apart from shadow maps that are used to calculate shadows, as calculations will drastically lower without losing much visual quality.

Nonetheless, it is interesting to look at the effect on computational performance when increasing and decreasing shadow map resolutions to 2048x2048 and 512x512, respectively. This as opposed to the standard 1024x1024 resolution that was used during the evaluation so far.

To this end, we will be revisiting the Yellow Planes scene and the Foliage scene, and look at the computational performance for the reference image, and both the minimum amounts and the optimal amounts of zones for MSTDZ and Histogram MSTDZ, based on PSNR values that were reported earlier (see Tables 12 and 13). The outcomes of this experiment are shown in Tables 14 and 15 for the Yellow Planes scene and the Foliage scene, respectively.

One of the observations that we can make from these Tables is that Histogram MSTDZ and Texel Groups are fairly strongly affected by decreasing or increasing shadow map sizes, more than regular MSTDZ is. Furthermore, Depth Peeling seems to be least affected by changing shadow map sizes. This is because of the fact that Depth Peeling always knows exactly when to stop iterating over shadow maps during ray-marching, as opposed to MSTDZ, Histo MSTDZ, and Texel Groups, which always go over $numberOfZones$ shadow maps per texel, even if only one of these actually contains a value. This leaves a large area of optimization for future research, alongside the improving of the ray-marching phase.

| **Method** | **Time per frame (in ms) Color gathering (1024 / 512 / 2048)** | **Time per frame (in ms) Ray-marching (1024 / 512 / 2048)** |
|---|---|---|
| $DP_6$ | 0.49 / 0.34 / 0.63 | 8.83 / 8.60 / 9.68 |
| $MSTDZ_3$ | 0.07 / 0.03 / 0.17 | 8.72 / 8.31 / 9.12 |
| $MSTDZ_8$ | 0.08 / 0.04 / 0.26 | 9.46 / 9.05 / 10.72 |
| $HistoMSTDZ_3$ | 0.51 / 0.17 / 1.70 | 8.83 / 8.40 / 9.19 |
| $HistoMSTDZ_8$ | 0.74 / 0.24 / 2.52 | 9.59 / 9.20 / 11.90 |
| $TexelGroups_{3,3,3}$ | 0.46 / 0.24 / 1.23 | 9.90 / 9.62 / 10.62 |
| $TexelGroups_{8,8,8}$ | 0.62 / 0.33 / 1.68 | 11.27 / 11.09 / 12.97 |

*Table 14.* Comparison of computational performance of MSTDZ, Histo MSTDZ, Texel Groups and Depth Peeling in the Yellow Planes simple scene with shadow map sizes of 1024x1024, 512x512 and 2048x2048, respectively.

| **Method** | **Time per frame (in ms) Color gathering (1024 / 512 / 2048)** | **Time per frame (in ms) Ray-marching (1024 / 512 / 2048)** |
|---|---|---|
| $DP_{12}$ | 11.14 / 10.39 / 11.59 | 12.68 / 12.36 / 15.35 |
| $MSTDZ_3$ | 2.16 / 2.13 / 2.31 | 12.46 / 12.29 / 13.42 |
| $MSTDZ_7$ | 2.17 / 2.16 / 2.48 | 14.71 / 14.25 / 16.74 |
| $HistoMSTDZ_3$ | 5.02 / 3.38 / 8.71 | 12.40 / 12.18 / 13.52 |
| $HistoMSTDZ_8$ | 6.69 / 3.36 / 13.85 | 15.28 / 13.50 / 17.35 |
| $TexelGroups_{3,3,3}$ | 6.27 / 4.91 / 11.37 | 15.00 / 13.85 / 17.79 |
| $TexelGroups_{7,7,7}$ | 7.55 / 6.12 / 15.33 | 18.84 / 16.77 / 25.78 |

*Table 15.* Comparison of computational performance of MSTDZ, Histo MSTDZ, Texel Groups and Depth Peeling in the Foliage complex scene with shadow map sizes of 1024x1024, 512x512 and 2048x2048, respectively.

**Individual render pass timings**

Last of all, to give more insight into the various methods and to see where it's most interesting to try and optimize in the future, we will look into timings of individual render passes for each method. We do this in Scene 1: the Five Primary Planes scene, as a simple scene like this already sketches a good idea of the weights of each render pass for each method. We will look at Depth Peeling, MSTDZ, Histogram MSTDZ and Texel Groups and use the optimal amount of zones for each of them, which is 3 for Depth Peeling, MSTDZ and Histogram MSTDZ, and 3/3/3 for Texel Groups. MRT is left out as we have already seen on multiple occasions that MSTDZ outperforms MRT in almost all cases. In all remaining cases MSTDZ performs as well as MRT. Results for this part of the evaluation are found in Table 16.

For Depth Peeling, we see that the first pass, in which the first opaque depth is gathered, takes a relatively short amount of time. All other passes are peel passes, which take approximately the same amount of time. However, the timings of the peels can vary significantly depending on the placement of semi-transparent geometry.

For all other methods, we see that the last transparent depth pass costs less time than the first transparent or first opaque depth passes. Furthermore, all 'pre-passes' cost significantly less time than the actual color gathering render pass (or passes, in case of Texel Groups) itself. This last pass always costs, by far, the largest amount of time for the entire method and this weight only becomes higher in more complex scenes. For Histogram MSTDZ this discrepancy is especially large. This implies that we should look into methods of improving the way each texel obtains a zone distribution, e.g. by grouping texels and letting them share the same distribution.

| **Method** | **Render pass** | **Time per frame (in ms)** |
|---|---|---|
| | | **Color gathering** |
| $DP_3$ | 0 | 0.021 (opaqueDepth) |
| | 1 | 0.049 (peel 1) |
| | 2 | 0.071 (peel 2) |
| | 3 | 0.071 (peel 3) |
| $MSTDZ_3$ | 0 | 0.016 (firstTranspDepth) |
| | 1 | 0.007 (lastTranspDepth) |
| | 2 | 0.010 (opaqueDepth) |
| | 3 | 0.022 (write multisample texture) |
| Histo $MSTDZ_3$ | 0 | 0.016 (firstTranspDepth) |
| | 1 | 0.012 (lastTranspDepth) |
| | 2 | 0.016 (opaqueDepth) |
| | 3 | 0.016 (histogram pass) |
| | 4 | 0.095 (write multisample texture) |
| Histo MSTDZ + $TexelGroups_{3,3,3}$ | 0 | 0.008 (firstTranspDepth) |
| | 1 | 0.009 (lastTranspDepth) |
| | 2 | 0.013 (opaqueDepth) |
| | 3 | 0.018 (histogram pass) |
| | 4 | 0.010 (determine 2x2 pixels) |
| | 5 | 0.006 (determine 4x4 pixels) |
| | 6 | 0.027 (1/2/4 texture) |
| | 7 | 0.062 (write full reso. texture) |
| | 8 | 0.034 (write half reso. texture) |
| | 9 | 0.022 (write quarter reso. texture) |

*Table 16.* Comparison of individual render passes for all methods, except for MRT.

### Evaluation conclusions

All of the methods that were proposed in this thesis were evaluated, as well as compared to the reference method in different scenes. The most important conclusions that were drawn are summarized here.

Based on the evaluation, it looks like regular MSTDZ always outperforms MRT in terms of computational time at equal number of depth zones. Regular MSTDZ also outperforms Histogram MSTDZ when semi-transparent objects are spread uniformly and not in clusters. However, given the right scene, Histogram MSTDZ outperforms regular MSTDZ, as fewer zones can be used, and thus lower computational effort needs to be put in.

Furthermore, it seems that both MSTDZ as well as Histogram MSTDZ can outperform Depth Peeling - especially in more complex scenes - both in terms of speed. Sometimes Depth Peeling is even outperformed in terms of quality, meaning that an approximation obtained with MSTDZ or Histogram MSTDZ looks similar to Depth Peeling's result for the same scene, while using a lower amount of Depth Zones than Depth Peeling would use peels. Regular MSTDZ works best if semi-transparent geometry is uniformly spread over a certain region, while Histogram MSTDZ works better if semi-transparent geometry is spread out in clusters.

Texel Groups does not increase the computational performance of its predecessor. However, it has the ability to perform well in simpler scenes, where both time and memory can be saved (compared to Histogram MSTDZ) by sacrificing a fragment of quality of the results.

Something peculiar that was found is that the use of bias for the first and last transparent depth values could be the cause of performance issues and even artifacts. However, it could also be that something inherent with multisample textures is causing these issues. In the future, it should be attempted to make the bias dynamic based on the distance between the first and last transparent depth values.

Additionally, MSTDZ and all of its improved versions have the ability to go up to 32 zones (unlike the MRT method). Despite of this nice property, it looks like this is actually never required. It seems that in most scenes, light is already extinguished far before it could have ever gone through 32 semi-transparent objects. This is not necessarily bad, as MSTDZ still outperforms MRT, but it is something to keep in mind when working with the proposed methods.

# 6 Conclusion

The goal of this thesis was to render colored scattering effects in real-time, or as close to it as possible. These effects are caused by the combination of participating media and semi-transparent objects within a 3D scene. To this end, a method was designed, implemented, optimized and evaluated.

A slight adaptation of Depth Peeling, combined with standard ray-marching, was used to solve colored participating media effects caused by semi-transparent objects. Two new methods were proposed to improve upon the color gathering phase, i.e. Depth Peeling. These methods are the Multi Render Target (MRT) method and the Multisample Texture Depth Zones (MSTDZ) method.

MRT worked well, but had the major disadvantage of being limited by the amount of concurrent render targets that can be used. To overcome this limitation, MSTDZ was proposed, in which the specific data structure of multisample textures is used to save colors in subsamples of such a texture, while also being able to gather more information. This is due to the fact that most graphics cards allow for more subsamples in a multisample texture than concurrent render targets. MSTDZ is thus a method that does what MRT does, but with potentially more accuracy. Some optimizations were applied, in which the first transparent and first opaque depth values, and later on last transparent depth values, were used to bound the area in which depth zones are distributed.

Certain complex scenes can not be properly ‘caught’ by MSTDZ, and to counteract this, an addition was made to MSTDZ that uses what could be seen as a sort of histogram binning mechanic to determine where to place fewer or more zones. Finally, an experiment was done with trading off visual detail for memory and vice versa, by grouping texels with similar histogram distributions together in lower resolution textures.

In addition, an extra idea was explored and implemented as an alternative (see Appendix B). This was done in parallel to the main research direction, as to possibly speed up any of the methods at the cost of quality. Since (colored) scattering effects have a low-frequency nature, a few missing light beams would not be noticeable, especially from far away. Sadly, early results looked not very promising, and since the idea was a sidetrack, it was abandoned. However, it is expected that with some changes, the method might become an interesting endeavor for future research.

An extensive evaluation was performed, as to find out whether the reference method, the MRT method and the MSTDZ method and all of its optimizations and additions were doing well at what they were theorized to do well. The corresponding results were the cause of optimism, but they also led to more questions and ideas for future improvement. Very good results were found, as it turned out, in the fact that MRT, MSTDZ and MSTDZ with its histogram addition can all be used in ways to approximate Depth Peeling's result in a faster manner, making the color gathering phases more efficient than that of the reference method. However, ray-marching was always either on par, or even slightly slower than it is for Depth Peeling. This was an expected result, but nonetheless it is important that this is tackled as one of the first topics of future research. Furthermore, using the 'Texel Groups' trade-off idea, it turns out to be possible to sacrifice accuracy in detailed parts of the scene to save time and memory. However, if used in complex scenes, computational results will become poor and the method will also fail to preserve memory.

Researching this thesis' specific problem turned out to be a completely new venture and, expectedly, this leaves open a large amount of options to explore and research in the future. Potential areas that could be improved are, for example, extending or adapting the histogram pass of the histogram addition, or, more importantly, decreasing the amount of ray-marching steps in various ways.

Once again: the goal of this thesis was to render colored scattering effects in real-time, or as close to it as possible. The goal was met, but depends on factors such as output resolution, shadow map resolutions and scene complexity. The focus was laid on improving the color gathering phase by approximating the reference method's result as accurately as possible. It turns out that the ray-marching phase has a larger influence on the total amount of time spent rendering each frame and future research should be invested into the optimization of this component, e.g. by applying prefiltering. However, for the color gathering phase a large speed-up is gained by applying either MSTDZ or histogram-based MSTDZ. Which one to use specifically depends on the distribution of semi-transparent objects within a scene. These methods are applicable in all settings, use a constant, predictable amount of render passes, possibly consume less memory and approximate the quality of the reference method to a fitting degree.

# 7
# Future Work

Various elements about the methods proposed in this thesis are worth discussing in terms of steps that could be made to improve upon these methods in future research. One of these points is the fact that shadows in scenes resulting from the various proposed methods are all of poor quality. This is a side-effect of having this thesis focus only on the colored scattering effects caused by the combination of participating media and transparent objects. It would be interesting to investigate whether improvements to shadow effects could be incorporated in the proposed methods, or whether existing methods can be applied in an efficient way.

A more general question that arises is whether a scene would ever need to use 32 zones. In extremely complex cases (e.g. trees, or even forests), this might be the case, but then a follow-up question would be: is the contribution of light still relevant after having passed through a high number of transparent objects? Or will the light have been extinct after a certain amount of transparent objects? In this case some kind of visibility or transmittance component should be kept track of. This component could then be used to cut off ray-marching early as soon as no more light is being propagated and only shadow would still be visible.

Certain parts of the final method may also be adapted to see if results can be improved. For example, the histogram algorithm bases its zone distribution on the amount of fragments found within each zone (per pixel). However, this pass of the algorithm takes up a relatively large chunk of time. There is the possibility that other density-based solutions exist, which could be investigated to see if efficiency of this specific part of the algorithm could be increased. Furthermore, in the method's current state, pixels are fused into pixel groups based on whether their histogram bins are exactly equal. However, if histogram distributions differ slightly, pixels that could be seen as similar, will not be grouped. This grouping condition could be tweaked to be more flexible, which would make the algorithm more effective in terms of quality, while also potentially increasing the algorithm's efficiency.

Multiple other aspects of the histogram addition could also be potentially improved. One idea that could be explored is swapping the 'histogram texture', which right now allows for four bins, for a multisample texture. In this case, when using 32 subsamples, up to 128 bins can potentially be used, allowing for a much more accurate allocation of zones to said bins. This could prove to be very useful in complex scenes, improving quality of the results, without crippling the speed at which these are gained.

Another issue that one could run into is that histogram-based MSTDZ is equal to regular MSTDZ if for a pixel all histogram zones have an equal amount of fragments. Having an equal amount of fragments in each histogram zone will uniformly distribute the corresponding depth zones over the histogram zones, which gives the exact same distribution of depth zones that regular MSTDZ would use. In such scenarios, the result of histogram-based MSTDZ will be obtained at a slower rate than regular MSTDZ would. This is due to the addition of the histogram binning algorithm. It would be a good optimization to skip the depth zone distribution phase if fragments in histogram zones are (nearly) equal for each histogram zone of a pixel. For example, one could add a condition that skips the depth zone distribution part immediately after it turns out that the amounts of fragments during the fragment counting phase are all (nearly) equal. Alternatively, the histogram partitioning could be reused for multiple frames and only be computed every few frames (instead of every frame). This, however, might lead to certain visual artifacts presenting themselves, but this depends on how often the recomputation is done.
In addition, it will not always be necessary to use the maximum amount of zones for each scene. The optimal amount of zones depends highly on the scene, and right now, finding it is a matter of trial-and-error. It is interesting to start a study into determining or approximating the ideal amount of zones to use in each scene. This would not only take away the trial-and-error task, but, most importantly, unnecessary extra steps in ray-marching can be avoided. On top of that, memory consumption would be dramatically lower, e.g. if an 8-sample MS texture can be used instead of a 32-sample MS texture.

In all methods, the ray-marching step could potentially be improved by skipping sampling steps whenever the current fragment maps to the same texel as the next fragment, or fragments. This would be especially useful if lower resolution textures are used as the method's color textures during the color gathering phase, since a fragment would project to the same texel more often. As soon as the next fragment, does not map to the same texel anymore, the results for all skipped steps can be approximately summed up. Using this optimization, multiple ray-marching steps can be skipped outright, leading to an increase in computational performance. This optimization does assume that color values of the skipped ray-marching fragments remain constant, which does not necessarily have to be the case. Other potential ray-marching optimizations could possibly be derived from the research of Engelhardt & Dachsbacher (2010, [8]) or Chen et al. (2011, [6]). Using their techniques, it might be possible to skip certain ray-marching steps. Potentially, more efficient ray-marching improvements could be inspired by Prefiltered Single Scattering (Klehm et al., 2014, [12]), or by the method proposed by Peters et al. (2016, [18]). As was explained earlier, these methods attempt to approximate the visibility function of pixels in participating media. The functions that are approximated in these papers are all single (decreasing) step functions, whereas visibility functions in pixels with semi-transparent objects usually have multiple (decreasing) steps. One possible attempt that could be made is to represent the visibility function using a Heaviside basis as was proposed by Salvi et al. (2011, [19]).

# A
# Pseudocode

## Depth Peeling

**Algorithm 1:** Depth Peeling

```
for $pass \leftarrow 0$ to $numberOfPeels + 1$ do
    if $pass == 0$ then
        render opaque geometry;
    end
    if $pass == 1$ then
        render transparent geometry;
        $texture\_color[pass] \leftarrow$ colors within framebuffer's color attachment;
        $texture\_depth[pass] \leftarrow$ colors within framebuffer's depth attachment;
    end
    if $pass > 1$ then
        if $texture\_depth[pass] < texture\_depth[pass - 1]$ then
            discard fragment;
        end
        else
            $texture\_color[pass] \leftarrow$ colors within framebuffer's color attachment;
            $texture\_depth[pass] \leftarrow$ colors within framebuffer's depth attachment;
        end
    end
end
```

## Multi Render Target method

**Algorithm 2:** Multi Render Target method

```
for $pass \leftarrow 0$ to 3 do
    if $pass == 0$ then
        obtain depths of nearest transparent fragments;
        write depths to firstTranspDepth texture;
    end
    if $pass == 1$ then
        obtain depths of last transparent fragments;
        write depths to lastTranspDepth texture;
    end
    if $pass == 2$ then
        obtain depths of nearest opaque fragments;
        write depths to opaqueDepth texture;
    end
    if $pass == 3$ then
        for $i \leftarrow 0$ to $numberOfZones$ do
            depthRange ← currLastDepth − currFirstDepth;
            depthZoneSize ← depthRange / $numberOfZones$;
            if $i *$ depthZoneSize $<$ currentFragmentDepth $<=$ $(i+1) *$ depthZoneSize then
                $colorTexture[i] \leftarrow [color, alpha, depth]$
            end
        end
    end
end
```

## MSTDZ

**Algorithm 3:** Multisample Texture Depth Zone method

```
for $pass \leftarrow 0$ to 3 do
    if $pass == 0$ then
        obtain depths of nearest transparent fragments;
        write depths to firstTranspDepth texture;
    end
    if $pass == 1$ then
        obtain depths of last transparent fragments;
        write depths to lastTranspDepth texture;
    end
    if $pass == 2$ then
        obtain depths of nearest opaque fragments;
        write depths to opaqueDepth texture;
    end
    if $pass == 3$ then
        for $i \leftarrow 0$ to $numberOfZones$ do
            depthRange ← currLastDepth − currFirstDepth;
            depthZoneSize ← depthRange / $numberOfZones$;
            if $i *$ depthZoneSize $<$ currentFragmentDepth $<= (i+1) *$ depthZoneSize then
                msColorTexture.sample[i] ← color;
                msDepthTexture.sample[i] ← depth;
            end
        end
    end
end
```

## Histogram-based MSTDZ

**Algorithm 4:** Histogram-based MSTDZ method

```
for $pass \leftarrow 0$ to 4 do
    if $pass == 0$ then
        obtain depths of nearest transparent fragments;
        write depths to firstTranspDepth texture;
    end
    if $pass == 1$ then
        obtain depths of last transparent fragments;
        write depths to lastTranspDepth texture;
    end
    if $pass == 2$ then
        obtain depths of nearest opaque fragments;
        write depths to opaqueDepth texture;
    end
    if $pass == 3$ then
        obtainZoneDistribution();
    end
    if $pass == 4$ then
        for $i \leftarrow 0$ to $numberOfZones$ do
            if zone[i].lowerBound < currentFragDepth <= zone[i].upperBound then
                msColorTexture.sample[i] ← color;
                msDepthTexture.sample[i] ← depth;
            end
        end
    end
end
```

**Algorithm 5:** obtainZoneDistribution - Histogram-based MSTDZ pass 3

```
totalHistoZones ← 4
for $i \leftarrow 0$ to $totalHistoZones$ do
    fragDepth ← gl_FragCoord.z if $0 <= fragDepth < 0.25$ then
        out_color ← vec4(1,0,0,0);
    end
    if $0.25 <= fragDepth < 0.5$ then
        out_color ← vec4(0,1,0,0);
    end
    if $0.5 <= fragDepth < 0.75$ then
        out_color ← vec4(0,0,1,0);
    end
    if $0.75 <= fragDepth < 1.0$ then
        out_color ← vec4(0,0,0,1);
    end
end
```

## Histogram-based MSTDZ with Texel groups

**Algorithm 6:** Histogram-based MSTDZ with Texel Groups

```
for $pass \leftarrow 0$ to 4 do
    if $pass == 0$ then
        obtain depths of nearest transparent fragments;
        write depths to firstTranspDepth texture;
    end
    if $pass == 1$ then
        obtain depths of last transparent fragments;
        write depths to lastTranspDepth texture;
    end
    if $pass == 2$ then
        obtain depths of nearest opaque fragments;
        write depths to opaqueDepth texture;
    end
    if $pass == 3$ then
        obtainZoneDistribution();
    end
    if $pass == 4$ then
        determine groups of 2x2;
    end
    if $pass == 5$ then
        determine groups of 4x4;
    end
    if $pass == 6$ then
        create 1/2/4-texture;
    end
    if $pass == 7$ then
        write all 1x1 texels to full resolution texture;
    end
    if $pass == 8$ then
        write all 2x2 texels to half resolution texture;
    end
    if $pass == 9$ then
        write all 4x4 texels to quarter resolution texture;
    end
end
```

## Ray-marching

*Due to issues with LaTeX float objects, the algorithm has moved a page. Please view the next page to see the Ray-marching algorithm.*

**Algorithm 7:** Ray-marching

```
color ← lightColor;
while inFrustum do
    stepForward();
    if currentFragmentIsVisible then
        if useDepthPeeling then
            for $i \leftarrow 0$ to $numberOfPeels$ do
                sampleColor ← sample(colorTexture[i]);
                sampleDepth ← sample(depthTexture[i]);
                if currentFragDepth > sampleDepth then
                    color ← blendColors(color, sampleColor);
                end
            end
        end
        if useMRT then
            for $i \leftarrow 0$ to $numberOfZones$ do
                sampleColor ← sample(colorTexture[i].x);
                sampleDepth ← sample(colorTexture[i].z);
                if currentFragDepth > sampleDepth then
                    color ← blendColors(color, sampleColor);
                end
            end
        end
        if useMSTDZ then
            for $i \leftarrow 0$ to $numberOfZones$ do
                sampleColor ← sampleMSTexture(MSColorTexture, projectedFragment, i);
                sampleDepth ← sampleMSTexture(MSDepthTexture, projectedFragment, i);
                if currentFragDepth > sampleDepth then
                    color ← blendColors(color, sampleColor);
                end
            end
        end
        if useHistoMSTDZ then
            for $i \leftarrow 0$ to $numberOfZones$ do
                sampleColor ← sampleMSTexture(MSColorTexture, projectedFragment, i);
                sampleDepth ← sampleMSTexture(MSDepthTexture, projectedFragment, i);
                if currentFragDepth > sampleDepth then
                    color ← blendColors(color, sampleColor);
                end
            end
        end
        if useHistoMSTDZTexelGroups then
            useHistoMSTDZTexelGroups();                         // See next page
        end
    end
    else
        color ← blendColors(color, vec3(0,0,0));
    end
end
```

**Algorithm 8:** useHistoMSTDZTexelGroups - Ray-marching

```
if projectedFragment.pixelGroupNumbers == 1 then
    for $i \leftarrow 0$ to fullReso.numberOfZones do
        sampleColor ← sampleMSTexture(MSColorTextureFullRes, projectedFragment, i);
        sampleDepth ← sampleMSTexture(MSDepthTextureFullRes, projectedFragment, i);
        if currentFragDepth > sampleDepth then
            color ← blendColors(color, sampleColor);
        end
    end
end
else if projectedFragment.pixelGroupNumbers == 2 then
    for $i \leftarrow 0$ to halfReso.numberOfZones do
        sampleColor ← sampleMSTexture(MSColorTextureHalfRes, projectedFragment, i);
        sampleDepth ← sampleMSTexture(MSDepthTextureHalfRes, projectedFragment, i);
        if currentFragDepth > sampleDepth then
            color ← blendColors(color, sampleColor);
        end
    end
end
else if projectedFragment.pixelGroupNumbers == 4 then
    for $i \leftarrow 0$ to quarterReso.numberOfZones do
        sampleColor ← sampleMSTexture(MSColorTextureQuarterRes, projectedFragment, i);
        sampleDepth ← sampleMSTexture(MSDepthTextureQuarterRes, projectedFragment, i);
        if currentFragDepth > sampleDepth then
            color ← blendColors(color, sampleColor);
        end
    end
end
```

# B Alpha-based Discarding of Fragments

During the work on this thesis, an alternative method was also developed. The alternative was inspired by Enderton et al. (2010, [7]) and can be applied to any of the methods proposed in this thesis. However, no multisampling was used to determine the random sampling patterns that are used in Stochastic Transparency. This makes the alternative look similar to alpha-to-coverage, which is often used in modern graphics hardware.

An interesting, inherent feature of the participating media's single scattering effect, is that it is a low-frequency effect. An idea for optimization was to use this property by randomly discarding fragments as seen from the light source, based on the alpha value of the semi-transparent objects that are seen. Whether or not a fragment is discarded, is based on a hashed function over the fragment's x and y coordinates, which returns a value between 0 and 1.
If for a certain fragment $hash(fragment.x, fragment.y) = 0.6$ and the alpha value corresponding to the current fragment is 0.4, the fragment is discarded. For a neighboring fragment, for which alpha is also 0.4, $hash(fragment2.x, fragment2.y) = 0.2$, and thus, fragment2 is not discarded. A schematic example is given in Figure 57.

As can be seen in Figure 58, certain subsamples will be discarded when applying this technique. The corresponding visual effect is that certain colored light beams that would normally be propagated through the transparent geometry will not be rendered. This will show itself as artifacts within the light scattered by the participating media. The effect is especially noticeable when viewing it from close by. However, since the single scattering effect is a low-frequency one, the effect that alpha-based discarding has, is mostly invisible from a distance.This makes the application of this method appealing to e.g. game developers. They could apply it on parts of a scene that a player can see, but not necessarily reach as to view it from nearby.

Shadows suffer greatly from applying this technique. Stipple patterns become extremely apparent. Improvements could be made, e.g. by applying a Gauss filter (or some other blurring filter) as a post-process, or by applying Percentage-Closer Filtering. A blurring filter could also be applied to the colored scattering effect areas to compensate for the artifacts that appear.

Another issue of this method is that subsequent discarded fragments that project to the same shadow map texel will lead to loss of energy and complete loss of color for that pixel. To solve this issue an exponential factor to compensate for energy loss is added, based on the depth of a fragment. This basically comes down to weighing further away semi-transparent objects higher.

During evaluation it was found that the addition of this method did not lead to significant speed-ups and, sadly, no time was left to deeply research and evaluate the cause of this. Nevertheless, we wanted to report this alternative to pave the way for potential future researchers who could improve upon the idea.

One suggestion for future improvement of this alternative would be to let the hash function somehow depend on the number of the zone a color is in. This means that each depth zone would have its own hash function, leading to more random discarding of fragments and better throughput of colors from semi-transparent objects in lower depth zones (i.e. those with lower depth values). This would make it possible to randomize the sampling pattern per zone, causing the alternative to approximate the results of Stochastic Transparency, with only a slight fragment of the noise issues which the original method suffers from.

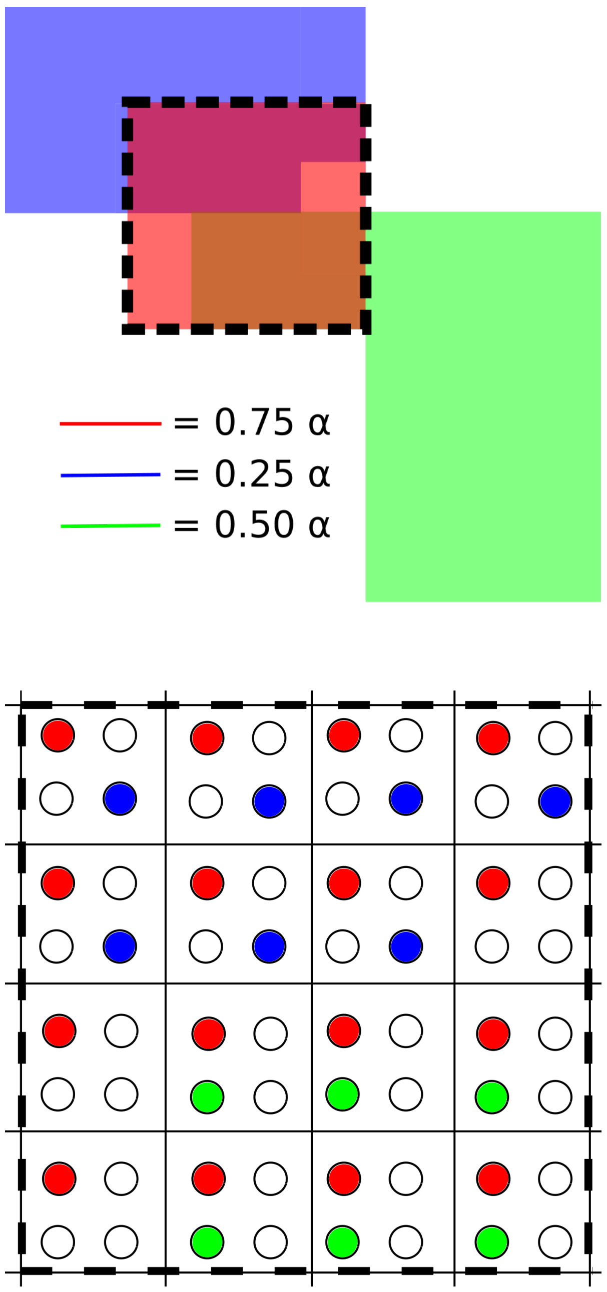


*Figure 57.* Top: Shadow map of a scene. Bottom: Corresponding part of texture of area within dashed lines. Since the red geometry has $alpha = 0.75$, approximately 25% of the corresponding subsamples are discarded for that layer. Approximately 75% of all blue subsamples are discarded and around 50% of green subsamples are discarded. The resulting texture can be found in Figure 58.

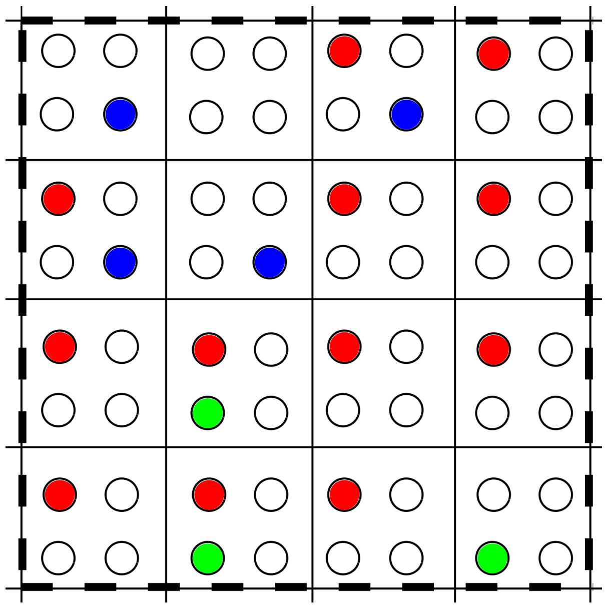


*Figure 58.* Resulting texture of the scene in Figure 57 after discarding fragments based on their alpha values.

# C

# Results and Evaluation Images

All images used in Chapter 5 are placed in higher resolution in this Appendix. For ease of reference, the same subsection names were used as they were in Chapter 5.

## Basic Performance of Individual Methods

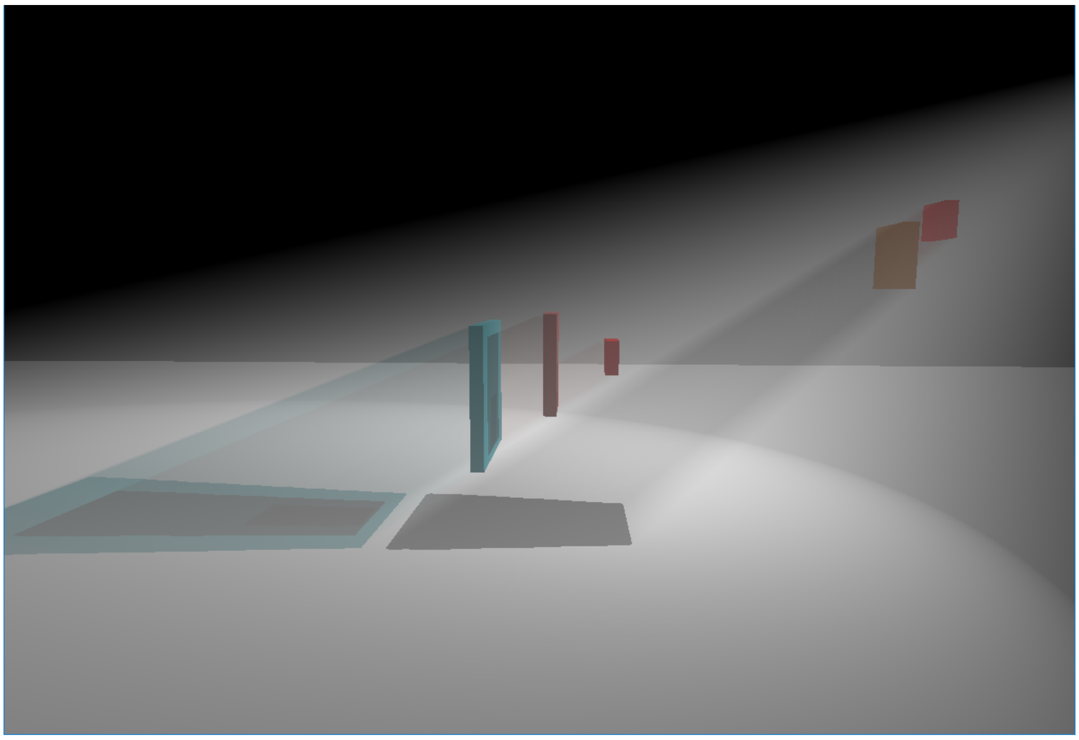


*Figure 33.* The Five Primary Planes scene rendered with Depth Peeling, 3 peels.

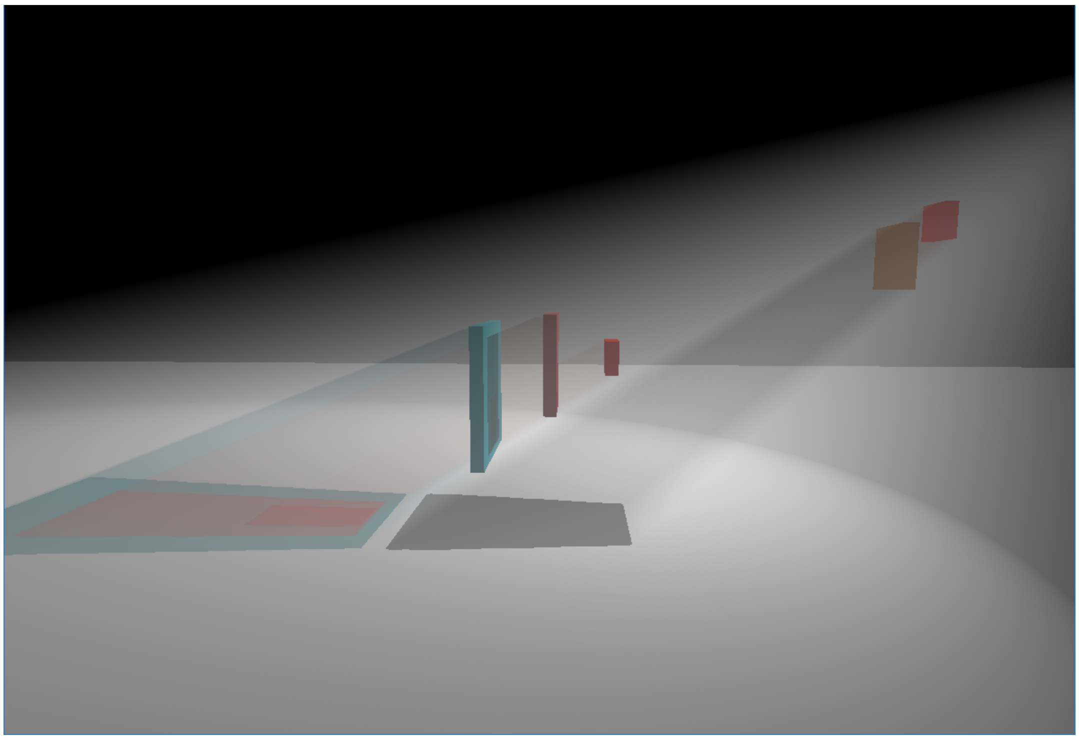


*Figure 33.* The Five Primary Planes scene rendered with Depth Peeling, 1 peel.

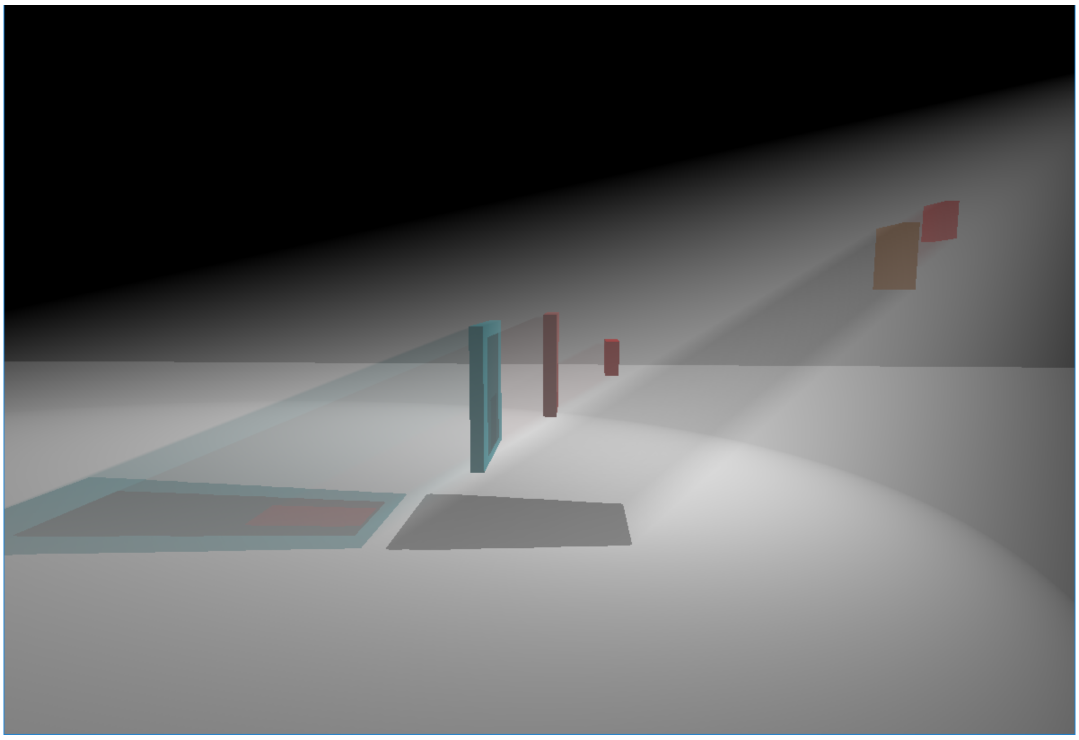


*Figure 33.* The Five Primary Planes scene rendered with Depth Peeling, 2 peels.

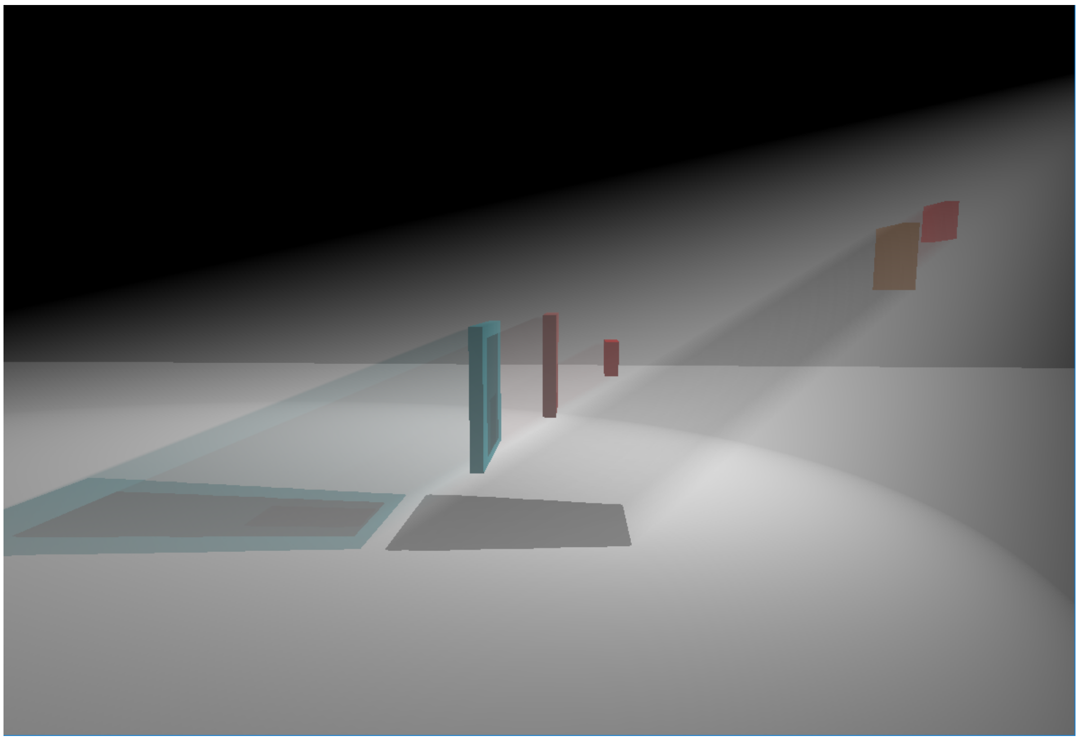


*Figure 34.* The Five Primary Planes scene rendered with the Multi Render Target method, 3 zones.

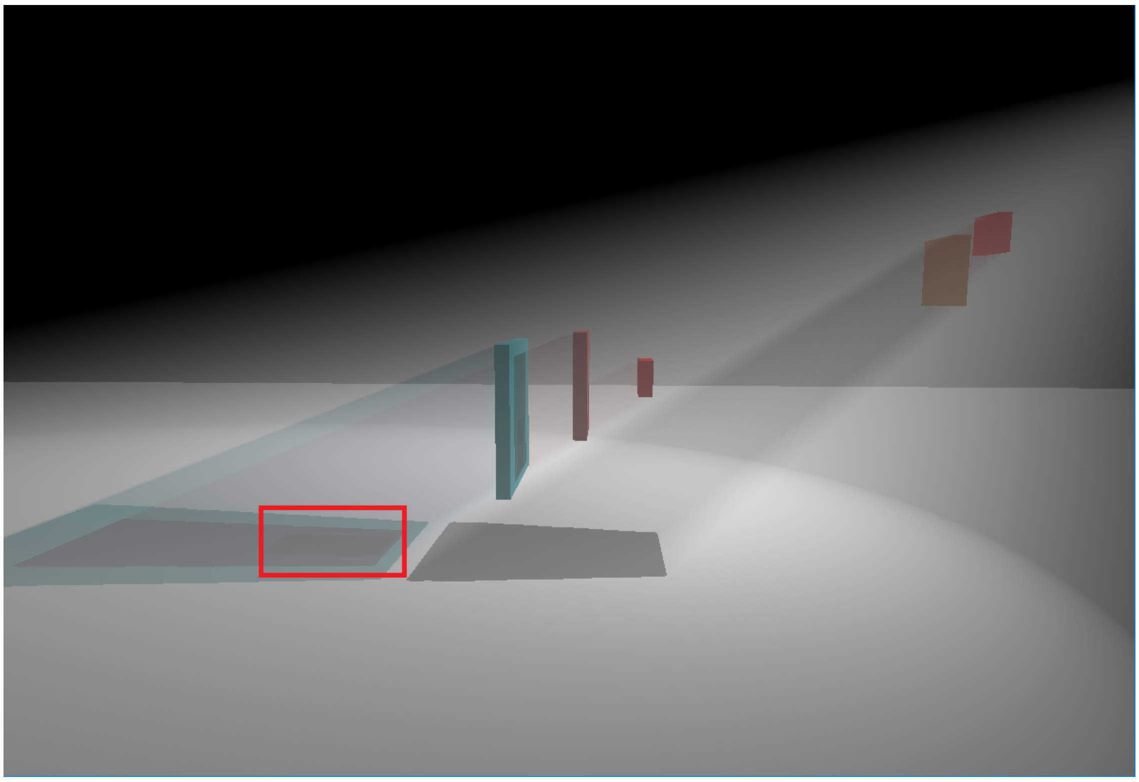


*Figure 34.* The Five Primary Planes scene rendered with the Multi Render Target method, 2 zones.

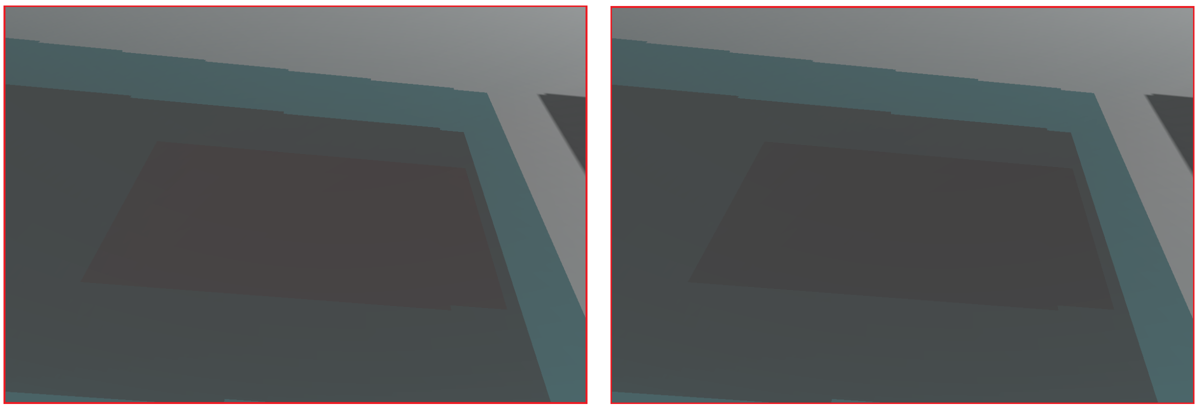


*Figure 35.* Example of barely visible artifact using MRT in the Five Primary Planes scene. Left: 2 zones are used. Right: 3 zones are used.

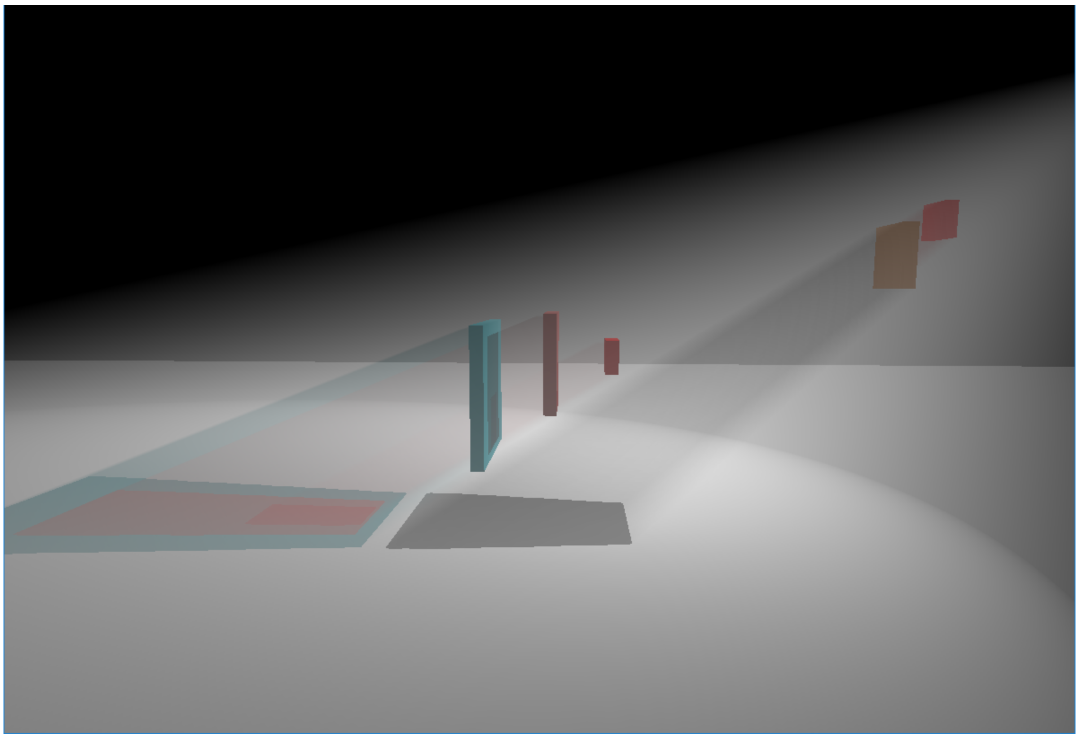


*Figure 34.* The Five Primary Planes scene rendered with the Multi Render Target method, 1 zone.

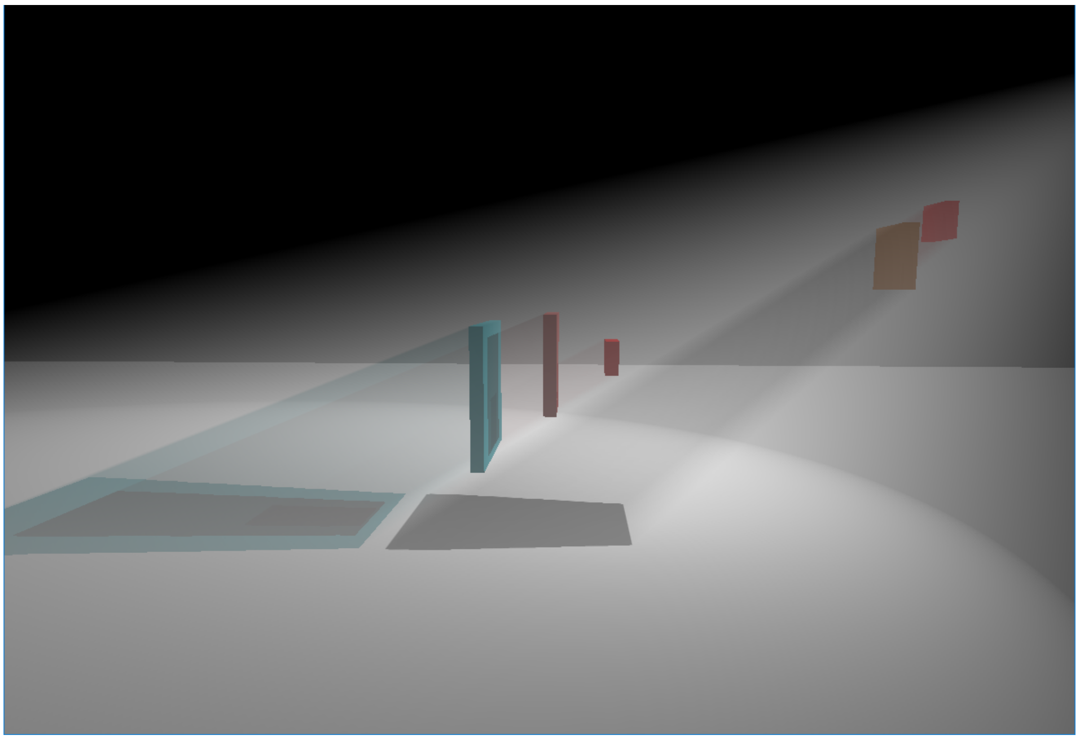


*Figure 36.* The Five Primary Planes scene rendered with MSTDZ, 3 zones.

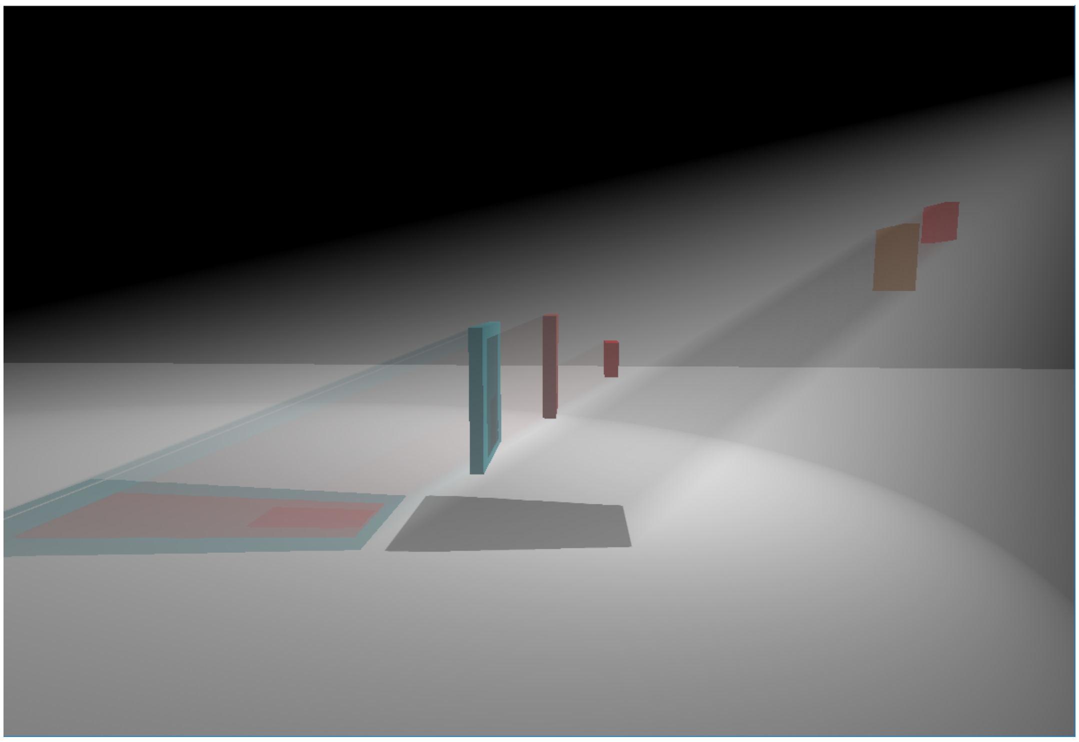


*Figure 36.* The Five Primary Planes scene rendered with MSTDZ, 1 zone.

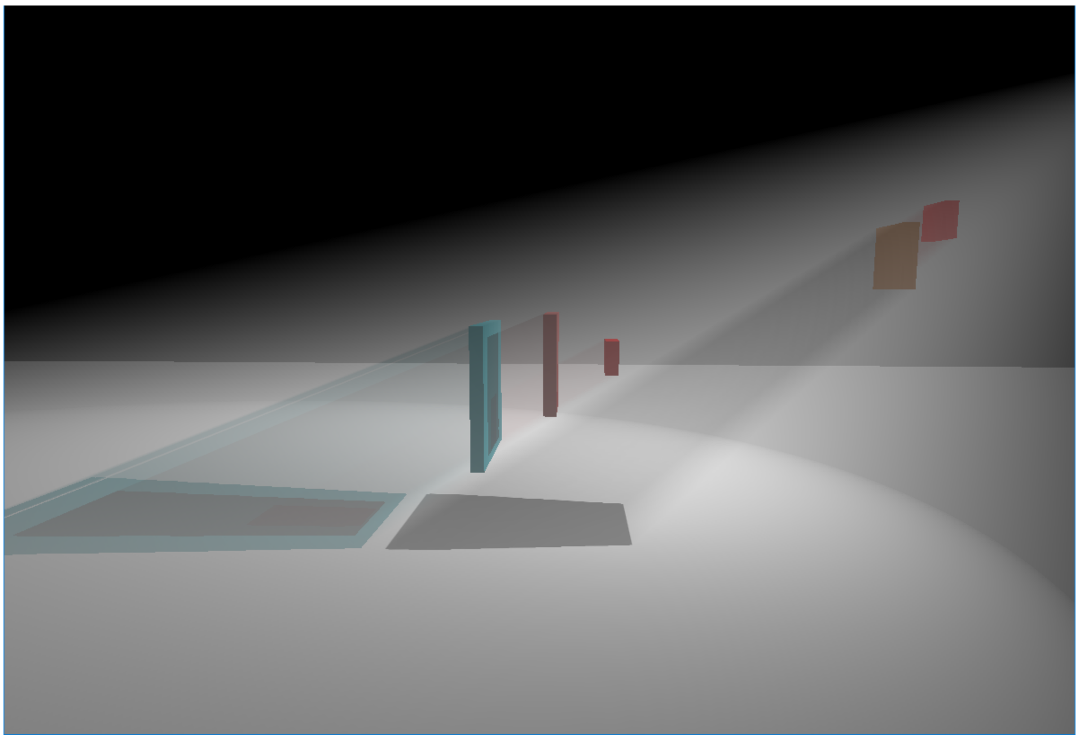


*Figure 36.* The Five Primary Planes scene rendered with MSTDZ, 2 zones.

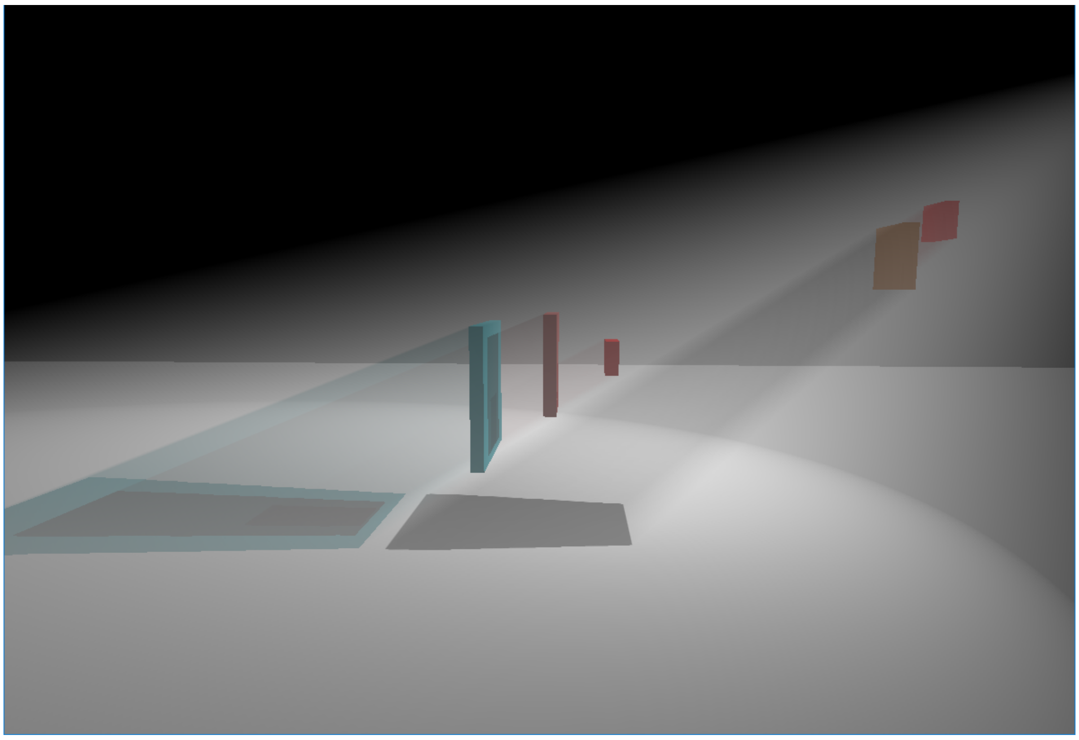


*Figure 37.* The Five Primary Planes scene rendered with Histogram MSTDZ, 3 zones.

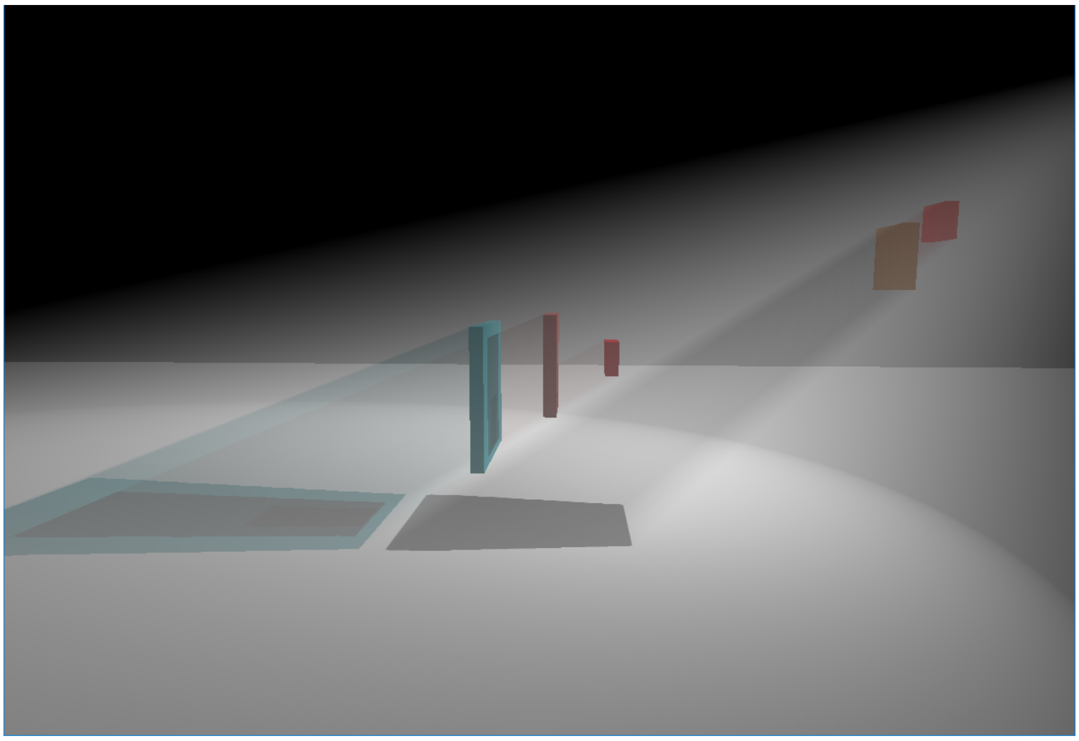


*Figure 37.* The Five Primary Planes scene rendered with Histogram MSTDZ, 3 zones.

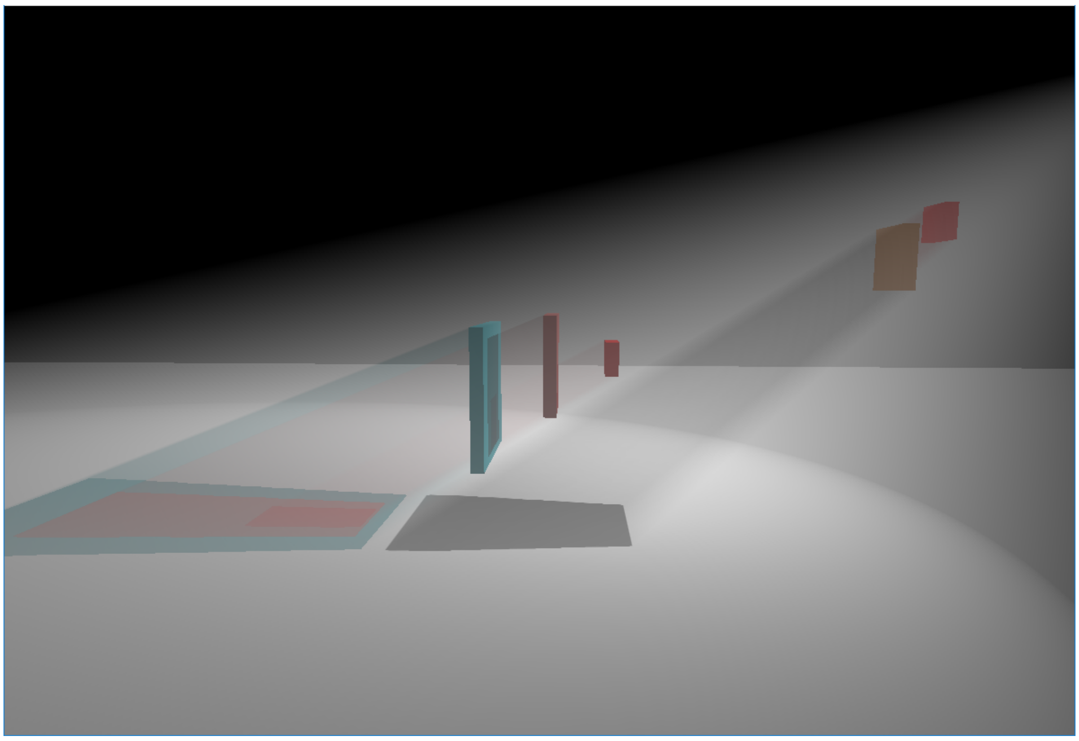


*Figure 37.* The Five Primary Planes scene rendered with Histogram MSTDZ, 1 zone.

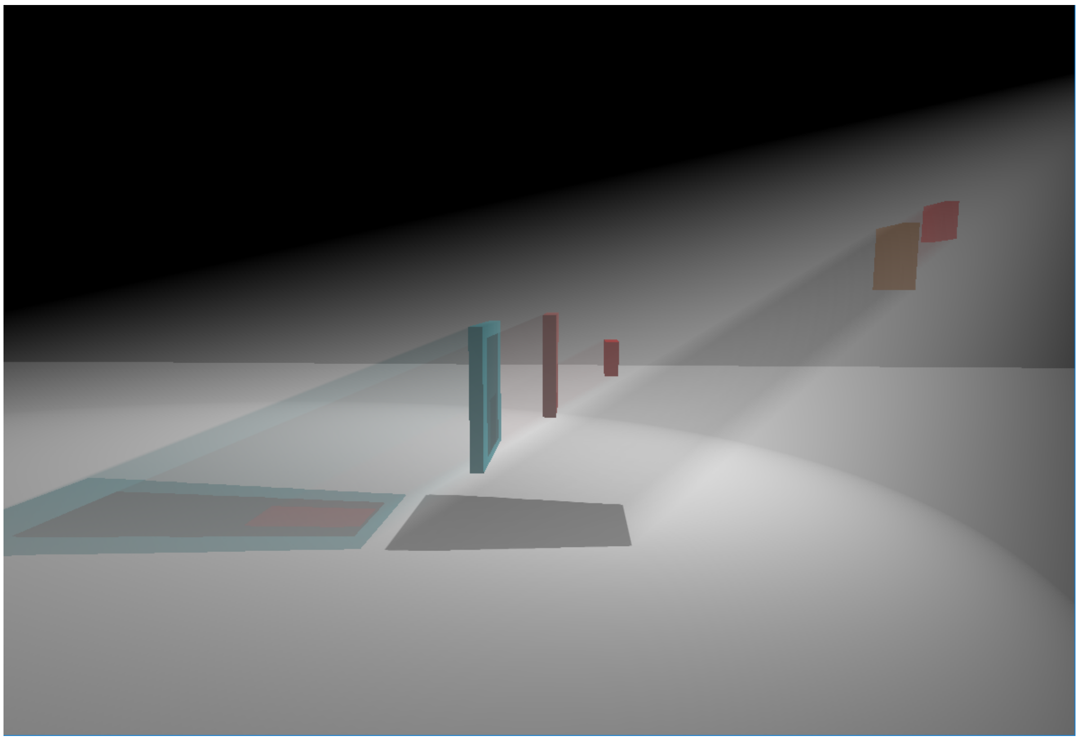


*Figure 37.* The Five Primary Planes scene rendered with Histogram MSTDZ, 2 zones.

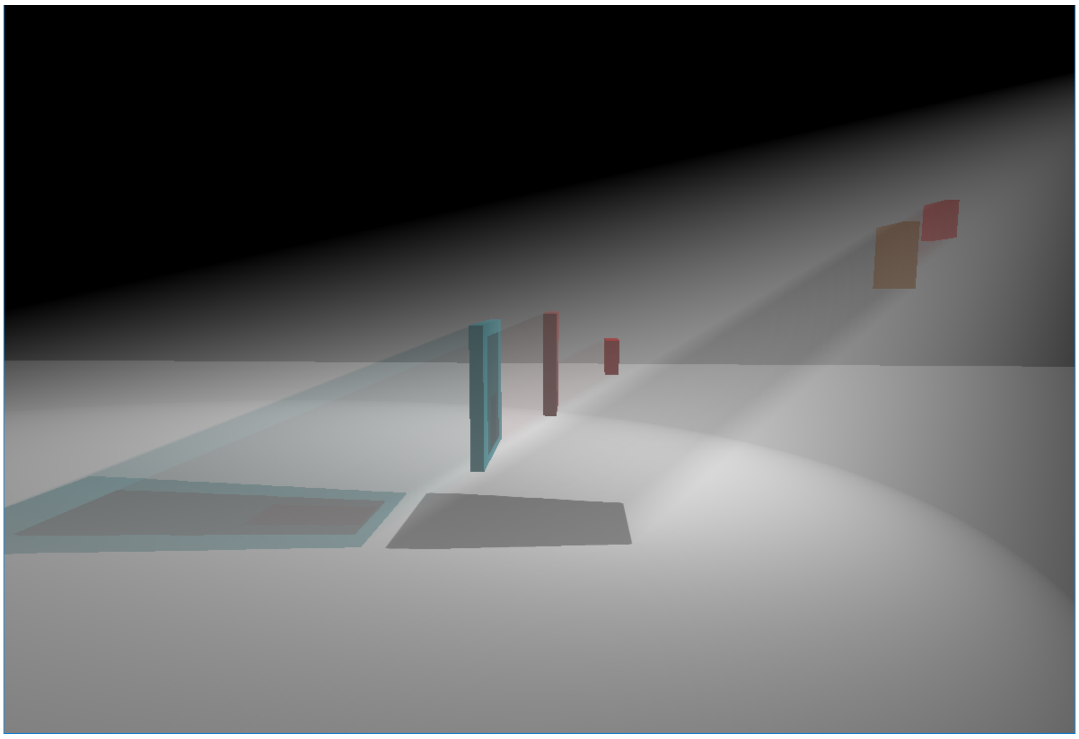


*Figure 38.* The Five Primary Planes scene rendered with Histogram MSTDZ with Texel Groups, 3/3/3 zones.

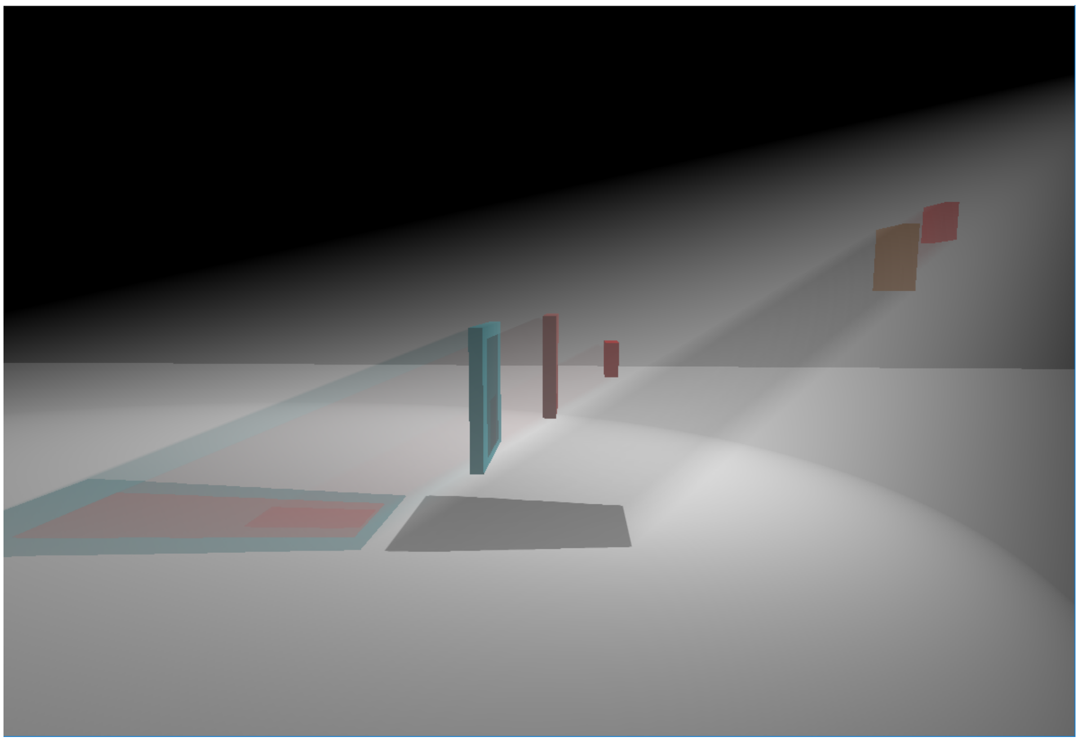


*Figure 38.* The Five Primary Planes scene rendered with Histogram MSTDZ with Texel Groups, 1/1/1 zones.

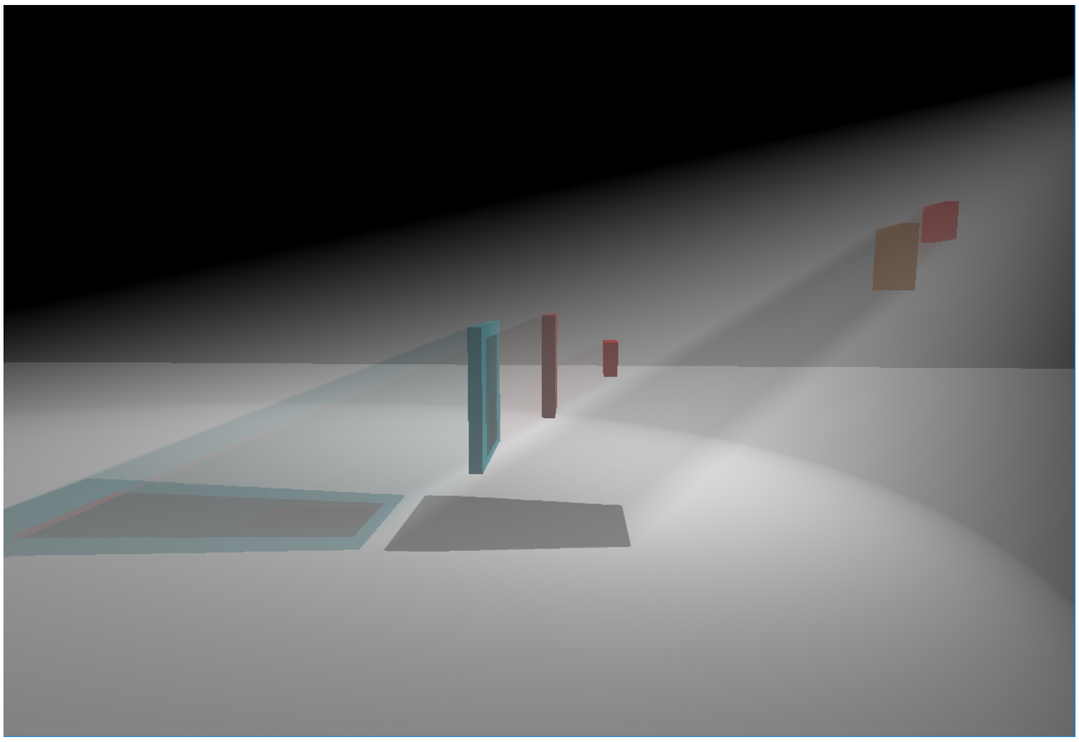


*Figure 38.* The Five Primary Planes scene rendered with Histogram MSTDZ with Texel Groups, 2/2/2 zones.

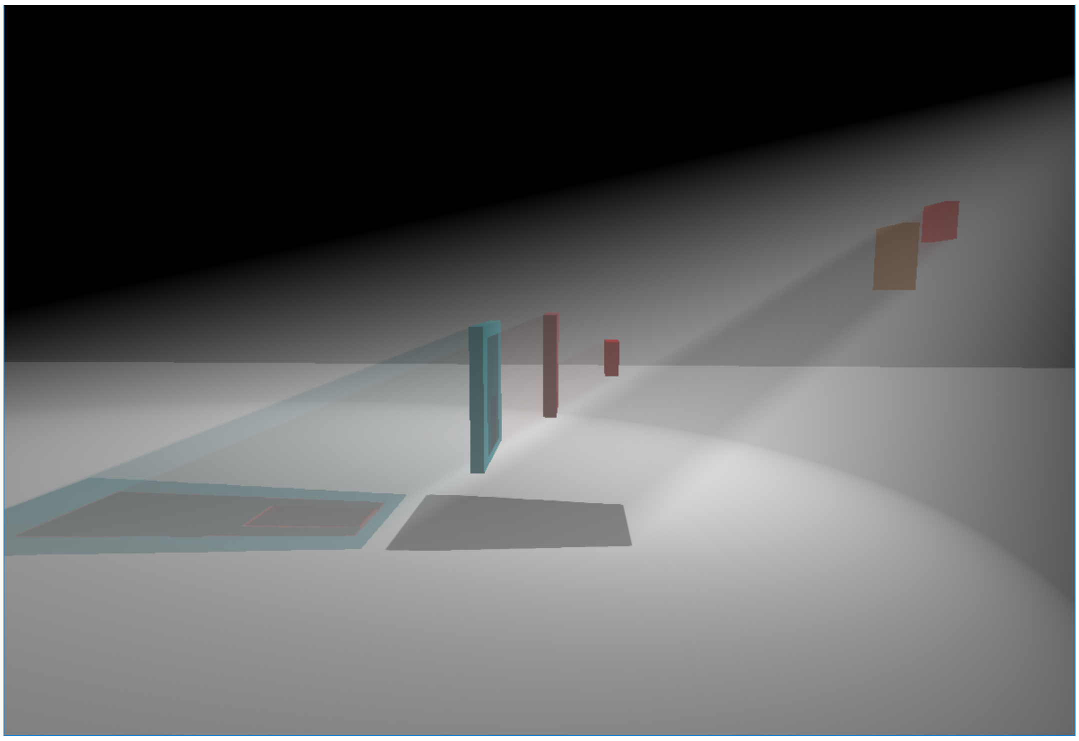


*Figure 38.* The Five Primary Planes scene rendered with Histogram MSTDZ with Texel Groups, 1/3/3 zones.

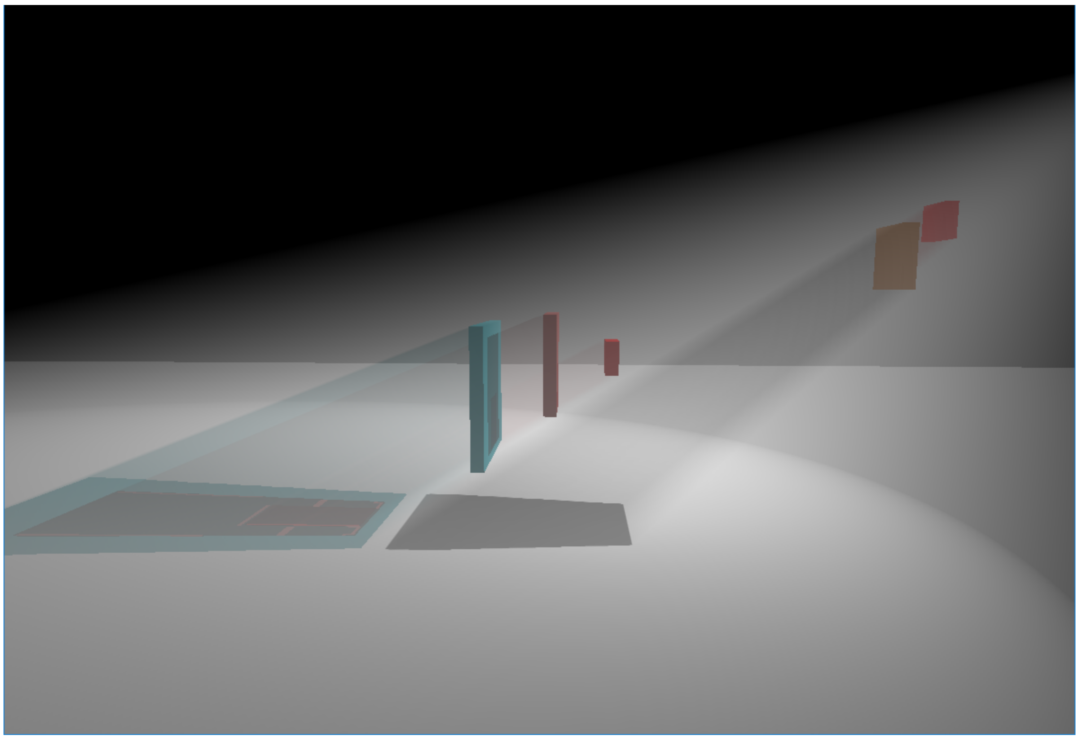


*Figure 38.* The Five Primary Planes scene rendered with Histogram MSTDZ with Texel Groups, 3/1/3 zones.

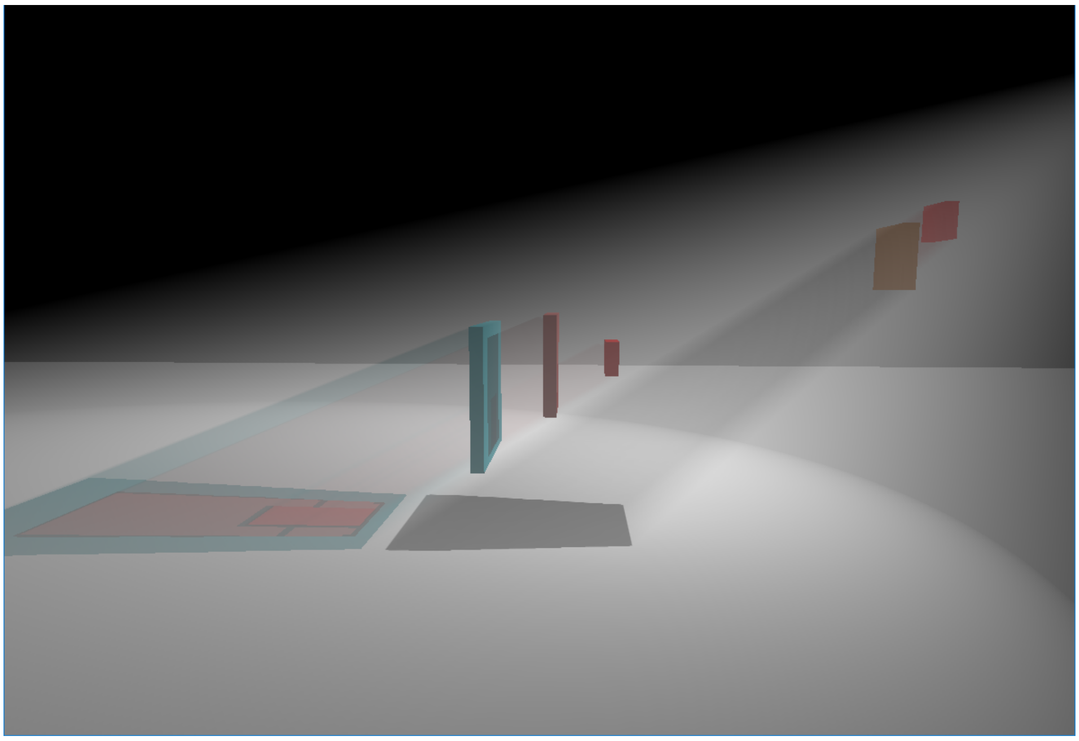


*Figure 38.* The Five Primary Planes scene rendered with Histogram MSTDZ with Texel Groups, 3/3/1 zones.

## Depth Zone Restrictions

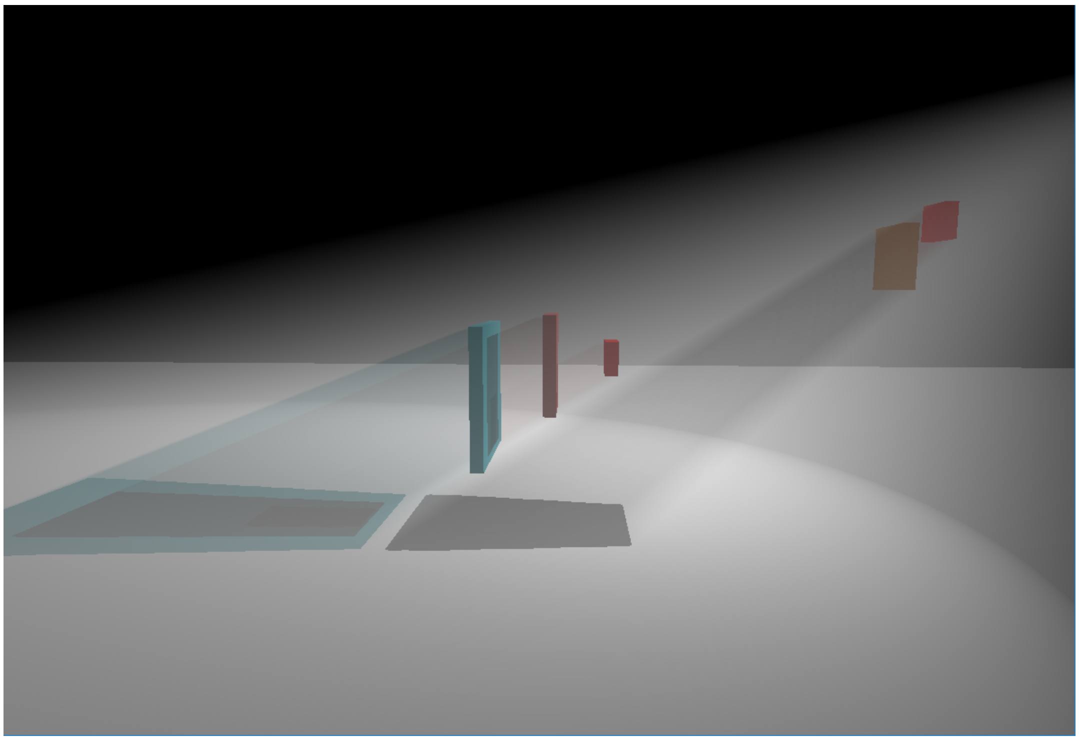


*Figure 40.* The Five Primary Planes scene rendered with the MRT method, 3 zones.

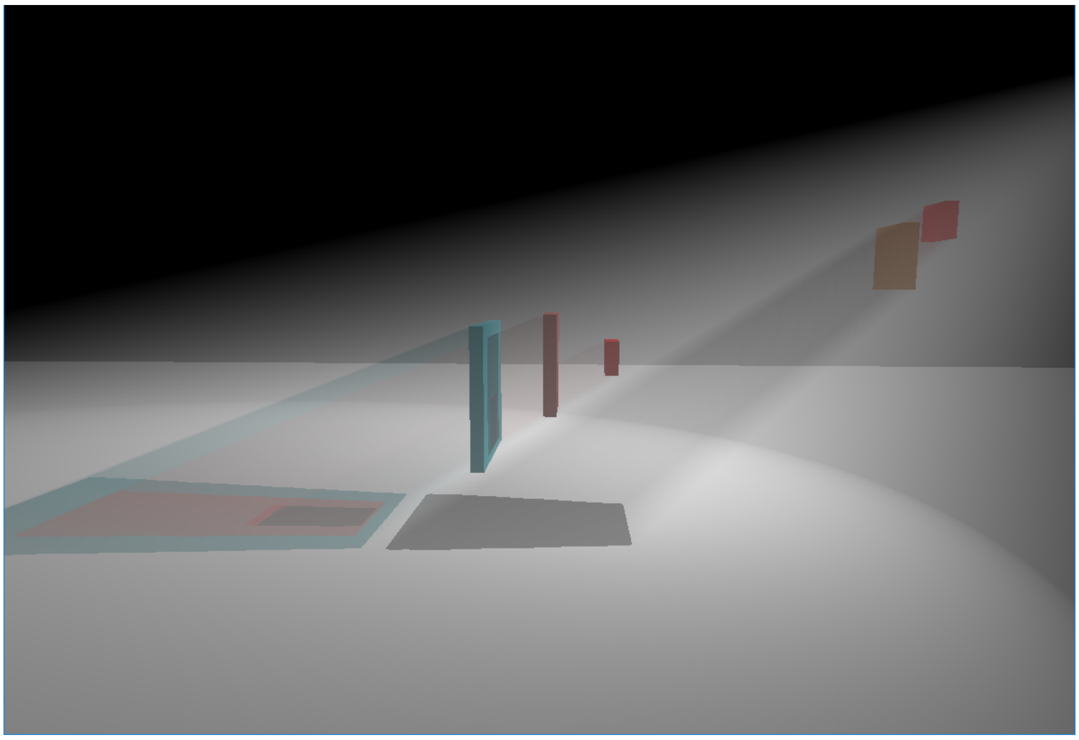


*Figure 40.* The Five Primary Planes scene rendered with the MRT method, replacing the last transparent depth values with first opaque depth values, 3 zones.

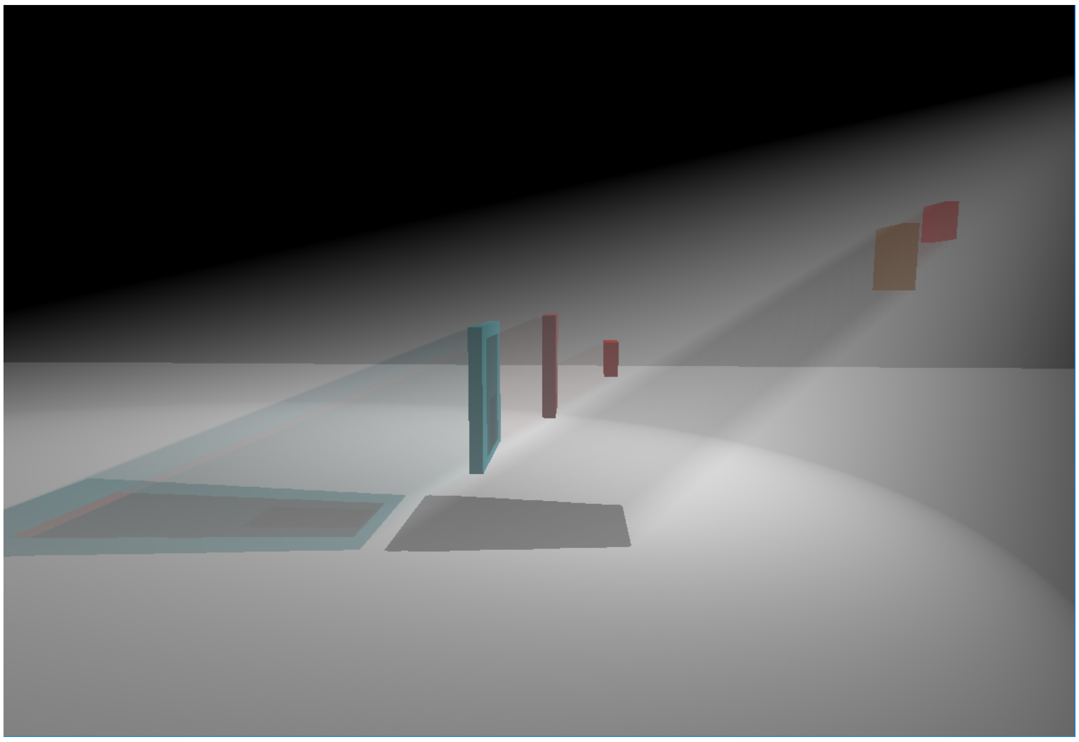


*Figure 40.* The Five Primary Planes scene rendered with the MRT method, replacing the last transparent depth values with first opaque depth values, 8 zones.

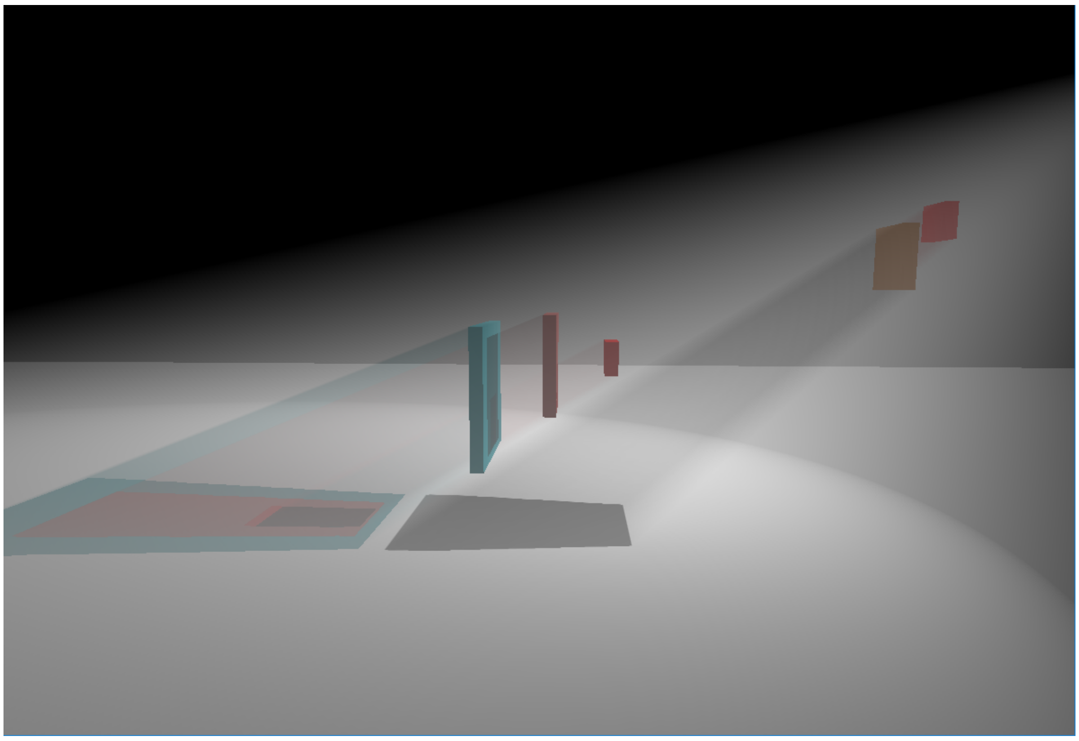


*Figure 40.* The Five Primary Planes scene rendered with MSTDZ, replacing the last transparent depth values with first opaque depth values, 3 zones.

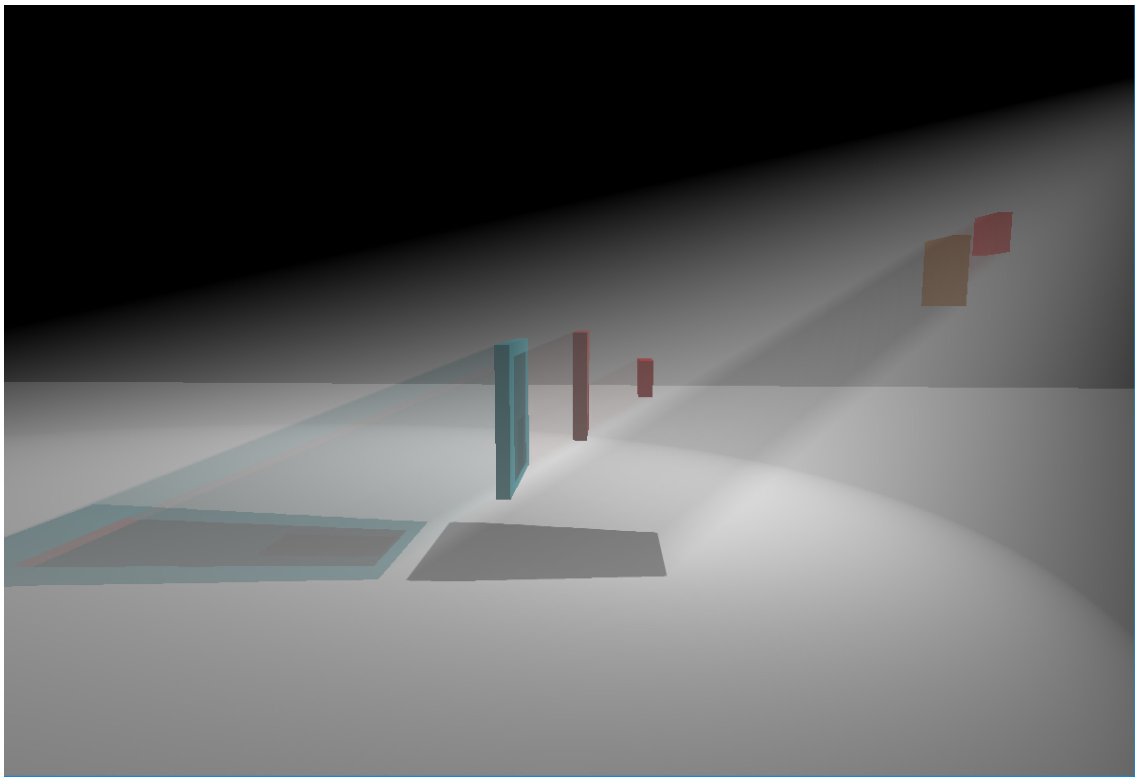


*Figure 40.* The Five Primary Planes scene rendered with MSTDZ, replacing the last transparent depth values with first opaque depth values, 8 zones.

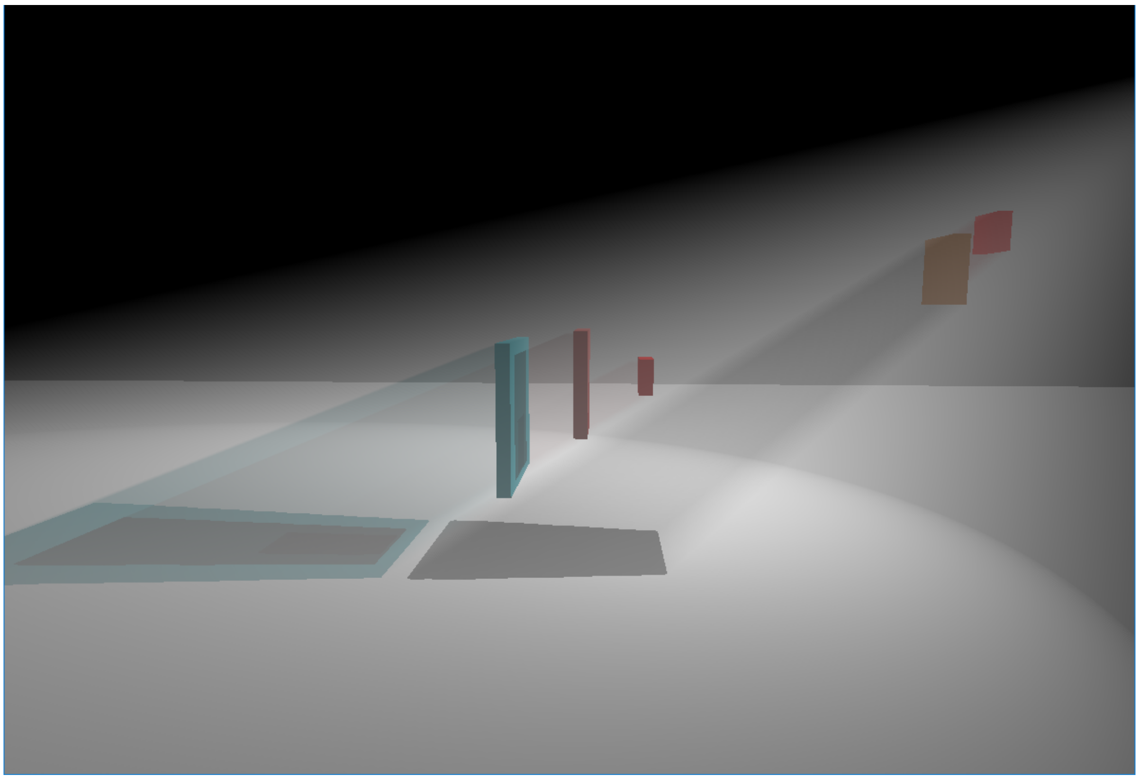


*Figure 40.* The Five Primary Planes scene rendered with MSTDZ, replacing the last transparent depth values with first opaque depth values, 9 zones.

## MRT versus MSTDZ

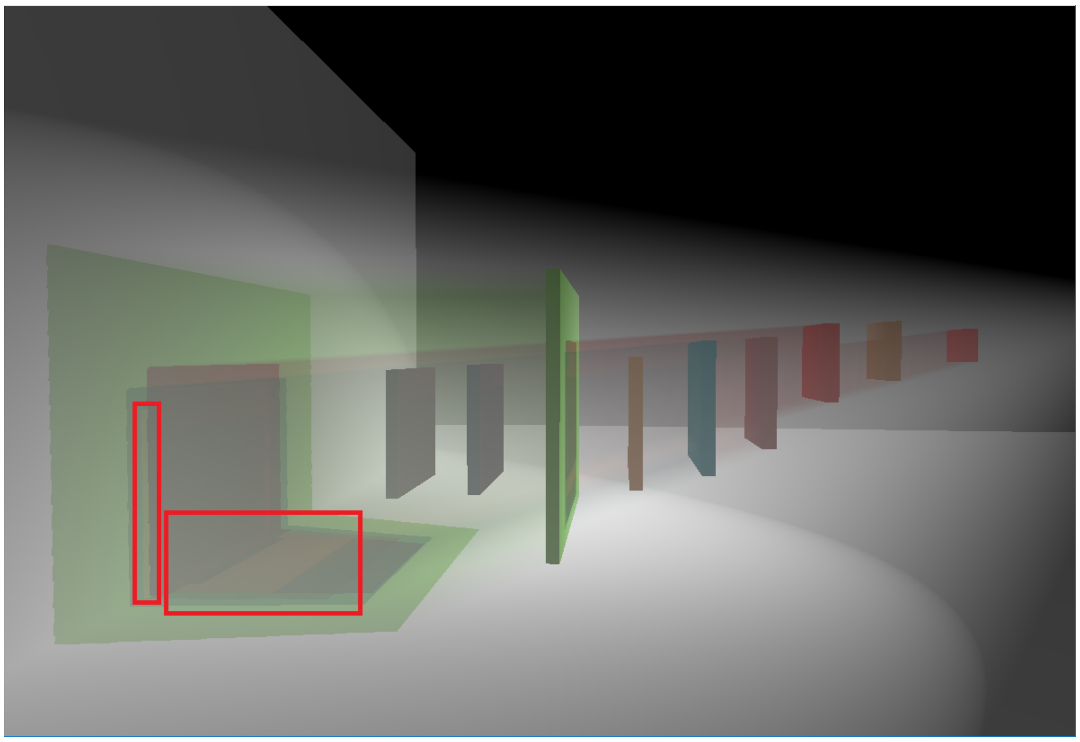


*Figure 41.* The Nine Planes scene rendered using the MRT method, 8 zones. Obvious artifacts arise, which are outlined in red.

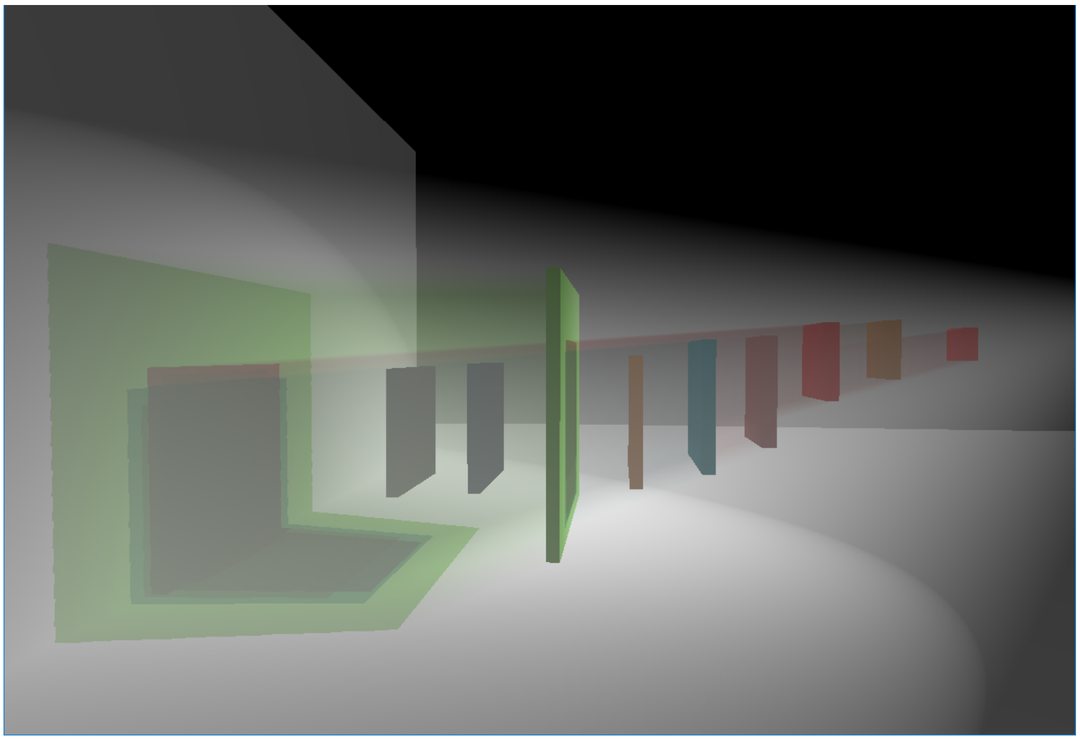


*Figure 41.* The Nine Planes scene rendered using MSTDZ, 9 zones.

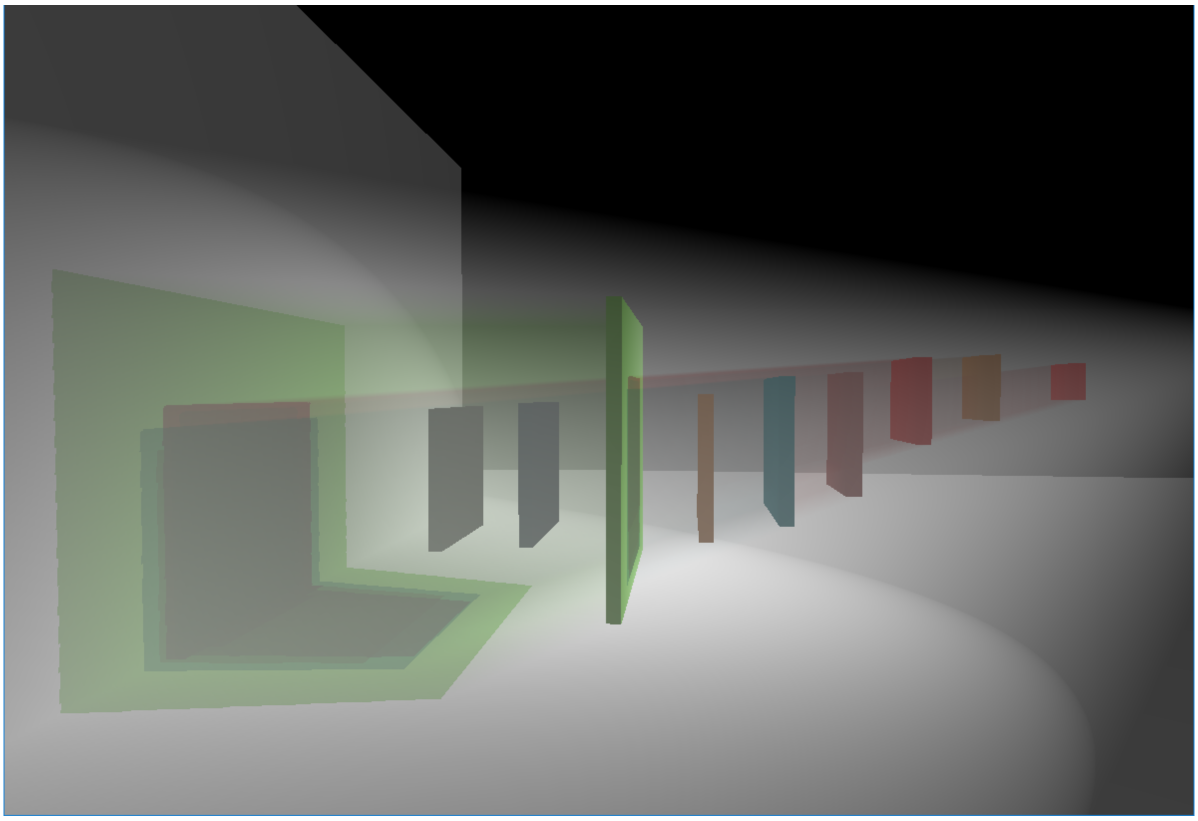


*Figure 41.* The Nine Planes scene using Depth Peeling, 9 peels are used to gain reference quality.

## MRT and MSTDZ versus Histogram MSTDZ

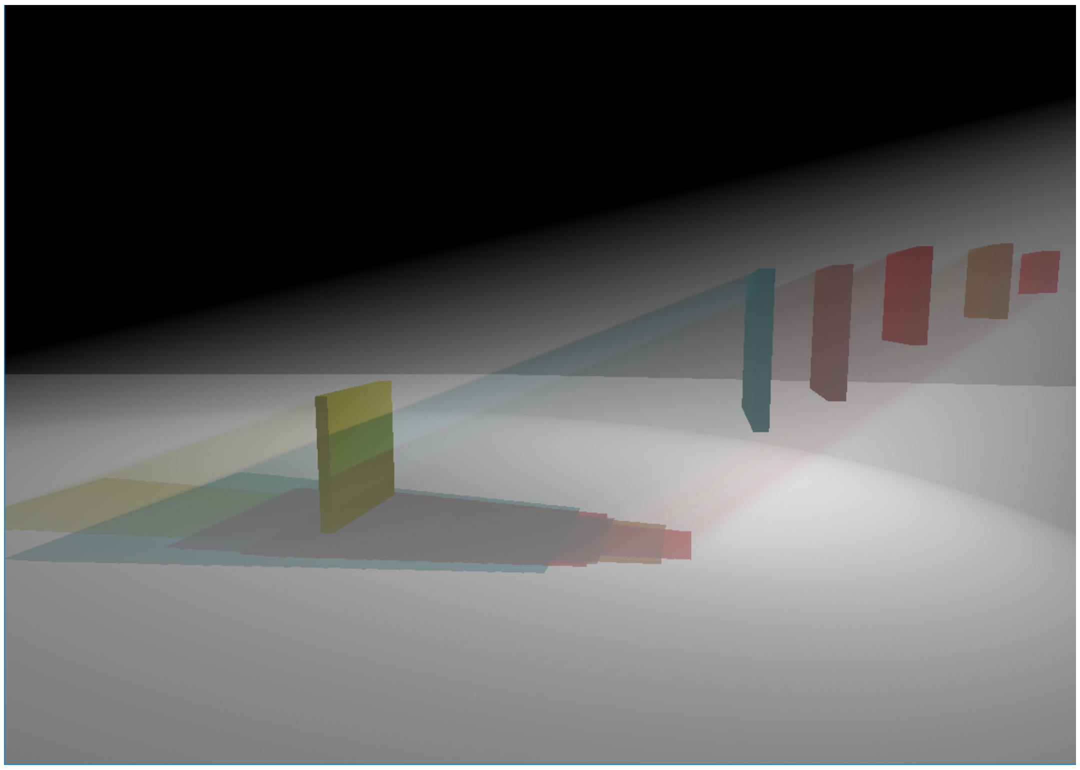


*Figure 42.* The Far-away Plane scene rendered using Depth Peeling. 6 peels were required to obtain reference quality.

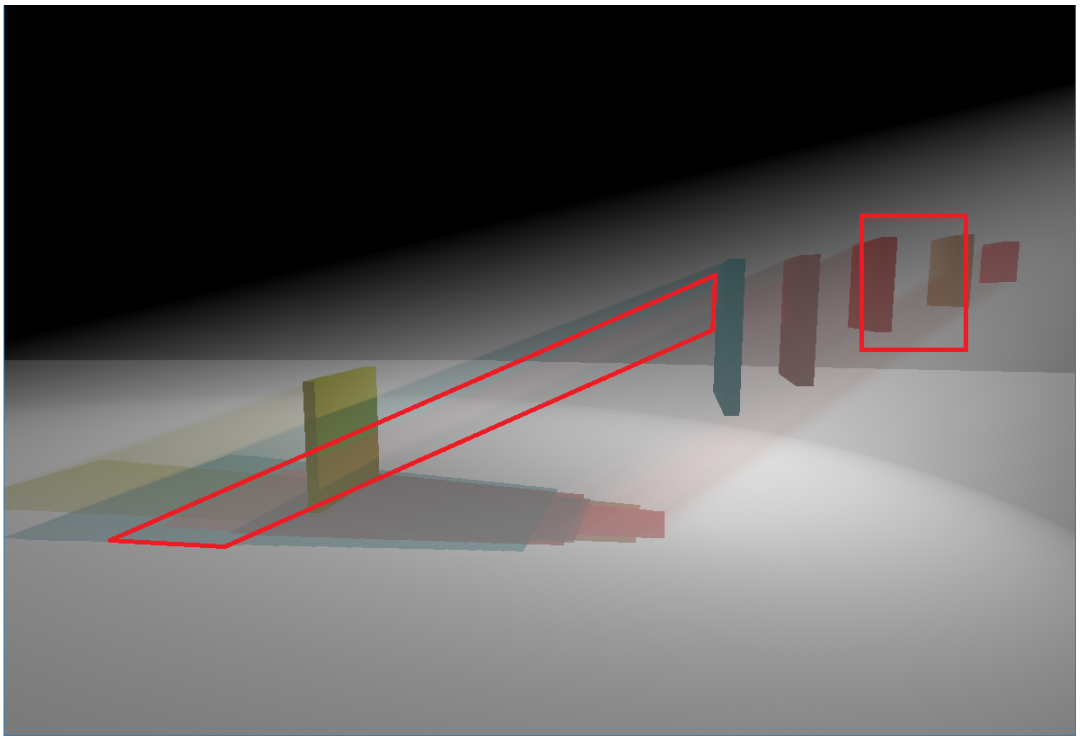


*Figure 42.* The Far-away Plane scene rendered using the MRT method, 6 zones. Obvious artifacts arise, which are outlined in red.

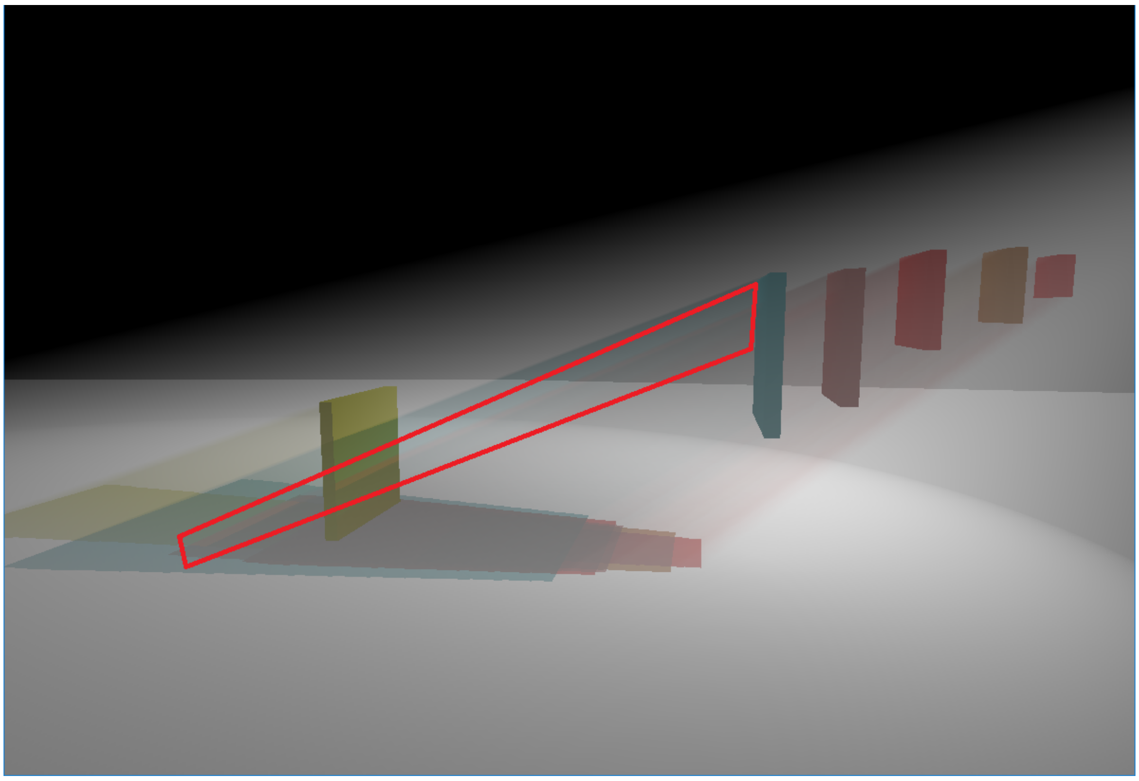


*Figure 42.* The Far-away Plane scene rendered using the MRT method, 8 zones. small artifacts still remain, which are outlined in red.

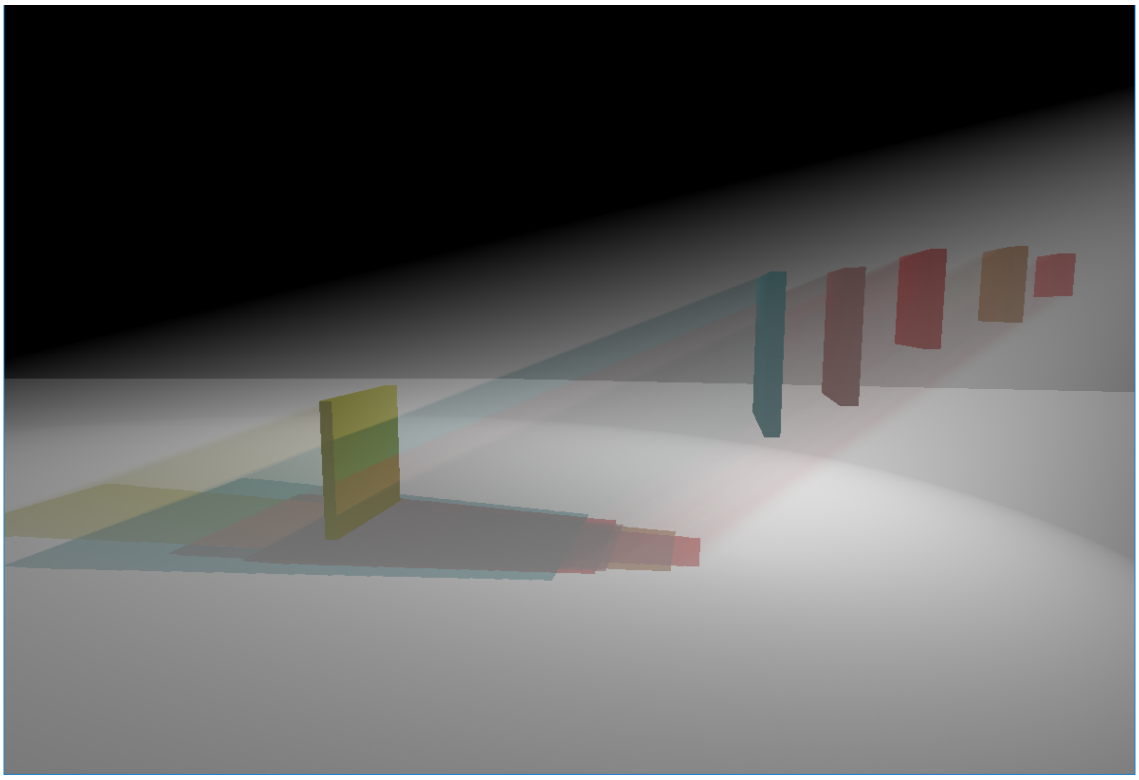


*Figure 42.* The Far-away Plane scene rendered using MSTDZ, 6 zones. The same artifact that arose using $MRT_6$ is observed.

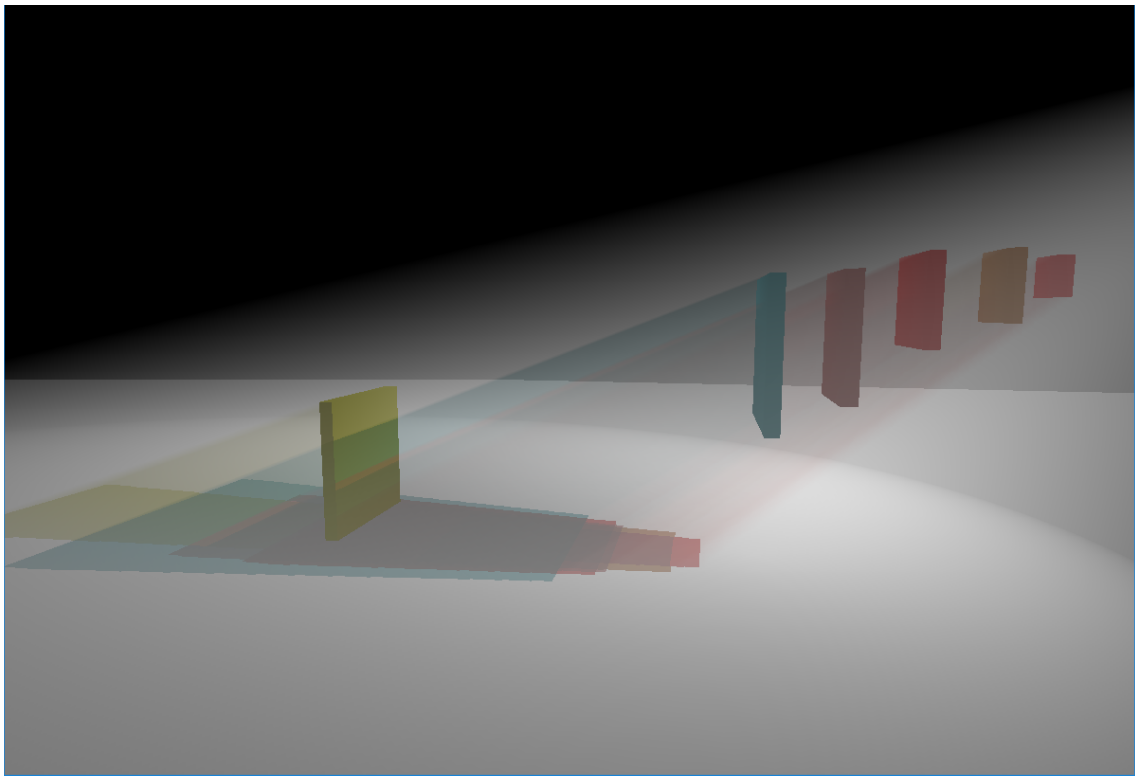


*Figure 42.* The Far-away Plane scene rendered using MSTDZ, 8 zones. The same artifact as in $MRT_8$ is observed.

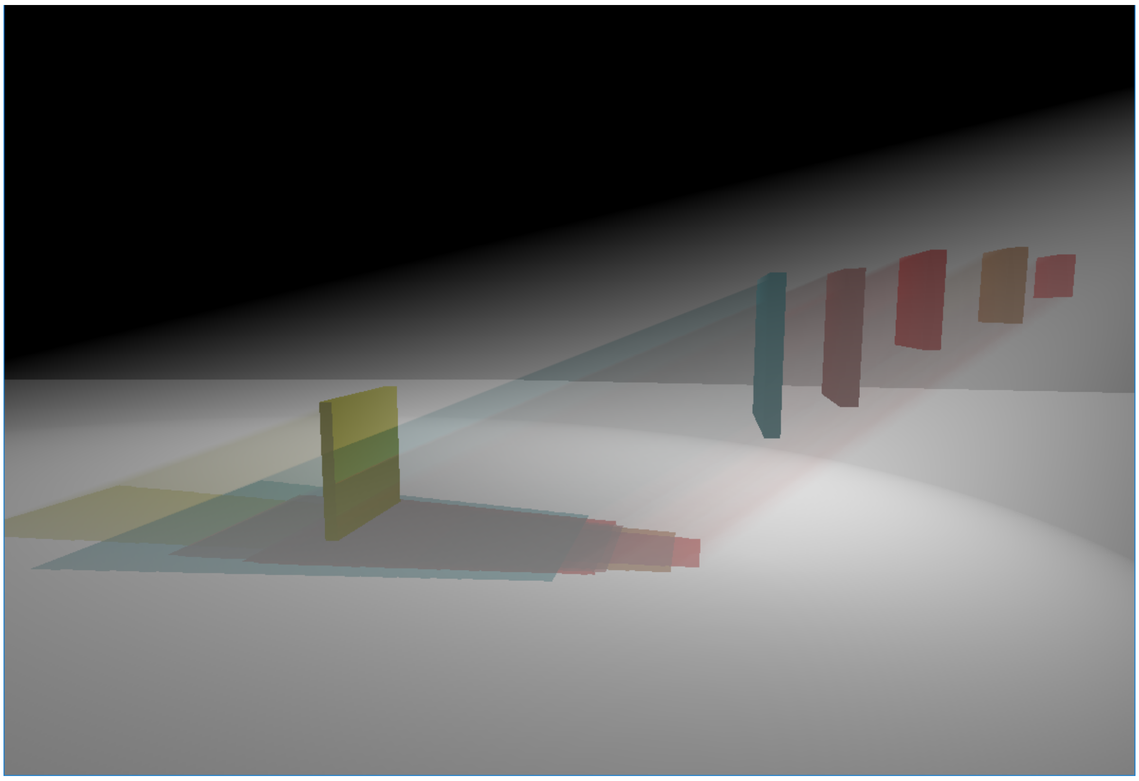


*Figure 42.* The Far-away Plane scene rendered using MSTDZ, 9 zones. No artifacts remain.

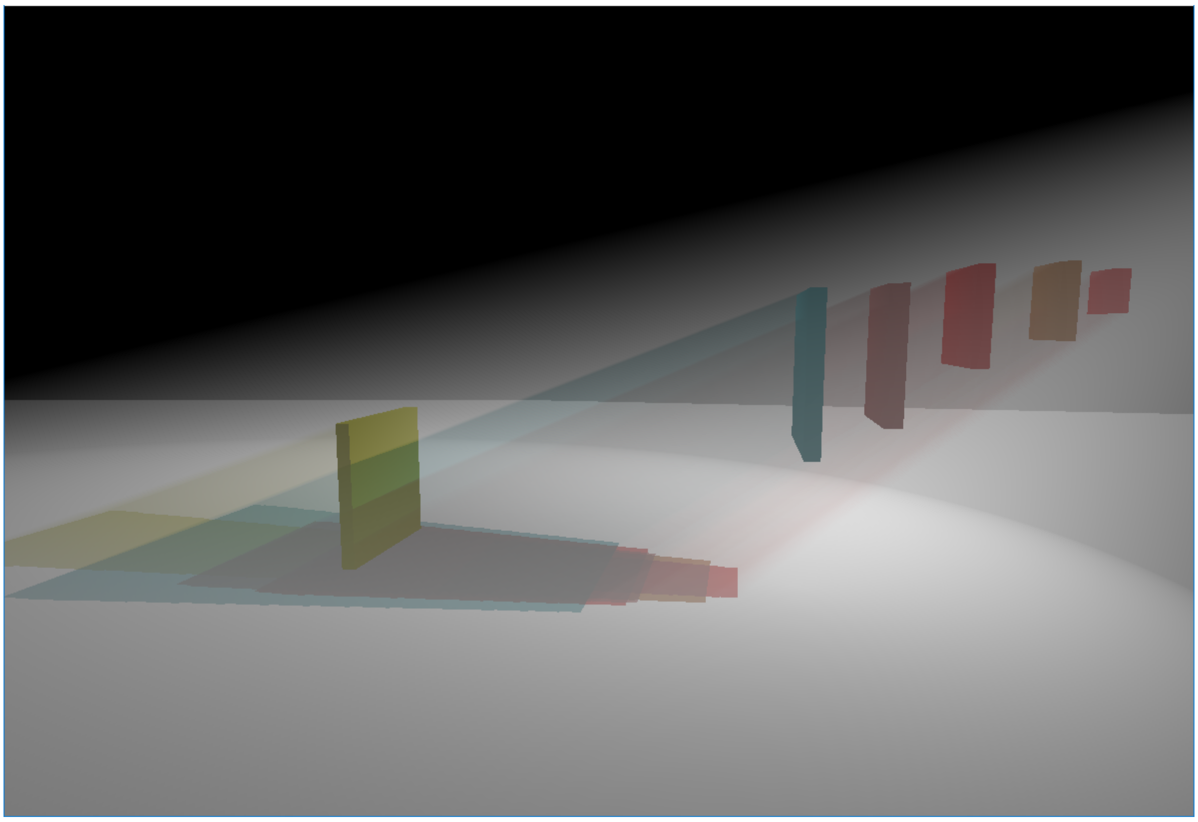


*Figure 42.* The Far-away Plane scene rendered using Histogram MSTDZ, 6 zones. Using fewer zones than with MRT or MSTDZ, we are able to obtain reference quality.

## Comparing Depth Peeling to MSTDZ, Histo MSTDZ and Texel Groups

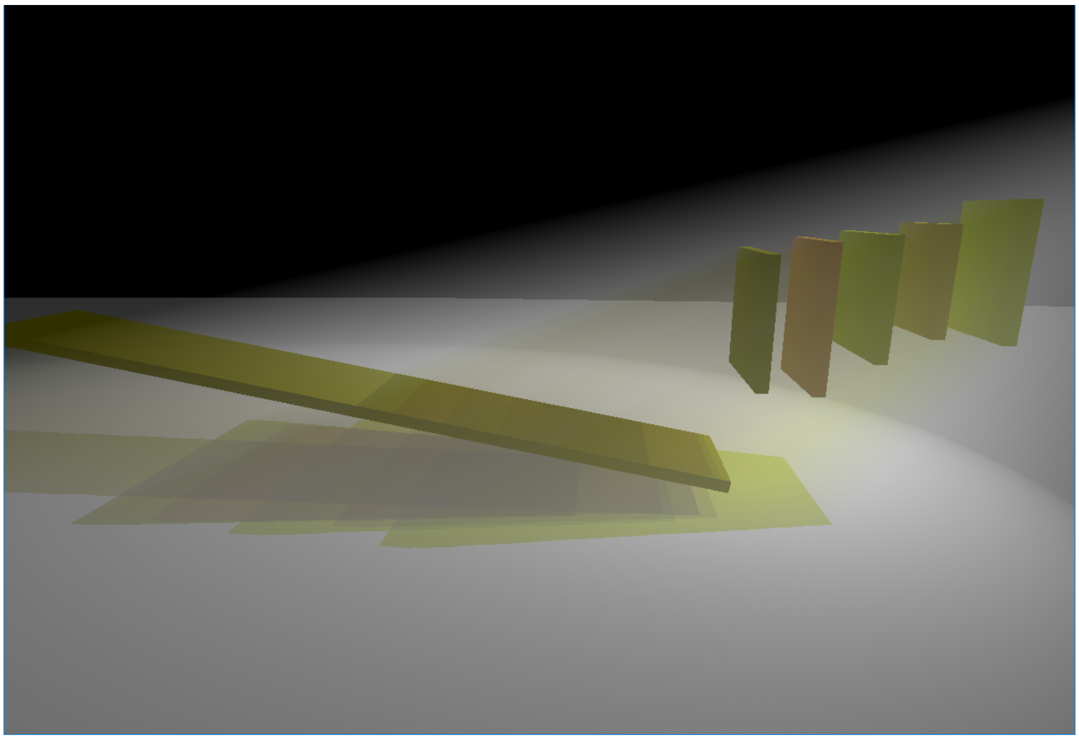


*Figure 47.* The Yellow Planes scene rendered using Depth Peeling, 6 peels were required to obtain reference quality.

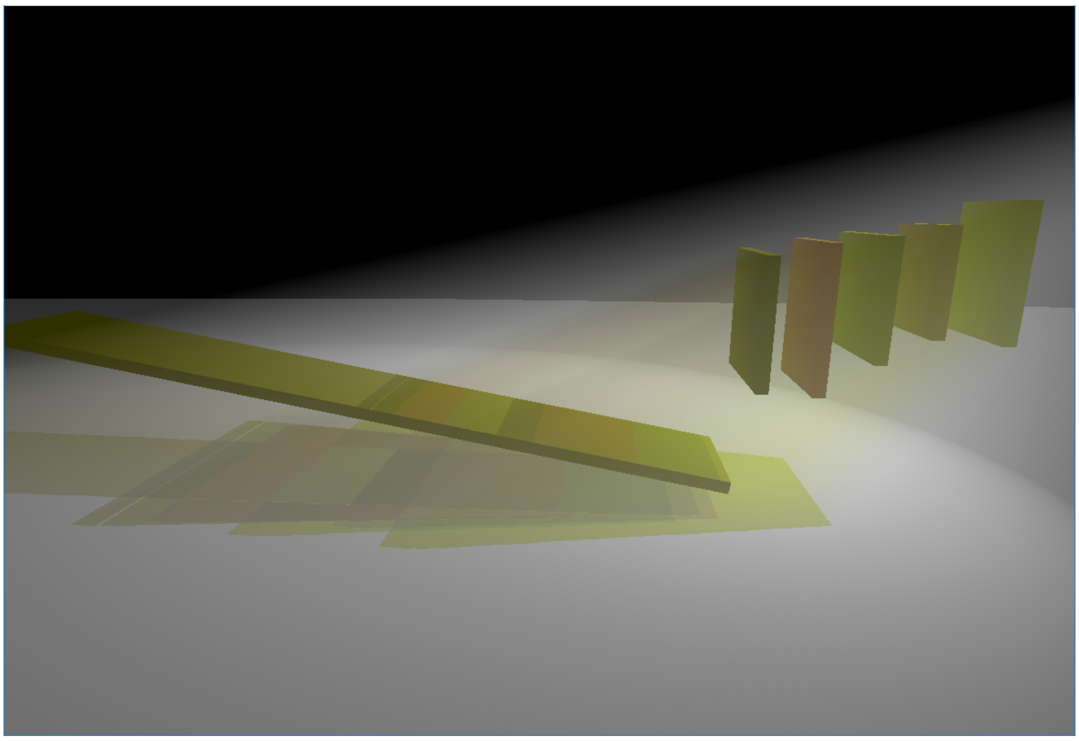


*Figure 47.* The Yellow Planes scene rendered using MSTDZ, 3 zones.

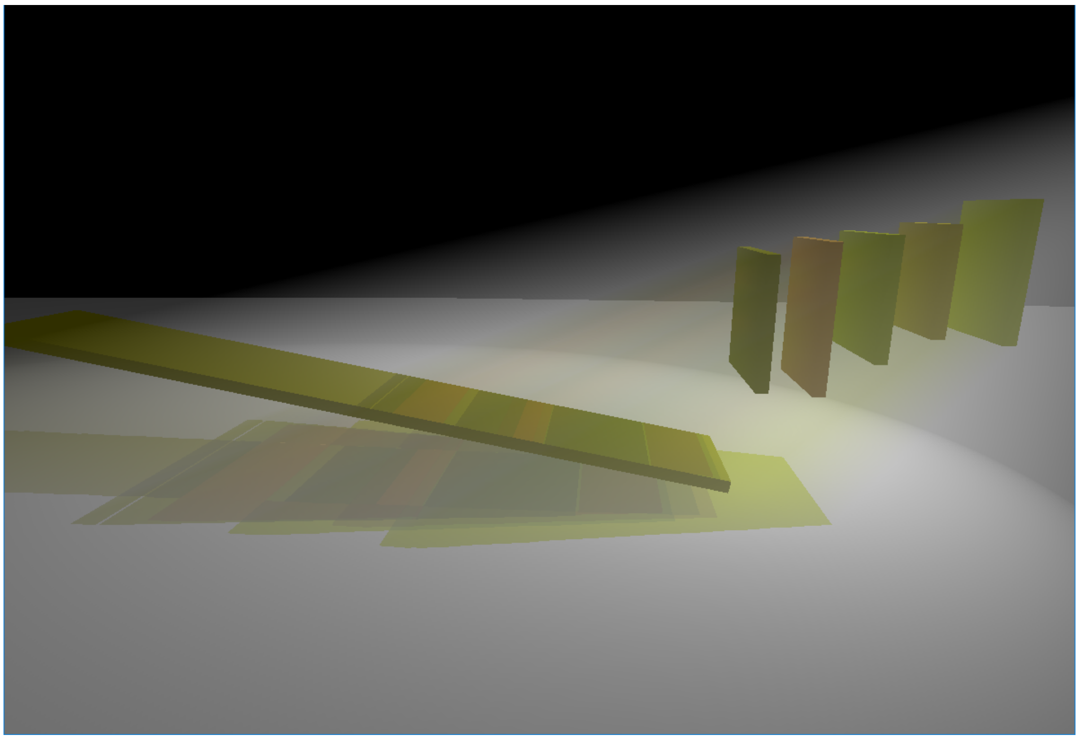


*Figure 47.* The Yellow Planes scene rendered using MSTDZ, 4 zones.

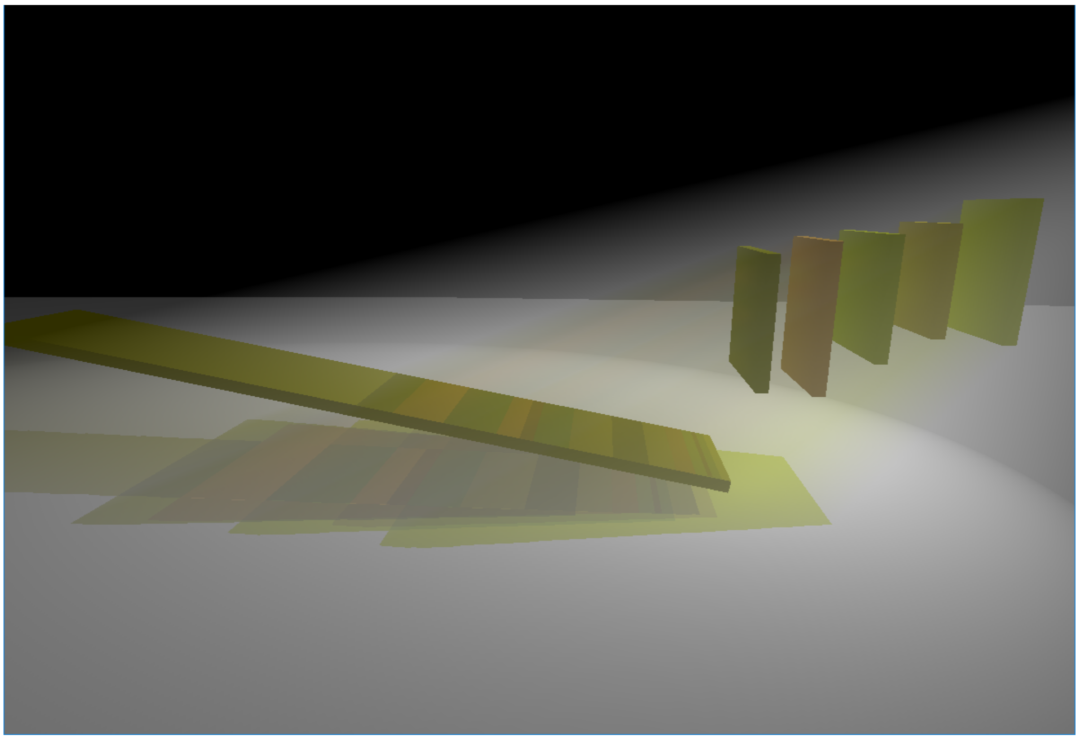


*Figure 47.* The Yellow Planes scene rendered using MSTDZ, 5 zones.

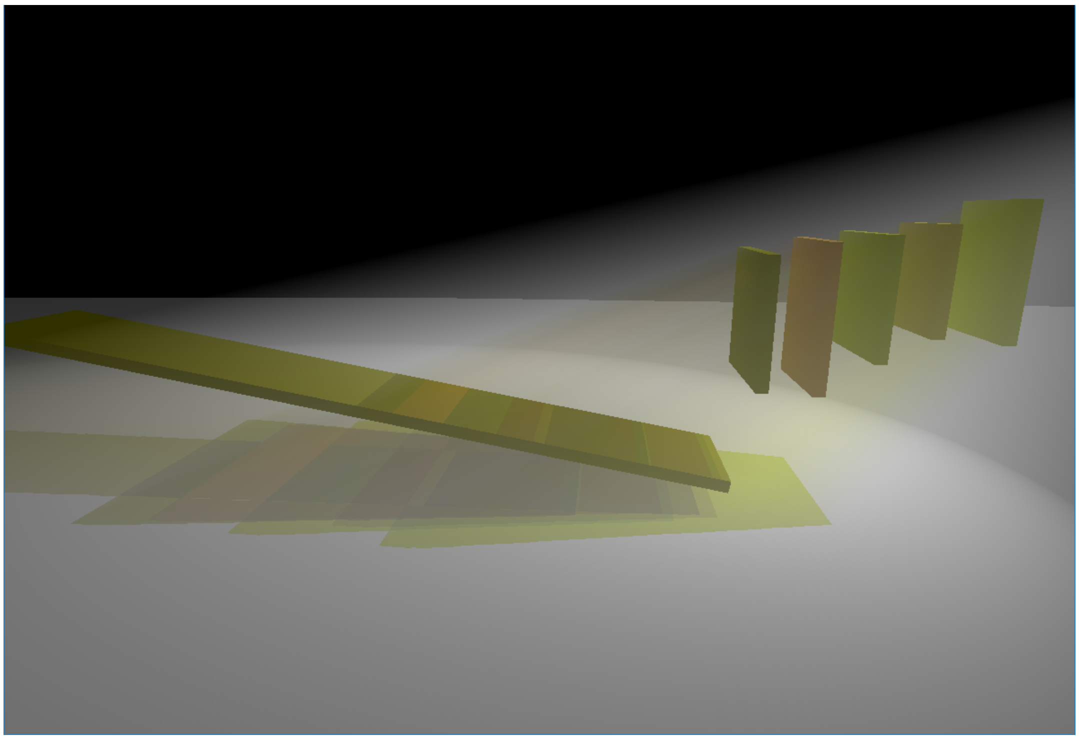


*Figure 47.* The Yellow Planes scene rendered using MSTDZ, 6 zones.

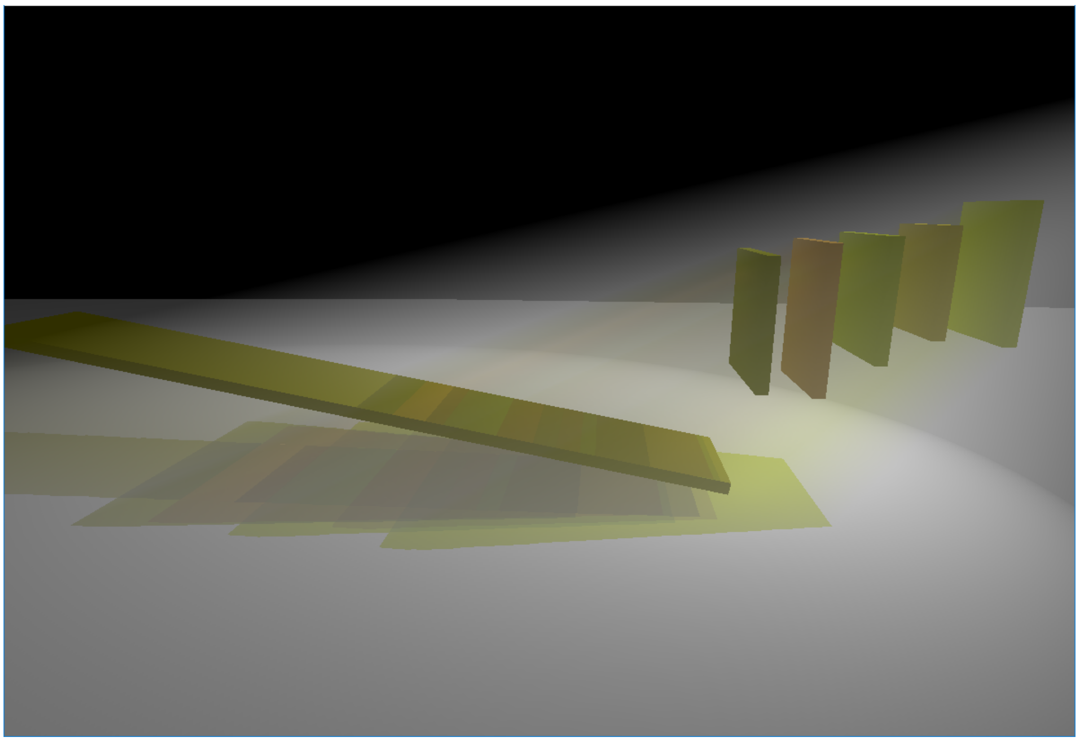


*Figure 47.* The Yellow Planes scene rendered using MSTDZ, 7 zones.

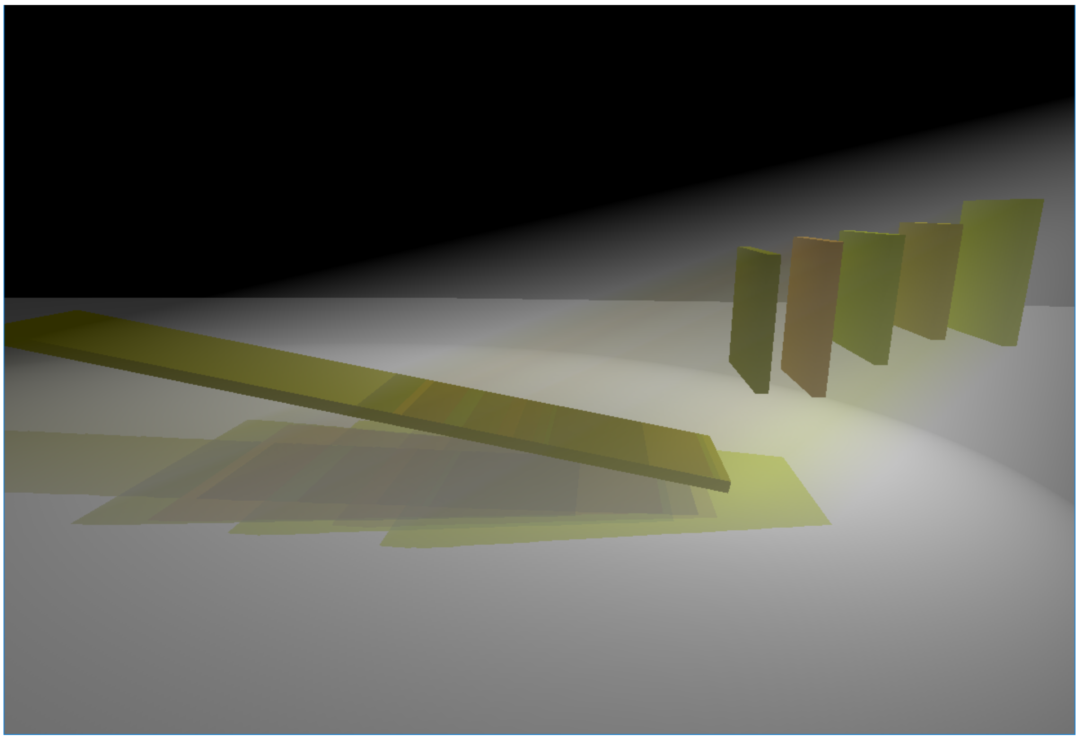


*Figure 47.* The Yellow Planes scene rendered using MSTDZ, 8 zones.

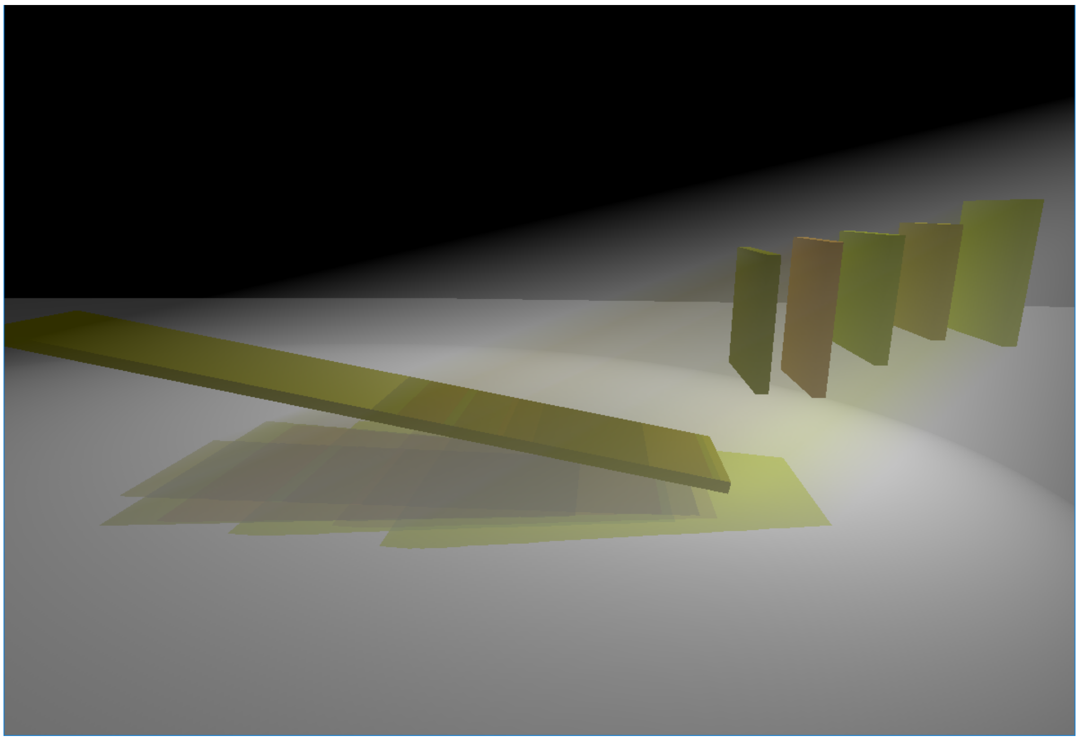


*Figure 47.* The Yellow Planes scene rendered using MSTDZ, 9 zones.

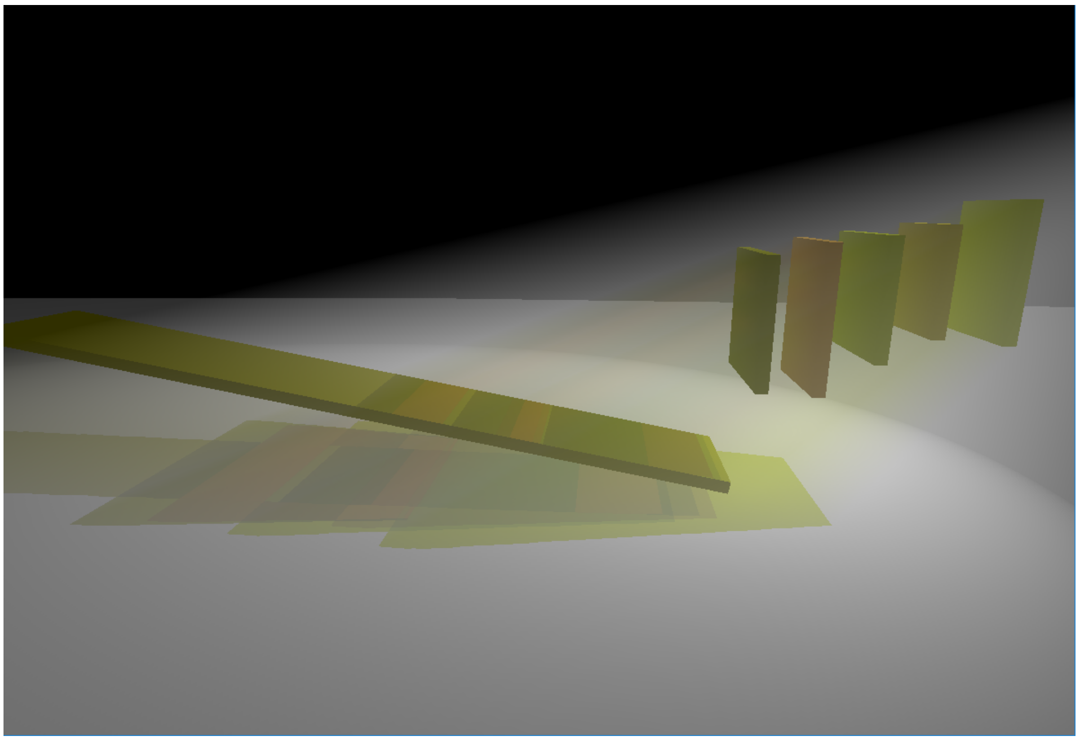


*Figure 48.* The Yellow Planes scene rendered using Histogram MSTDZ, 3 zones.

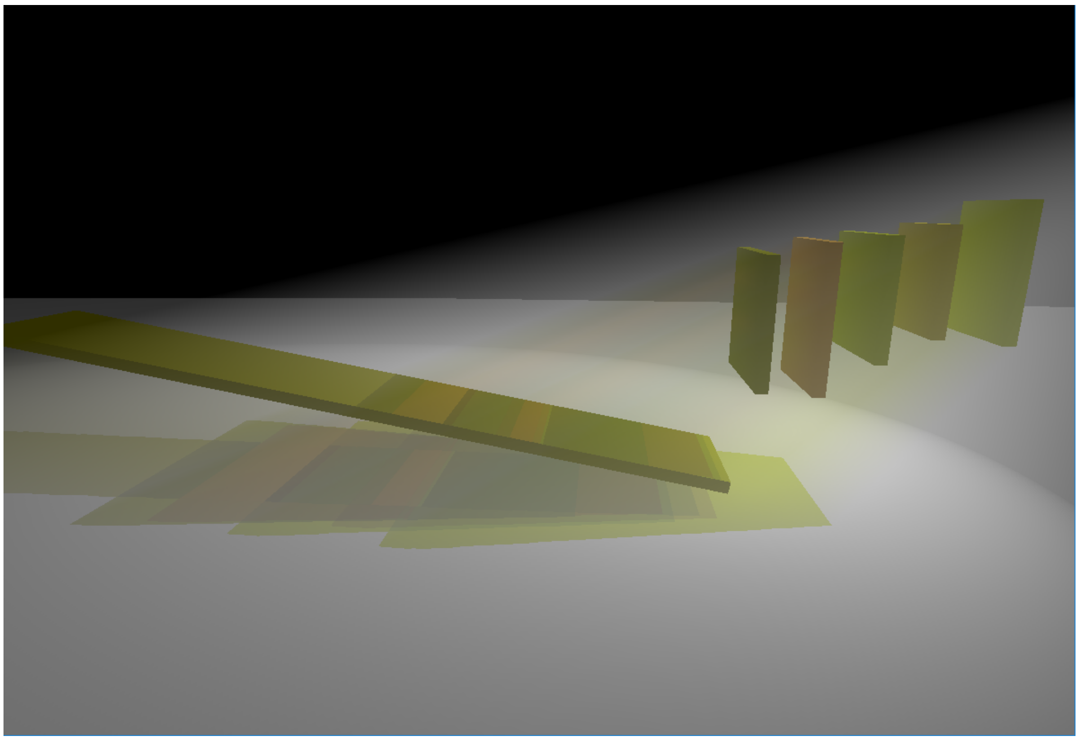


*Figure 48.* The Yellow Planes scene rendered using Histogram MSTDZ, 4 zones.

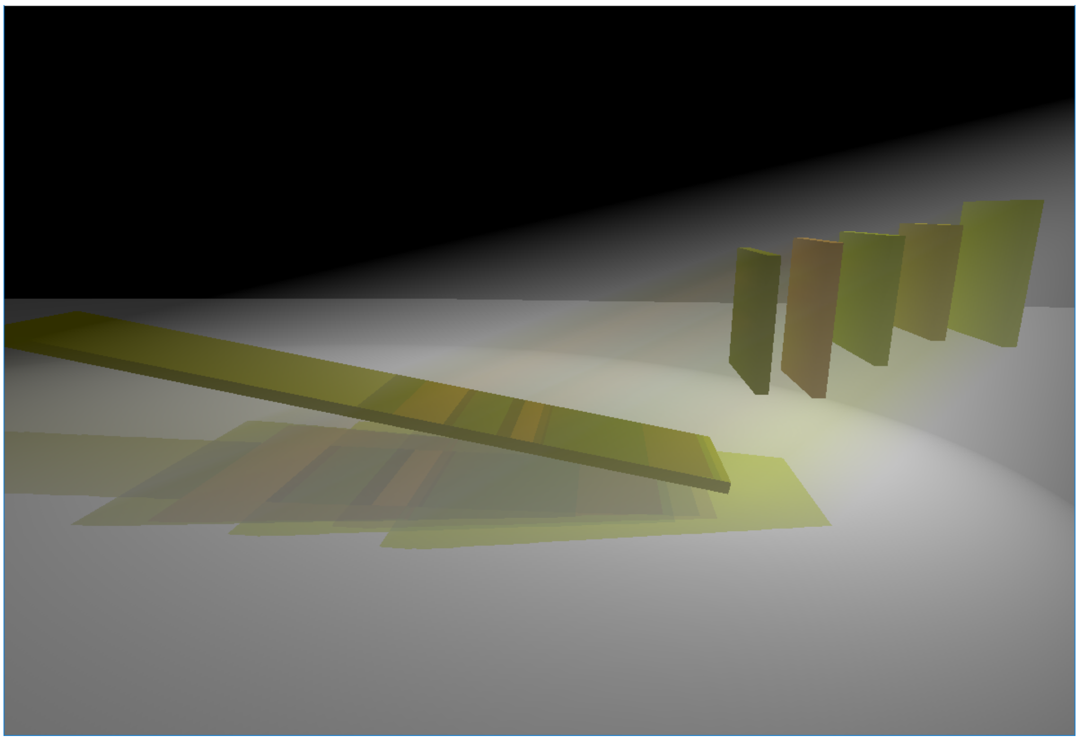


*Figure 48.* The Yellow Planes scene rendered using Histogram MSTDZ, 5 zones.

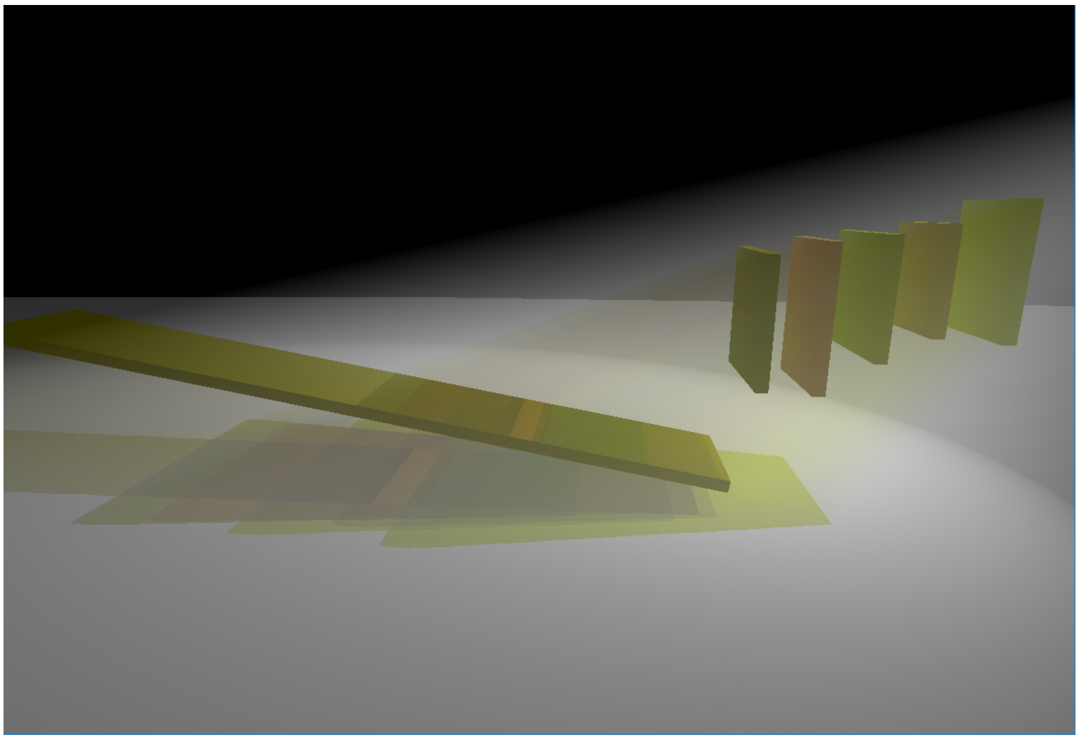


*Figure 48.* The Yellow Planes scene rendered using Histogram MSTDZ, 6 zones.

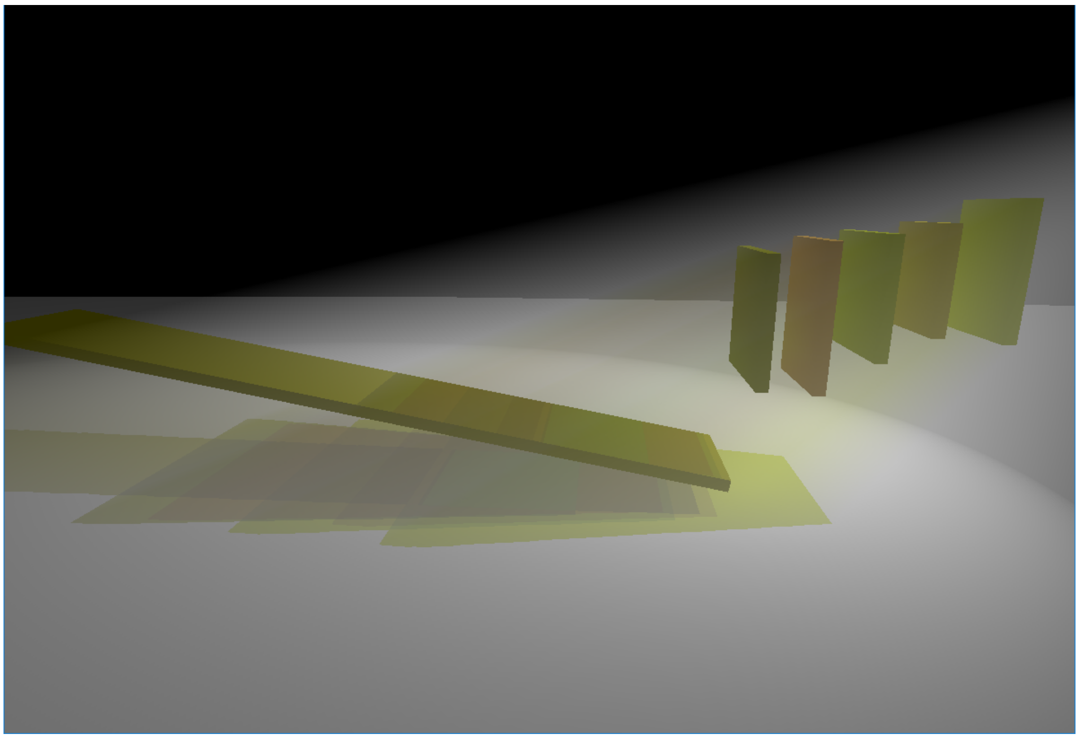


*Figure 48.* The Yellow Planes scene rendered using Histogram MSTDZ, 7 zones.

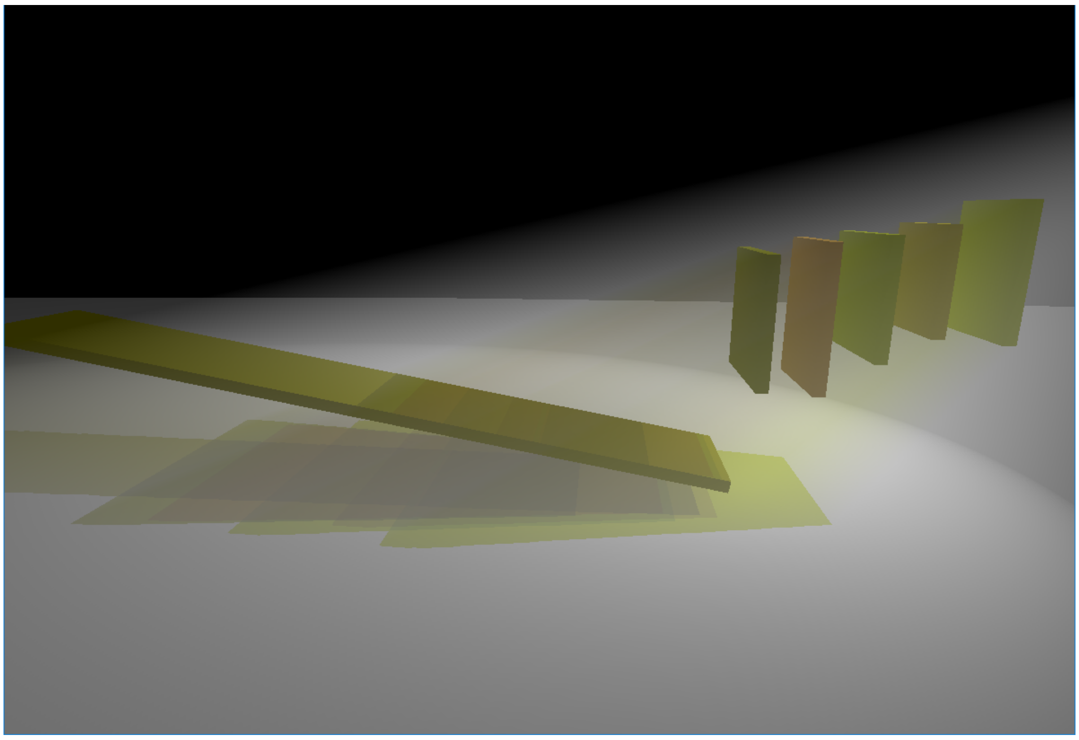


*Figure 48.* The Yellow Planes scene rendered using Histogram MSTDZ, 8 zones.

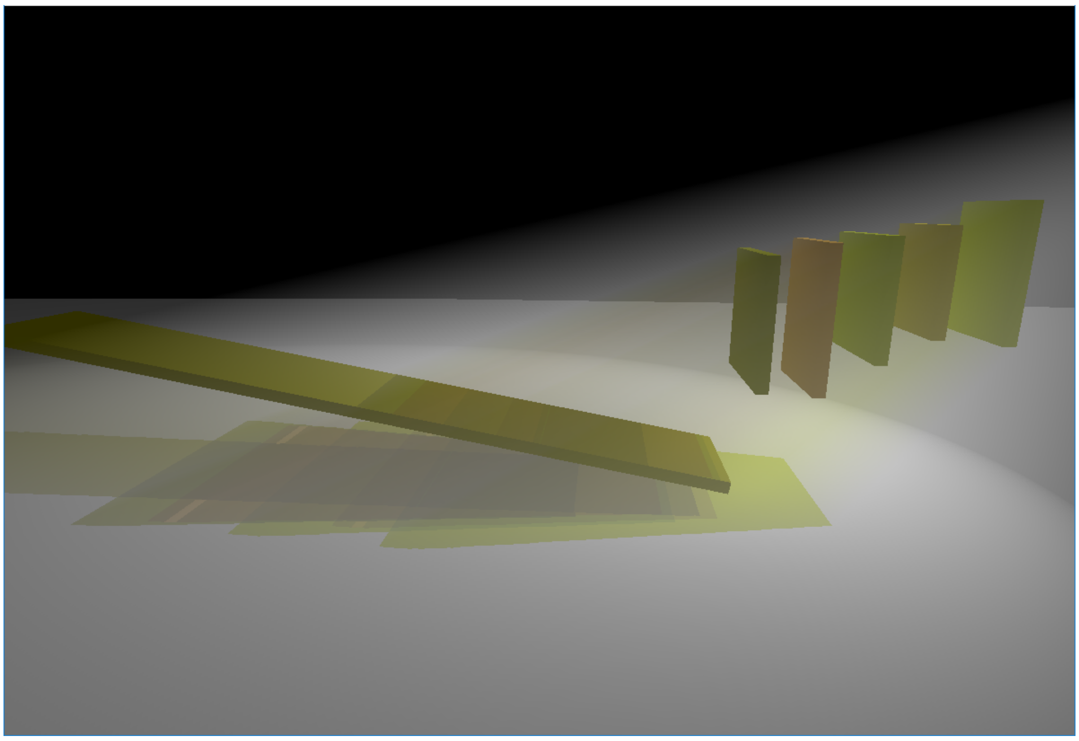


*Figure 48.* The Yellow Planes scene rendered using Histogram MSTDZ, 9 zones.

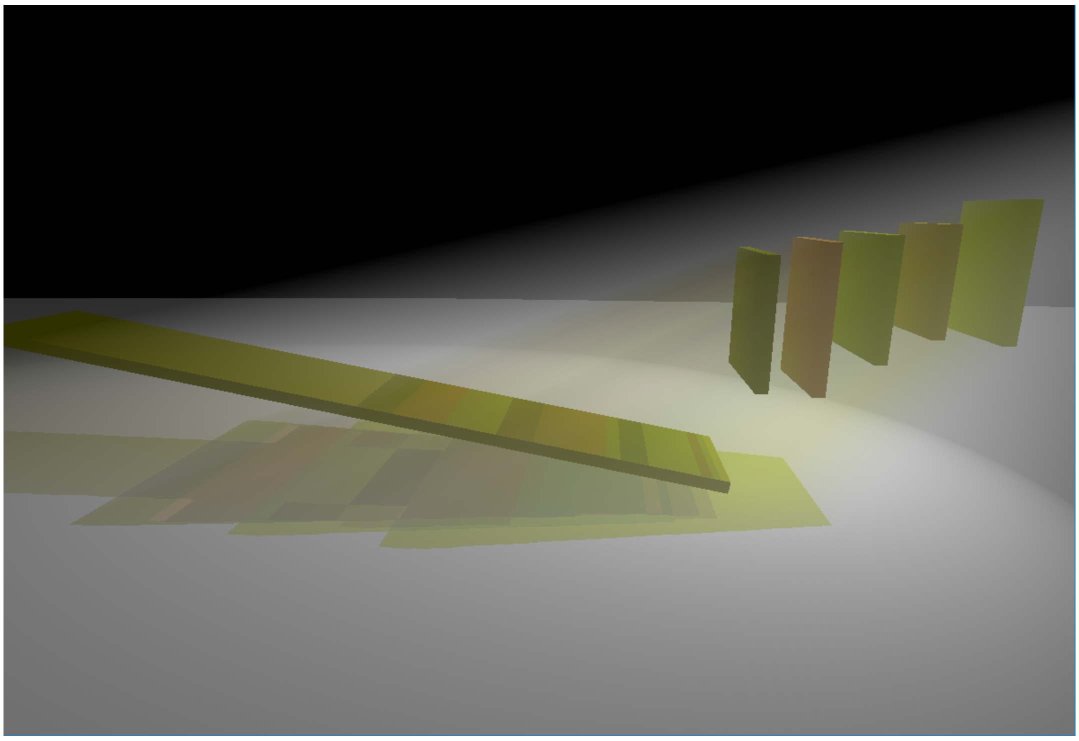


*Figure 49.* The Yellow Planes scene rendered using Histogram MSTDZ with Texel Groups, 3/3/3 zones.

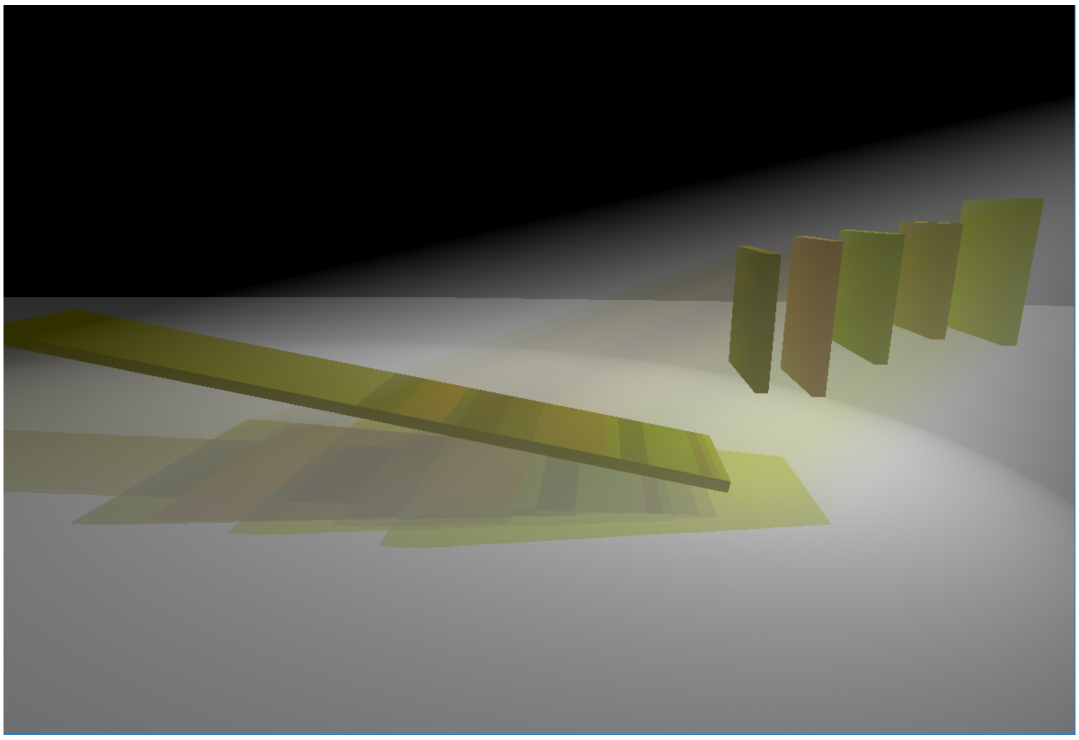


*Figure 49.* The Yellow Planes scene rendered using Histogram MSTDZ with Texel Groups, 4/4/4 zones.

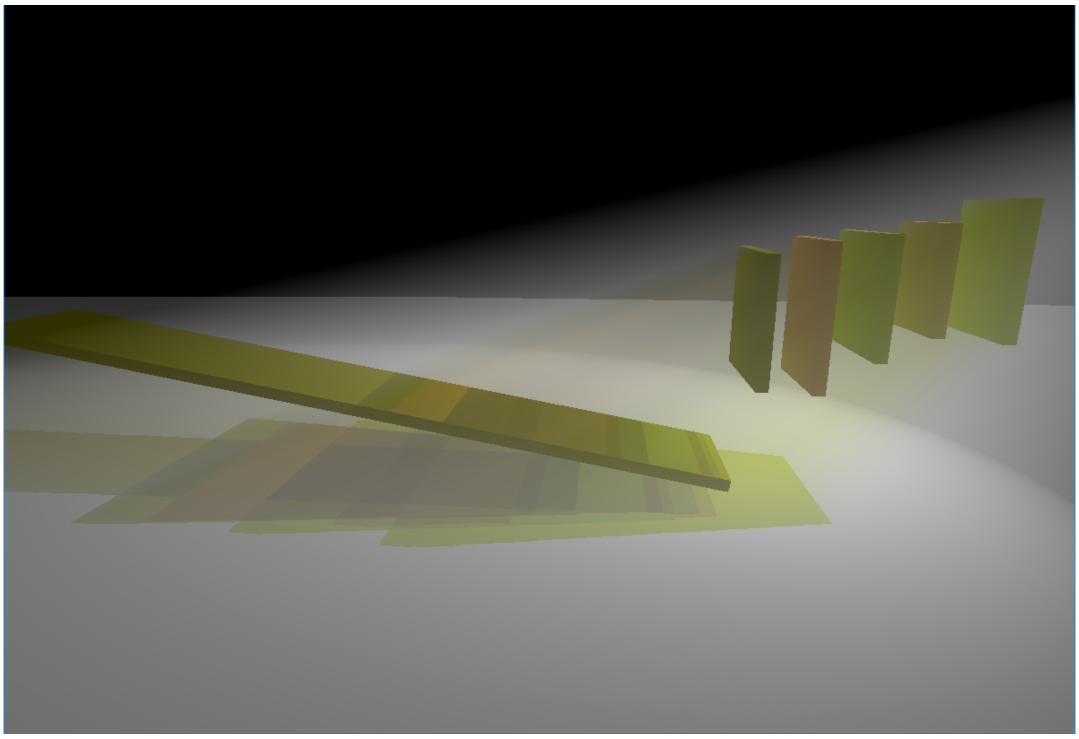


*Figure 49.* The Yellow Planes scene rendered using Histogram MSTDZ with Texel Groups, 5/5/5 zones.

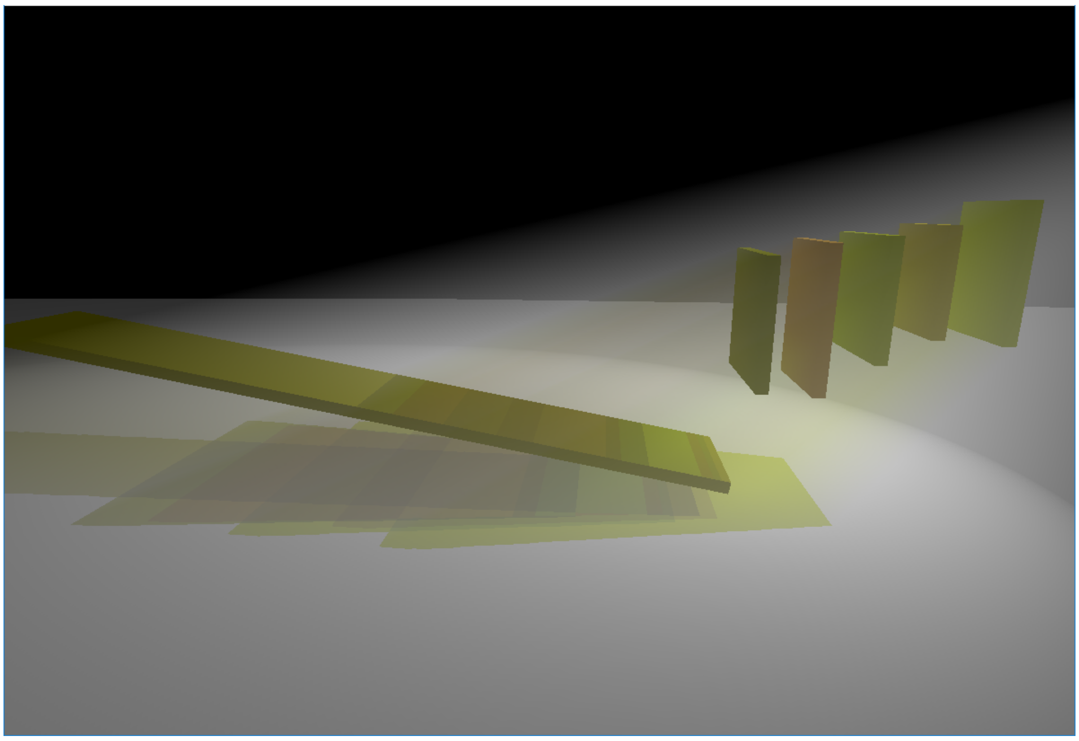


*Figure 49.* The Yellow Planes scene rendered using Histogram MSTDZ with Texel Groups, 6/6/6 zones.

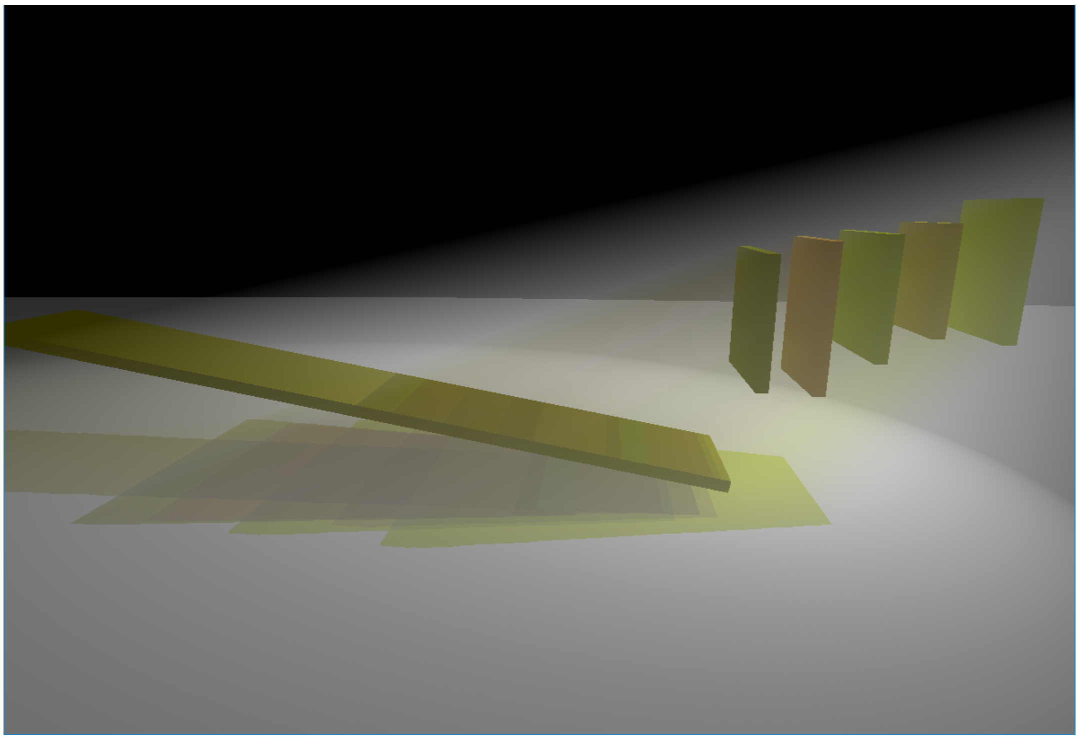


*Figure 49.* The Yellow Planes scene rendered using Histogram MSTDZ with Texel Groups, 7/7/7 zones.

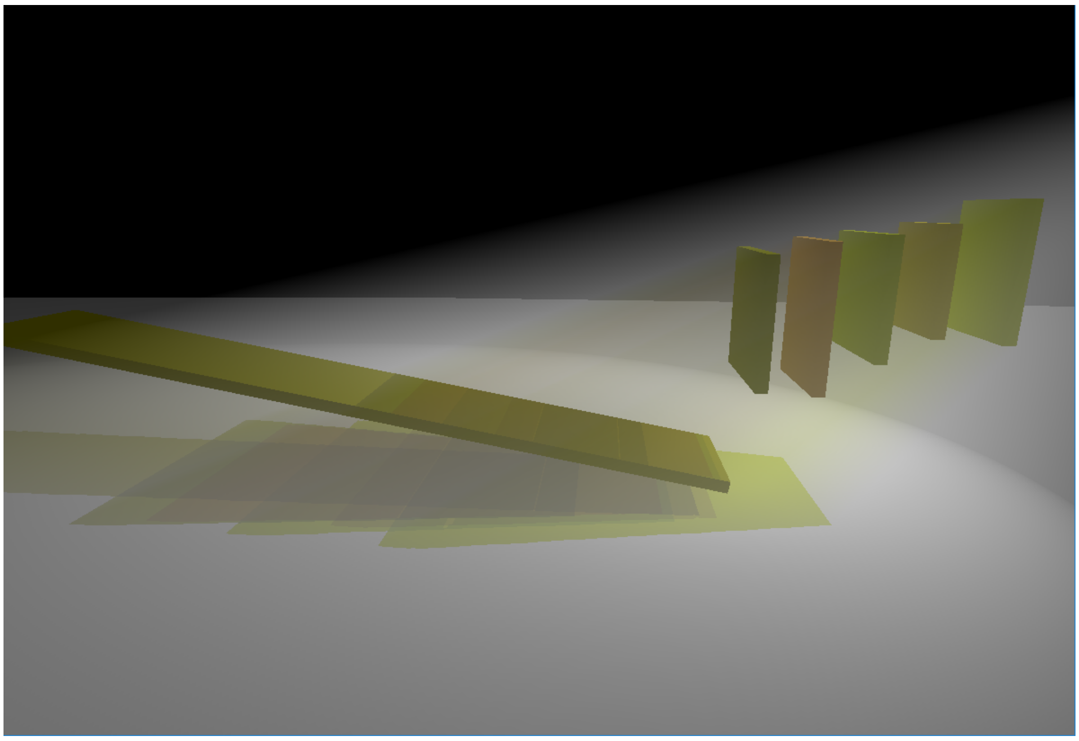


*Figure 49.* The Yellow Planes scene rendered using Histogram MSTDZ with Texel Groups, 9/9/9 zones.

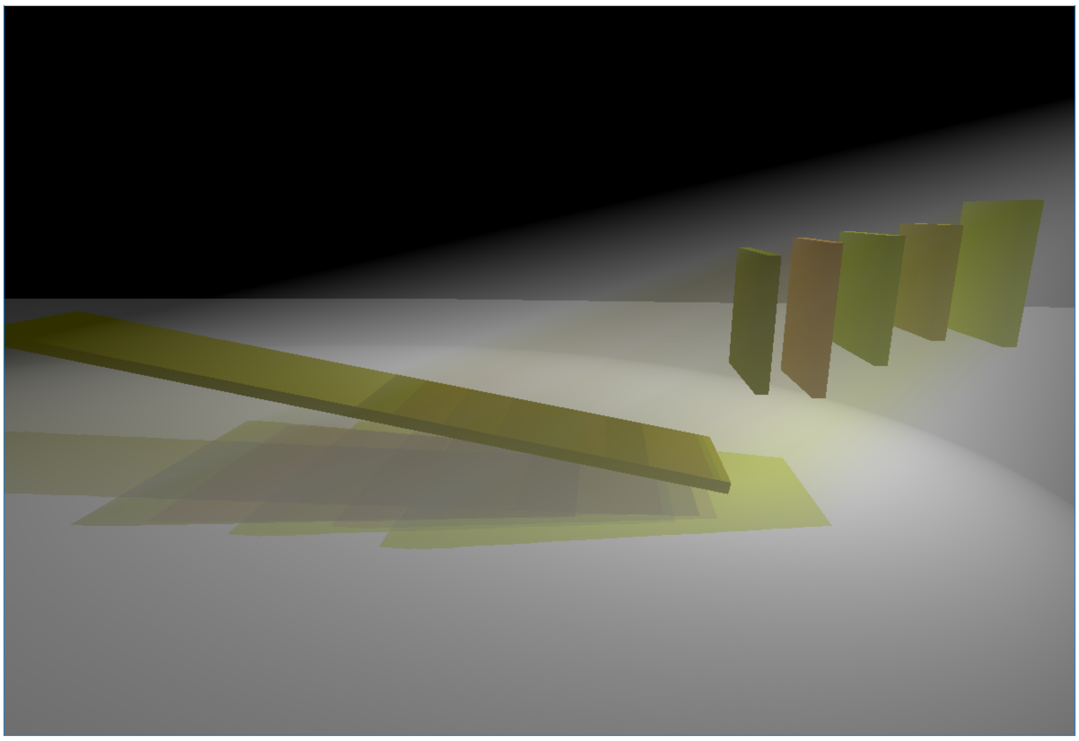


*Figure 50.* The Yellow Planes scene rendered using Histogram MSTDZ with Texel Groups, 8/8/8 zones.

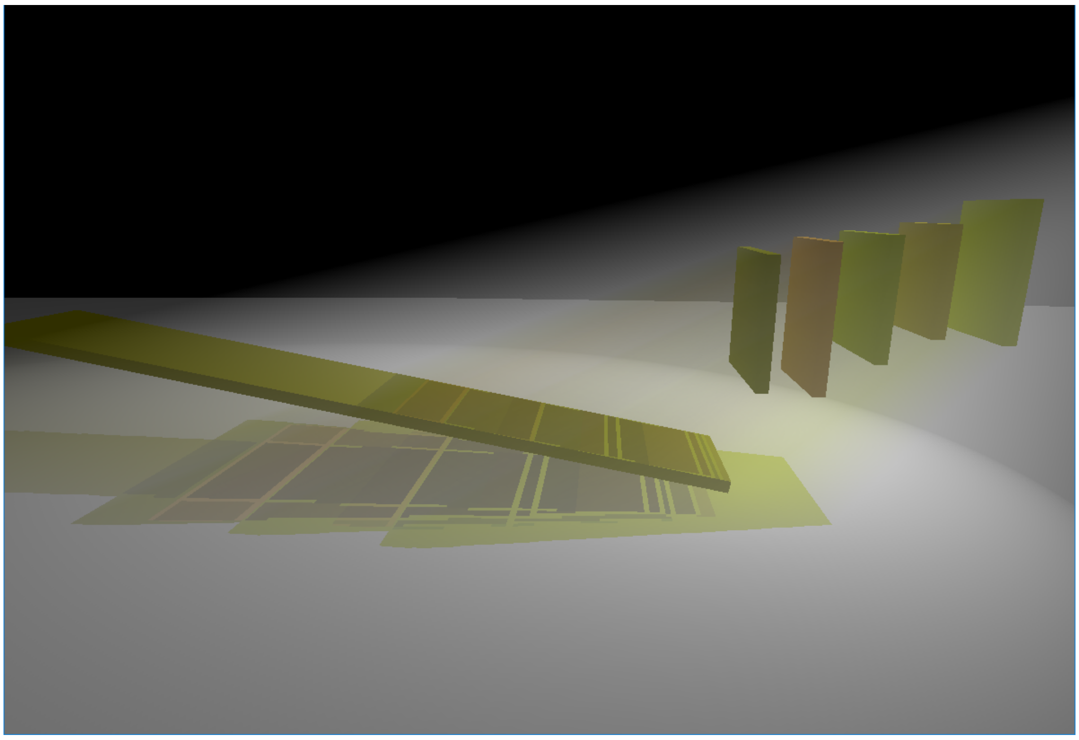


*Figure 50.* The Yellow Planes scene rendered using Histogram MSTDZ with Texel Groups, 1/1/8 zones.

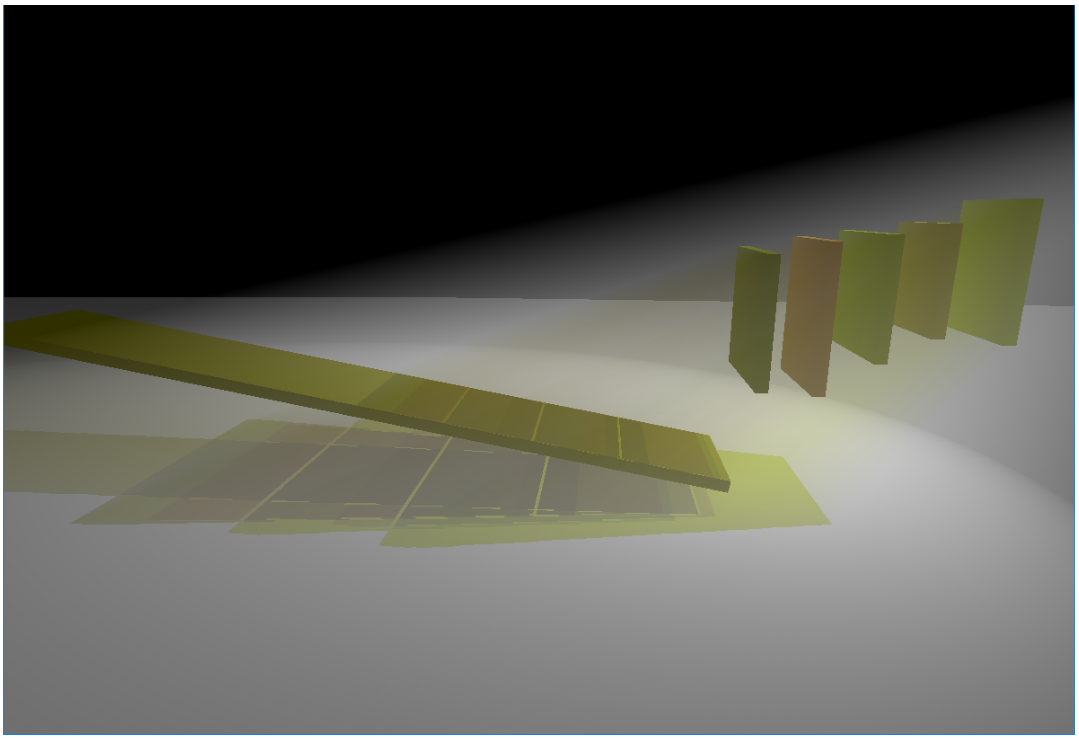


*Figure 50.* The Yellow Planes scene rendered using Histogram MSTDZ with Texel Groups, 1/8/8 zones.

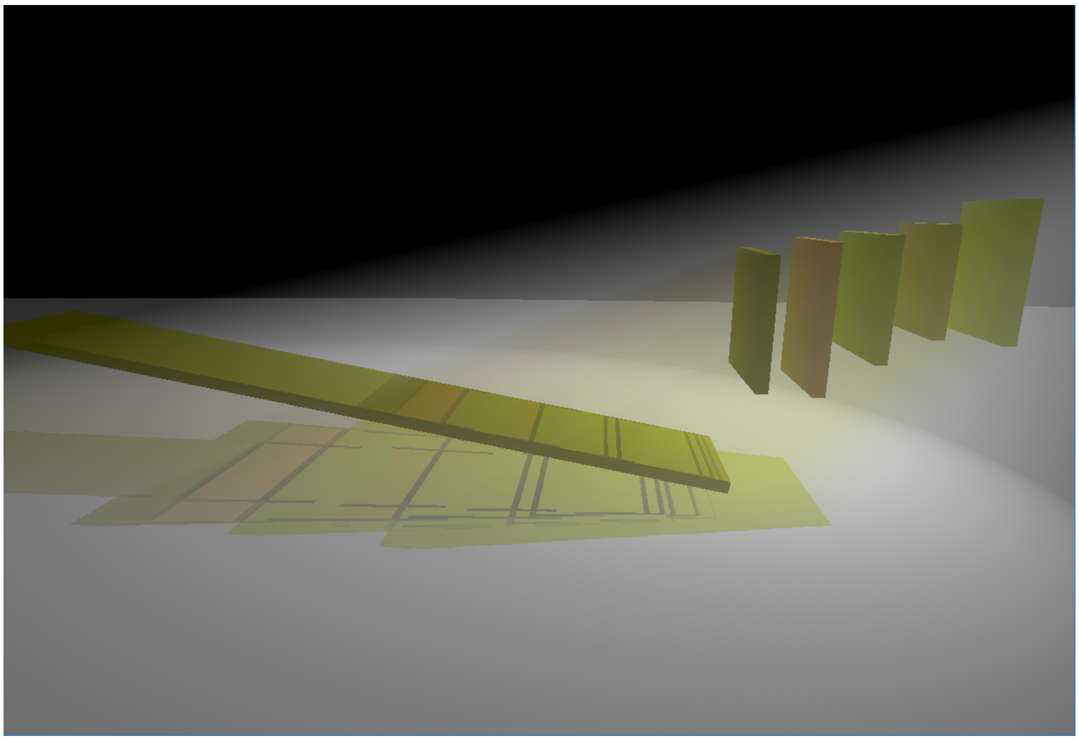


*Figure 50.* The Yellow Planes scene rendered using Histogram MSTDZ with Texel Groups, 8/8/1 zones.

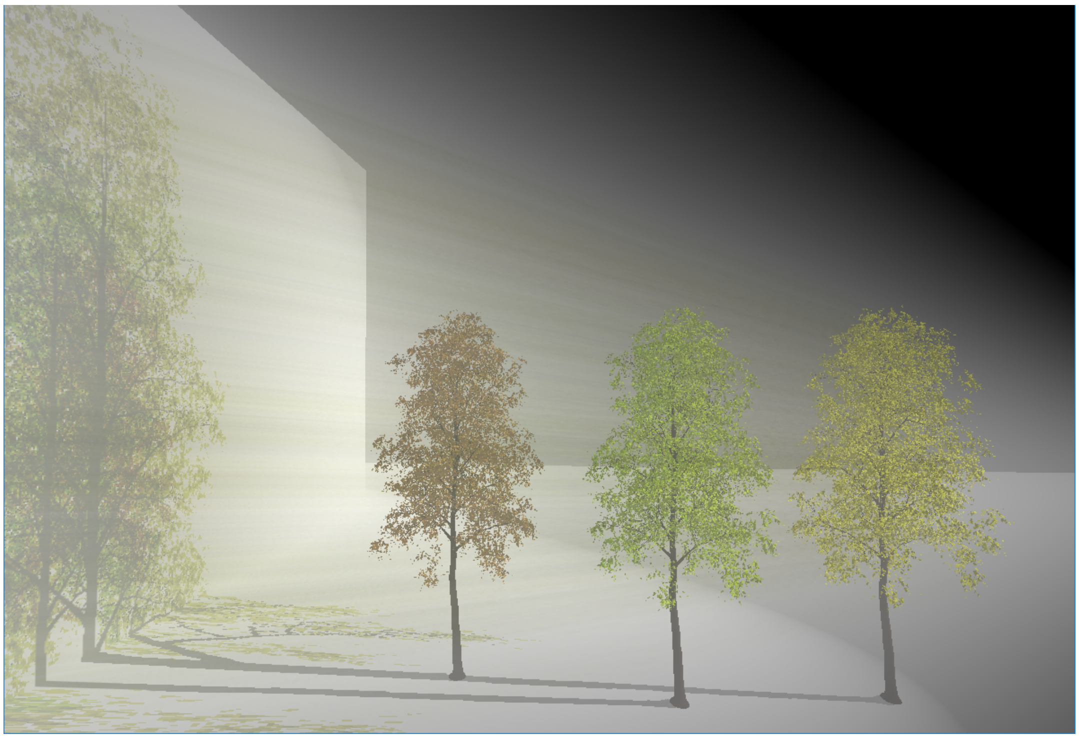


*Figure 53.* The Foliage scene rendered using Depth Peeling, 12 peels were required to obtain reference quality.

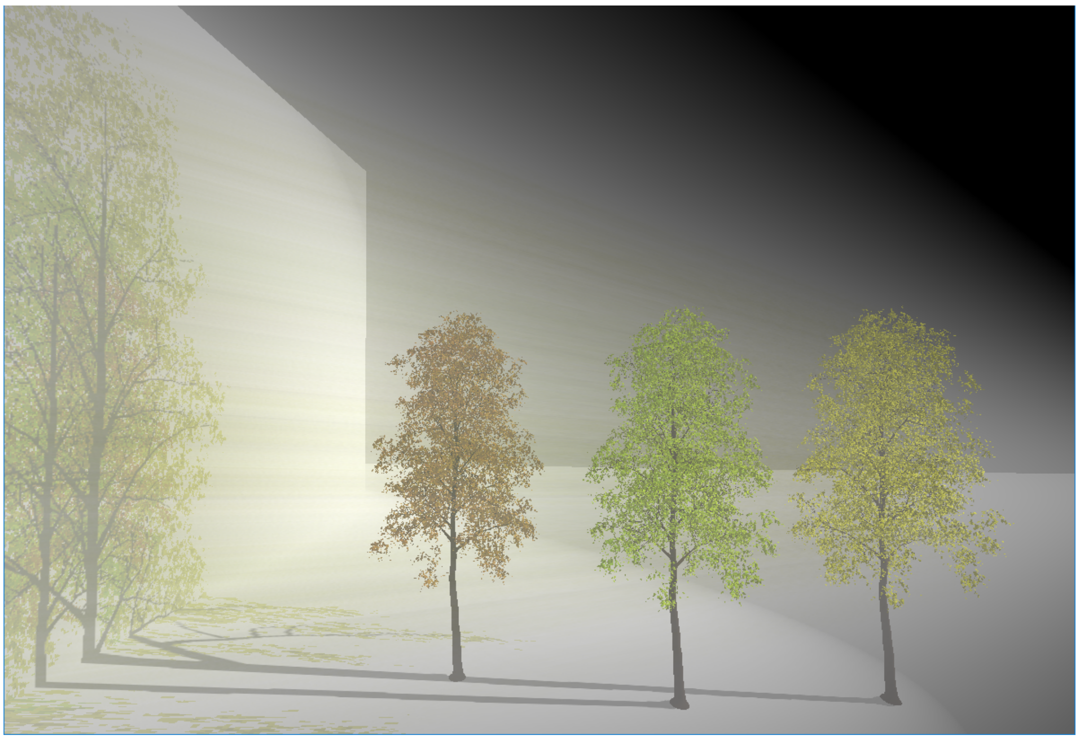


*Figure 53.* The Foliage scene rendered using MSTDZ, 3 zones.

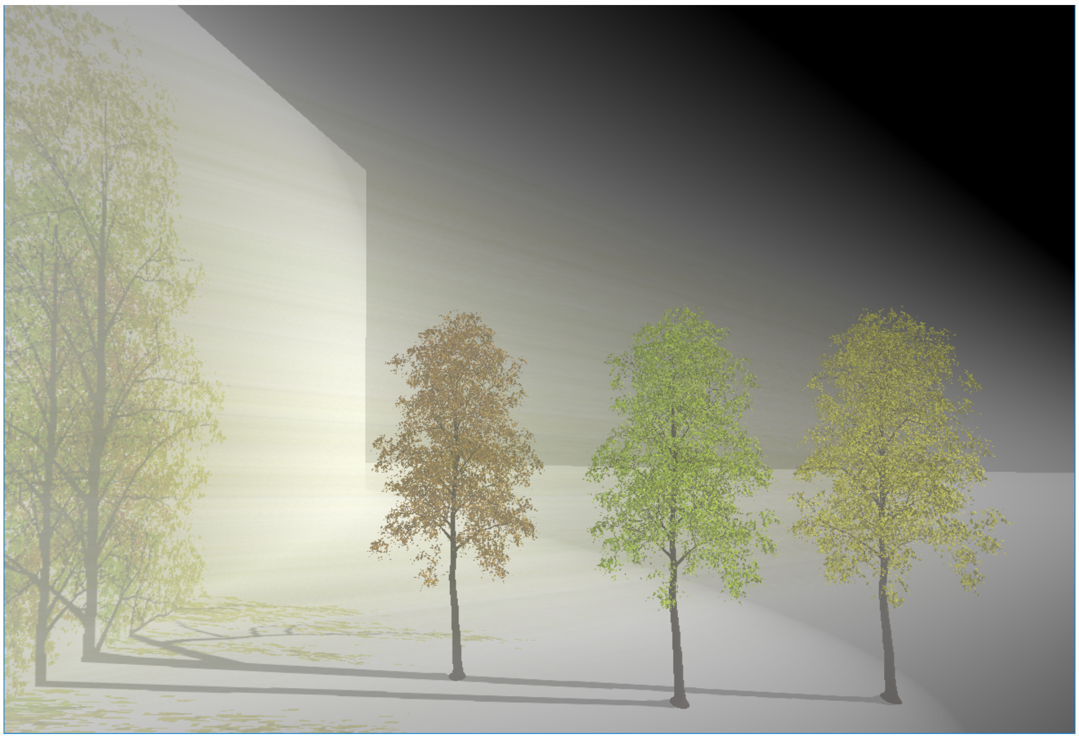


*Figure 53.* The Foliage scene rendered using MSTDZ, 4 zones.

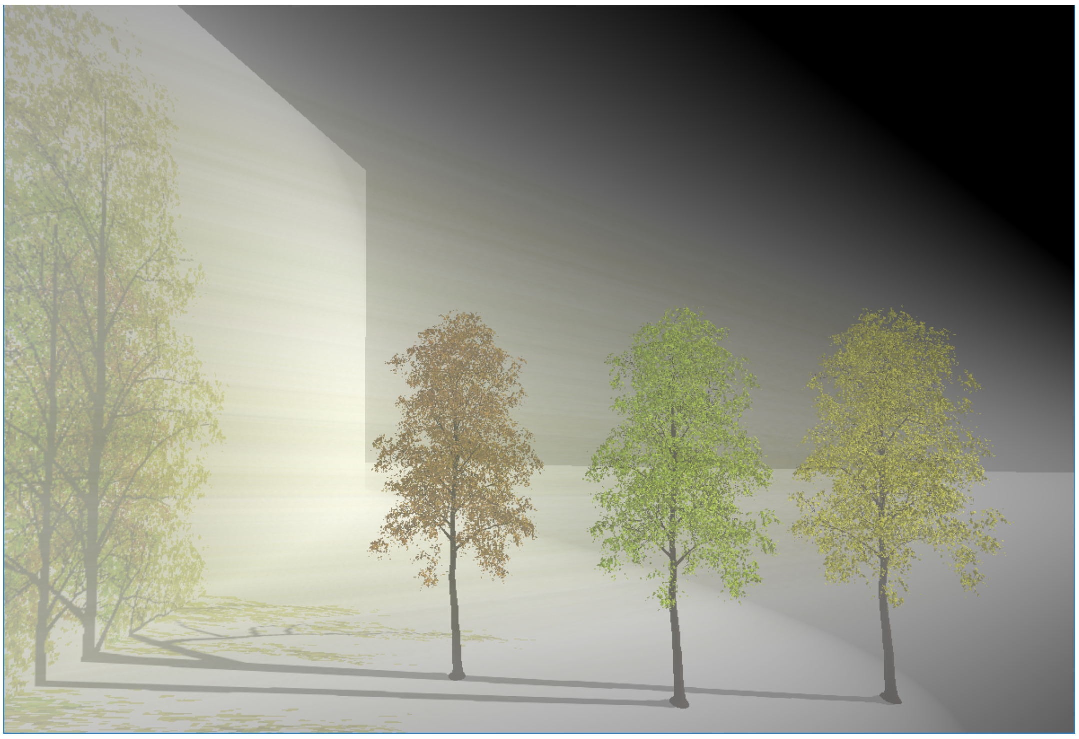


*Figure 53.* The Foliage scene rendered using MSTDZ, 5 zones.

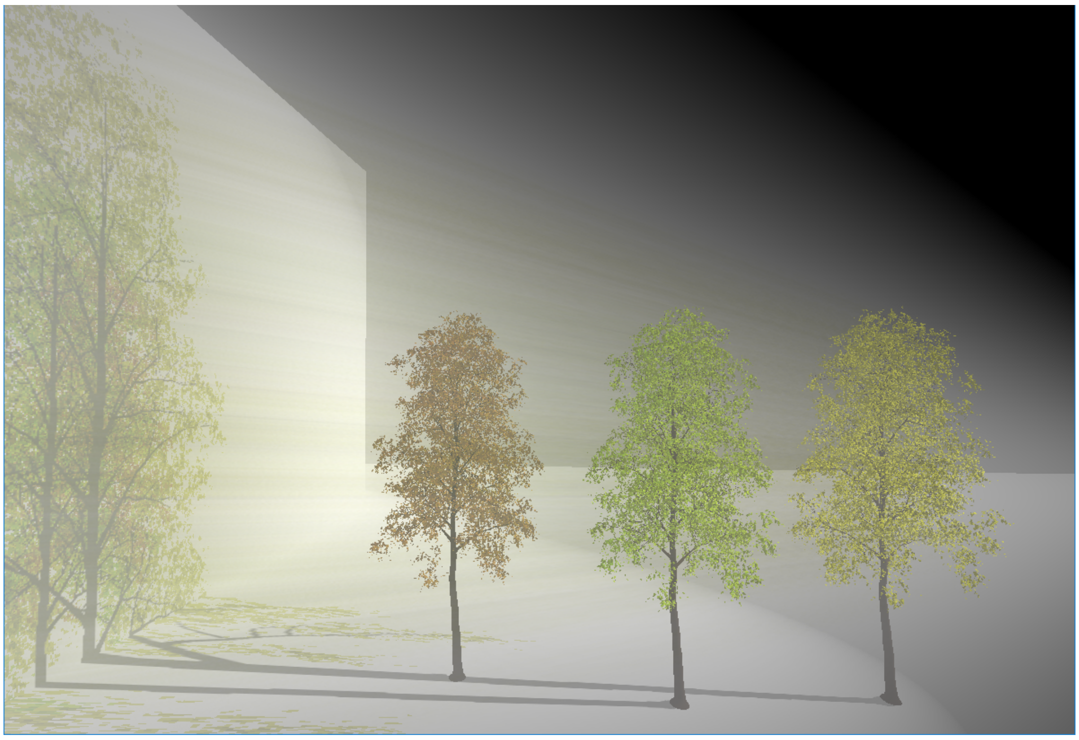


*Figure 53.* The Foliage scene rendered using MSTDZ, 6 zones.

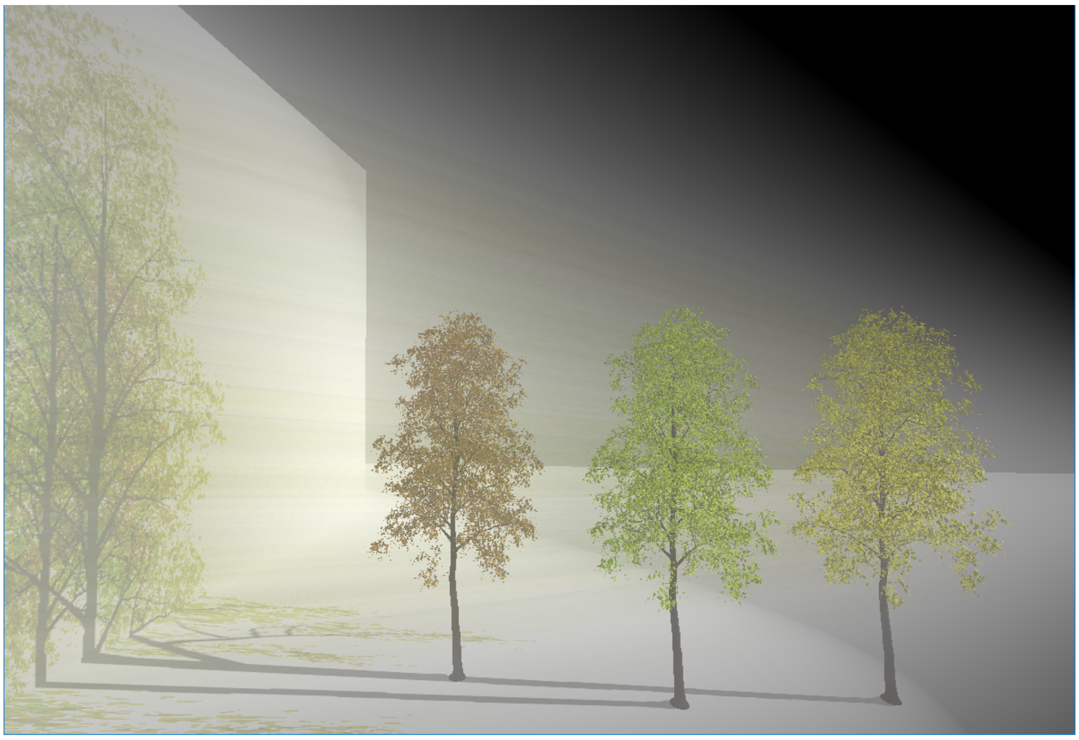


*Figure 53.* The Foliage scene rendered using MSTDZ, 7 zones.

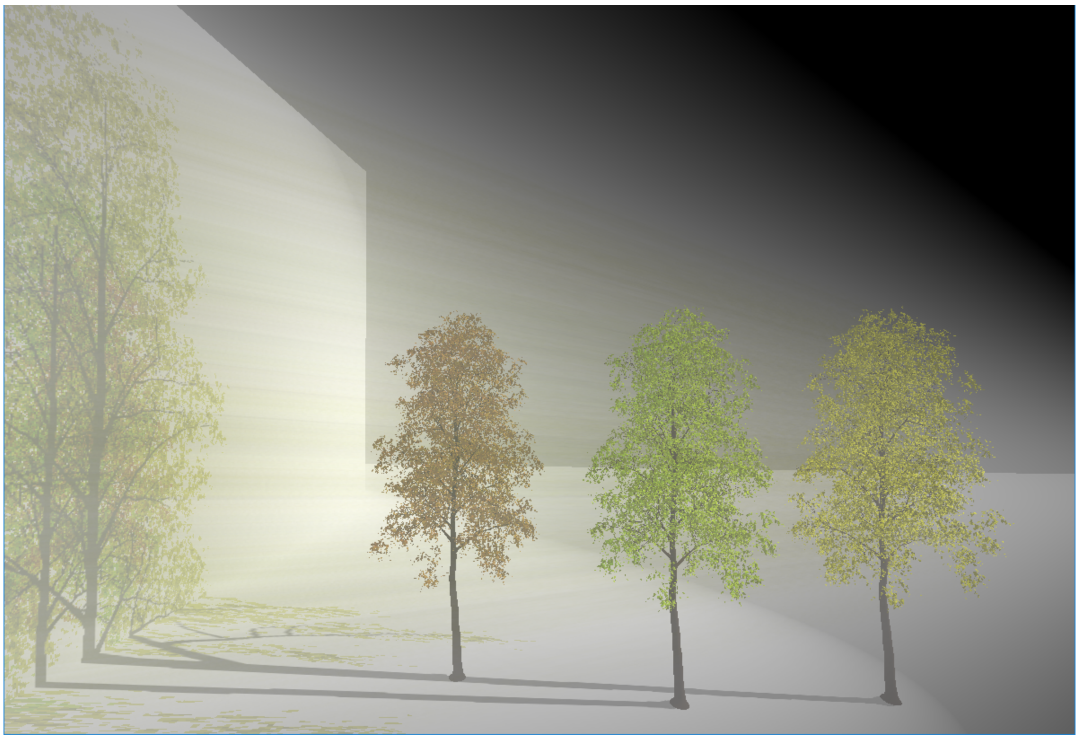


*Figure 53.* The Foliage scene rendered using MSTDZ, 8 zones.

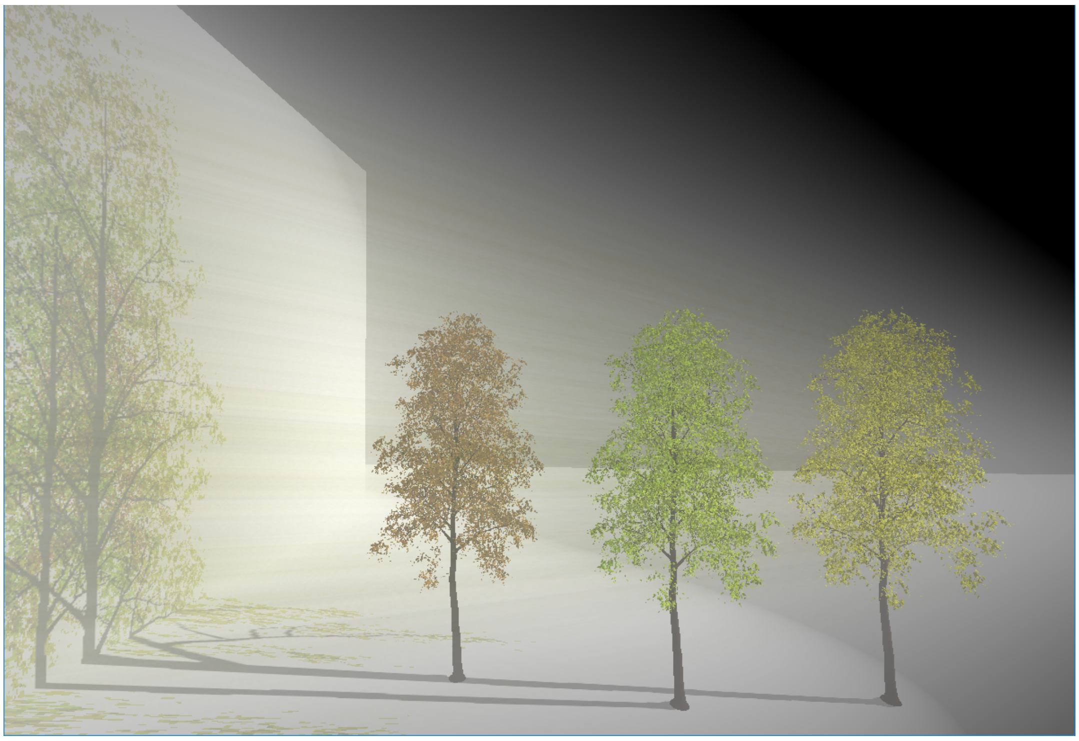


*Figure 53.* The Foliage scene rendered using MSTDZ, 9 zones.

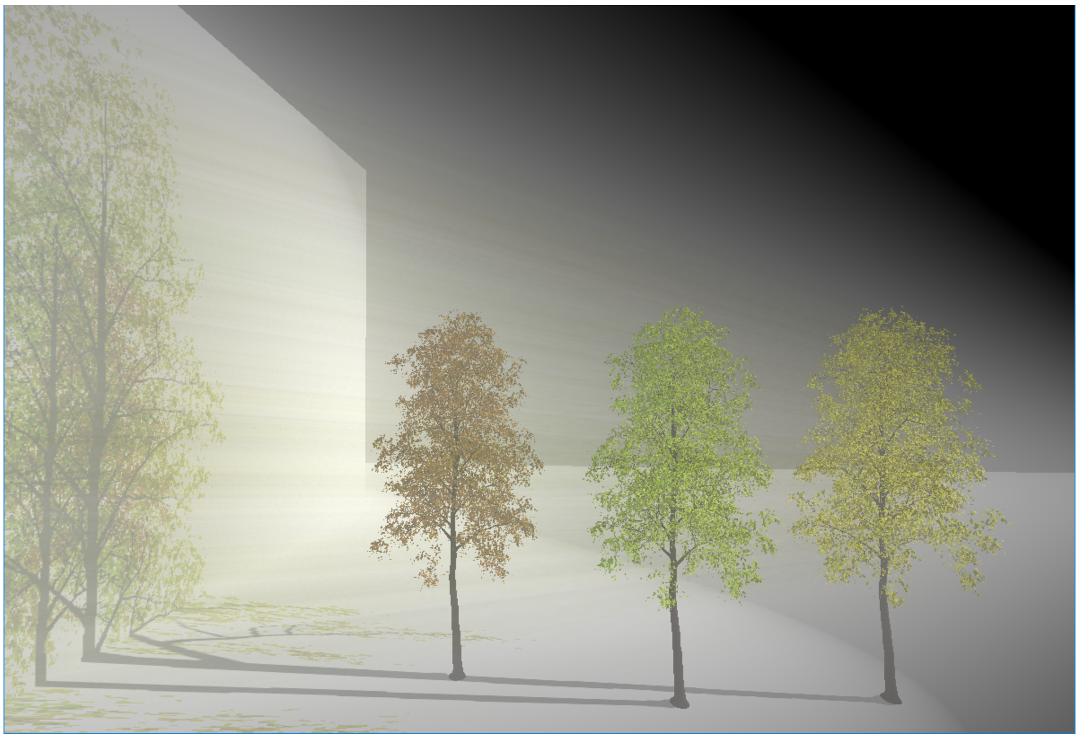


*Figure 53.* The Foliage scene rendered using MSTDZ, 10 zones.

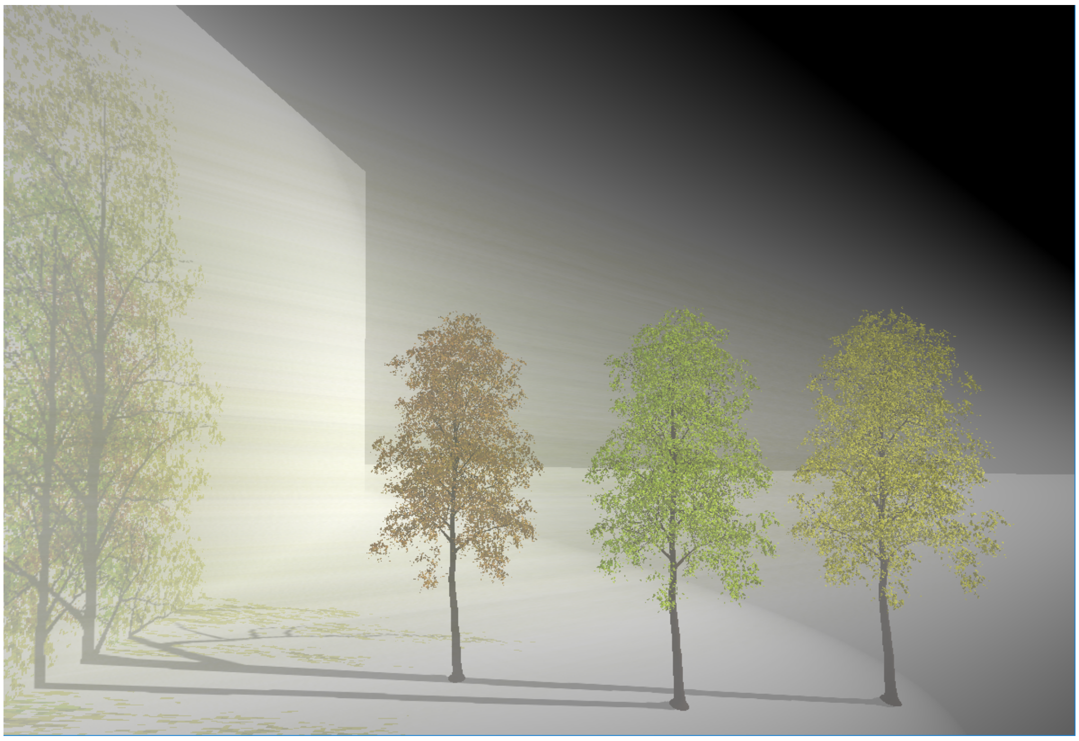


*Figure 53.* The Foliage scene rendered using MSTDZ, 11 zones.

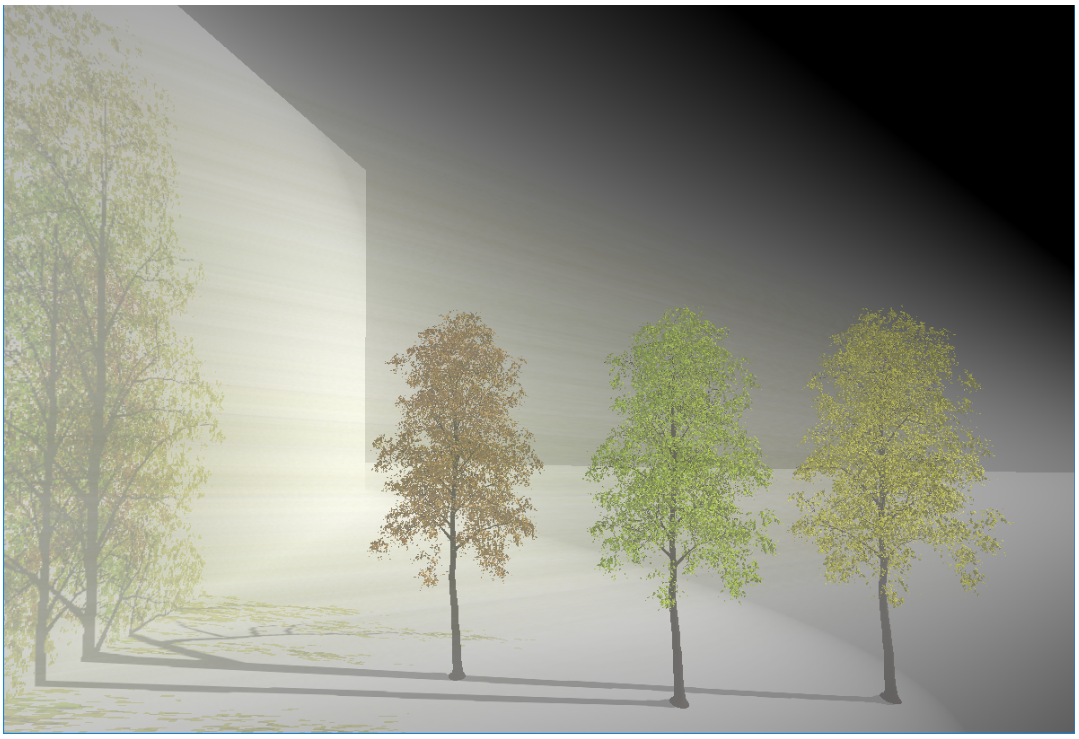


*Figure 53.* The Foliage scene rendered using MSTDZ, 12 zones.

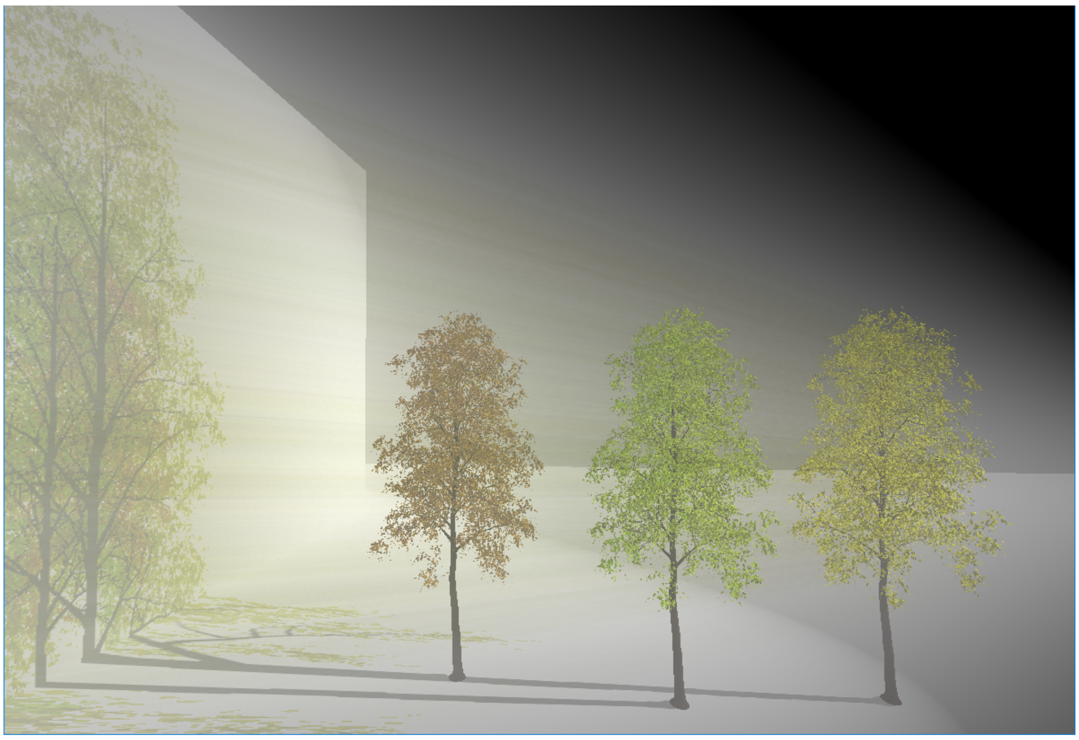


*Figure 54.* The Foliage scene rendered using Histogram MSTDZ, 3 zones.

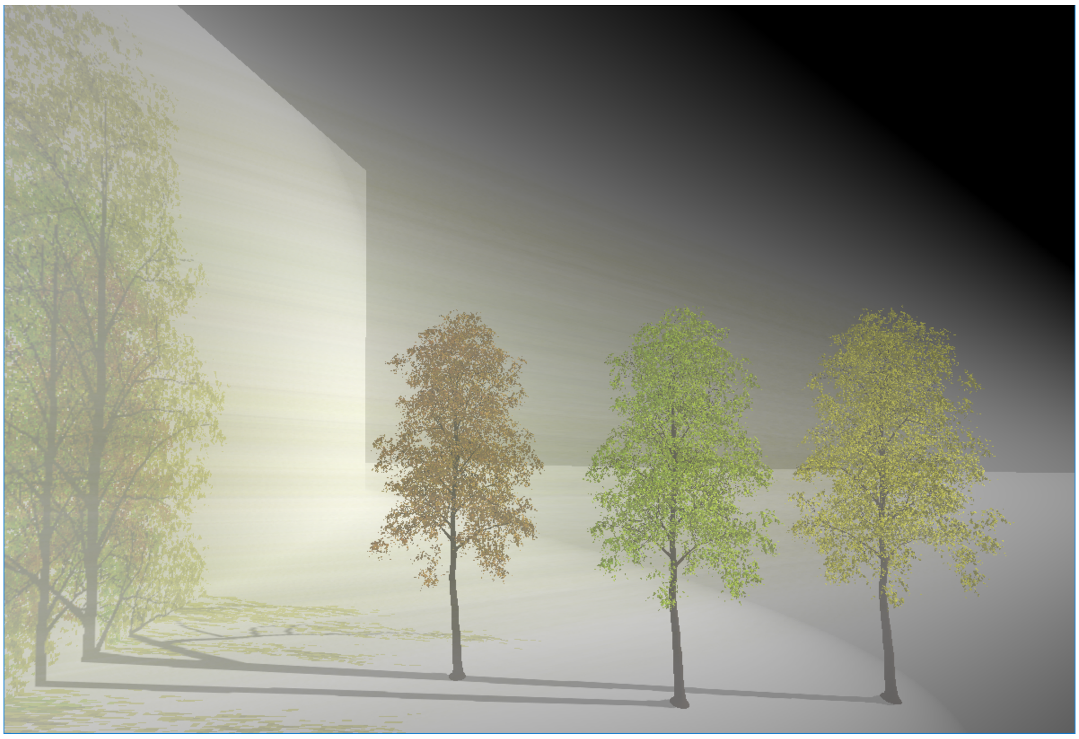


*Figure 54.* The Foliage scene rendered using Histogram MSTDZ, 4 zones.

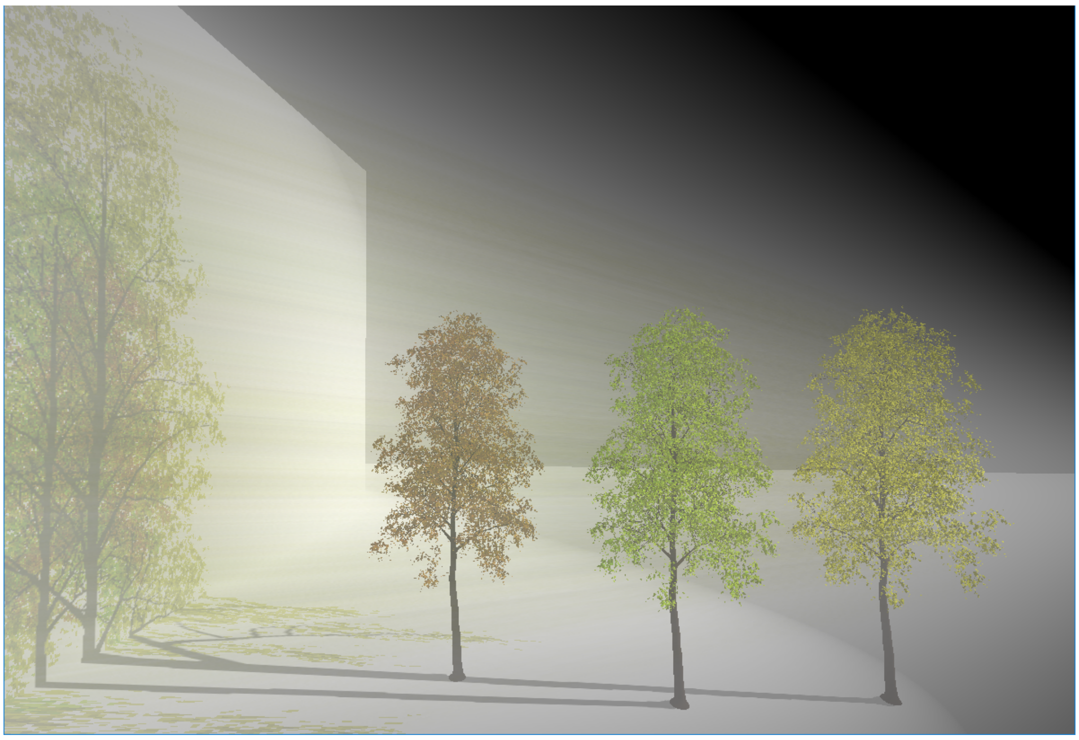


*Figure 54.* The Foliage scene rendered using Histogram MSTDZ, 5 zones.

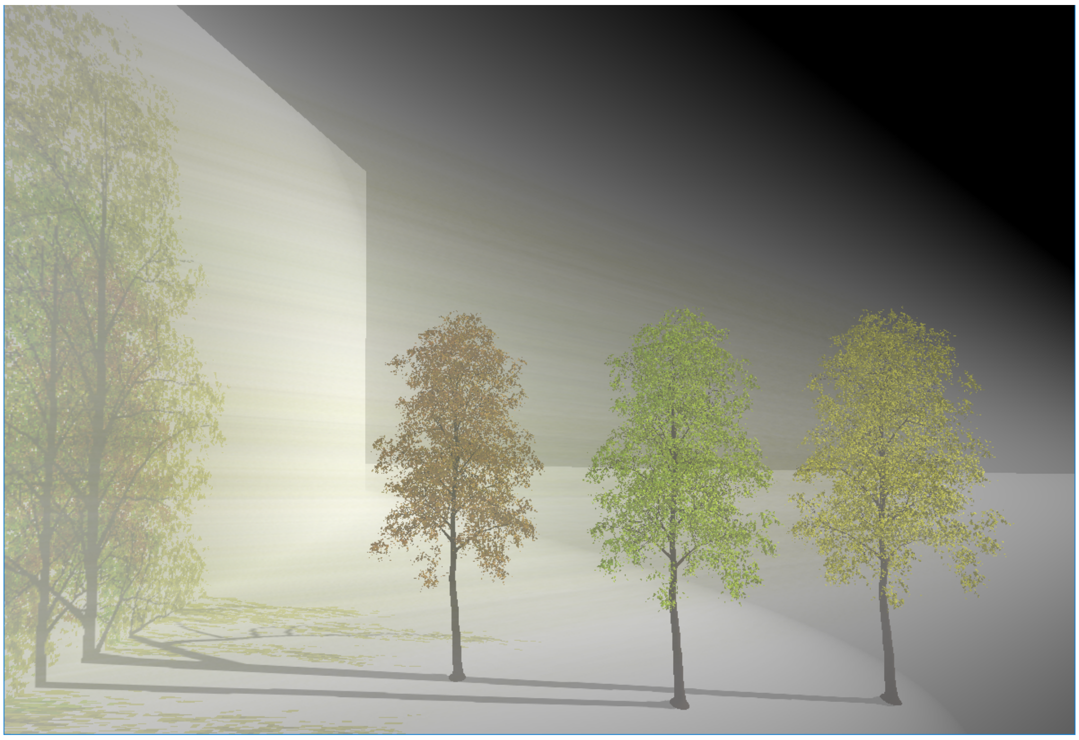


*Figure 54.* The Foliage scene rendered using Histogram MSTDZ, 6 zones.

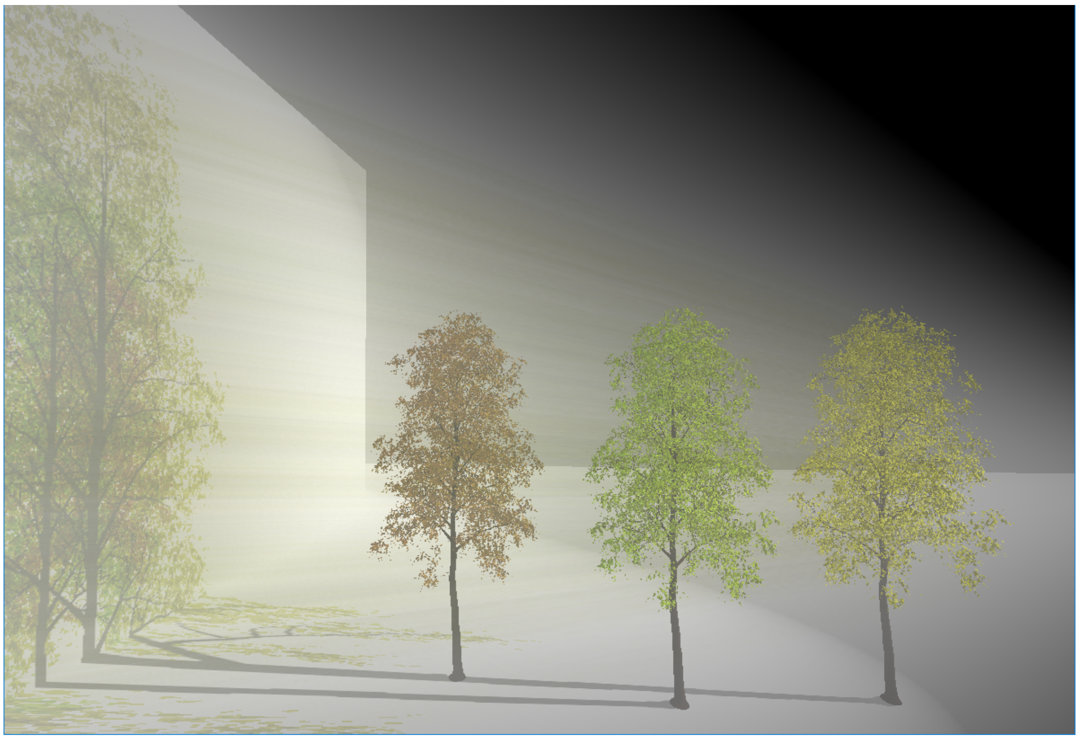


*Figure 54.* The Foliage scene rendered using Histogram MSTDZ, 7 zones.

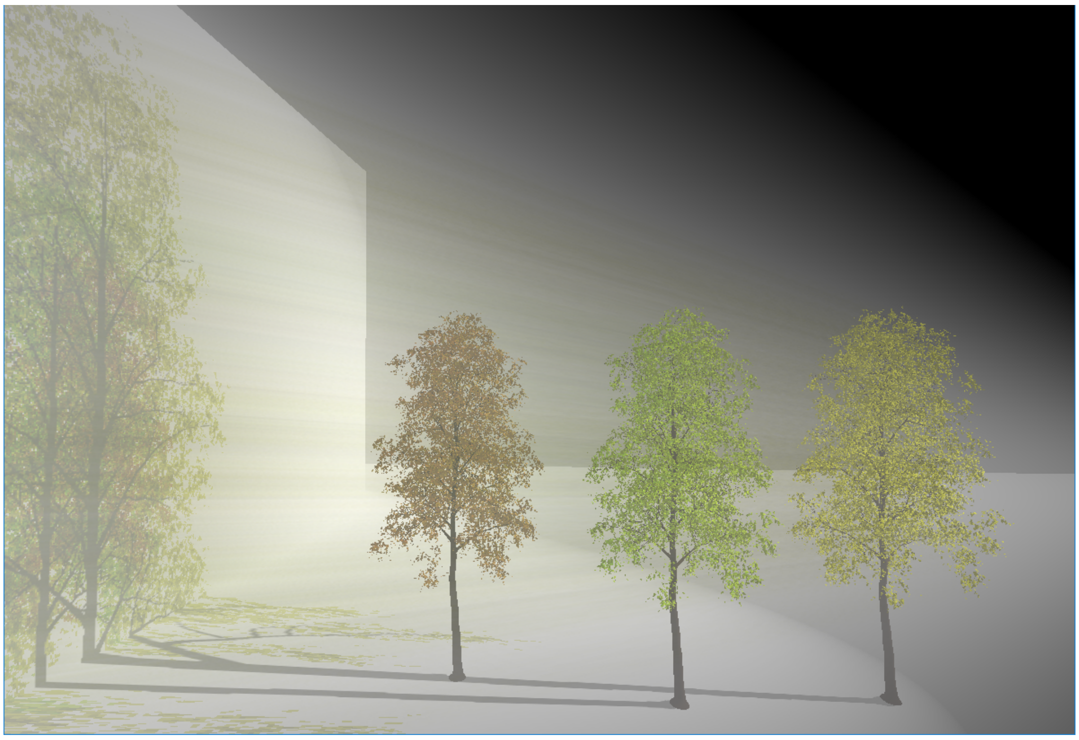


*Figure 54.* The Foliage scene rendered using Histogram MSTDZ, 8 zones.

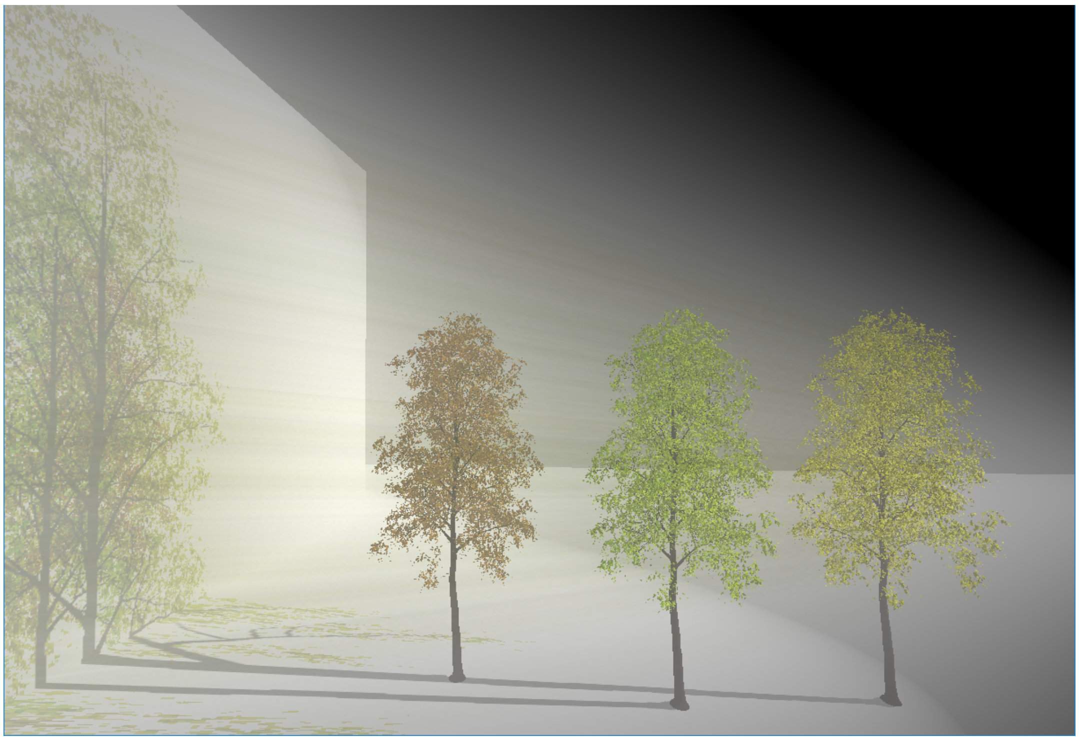


*Figure 54.* The Foliage scene rendered using Histogram MSTDZ, 9 zones.

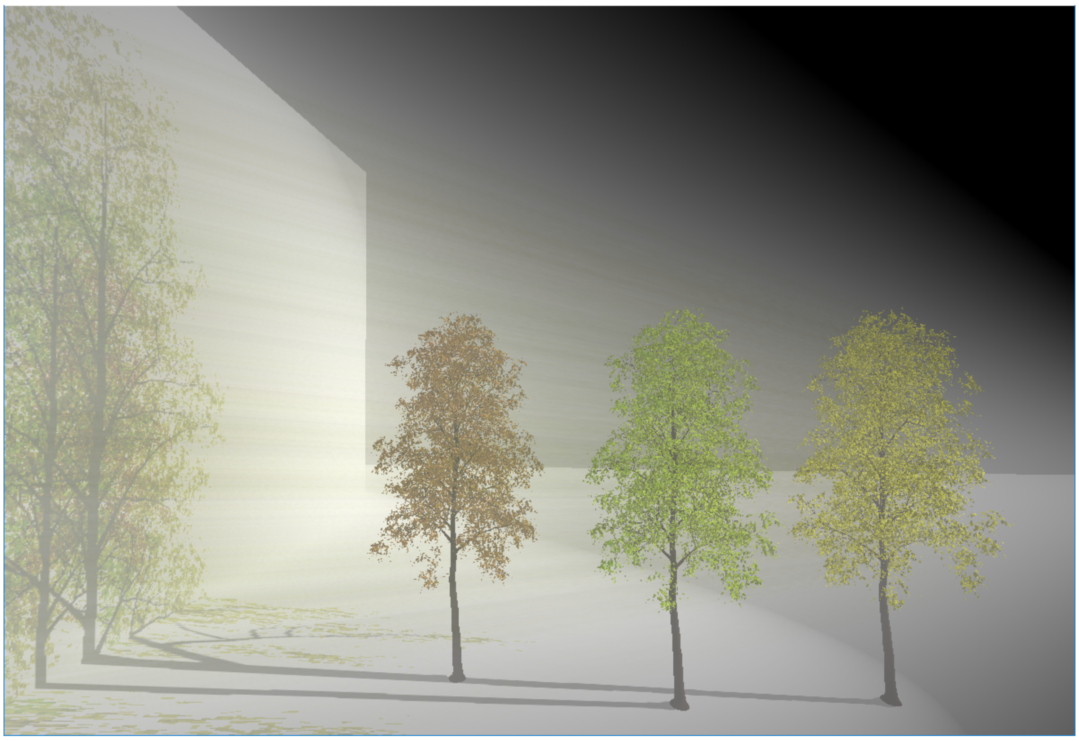


*Figure 54.* The Foliage scene rendered using Histogram MSTDZ, 10 zones.

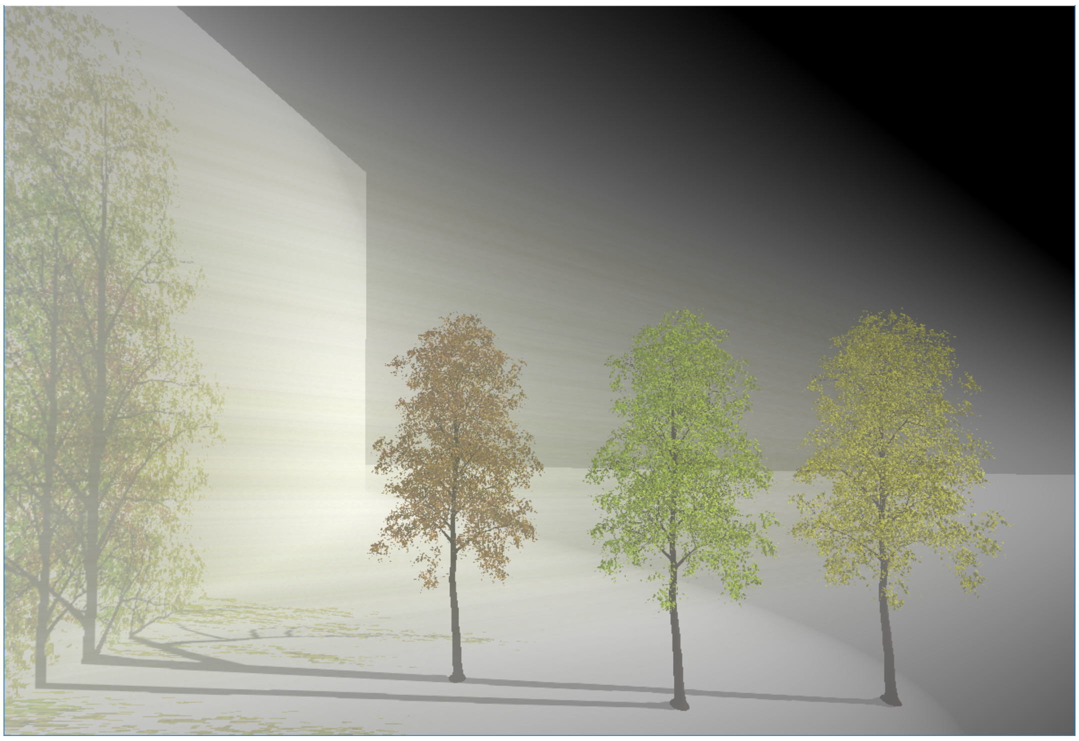


*Figure 54.* The Foliage scene rendered using Histogram MSTDZ, 11 zones.

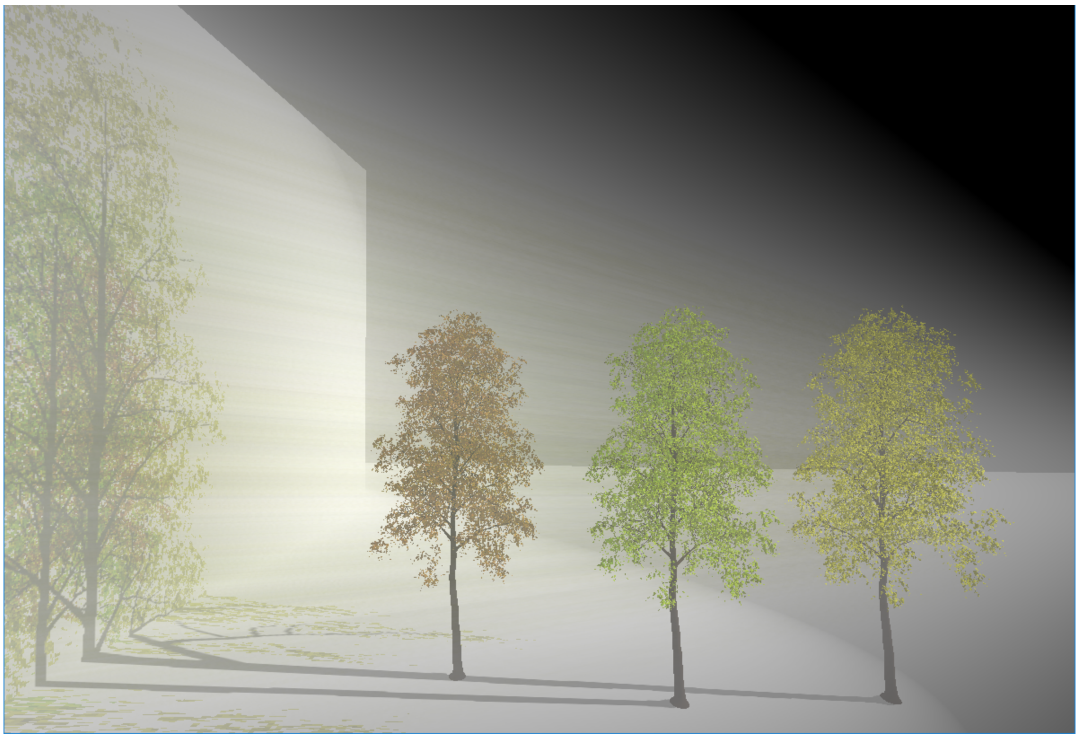


*Figure 54.* The Foliage scene rendered using Histogram MSTDZ, 12 zones.

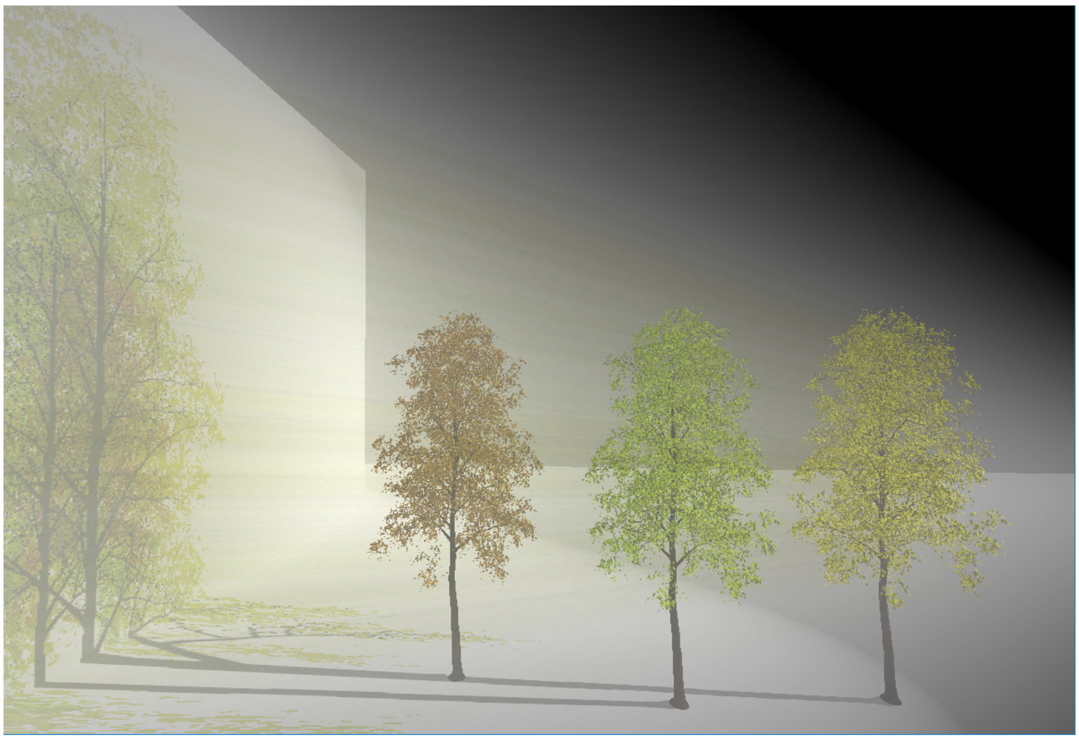


*Figure 55.* The Foliage scene rendered using Histogram MSTDZ with Texel Groups, 3/3/3 zones.

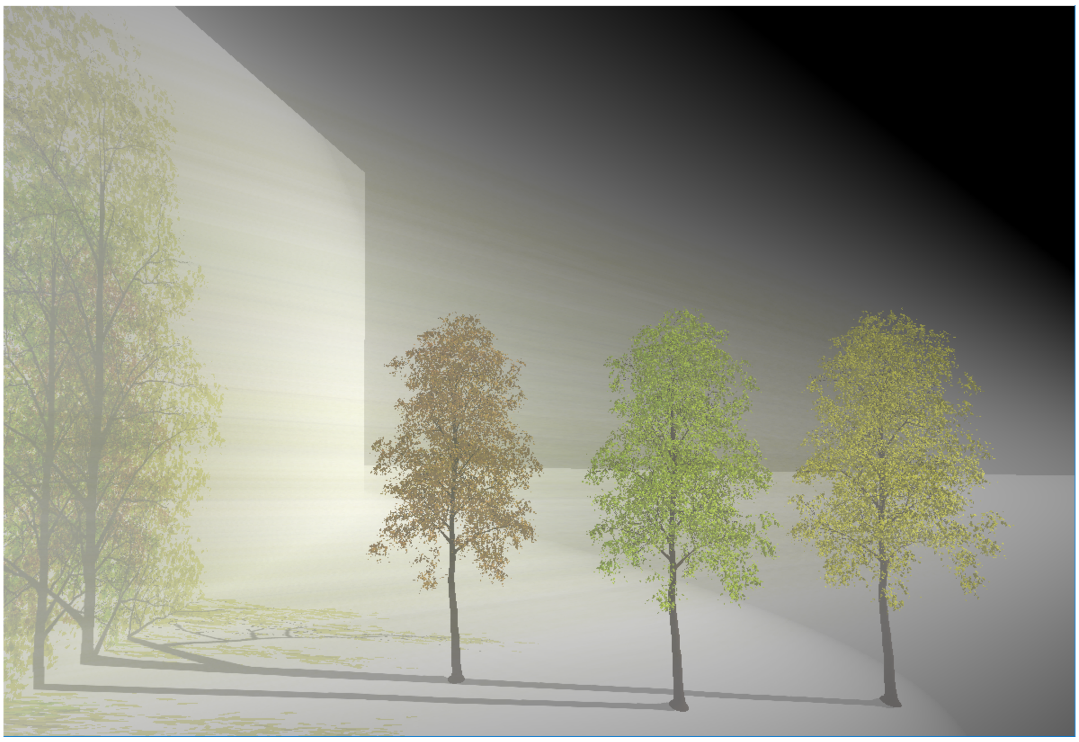


*Figure 55.* The Foliage scene rendered using Histogram MSTDZ with Texel Groups, 12/12/12 zones.

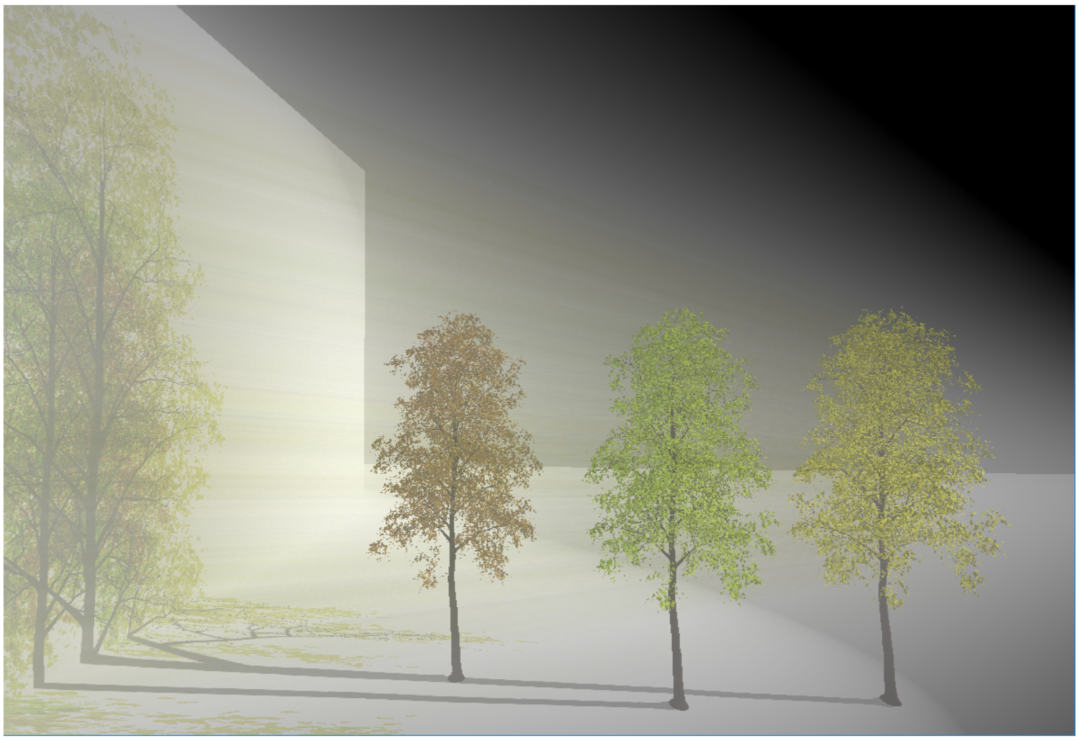


*Figure 55.* The Foliage scene rendered using Histogram MSTDZ with Texel Groups, 7/7/7 zones.

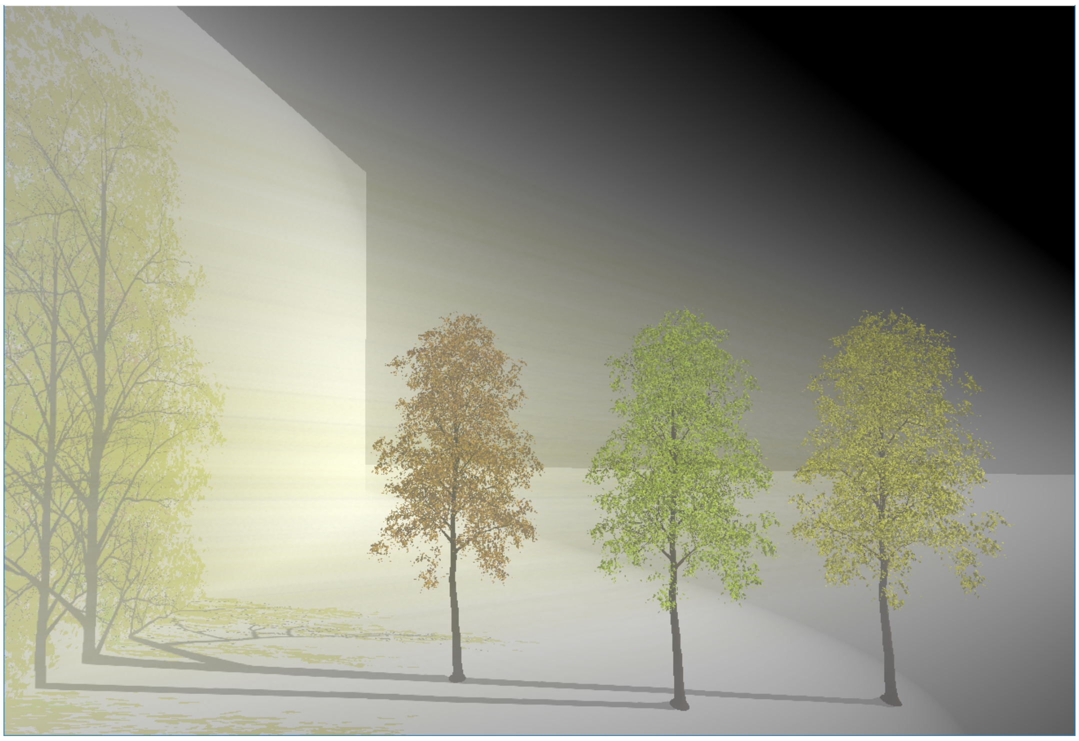


*Figure 56.* The Foliage scene rendered using Histogram MSTDZ with Texel Groups, 1/1/7 zones.

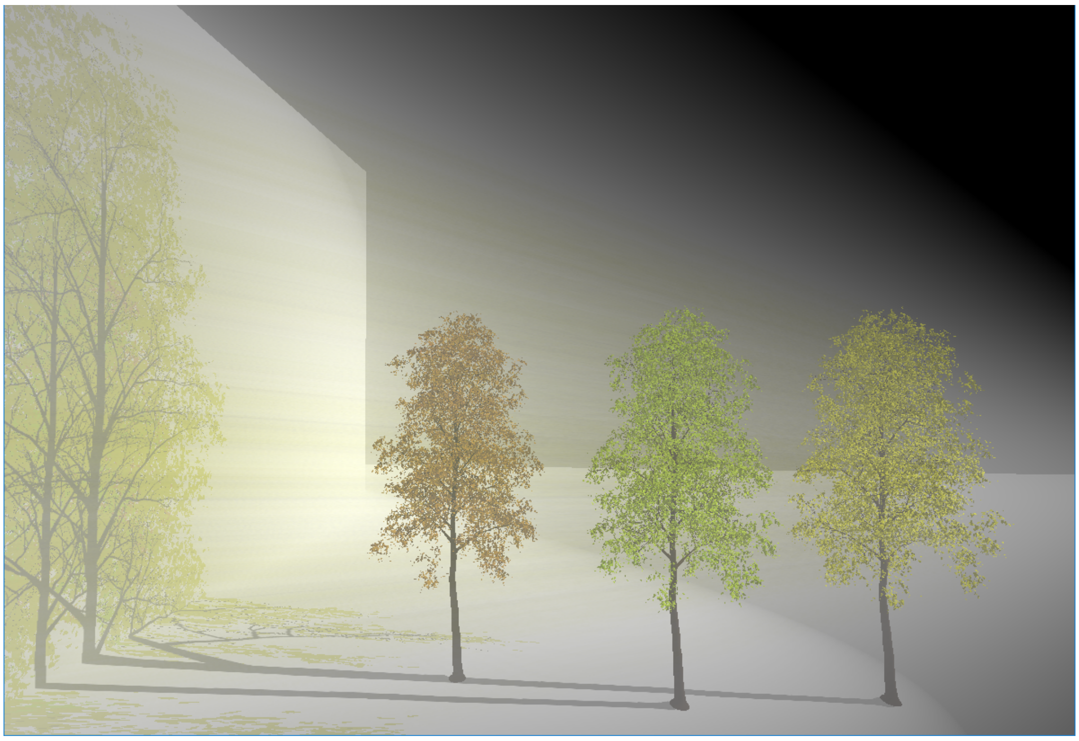


*Figure 56.* The Foliage scene rendered using Histogram MSTDZ with Texel Groups, 1/7/7 zones.

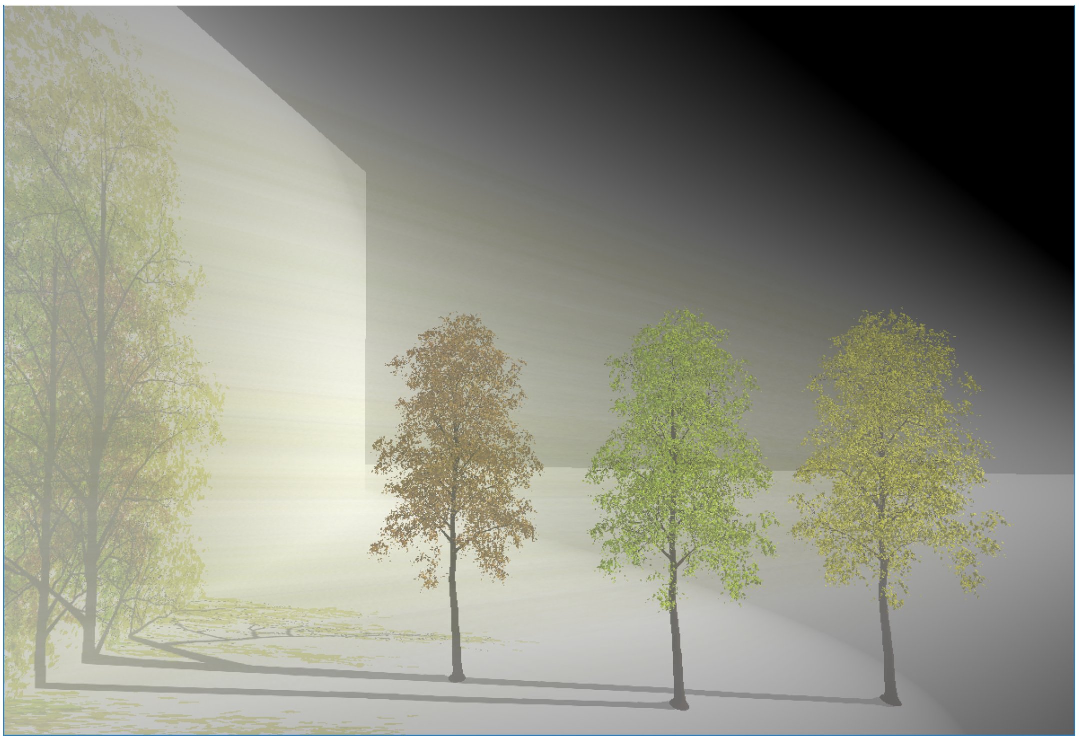


*Figure 56*. The Foliage scene rendered using Histogram MSTDZ with Texel Groups, 7/1/1 zones.

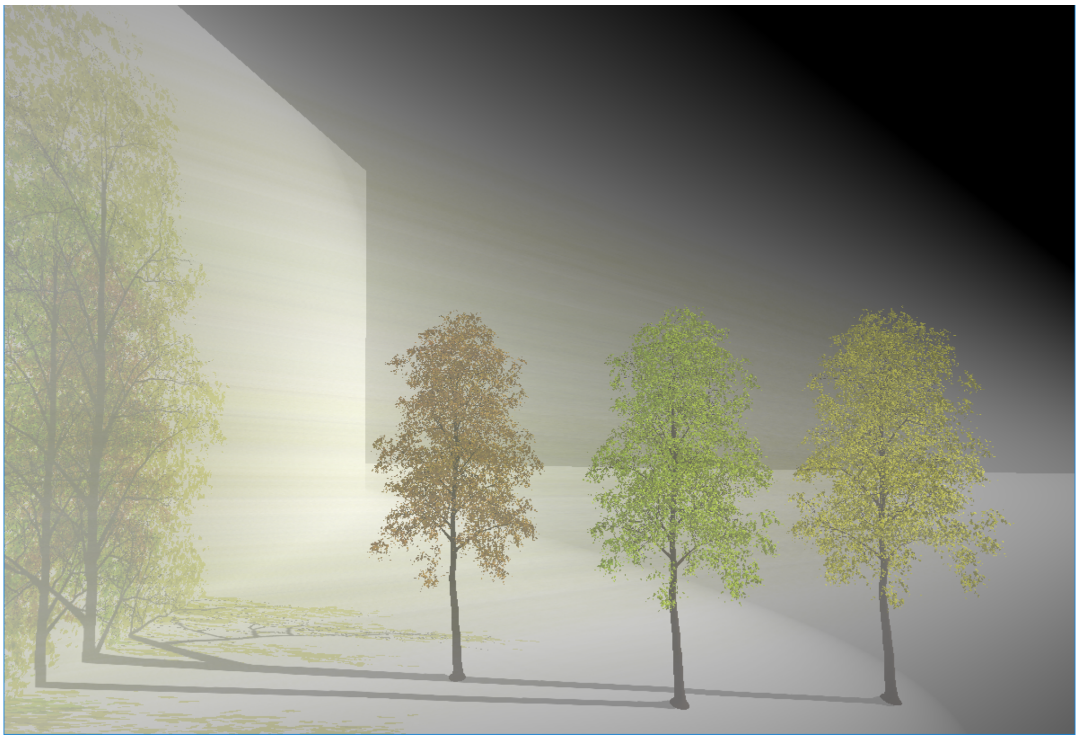


*Figure 56*. The Foliage scene rendered using Histogram MSTDZ with Texel Groups, 7/7/1 zones.

# Bibliography

[1] Baran, I., Chen, J., Ragan-Kelley, J., Durand, F., & Lehtinen, J. (2010). A Hierarchical Volumetric Shadow Algorithm for Single Scattering. *ACM Transactions on Graphics (TOG)*, 29, 178-186.

[2] Bavoil, L., & Myers, K. (2008). Order independent transparency with dual depth peeling. *NVIDIA OpenGL SDK*, 1-12.

[3] Billeter, M., Sintorn, E., & Assarsson, U. (2012). Real-time Multiple Scattering Using Light Propagation Volumes. *I3D, 12*, 119-126.

[4] Blinn, J.F. (1982). Light Reflection Functions for Simulation of Clouds and Dusty Surfaces. *Computer Graphics (ACM SIGGRAPH ?82 Proceedings)*, 16(3):21?29.

[5] Chandrasekhar, S. (1960). *Radiative Transfer*. New York, NY: Dover.

[6] Chen, J., Baran, I., Durand, F., & Jarosz, W. (2011). Real-time Volumetric Shadows Using 1D Min-max Mipmaps. *Symposium on Interactive 3D Graphics and Games*, 7-13.

[7] Enderton, E., Sintorn, E., Shirley, P., & Luebke, D. (2011). Stochastic transparency. *IEEE transactions on visualization and computer graphics*, *17*(8), 1036-1047.

[8] Engelhardt, T., & Dachsbacher, C. (2010). Epipolar Sampling for Shadows and Crepuscular Rays in Participating Media with Single Scattering. *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, 119-125.

[9] Everitt, C. (2001). Interactive order-independent transparency. *White paper, nVIDIA*, 2(6), 7.

[10] Jarosz, W. (2008). Efficient Monte Carlo Methods for Light Transport in Scattering Media. ProQuest.

[11] Kajiya, J.T. (1986). The Rendering Equation. *Computer Graphics (ACM SIGGRAPH '86 Proceedings)*, 20, 143-149.

[12] Klehm, O., Seidel, H. P., & Eisemann, E. (2014). Prefiltered Single Scattering. *Proceedings of the 18th meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, 71-78.

[13] Liu, F., Huang, M. C., Liu, X. H., & Wu, E. H. (2009). Bucket depth peeling. *SIGGRAPH 2009: Talks*, 50. Washington, WA: ACM.

[14] McGuire, M., & Bavoil, L. (2013). Weighted blended order-independent transparency. *Journal of Computer Graphics Techniques.*

[15] Pegoraro, V., Parker, S. G. (2009). An Analytical Solution to Single Scattering in Homogeneous Participating Media. *Computer Graphics Forum* , 28, 329-335. Chicago, IL: Blackwell.

[16] Pegoraro, V., Schott, M., & Parker, S. G. (2010). A Closed-Form Solution to Single Scattering for General Phase Functions and Light Distributions. *Computer Graphics Forum*, 29, 1365-1374. Chicago, IL: Blackwell.

[17] Peters, C., & Klein, R. (2015). Moment shadow mapping. *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games*, 7-14. Washington, WA: ACM.

[18] Peters, C., Münstermann, C., Wetzstein, N., & Klein, R. (2016). Beyond Hard Shadows: Moment Shadow Maps for Single Scattering, Soft Shadows and Translucent Occluders. *Proceedings of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, 159-170. Washington, WA: ACM.

[19] Salvi, M., Montgomery, J., & Lefohn, A. (2011, August). Adaptive transparency. *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, 119-126. Washington, WA: ACM.

[20] Sun, B., Ramamoorthi, R., Narasimhan, S.G., & Nayar S.K. (2005). A Practical Analytic Single Scattering Model for Real Time Rendering. *ACM Trans. Graph. 24*, 3, 1040?1049.

[21] Wyman, C., & Ramsey, S. (2008). Interactive Volumetric Shadows in Participating Media with Single-Scattering. *Interactive Ray Tracing, 2008. RT 2008. IEEE Symposium on (pp. 87-92).*

[22] Wyman, C. (2016). Exploring and expanding the continuum of OIT algorithms. *High Performance Graphics*, 1-11.