DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics: Games Engineering

Error-Based Adaptive Voxel Raytracing

Alexander Müller

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master´s Thesis in Informatics: Games Engineering

Error-Based Adaptive Voxel Raytracing

Fehlerbasiertes Adaptives Voxel Raytracing

Author:	Alexander Müller
Supervisor:	Prof. Dr. Rüdiger Westermann
Advisor:	Dr. Matthäus G. Chajdas
Submission Date:	15.12.2020

I confirm that this master's thesis in informatics: games engineering is my own work and I have documented all sources and material used.

Munich, 15.12.2020 Alexander Müller

Acknowledgments

I'd like to thank everyone who made this thesis possible and supported it.

Prof. Westermann for allowing me to do this thesis even though NDA's are involved (which is not a given).

Matthäus for guiding me through all of it and teaching me a lot along the way.

Hugh for spending a lot of time on preparing the data for me, answering all the question I had regarding it and providing me with valuable insights into production techniques. Michiel and Jeroen for allowing this thesis to happen at all, the time they spend and their support.

Everyone else from Guerrilla Games and AMD who was involved.

Furthermore, I'd like to thank Max, Sahin, Laura, Markus, Walentin, An and Minh for their help proofreading the thesis and for providing valuable feedback.

Abstract

Voxel rendering has been around for decades. Given the wide variety of use-cases, especially in the games and vfx industry, it is important to find new and improved algorithms for handling this specific rendering scenario. Such an algorithm has to increase rendering quality and/or decrease the performance necessary to achieve it, while at the same time being flexible enough to adapt to different data situations.

Raytracing an image at a high quality requires lots of rays per pixel and while the quality of the results is typically good, the performance costs to render those images are high. Adaptive renderers are a possible solution to this problem as they render the image by first using only a single ray per pixel and only increase ray counts where necessary. They do, however, not scale well across a wide range of possible scenarios and cannot handle some datasets without extremely high ray counts.

This thesis introduces a new rendering algorithm for raytracing of a (sparse) voxel octree: error-aware adaptive voxel raytracing. Instead of analyzing the rendered image and improving the quality of it based on variance in it, this approach tries to refine image quality during the raycasts themselves. It allows for the use of a level of detail system and dynamically determines if the ray count and level of detail for each pixel needs to be increased.

To achieve that, the error of sampling an octree node instead of its children is pre-calculated and stored for fast use at runtime. As a result, there's is only a minor overhead added to each raycast's performance cost, one that is easily offset by the overall performance and quality improvements of the algorithm.

When tested on the production dataset of the video game Horizon: Zero Dawn, the algorithm was able to outperform an adaptive approach easily for every single dataset. Three different image similarity metrics are used to determine the quality of the adaptive renderer and the approach from this paper compared to a reference image. The custom algorithm showcases great scalability across different resolutions, for different ray count ranges and for different quality levels.

Due to visibility being the metric used to determine the error made during sampling, the algorithm falls short of its potential when lighting information is being introduced to the rendering scenario. While it still outperforms the adaptive renderer in those tests, other metrics need to be tested in order to further advance the quality of the images that are being rendered.

Contents

Acknowledgments **iii**

Abstract **iv**

1 Introduction **1**
1.1 High Variance in Dataset 1
1.2 Speed-Quality Trade-Off 1
1.3 Resolution 2

2 Octree Voxel Rendering **3**
2.1 Sparse Voxel Octree 3
2.2 Octree Raytracing 3

3 Error-Aware Adaptive Rendering **4**
3.1 Theory 4
3.1.1 Analytical Approach 5
3.1.2 Raycast Approach 6
3.2 Practical Implementation 8
3.2.1 Sample Locations 9
3.2.2 Rendering Tests 9

4 Dataset **14**
4.1 Desert 15
4.2 Forest 16
4.3 Mountain 17
4.4 Biomes 18

5 Testing **19**
5.1 Methodology 19
5.1.1 Setup 19
5.1.2 Adaptive Rendering 20
5.1.3 Error-Aware Rendering 21
5.1.4 Decay Factor and Error Threshold 22

5.1.5 Image Similarity Metrics 24
5.2 Results 24
5.2.1 Desert Dataset 25
5.2.2 Forest Dataset 27
5.2.3 Mountains Dataset 29
5.2.4 Biomes Dataset 31
5.3 Lighting 33
5.3.1 Desert Dataset 33
5.3.2 Forest Dataset 36
5.3.3 Mountain Dataset 38
5.3.4 Biomes Dataset 40
5.4 Error Threshold 42
5.5 Resolution 48

6 Future Work **53**
6.1 Error Metrics 53
6.2 Optimization 53

7 Conclusion **55**

List of Figures **56**

List of Tables **58**

Bibliography **60**

1 Introduction

When rendering an image using voxel raytracing, there are two possibilities. One can either rely on an octree level of detail system and use only few rays per pixel or the level of detail system isn't used and a lot more rays are being cast. The advantages of the first approach is the massive performance benefit one gets from using it. The disadvantage, however, is a heavy loss in terms of image quality. The level of detail system introduces an error and areas of the voxel data with high variance lose a lot of detail.

Adaptive renderers ([JW88], [Wid+15]) try to handle this issue by rendering the image first using a single ray per pixel. Once the image is rendered it is analyzed and the variance of it is calculated. Pixels with high variance are rendered once again using a higher ray count and without the level of detail system. This can be done over and over again, depending on the set iteration count.

1.1 High Variance in Dataset

The first issue for the adaptive renderer is extremely high variance within the dataset. An example here would be a highly emissive voxel, e.g. one representing a torch on a wall. A single pixel's variance can be low even if it contains such a voxel. The renderer therefore does not know that it needs to refine the image at the pixel's position and the information of this voxel will be lost.

1.2 Speed-Quality Trade-Off

There are two parameters that control the quality and respective ray counts of an adaptive renderer. The first one is the iteration count, i.e. the amount of refinement steps the renderer performs on the whole image. The second parameter is the threshold for the variance at which the renderer starts to perform those refinements. Both parameters do not scale well, and even slight improvements in quality result in a large additional cost of rendering.

In order to be able to better adapt the raytracer to a wide range of rendering scenarios, it is important for the scalability to be as good as possible.

1.3 Resolution

Various tasks, e.g. the rendering of a cubemap for a lightprobe in a video game's global illumination system, require images with a low resolutions. For large datasets, e.g. if the amount of information for each pixel is high, an adaptive renderer tends to struggle even more. This is due to effects like the one described in 1.1 being more prominent at lower resolutions.

To properly support those use cases as well, an improvement to the adaptive rendering approach needs to be able to better scale across a wide range of resolutions, ideally linearly.

This defines the core design goals for the rendering algorithm introduced in this paper. It has to

- be able to handle high variance in the dataset
- allow for speed-quality trade-offs
- scale across a wide range of resolutions

2 Octree Voxel Rendering

2.1 Sparse Voxel Octree

Sparse voxel octrees (SVOs) are structures used primarily for speeding up raytracing of large datasets [LK10]. It is a hierarchical structure of voxels where a node in the tree can consist of up to 2^3 child nodes. During ray traversal, the structure is used to reduce the amount of intersection tests required as the hierarchy of the octree is used to determine which nodes need to be checked for them. An octree is sparse if it only creates parent nodes where necessary, i.e. when there is at least one child node for it with valid data.

In addition to that, it can serve as level of detail system by combining the values of the child nodes during generation and returning them during traversal. A level of detail system always approximates the data, and therefore using it introduces an error into the process. The algorithm introduced in this thesis (3) uses this property to refine raytracing results during rendering.

2.2 Octree Raytracing

A naive approach to octree raytracing would be to perform ray-box intersections for every node along the ray. Since nodes have to be checked from back to front for optimal performance (to stop the raytrace as soon as there is a hit), it is more efficient to compute the intersection with the octree once and step through the octree from there using the mid-planes of the nodes. The octant closest to the ray's entry-point can be found by checking on which side of each mid-plane it is. This provides the child voxel that needs to be checked first. If there is no result, the entry-point is moved to the closest mid-plane and the same algorithm is performed once again. If the level of the octree node corresponds to the set level of detail, it is sampled and the raytrace is done.

Different approaches to ray traversal over the years can be found here [AGL91], [RUL00],

3 Error-Aware Adaptive Rendering

This thesis proposes a novel algorithm for rendering voxel datasets. Its goal is to improve upon the performance and quality of existing, adaptive rendering approaches and to allow for good scalability of quality and render resolution. To achieve this, a solution needs to be found that enables the use of the level of detail system a (sparse) voxel octree provides.

While adaptive rendering relies on variance in the rendered image in order to determine which pixels need to be rendered at a higher quality, the principle behind this new approach is to determine the error made during sampling of a specific node in the octree when compared to a reference renderer. Thus, the name "error-aware adaptive rendering".

3.1 Theory

In order to understand the idea of the algorithm, it is important to understand how a level of detail system introduces an error to the rendering process. When generating an octree, a node's value is determined by a combination of its children's values. The simplest case possible is to set the visibility of the parent node to true if it contains at least 1 child and to calculate the color of the node by averaging the children's colors. If it is now decided to sample the parent node instead of the children, an error is introduced. While the precise color sampled should be one of the children's, it is now the average of all of them.

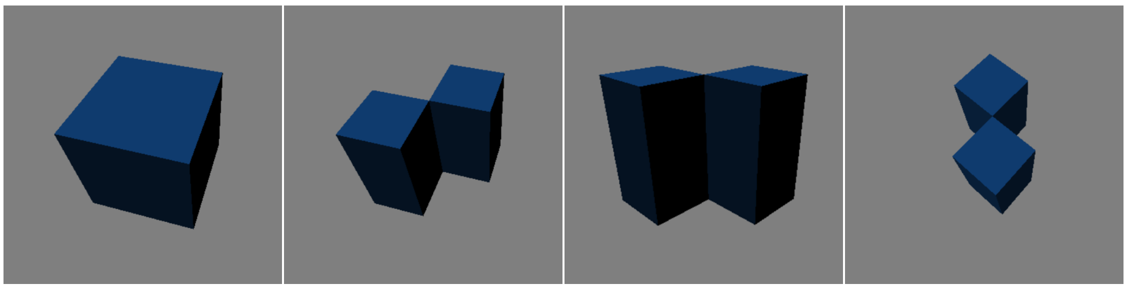

(a) Parent node (b) Child nodes (c) Side view (d) Top view

Figure 3.1: Octree node example

This error might be even bigger when using visibility. 3.1(a, b) shows an example setup for a parent node and the 4 child nodes it contains. The colors are identical as they are not important here. Assuming there is a single ray that hits the parent box from the given camera direction, it will always return the color of the node. If the level of detail is now one level higher, i.e. if the children of the parent node are being sampled instead, a large number of rays that would have hit the parent now miss. This, essentially, is the error that occurs due to the level of detail system.

When using a cone tracer [Cra+11], the cone is usually set up so one node of the desired level of detail covers a single pixel in the final image. If a higher level of detail is used for a pixel, the ray count for this pixel needs to be adjusted as well to avoid under-sampling. For voxels, this means using 2x2 rays for the pixel and sampling the children of the node.

In addition to that, there is also the factor viewing direction. The error made when sampling the parent instead of the children is minimal for 3.1(c). For 3.1(d), however, it is vastly larger. The error is a function that is defined on the surface of a sphere around the node rather than a constant. Furthermore, the number of children and their position play a pivotal role.

Therefore, the actual question that needs to be asked by the algorithm when sampling is: How large is the error made by sampling the parent node with one ray instead of tracing 4 rays against the children given the constellation of children and the viewing angle?

Fortunately, calculating the error values can be done offline and stored for use in the real-time application. For an octree node, there are $2^8 = 256$ possible combinations of children.

3.1.1 Analytical Approach

The solid angle Ω is defined as

$$\Omega = \frac{v * d_a}{r^2} \tag{3.1}$$

with v being the directional vector from the origin, $\bar{v} = 1$, the differential area of a surface patch d_a and the distance between the origin and the patch r. This represents the area of a unit sphere for a given angle that is covered by a projection of a given surface.

The solid angle of a voxel Ω_{voxel} for a vector v is therefore defined as

$$\Omega_{voxel} = \bigcup_{s \in S} \left(\frac{n * d_s}{r^2} \right) \tag{3.2}$$

with the set of voxel surface patches S. This can easily be expanded to calculate the solid angle for a set of child voxels C

$$\Omega_{children} = \bigcup_{c \in C} \bigcup_{s \in S_c} \left(\frac{n * d_s}{r^2} \right) \tag{3.3}$$

with S_c being a set of voxel surface patches for a specific child c. The error ε can now simply be computed by calculating the difference between Ω_{voxel} and $\Omega_{children}$:

$$\varepsilon = \Omega_{voxel} - \Omega_{children} \tag{3.4}$$

While ε could be computed explicitly for a single angle, it cannot be pre-computed to speed-up rendering. A possible solution is to discretize the vector space for n and to approximate the function using raycasting.

3.1.2 Raycast Approach

The concept of this approach is to use an orthographic camera and to place it at a defined, limited number of angles around a unit sphere, looking inwards. The size of the camera is set so it covers the entire Unit sphere (width and height are 2). The parent voxel node is placed as a unit cube inside the unit sphere with a set combination of child nodes.

For each direction and child combination, the parent mean is either 0 or 1 (node is visible or not). The child mean value needs to be calculated next. For that, the camera view is divided into a regular 2x2 grid (i.e. a small frame buffer) and for each cell a single ray is cast through the center as well. If this ray doesn't hit the parent node, it is ignored. If it does, it is checked whether it hits one of the child nodes. The amount of child nodes hit is divided by the number of rays that hit the parent node to calculate the mean value for the children. The error can now be calculated by subtracting the child mean from the parent mean.

Raycasting is always an approximation and there is an error due to sample frequency (the limited resolution of the regular grid used). The maximum error ϵ_{max} is half the size of a cell. As it can occur at every edge of the nodes, the absolute error is ϵ_{max} multiplied by the length of all edges. Since there is a maximum of 8 children, each with 12 edges and each edge being 0.5 units long (since a child is half the size of the parent unit cube), the total edge length of all children is 48 units. This results in the following total errors introduced through the raycasting approximation:

Resolution	512x512	256x256	128x128	64x64
Error	0.09375	0.1875	0.375	0.75

Table 3.1: Error introduced through raycasting approximation

Small resolutions like 256^2 are already enough to be able to keep the maximum error small enough so it can be neglected. Since there are always exactly 4 rays supposed to be cast against the children, this higher resolution needs to be mapped to the 2x2 grid. This can simply be done by dividing the large grid into 4 regular segments and to only cast a ray for a single pixel in the segment. This can be done for every pixel in the segment and the average is calculated.

As an example, a resolution of 256^2 is used. It is divided into 4 quarters, each one being 128^2 pixels in size, or a total of 16,384 pixels in total. The mean value for the children would be calculated by casting 16,384 sets of 4 rays (distributed across the quarters) and calculating the average of all results.

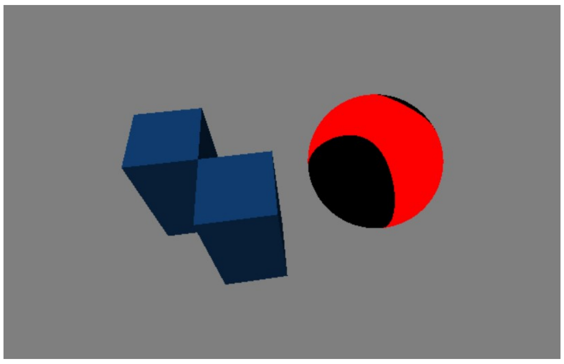


Figure 3.2: Sample error values

3.2 shows the offline calculated error values for the example setup from before projected onto a sphere. Each point on the surface corresponds to the error value for the inverse of its normal vector. If it is marked red, the error is above a set threshold. It is clearly visible, that the algorithm is working as expected, as the error is lower than the threshold when looking at the setup from the side, but higher than the threshold when looking at it from the top (compare to 3.1 (c, d)).

```
foreach Child node combination do
    foreach Direction do
        $pmean \leftarrow parent\ mean$;
        $cmean \leftarrow 0$;
        $ccount \leftarrow 0$;
        for $i \leftarrow 0$ to $pixel\ count/4$ do
            Select pixel positions from each quarter;
            foreach Pixel position do
                if ray hits parent node then
                    $ccount++$;
                    if ray hits any child node then
                        $cmean++$;
                    end
                end
            end
        end
        $cmean = cmean/ccount$;
        $error \leftarrow pmean - cmean$;
        Store error for direction and child combination;
    end
end
```

Algorithm 1: Pseudo-code of the error calculator

3.2 Practical Implementation

For the error data used throughout this thesis, error values for 256 directions were computed for each child combination and a 512^2 frame buffer resolution was used. To save memory and to speed up computation during runtime, the error values for each child combination were encoded as spherical harmonics (first 3 orders, 9 parameters).

3.2.1 Sample Locations

When sampling on a regular grid, a large systematic error could be introduced. If every sample is located at the centre of each grid cell, entire rows and columns of samples are aligned. During raycasting, they thus project onto the same x and y axes. If the edge of one of the octree cubes is aligned with one of said axes, all samples along this axis are either intersecting the cube or not. This is more commonly referred to as aliasing.

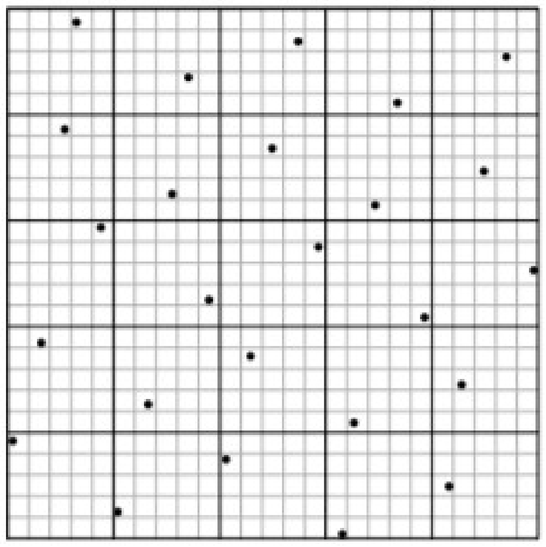


Figure 3.3: Sample jittering from [Ken13]

To counter this issue, the pixel position selected from each quarter is randomly jittered according to the algorithm introduced by [Ken13].

3.2.2 Rendering Tests

In order to confirm the viability of the approach, a purpose-built test scene is used. It contains a Cornell box with a light source at the back and a grid at the front. The box has a size of 32^3, the empty spaces in the grid are 1 voxel wide. To better visualize the effects of the error-aware refinement of the level of detail, it is not used for the primary, but rather for a secondary (shadow) ray. This secondary ray is importance-sampled. Inverse square falloff over distance is used to calculate the light intensity.

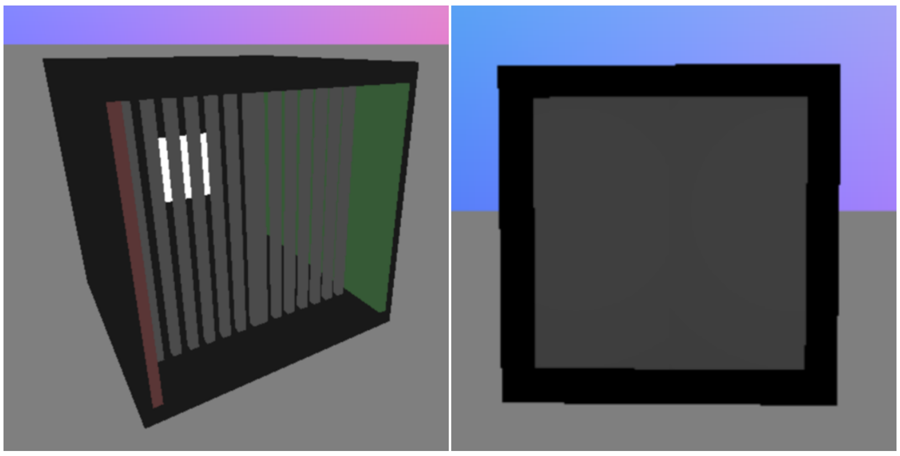

(a) LOD 0 (b) LOD 1

Figure 3.4: Test scene setup

As seen in 3.4, the second-highest level of detail is enough to convert the grid to a plane that covers the whole front of the box. As a result, no light is able to emit from the box any more. The level of detail that is sampled by the shadow ray is based on the distance between the ray's origin and the intersection point with the geometry. This leads to the light emission being cut off abruptly instead of fading into the distance.

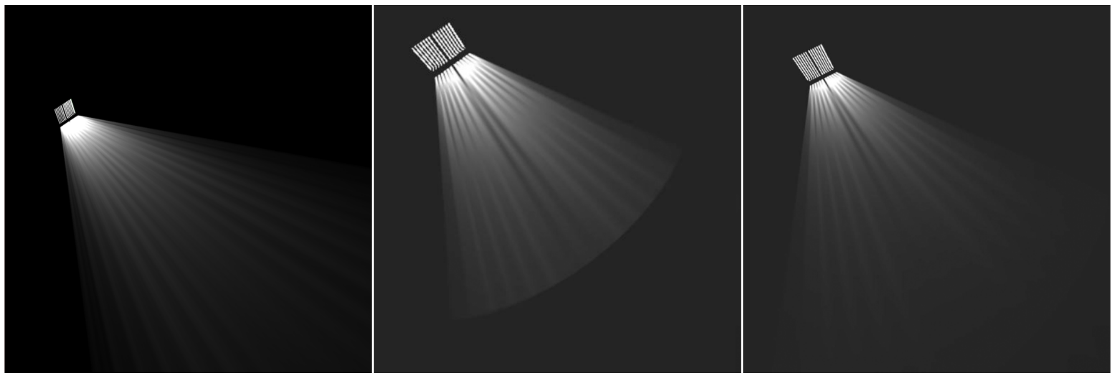

(a) Reference (b) Error-aware disabled (c) Error-aware enabled

Figure 3.5: Test scene shadow ray quality comparison

3.5 shows a comparison between a high ray count reference image and the two real-time rendered results. 3.5(b) is the version without the error-aware level of detail enabled and shows the expected result. A clear cutoff at the distance where the sampled level of detail changes is visible. The error-aware renderer is able to dynamically adjust the level of detail used when sampling octree and as a result it is able to produce an image that is way closer to the reference.

This test highlights a situation the adaptive renderer simply cannot handle at reasonable cost. to achieve the same image quality, hundreds if not thousand of iterations would be required to gradually increase the level of detail of the beam area pixel by pixel. The open question is whether the level of detail is increased overall for the error-aware renderer or whether it only increases the quality for the pixels necessary.

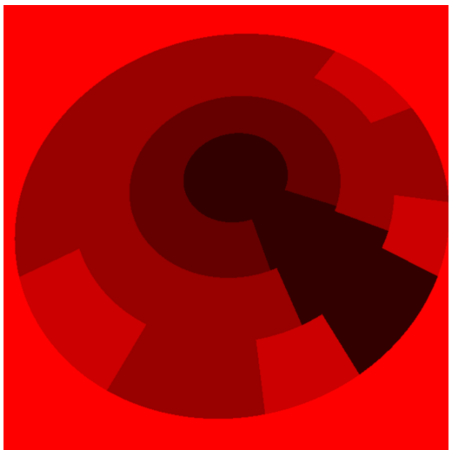


Figure 3.6: Test scene level of detail visualization

3.6 is a visualization of the level of detail for a larger area. A darker color is relative to a higher level of detail. The Cornell box is placed at the center of the circles. As is apparent from the image, there are very few areas where the level of detail is higher than it is supposed to be, and even in the cases where it is, it is only increased by a single level. In the area of the light beam, the error-aware renderer appears to be doing a remarkable job refining the image quality.

An aspect of the render that wasn't previously discussed but is a part of these images is the inclusion of a decay factor. The error made by sampling 3.4(b) instead of 3.4(a) is obviously extremely high, however there is basically no error for levels of detail beyond LOD 1. Simplifying a plane does not result in a large error. To tell the renderer during the ray traversal that it needs to refine e.g. a LOD 2 voxel that is part of the grid, it is necessary to add the child errors to a parent's own error value. A decay factor is needed to not propagate this error through the whole octree at full strength, more on this in 5.

To show that this approach doesn't just work for a single, purpose-built scene, a section of the forest dataset (4) was selected for early tests. The area features a slackline which has a high geometric variance.

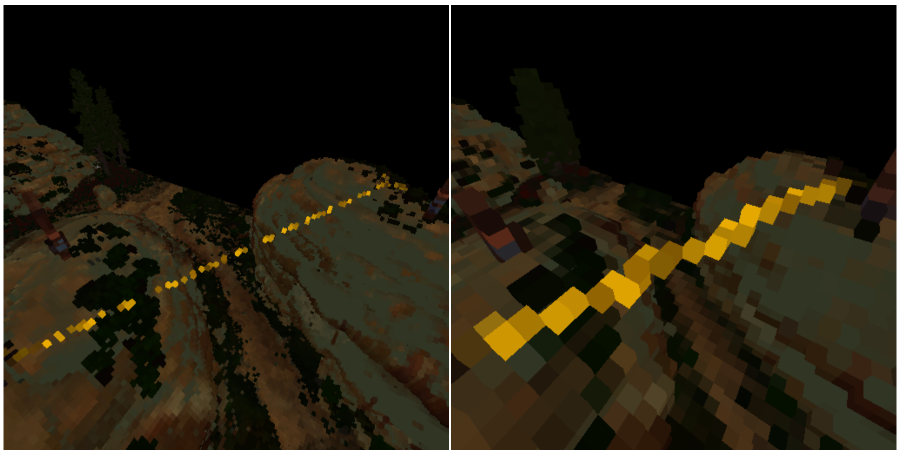

(a) LOD 0 (b) LOD 2

Figure 3.7: Slackline dataset example

3.7 shows the issue with the slackline due to the level of detail system. Since a node in the octree is created as soon as there is at least one child node, the thickness of the voxel line increases with lower levels of detail which leads to the slackline covering an disproportional area of the rendered image in those cases.

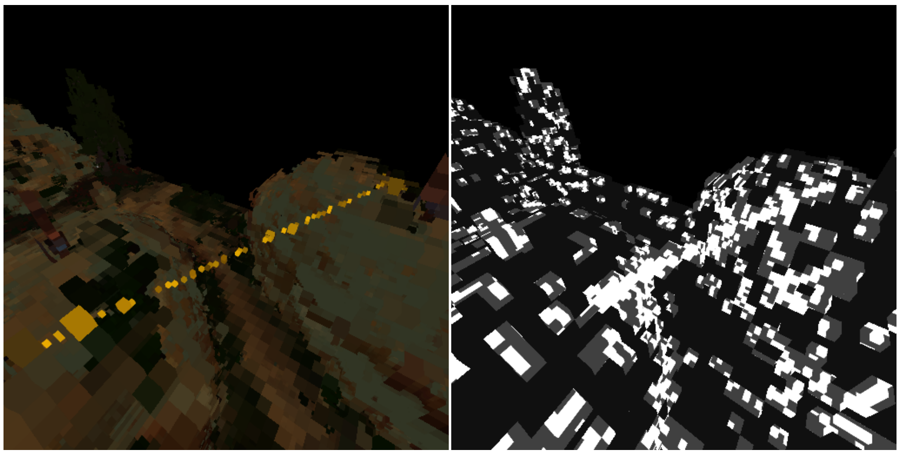

(a) Error-aware rendering result (b) Refinement level

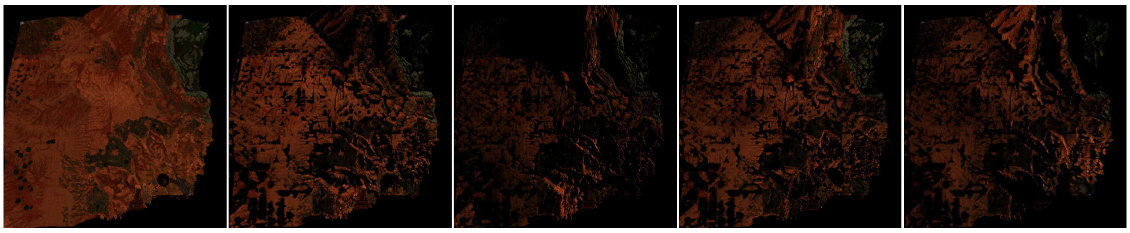
Figure 3.8: Slackline error-aware rendering test results

3.8(a) shows the rendered image with the error-aware level of detail enabled. 3.8(b) visualizes the nodes that were refined more drastically. The results here confirm that the algorithm works just as well for this area of the dataset. A full set of tests and comparisons can be found in 5.

4 Dataset

In addition to the purpose-built test case shown previously, the error-aware adaptive rendering algorithm introduced in this paper was tested on a production dataset. This allows for a detailed investigation of its performance compared to an adaptive rendering approach. It shows what kind of data is handled well by the algorithm and in which areas it fails to outperform existing solutions.

The data is from the Horizon: Zero Dawn video game, which was developed by Guerrilla Games. It was created by voxelizing the game's environment using an orthographic camera. Testing was done on 4 different areas of the game, each one selected to be vastly different from the others in terms of terrain, vegetation and structures. They feature a wide range of colors, texture and geometric detail. The selection therefore covers a wide variety of possible data situations and reflects accurately what results can be expected in a production environment.



(a) Albedo (b) Time of day 0 (c) Time of day 1 (d) Time of day 2 (e) Time of day 3

Figure 4.1: Lighting scenarios in dataset

For each area there are 5 different lighting setups available, as shown in 4.1. The first one is Albedo, i.e. the color of the voxels without any lighting applied to them. The other 4 are the emission of the voxels during different times of day. This lighting information depends only on the position of the sun and the self-shadowing of the terrain that occurs because of it. It is expected that the image variance-based adaptive rendering approach profits from large, dark areas in the images in terms of required ray counts and therefore performance. The opposite is expected for the quality of the image in those areas. The error-aware adaptive rendering approach is geometry-based and ray counts are not dependant on image information. They are therefore identical across all lighting scenarios.

Each area covers 512^2 meters of terrain with a voxel resolution of $25cm^3$. The size of the voxel space is therefore 2048^3. As a result, the sparse voxel octree used to accelerate rendering has 12 levels (including 1^3).

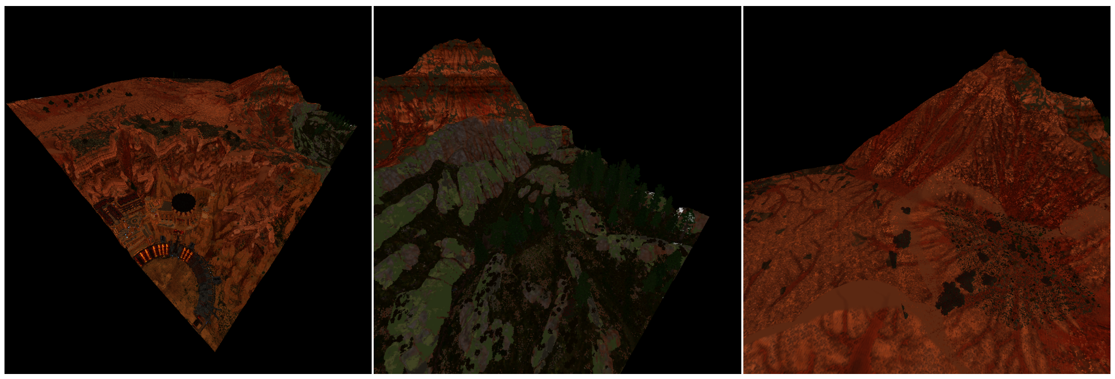
Due to the sparse nature of the voxel octree used here, the number of voxels on the lowest level is way less than 2048^3 and therefore the amount of nodes in the octree is a lot smaller as well. While this drastically lowers memory requirements, it leads to large differences in voxel and node count between different areas of the dataset.

Metric	Desert	Forest	Mountain	Biomes
Voxels	18,625,873	28,115,001	26,437,357	14,592,384
Octree Nodes	26,012,147	40,829,320	38,335,931	21,162,056

Table 4.1: Voxel and octree node counts in dataset

4.1 Desert

The desert area is located at the outer edge of the game's world. It features large structures, rock formations and the least amount of vegetation out of all of the areas in the dataset. The little vegetation present does however provide two test cases of interest. There are a couple of sparsely placed trees, as well as a small, dark forest. There is a lot of noise in the terrain's texture which is difficult to handle for all rendering approaches. The frequency of geometric and texture detail in the data varies drastically across the area and it allows for observations regarding ray counts in respect to the detail level.



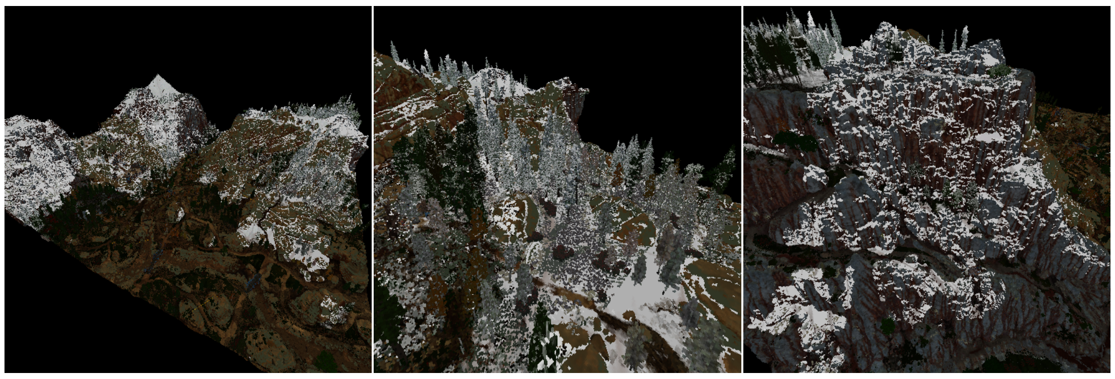
(a) Overview (b) Forest area (c) Noisy mountain

Figure 4.2: Desert area overview

In 4.2(a), the valley with the city is clearly visible in the foreground. There are a lot of small details in this area that need to be captured by the renderer. Contrary to that, it is apparent that there is not a lot of detail on the plateau as there are almost no assets placed on it. The aforementioned group of very isolated trees is placed towards the left corner of the terrain. 4.2(b) shows a closer look at the dense forest area on the right side of the overview image. While there is a high variance in geometry, there is only little variance in color. Lastly, 4.2(c) shows a closer look at the mountain on the right side of the overview image, behind the forest. It is an area with a very high amount of noise in the terrain's texture.

4.2 Forest

The forest area is the part of the dataset with the highest variance in the rendered images. This is due to the large amounts of trees in it, the altitude changes and exposed rock formations, as well as the snow that covers parts of the area. The surrounding mountains lead to large, shadowed areas in the data during different times of day which highlights the performance gains of the adaptive renderer for those scenarios.



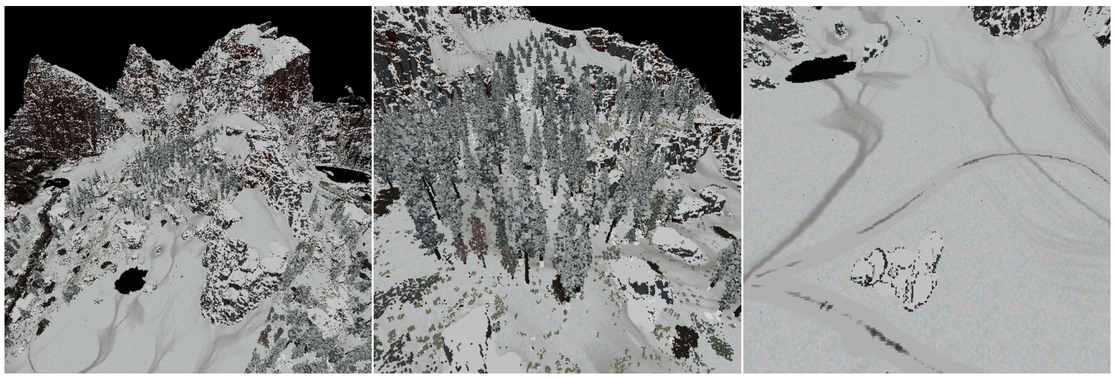
(a) Overview (b) Forest area (c) Rock formations

Figure 4.3: Forest area overview

4.3(a) better highlights the density of the vegetation, as well as the high changes in altitude between the valley and the mountain range. It provides a better impression of the rendering challenges in this area. 4.3(b) shows a section that is especially difficult to render in more detail. In addition to the noisy, dense vegetation, there are sections of the forest that are covered in snow as well. The high geometric variance there is paired with a high variance in color. This allows for good comparisons of the two rendering approaches, as they each base their refinements on one of the two aspects. 4.3(c) shows a closer look at another challenging aspect of this region. The various rock formations have a wide range of frequencies in geometric detail as well as a variety of colors. The additional snow makes those areas great test cases for direct comparisons between the two rendering approaches as well, without relying on vegetation for high geometric detail.

4.3 Mountain

Since snow appears to drastically affect the variance in color within the data, the mountain dataset was chosen as a reference for an area covered entirely by it. In addition to that, the mountain dataset stretches across even more vertical space than the forest dataset. Images are being rendered from above during testing, and rays are usually intersecting the ground at a flat angle, for example in case of the desert dataset. Having mountains enables testing of the error-aware approach at steep angles without modifying the camera settings.



(a) Overview (b) Forest region (c) Plain region

Figure 4.4: Mountain area overview

The overview image (4.4(a)) shows the height differences in the dataset, ranging from the lowest point in the lower left corner to the highest in the upper right one. The different angles of the terrain are also clearly visible. As with the forest area from the previous section, there are snow-covered trees at medium to high density present here with the same effects as before (high variance in color and geometry, 4.4(b)). 4.4(c) shows one additional point of interest in the area. There are multiple smaller regions like it with low variance in color and geometric detail. Those regions should almost exclusively use a single ray per pixel and there should not be any refinements to image quality in them.

4.4 Biomes

The last part of the dataset used for testing is the biomes area. It features a wide variety of different biomes in it, effectively containing almost all of the previously mentioned features. It allows for comparisons within the same image and shows clearly how the algorithms perform on different data scenarios.

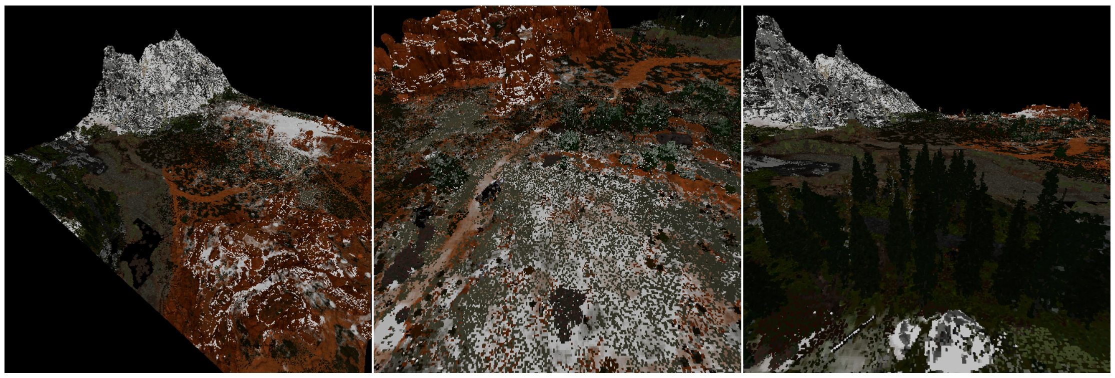

(a) Overview (b) Noisy snow (c) Forest

Figure 4.5: Biomes area overview

As shown in 4.5(a), the area combines forest, mountain, desert and even snow-covered desert areas. Therefore, the frequency of geometric detail and color varies extremely across it. Of special interest is the border between snow-covered and regular terrain, as the transition results in high levels of noise due to the partial snow coverage (4.5(b)). High variance in color results in more rays per pixel for the adaptive rendering approach, however, 4.5(c) also shows a couple of areas with varying geometric detail which should increase the ray count of the error-aware approach instead.

5 Testing

This chapter is dedicated to showing and discussing the results of the implementation when applied to the dataset. It discusses how different parameters influence the quality and ray count and how those parameters are chosen. Testing was done on all 4 datasets and with all 5 lighting situations for each of them. In addition to that, there are results for the desert dataset across different resolutions and for different error thresholds.

5.1 Methodology

In this first section of the chapter, the exact setup and methodology used for obtaining the results shown in the second chapter is discussed. The precise way of rendering using the different algorithms introduced in 2 and 3 is explained so the results can easily be comprehended.

5.1.1 Setup

A core aspect of this thesis is to use a real world, production dataset and to improve an actual production issue. The testing is purposefully set-up to keep this aspect in mind. Light probes are usually rendered as cubemaps, i.e. with a camera placed at the center pointing in the direction of the 6 faces of a cube surrounding it. Such a camera therefore has a vertical and horizontal field of view of 90 degrees and a resulting aspect ratio of 1:1.

The camera is placed above the map at the center of it, looking downwards. In the octree, the lowest level is 2048^3, the maximum size of the image is therefore 2048 by 2048 pixels if one pixel should roughly cover one voxel. Since refinements to specific pixels should be tested, the image size needs to be reduced to 1024 by 1024 pixels. A single pixel thus covers 2x2 voxels which allows for the renderer to use 4 rays instead of 1 per pixel and not to over-sample.

The error-aware renderer can now read the error value of the second-lowest level in the octree, compare it to the threshold and increase the ray count to 2x2 rays while simultaneously change the level of the octree sampled to the lowest level. This would, however, effectively not use the decay factor and not take the error of the lower octree levels into account. To properly test this aspect of the algorithm as well, an image resolution of 512x512 is used, which allows for 3 levels of the octree being sampled and the decay factor thus being taken into account for the highest of them.

Image quality is judged by rendering a reference image at a very high quality and comparing the images rendered by the adaptive renderer and the error-aware renderer to it. The reference image is rendered by using 4x4 rays for each pixel, each one sampling the lowest level of the octree (i.e. without using the LOD provided by the Octree). Thus, there is roughly one ray for each voxel in the scene (given the slight distortion due to the perspective camera).

5.1.2 Adaptive Rendering

The adaptive rendering approach refines the image quality based on variance within it. To achieve that, the whole image is rendered once with 1 ray per pixel. A kernel function is then used across the image to calculate the variance for each pixel. Depending on this variance, the respective pixels are rendered again using either 2x2 or 4x4 rays.

The octree level of detail is disabled for the refinement steps of the adaptive renderer, only the lowest level in the tree is sampled. This process can be repeated until either a set iteration count is reached or until there is not enough variance in the image to further refine it. The ray count thus increases steadily with additional iterations. Since that wouldn't be a fair comparison, the adaptive renderer in the case of this series of tests is limited to a single iteration as to not drastically increase the ray counts of its results.

In the adaptive renderer used here, the luminance of each pixel is calculated first based on its RGB value. The variance in the image is afterwards calculated by using a kernel of size 3x3 and calculating the mean luminance for this kernel. The sum of squared differences between the mean and each pixel within the kernel is then calculated. Pixels with rays that do not intersect any of the voxels (e.g. pixels that show the black background) aren't used for this as this would lead to the edges of the terrains drastically increasing the ray count. The sums of squared differences are normalized across all datasets tested (to make the ray counts comparable) and the pixels are split into 3 groups based on it.

A value of 0.75 or larger leads to a refinement for the respective pixel by casting an additional 4x4 rays for it. If the value lies between 0.25 and 0.75, 2x2 rays are being cast instead. For every other pixel, no refinement is made, and the result of the single ray is used for the pixel.

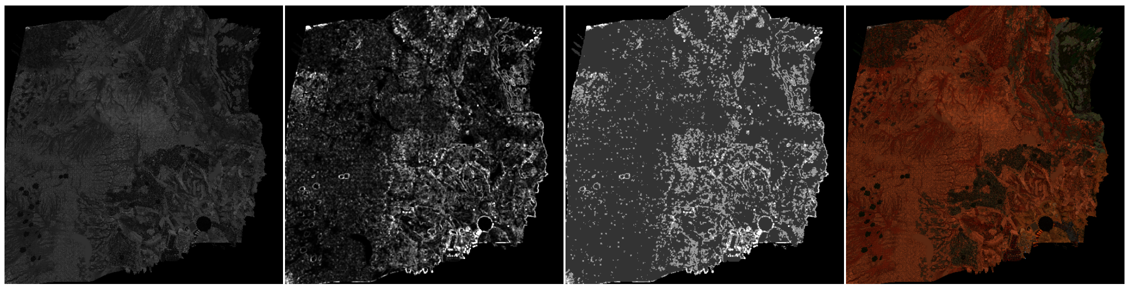

(a) Luminance (b) Variance (c) Ray count (d) Result

Figure 5.1: Adaptive rendering algorithm and results

5.1(a) shows the luminance for the single ray per pixel adaptive image. 5.1(b) is the normalized variance for it. For every of the following tests, the ray count image (5.1(c)) is displayed. For every pixel there are 4 possible values, each one representing a specific ray count used for the respective pixel. From black to white those represent the background (0 rays per pixel), 1 ray per pixel, 4 rays per pixel and 16 rays per pixel. The result is shown in 5.1(d), however, given the scale of the images in this thesis it is not feasible to use the overview images for visual comparisons between the different rendering approaches. The different image similarity metrics should be used for an assessment of image quality.

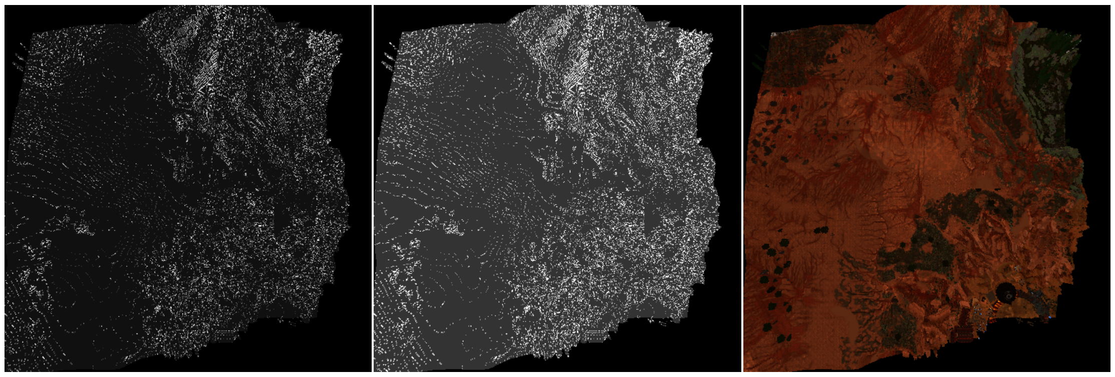
Lastly, the traversal step count (i.e. the amount of octree nodes stepped through) is calculated. In case of the adaptive renderer, the traversal step count is directly related to the ray count.

5.1.3 Error-Aware Rendering

The level of detail system is enabled for the error-aware renderer. As described in 5.1.1, the cone of the cone tracer is set up so that one node in the third-lowest level of the octree roughly covers a single pixel in the final image (slightly distorted due to the perspective camera). A single ray per pixel is cast and the error value is calculated for the node given the ray direction

If the error is larger than the threshold, the ray is discarded and 2x2 rays are being cast. At the same time the level of detail used for sampling is decreased by 1 (the second-to-lowest level of the octree). For each of the quadrants, the same error check is performed. If the error here is above the threshold once more, the ray for the quadrant is discarded again and 2x2 rays are being cast in its place. The level of detail sampled by those is now the lowest level of the octree.

The ray count for each pixel can therefore be a maximum of 21, which would be 5 discarded rays and 16 rays that actually produce samples. Contrary to the adaptive rendering approach, refinements in the error-aware approach come with the cost of discarding rays that aren't used and this cost is factored into the ray and traversal counts that are being presented in this chapter. In theory, it is possible to not discard rays for lower levels of refinement and instead use them as one of 4 rays. Since this ray would however go through the center of the 2x2 grid, the distribution of the 4 rays would not be uniform and as a result some information might get lost during sampling.



(a) Ray count (b) LOD level (c) Result

Figure 5.2: Error-aware rendering algorithm and results

5.2(a) shows the numbers of rays cast for each pixel. Black represents 1 ray per pixel, while white represents 21. 5.2(b) shows the maximum level of detail used for each pixel, i.e. a white color if at least one of the rays for the pixel samples the lowest level of the octree. This image is qualitatively comparable to the adaptive rendering ray count one, however not quantitatively as it is possible that only a couple of rays sample a higher level of detail. As with the adaptive renderer, rays that do not hit a voxel are not being counted.

While the two images allow for an overview of the regions for which the image quality is refined, the actual ray counts and traversal step counts should be used for performance comparisons. Since it is not possible to determine the traversal step counts from either 5.2(a) or 5.2(b), they are being counted during the rendering process.

5.2(c) shows the result of the error-aware rendering approach. As described before, image similarity metrics should be used for assessments of image quality.

5.1.4 Decay Factor and Error Threshold

Given the way the error values for a node are calculated (see 3), it is possible for a node to have a low value for a specific direction even though one of its child nodes has a high one. Since the octree is stepped through from top to bottom, such a scenario would result in no refinements and a lot of detail getting lost.

To counter this behaviour, the error values of the child nodes need to be considered when calculating the error value of an octree node during traversal. A decay factor needs to be multiplied with children's error to prevent higher nodes from having error that are always above the threshold (which would make the algorithm obsolete, the result would be the same as the reference rendering).

In addition to that, the decay factor needs to be scaled with the number of children to prevent nodes with high child counts to have disproportionate amounts of refinement steps. The resulting formula to calculate the error of a node N given a ray r is

$$E_N(r) = e(r, N) + \sum_{c \epsilon C} \frac{d * e(r, c)}{|C|}$$

where $e(r, N)$ is the error for the node itself, C are the children of the node and d is the decay factor. The combination of the decay factor and error threshold determines the ray count of the renderer. To properly compare the effectiveness of the error-aware approach across different areas in comparison to the adaptive renderer, it is necessary that the two variables are identical for all tests.

The desert dataset serves as a point of reference for all tests. It is used in some more specific tests and to determine the decay factor and error threshold. To achieve that, a series of renderings is done with varying values and the ray counts are compared to the ray count of an image rendered using the adaptive approach, which is 395,162.

Decay factors are tested for [0.05;0.3], with a step size of 0.05.

		Error Threshold					
		0.175	**0.15**	**0.125**	**0.1**	**0.075**	**0.05**
	0.05	135,677	285,193	478,125	810,805	2,029,727	2,379,774
	0.1	142,195	292,495	510,337	849,472	2,137,759	2,506,243
Decay	**0.15**	157,248	327,704	566,407	943,173	2,378,533	2,784,862
Factor	**0.2**	195,622	418,090	682,518	1,113,927	2,584,319	3,108,818
	0.25	248,535	515,262	898,454	1,291,458	2,964,559	3,835,181
	0.3	375,238	768,187	1,358,136	2,087,938	4,321,629	5,103,326

Table 5.1: Desert dataset ray counts for different decay factors and error thresholds

For each decay factor step, the error threshold is determined for which the ray count is as close to the adaptive result as possible. Those settings are compared in terms of image quality using the same image similarity metrics as all tests in this chapter (see next section). The best result is a decay factor of 0.2 in combination with an error threshold of 0.154. These values will be used for testing going forward.

5.1.5 Image Similarity Metrics

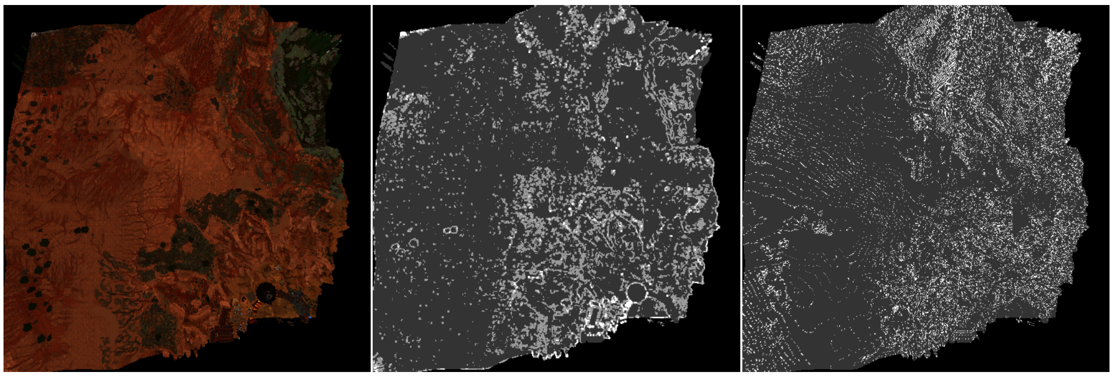
There are three different image similarity metrics [Mül+20] used to determine the quality of the renderings.

- Peak Signal-To-Noise Ratio (PSNR): PSNR was introduced in 1998 as an alternative to RMSE [Vel10]. It is often used for measuring the quality of image compression and reconstruction algorithms. However, it has a variety of shortcomings [KY12]. PSNR is a relative measure, which means that the result depends on the content visible on the images. Since it is only used to compare the effectiveness of different rendering algorithms on the same content this is not an issue here. The other major issue is, that the PSNR is measured in dB and does not operate on a linear scale. This has to be considered when judging differences in ratio between multiple datasets.
- Root Mean Square Error (RMSE): The root mean square error between two images provides an easy to understand baseline for image comparison. This value is related to image intensity and should therefore also only be used to compare images of the same content. For comparisons of multiple datasets only the ratio between the different rendering approaches should be used.
- Structural Similarity Index Measure (SSIM): While SSIM has some similarities to PSNR [HZ13], it delivers a more subjective comparison and more closely mirrors the impressions humans have when comparing visual quality [HB12].

5.2 Results

This section contains the results of all the tests done on the dataset with albedo lighting using the previously described testing methodology. Its purpose is to evaluate the comparability of the different areas without having shadows as an additional factor and to have a baseline for the lighting scenarios discussed in the next chapter.

5.2.1 Desert Dataset



(a) Error-aware rendering result (b) Adaptive rendering ray count (c) Error-aware rendering maximum LOD level

Figure 5.3: Desert dataset results

As expected, the adaptive renderer focuses its refinements on the city area, the forest in the upper right corner and the mountain area (5.3(b)). It does seem to fail handling the sparse vegetation in the lower left area, as well as the vegetation in the upper left image. It also struggles with the noisy area between the forest and the city on the right side of the image.

Contrary to that, the error-aware renderer appears to be handling the low-density vegetation better, refining the image quality for individual trees, as well as the forest more consistently. Is is also clearly visible in 5.3(c) that the refinements follow the terrain closely, e.g. improving the quality of the edges due to elevation changes on the left side of the image.

Rendering Method	Ray Count	Rays / Pixel	Traversal Steps
Adaptive	395,162	1.888	3,557,270
Error-Aware	394,242	1.886	3,582,041

Table 5.2: Desert dataset ray counts and traversal step counts

Since this rendering scenario is used to determine the decay factor and error threshold used, the ray counts are almost identical here (within 0.23%). Even though the ray count for the error-aware approach is slightly lower, the traversal step count is higher. This highlights the additional traversal cost of this approach.

The ray counts here are the lowest for all areas tested. This suggests that the overall variance in color and geometry is lower than in the other areas. This is supposedly mostly due to the lack of vegetation in most of the area, as this tends to add lots of noise to the data.

Rendering Method	Didn't Hit	1 Ray / Pixel	4 Rays / Pixel	16 Rays / Pixel
Adaptive	52,886	169,475	37,088	2,347

Table 5.3: Desert dataset adaptive rendering pixel counts

As described earlier, rays that do not hit the voxels are not being added to the total count in 5.2. Due to the low amounts of noise in the desert area and the noisy areas being handled poorly by the adaptive renderer (e.g. the isolated trees) the overall number of refined pixels is comparably also on the lower end of the results.

Rendering Method	Didn't Hit	LOD 2	LOD 1	LOD 0
Error-Aware	53,121	181,380	14,256	13,387

Table 5.4: Desert dataset error-aware rendering maximum LOD level used

The number of pixels with rays that do not hit any voxel is comparable between approaches for obvious reasons. The distribution of pixel counts between the refinement levels is shifted towards the maximum level of refinement. This can be adjusted using the decay factor, during testing this was, however, determined to be the best possible scenario for the error-aware renderer when rendering this dataset.

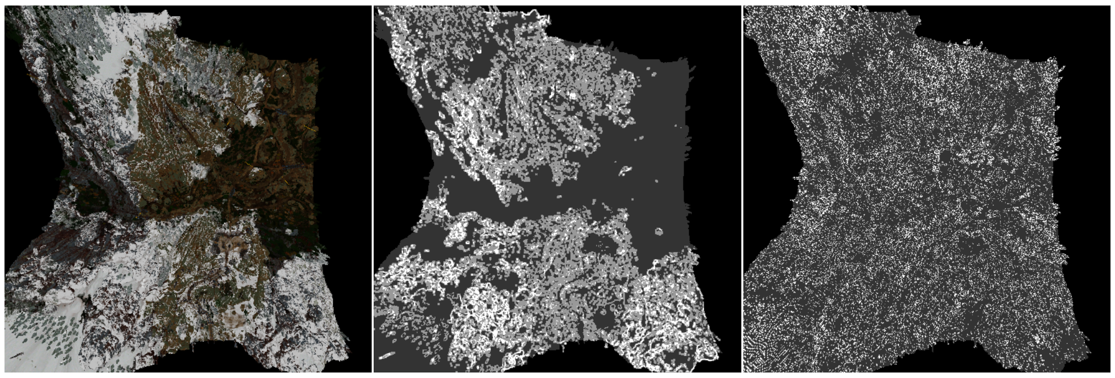
Rendering Method	PSNR	RMSE	SSIM
Adaptive	50.913	0.00285	0.991
Error-Aware	53.937	0.00201	0.995

Table 5.5: Desert dataset image similarity to reference image

Given the close difference in traversal step counts here, this test case is a good point of reference for image quality differences between the two rendering approaches when compared to the reference image. The PSNRs (higher is better) are the highest values of all 4 areas. In combination with the low ray counts this confirms that the noise in the desert area is comparably low.

The RMSEs (lower is better) confirm that the error-aware algorithm is achieving higher quality results in this case. The difference in the more subjective SSIMs (higher is better) is small, suggesting that a viewer might perceive the differences in image quality not as much.

5.2.2 Forest Dataset



(a) Error-aware rendering result   (b) Adaptive rendering ray count   (c) Error-aware rendering maximum LOD level

Figure 5.4: Forest dataset results

As can be seen in 5.4(b), the adaptive renderer does ignore almost the entire valley in terms of refinements. This appears to be the result of the low variance in luminance due to the high vegetation density. The error-aware renderer (5.4(c)) handles it better and refines lots of pixels throughout the valley.

While the error-aware rendering result looks noisy, there are multiple regions with a low refinement density, suggesting that the noise is part of the data rather than being introduced by the renderer. The number of trees and rock formations in the environment support this further, as the geometric detail is high throughout the area.

Opposite to the valley, the adaptive renderer has to drastically increase its ray count and thus the traversal steps for the mountains. The extremely noisy snow does seem to cause issues for it, while the error-aware renderer can rely on the level of detail system, as well as the more selective, geometry-based selection approach to keep its ray count comparably low.

Rendering Method	Ray Count	Rays / Pixel	Traversal Steps
Adaptive	656,205	3.425	6,562,050
Error-Aware	486,912	2.545	4,172,056

Table 5.6: Forest dataset ray counts and traversal step counts

The aforementioned results are highlighted clearly by the absolute ray count for the image, the adaptive render needs 3̃5% more rays and 5̃7% more traversal steps. All counts are higher than the ones for the desert area.

Rendering Method	Didn't Hit	1 Ray / Pixel	4 Rays / Pixel	16 Rays / Pixel
Adaptive	70,558	100,733	74,848	16,005

Table 5.7: Forest dataset adaptive rendering pixel counts

It is expected that the relative gap between ray counts and traversal steps is lower for the latter due to the rendering overhead of the error-aware renderer, however this is not the case here. The high amount of refined pixels, especially 4 ray pixels, provides a clear indication as to why this is the case.

Rendering Method	Didn't Hit	LOD 2	LOD 1	LOD 0
Error-Aware	70,800	149,667	19,330	22,347

Table 5.8: Forest dataset error-aware rendering maximum LOD level used

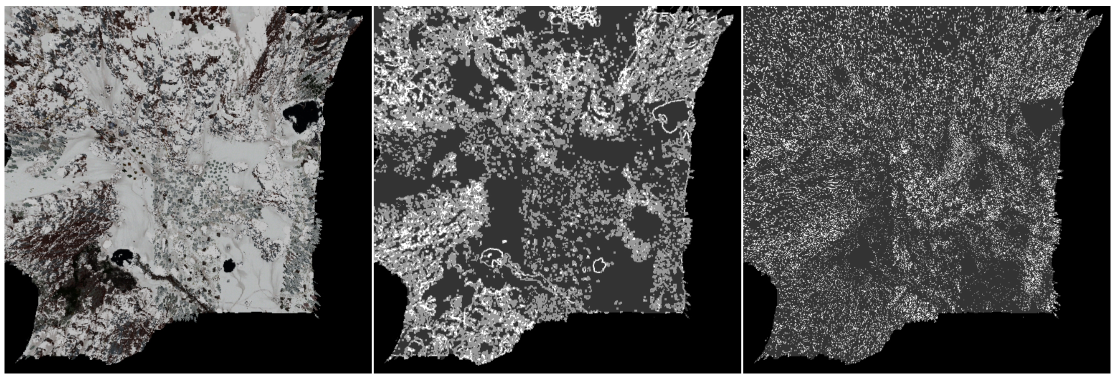
Contrary to that, the number of unrefined pixels is way higher in case of the error-aware approach. Those pixels can use the LOD system and thus save lots of traversal steps in the process. The results here suggest that the algorithm can handle noise within a dataset way better in terms of performance.

Rendering Method	PSNR	RMSE	SSIM
Adaptive	41.666	0.00825	0.939
Error-Aware	43.197	0.00692	0.957

Table 5.9: Forest dataset image similarity to reference image

In addition to the performance benefits the error-aware renderer still outperforms the adaptive one in all metrics. Compared to the desert dataset results the overall quality of both compared to the reference image is worse, suggesting that even though the traversal step counts are increased the image is still largely under-sampled. Given the noise in the dataset this is, however, to be expected.

5.2.3 Mountains Dataset



(a) Error-aware rendering result (b) Adaptive rendering ray count (c) Error-aware rendering maximum LOD level

Figure 5.5: Mountain dataset results

The sharp difference in luminance between the rocks and the snow leads to high levels of variance for the adaptive renderer (5.5(b)). This might also be the reason why the forest in the center of the area is detected more reliably by the adaptive renderer compared to the valley in the forest dataset.

This area shows the worst-case for the error-aware approach. Geometric variance in large, monotonous colored patches of ground leads to unnecessary refinements in 5.5(c).

Rendering Method	Ray Count	Rays / Pixel	Traversal Steps
Adaptive	749,468	3.762	6,533,670
Error-Aware	475,532	2.385	4,200,987

Table 5.10: Mountain dataset ray counts and traversal step counts

As a result, the performance numbers for this area show a similar pattern to the forest area in the previous section. The additional snow in this one, in combination with the forest refinements, appears to be the reason for an even larger performance gap between the two renderers.

Rendering Method	Didn't Hit	1 Ray / Pixel	4 Rays / Pixel	16 Rays / Pixel
Adaptive	62,908	103,135	82,910	13,662

Table 5.11: Mountain dataset adaptive rendering pixel count used

The pixel counts are comparable to the forest dataset as well. The number of pixels with 4 rays per pixel is high compared to the other areas which appears to be mostly due to the forest and some larger rock formations at the top of the area.

Rendering Method	Didn't Hit	LOD 2	LOD 1	LOD 0
Error-Aware	62,798	158,857	20,238	20,251

Table 5.12: Mountain dataset error-aware rendering maximum LOD levels

There are no such outliers for the error-aware renderer. The numbers here are in the same range as those from the desert dataset, suggesting that the overall level of geometric detail of both areas is comparable.

Rendering Method	PSNR	RMSE	SSIM
Adaptive	40.684	0.00924	0.928
Error-Aware	41.890	0.00804	0.945

Table 5.13: Mountain dataset image similarity to reference image

Relative to the pixel counts, the PSNR and SSIM metrics are comparable to the forest dataset as well. The error-aware renderer again produces higher quality results while at the same time needing significantly fewer rays and traversal steps to achieve this result.

5.2.4 Biomes Dataset

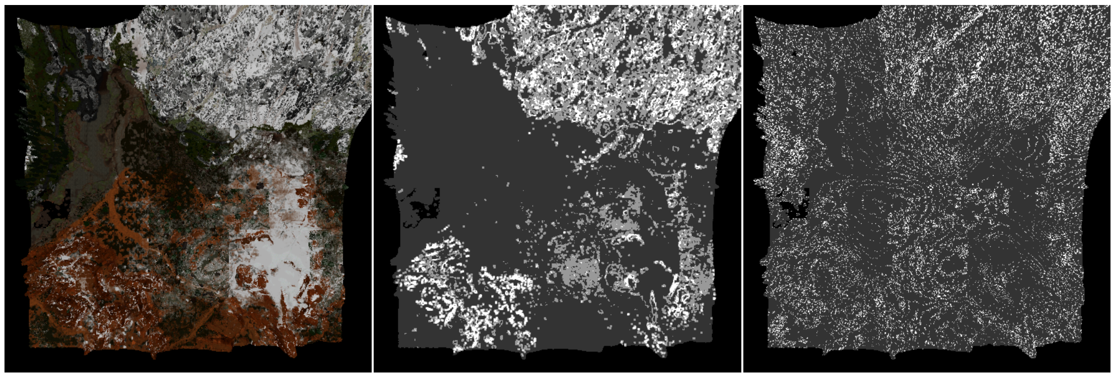

(a) Error-aware rendering result (b) Adaptive rendering ray count (c) Error-aware rendering maximum LOD level

Figure 5.6: Biomes dataset results

The combination of biomes in a single area achieves the desired result in showing clearly the advantages and disadvantages of the different renderers. 5.6(b) confirms that the snow is extremely difficult to handle by the adaptive renderer in terms of performance. Opposite to that, the renderer struggles with rendering the forest area once again as the color variance is not large enough. This makes it impossible for an adaptive renderer to be set up in a way to effectively render this mixed area. It either requires low variance thresholds in order to be able to increase the quality of the forest which would drastically increase the ray counts or do the opposite to decrease the ray count which would lower the quality even more.

5.6(c) shows that the error-aware approach is far more versatile and that the refinements are more evenly distributed across the area. This indicates that the solution allows for quality-performance trade-offs.

Rendering Method	Ray Count	Rays / Pixel	Traversal Steps
Adaptive	616,187	2.877	6,161,870
Error-Aware	456,034	2.130	3,992,801

Table 5.14: Biomes dataset ray counts and traversal step counts

Even though the adaptive renderer doesn't refine large portions of the area, the high variance in the snow-covered areas still leads to higher ray counts and traversal counts for the adaptive renderer. The ability of the error-aware renderer to use the LOD system outweighs the additional traversal step cost of the algorithm once again.

Rendering Method	Didn't Hit	1 Ray / Pixel	4 Rays / Pixel	16 Rays / Pixel
Adaptive	47,996	145,471	52,343	16,334

Table 5.15: Biomes dataset adaptive rendering pixel counts

The pixel counts for the adaptive renderer reflect what can be seen in 5.6(b). Compared to the previous two areas, the number of pixels with 2x2 rays is significantly lower and the number of single ray pixels respectively higher. Given the large area in the image without refinement this is to be expected. However, the number of pixels with 4x4 rays is comparable. The high frequency noise, especially around the mountain and in the snow-covered desert appears to be the reason for this.

Rendering Method	Didn't Hit	LOD 2	LOD 1	LOD 0
Error-Aware	48,087	179,243	17,837	16,977

Table 5.16: Biomes dataset error-aware rendering maximum LOD level used

For the error-aware renderer, the pixel counts are all in the range of previous datasets, suggesting that there are no major outliers in the data in terms of geometric variance. The desert datasets counts are lower for LOD 0 and LOD 1, but this is to be expected given the sparse vegetation.

Rendering Method	PSNR	RMSE	SSIM
Adaptive	41.535	0.00838	0.937
Error-Aware	44.221	0.00615	0.964

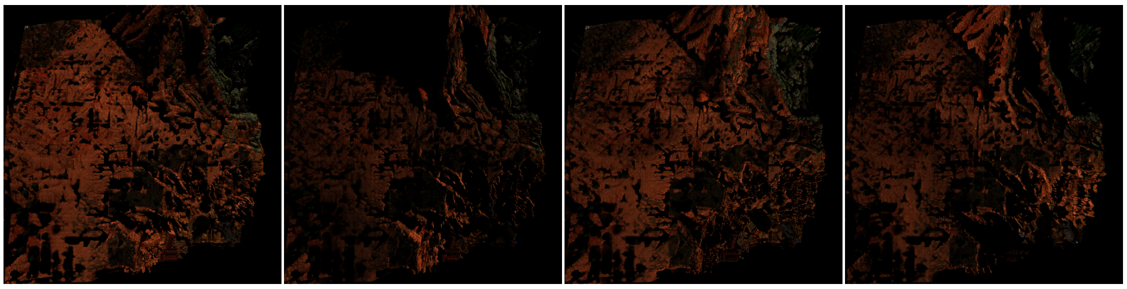
Table 5.17: Mountain dataset image similarity to reference image

The error-aware renderer outperforms the adaptive one by a wide margin given the difference in total ray counts. This is also mostly attributable to the large area of the adaptive rendering without any refinements.

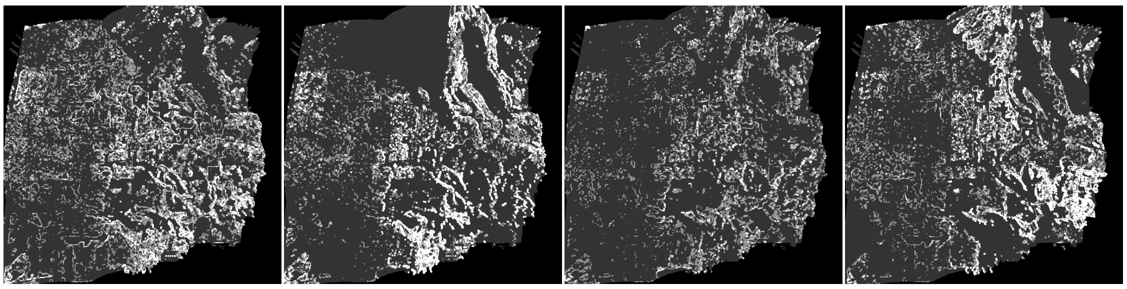
5.3 Lighting

This section shows comparisons of the different rendering algorithms for all 4 areas of the dataset (as described in 4) across the 4 different times of day.

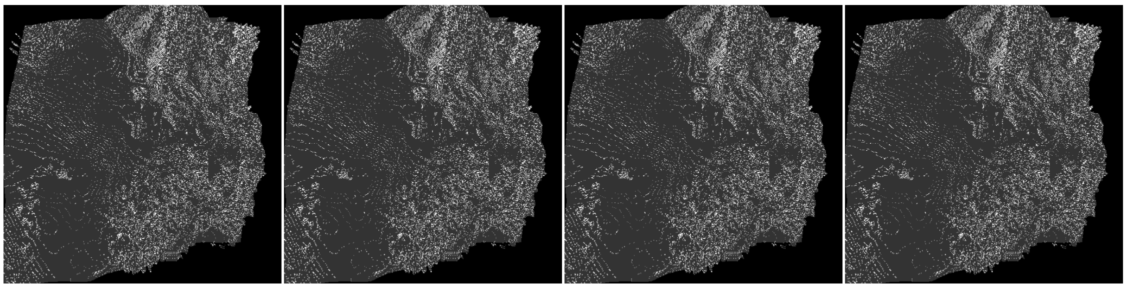
5.3.1 Desert Dataset



(a) Error-aware rendering results



(b) Adaptive rendering ray counts



(c) Error-aware rendering maximum LOD levels

Figure 5.7: Desert dataset results for different times of day

It is generally expected that shadowed areas help the adaptive renderer to catch up to the error-aware one in terms of traversal step counts due to the luminance being 0 in those areas (and thus no refinements being necessary). While this is the case here, even for large areas (as visible in 5.7(b, 2)), the overall additional noise introduced appears to far out-weight this advantage.

In addition to that, the sharp contrast between shadow and terrain luminance leads to an additional, large amount of pixels being refined using 4x4 rays instead of only 2x2.

Time of Day	Ray Count (A)	Ray Count (EA)	Traversal Steps (A)	Traversal Steps (EA)
Time 0	702,412	394,242	5,838,850	3,582,041
Time 1	709,204	394,242	6,055,420	3,582,041
Time 2	488,808	394,242	3,172,754	3,582,041
Time 3	705,836	394,242	5,970,920	3,582,041

Table 5.18: Desert dataset ray counts and traversal step counts for different times of day

Time 1 clearly shows this effect, while there are lots of shadowed areas there is also high variance and thus the adaptive renderer cannot improve as was expected. Overall, none of the lighting scenarios is close to the albedo ray count of the adaptive renderer (395,162). It is difficult to pinpoint why Time 2 performs that much better compared to the others. The average luminance of the images can be observed to see if there is a relation to the ray count of the image.

	Time 0	Time 1	Time 2	Time 3
Luminance	0.0864	0.0324	0.0732	0.0541

Table 5.19: Desert dataset average image luminance for different times of day

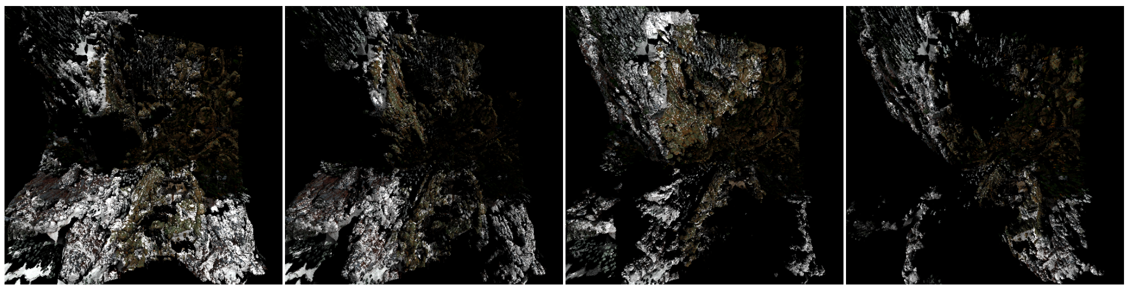
5.19 shows that there is no direct relation between the two. A closer look at 5.7 suggests that it is not the average luminance, but rather the distribution of it across the image that is important for the ray count.

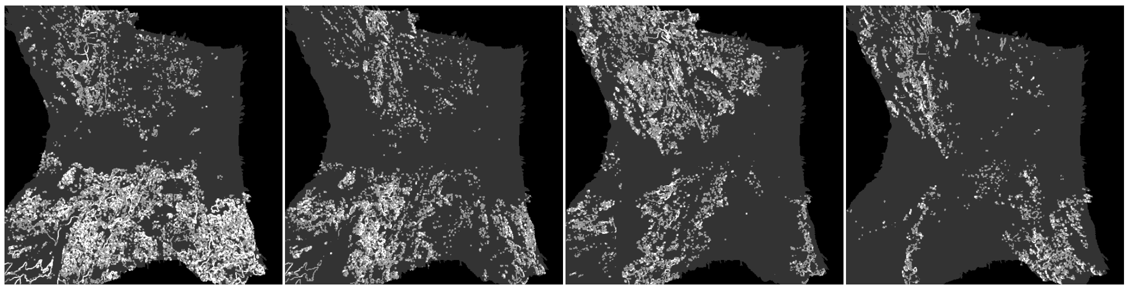
Time of Day	PSNR (A)	PSNR (EA)	RMSE (A)	RMSE (EA)	SSIM (A)	SSIM (EA)
Time 0	49.955	52.411	0.00318	0.00239	0.988	0.993
Time 1	53.116	54.901	0.00221	0.00179	0.994	0.996
Time 2	50.291	52.551	0.00306	0.00236	0.989	0.993
Time 3	51.551	54.528	0.00265	0.00188	0.992	0.996

Table 5.20: Desert dataset image similarities to reference image for different times of day - Adaptive to reference (A) and error-aware to reference (EA)

However, the image quality metrics hint at a relation between the difference in image quality and the average luminance of an image. Overall darker images shrink the margin between adaptive and error-aware. It is interesting to see that even though the error-aware renderer does not take lighting into consideration, it is still able to outperform the adaptive approach qualitatively and that the difference in ray count is extremely high at the same time.

5.3.2 Forest Dataset


(a) Error-aware rendering results


(b) Adaptive rendering ray counts

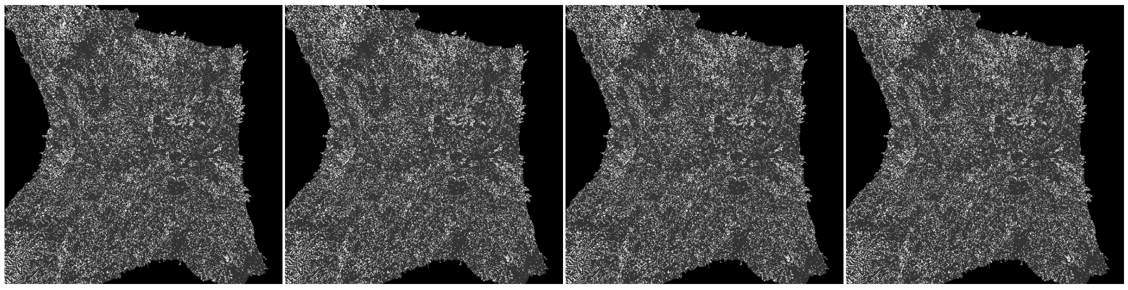

(c) Error-aware rendering maximum LOD levels

Figure 5.8: Forest dataset results for different times of day

The high mountains surrounding the area lead to a shadowed valley at every time of day (see 5.8). Since the valley caused issues for the albedo tests as well, this shouldn't affect the performance counts and quality metrics as much. As the mountains are already noisy without shadows, the lighting only reduces refinement counts in 5.8(b) compared to the albedo test.

Time of Day	Ray Count (A)	Ray Count (EA)	Traversal Steps (A)	Traversal Steps (EA)
Time 0	688,756	486,912	5,572,030	4,172,056
Time 1	472,644	486,912	3,655,450	4,172,056
Time 2	507,992	486,912	3,938,220	4,172,056
Time 3	390,532	486,912	2,967,970	4,172,056

Table 5.21: Forest dataset ray counts and traversal step counts for different times of day

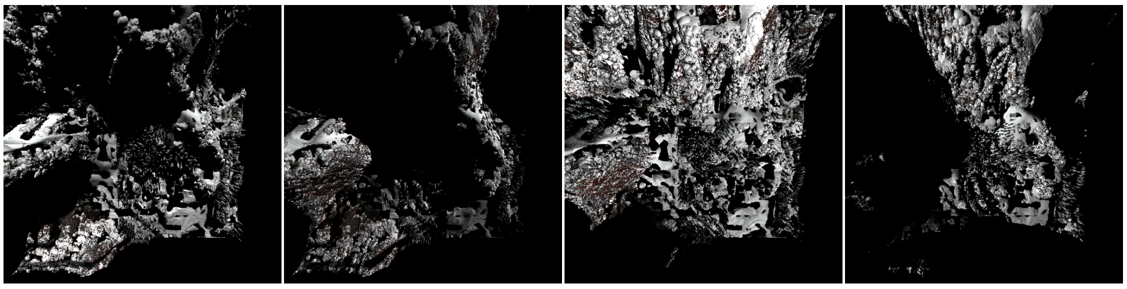
The ray counts reflect that the shadows do not add additional variance to the image but rather only reduce it. Compared to the albedo ray count (749,468), all times of day require less rays to render the image. Time 3 represents a rare and special case, as the ray count is lower than the one of the error-aware renderer. For the traversal step counts, this is even true for Time 1, Time 2 and Time 3.

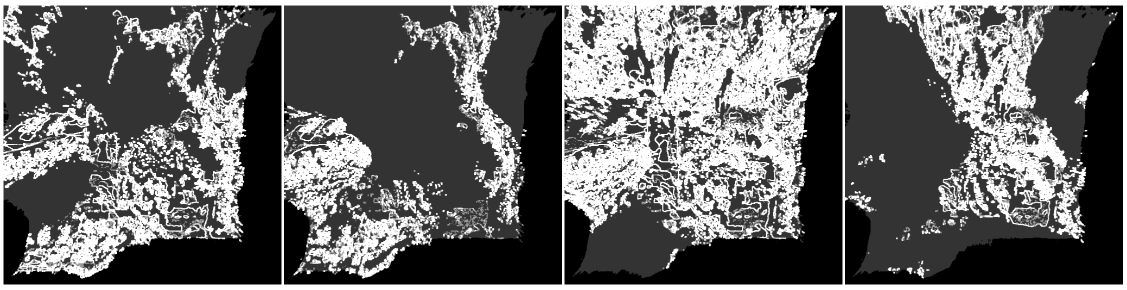
Time of Day	PSNR (A)	PSNR (EA)	RMSE (A)	RMSE (EA)	SSIM (A)	SSIM (EA)
Time 0	41.339	42.659	0.00857	0.00736	0.941	0.957
Time 1	43.928	44.697	0.00636	0.00582	0.963	0.970
Time 2	42.359	43.930	0.00762	0.00636	0.949	0.964
Time 3	44.364	47.204	0.00605	0.00436	0.966	0.982

Table 5.22: Forest dataset image similarities to reference image for different times of day - Adaptive to reference (A) and error-aware to reference (EA)

The high noise levels of the data, in addition to the large areas in shade, help the adaptive renderer to shorten the gap to the error-aware renderer. In the case of Time 2, one could even argue that both approaches produce similar results, given the lower performance cost of the adaptive approach.

5.3.3 Mountain Dataset


(a) Error-aware rendering results


(b) Adaptive rendering ray counts

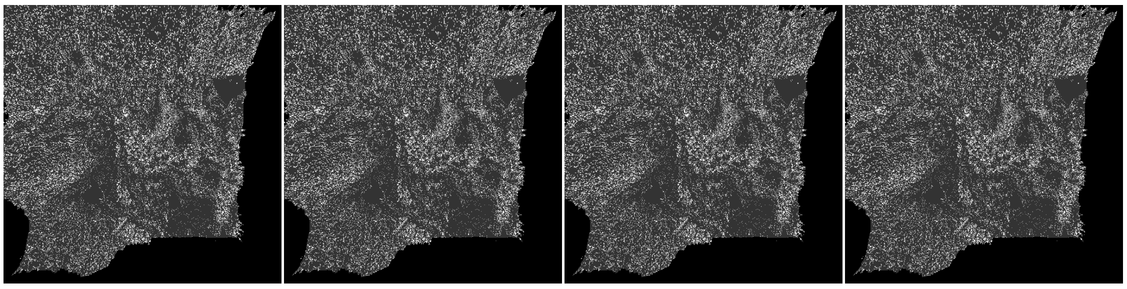

(c) Error-aware rendering maximum LOD levels

Figure 5.9: Mountain dataset results for different times of day

The effect of the shadows on the mountain dataset appears to be similar to that on the forest area. The overall variance of the albedo image already is quite noisy and there is not much noise being added by smaller shadows. Large dark areas, as clearly visible in 5.9(b), help to reduce the amount of refined pixels drastically. This is especially apparent in the second and the fourth image.

Time of Day	Ray Count (A)	Ray Count (EA)	Traversal Steps (A)	Traversal Steps (EA)
Time 0	544,532	475,532	4,386,570	4,200,987
Time 1	423,948	475,532	3,338,230	4,200,987
Time 2	878,388	475,532	7,283,190	4,200,987
Time 3	473,056	475,532	3,761,310	4,200,987

Table 5.23: Mountain dataset ray counts and traversal step counts for different times of day

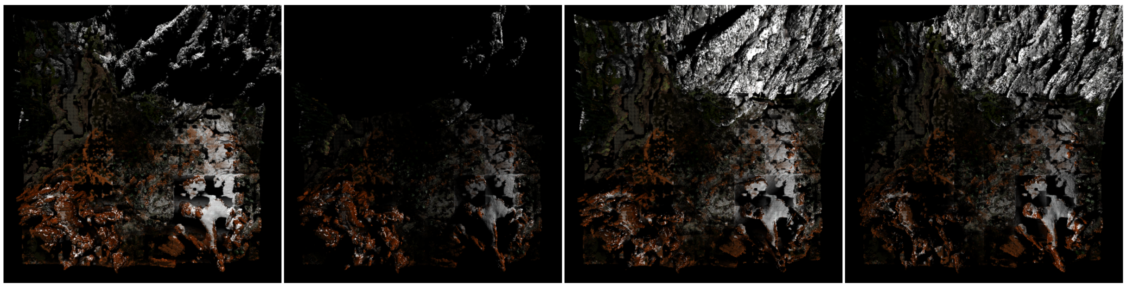
The ray and traversal step counts reflect this observation. Time 1 and Time 3 have a performance advantage on side of the adaptive renderer, while the rendering of the image for Time 2 requires additional rays and traversal steps compared to the albedo image (749,468 rays). Due to the lack of large regions being in shade, the added shadows only add to the high color variance in the image.

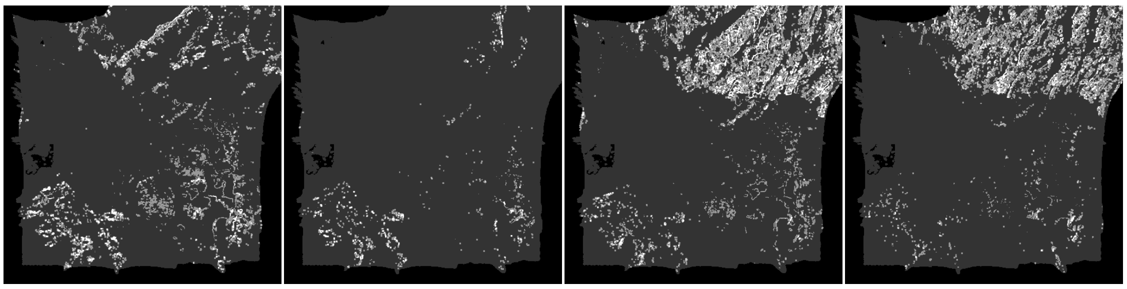
Time of Day	PSNR (A)	PSNR (EA)	RMSE (A)	RMSE (EA)	SSIM (A)	SSIM (EA)
Time 0	42.548	44.436	0.00746	0.00600	0.954	0.970
Time 1	44.869	45.704	0.00571	0.00519	0.972	0.977
Time 2	38.647	40.597	0.01168	0.00934	0.899	0.935
Time 3	42.507	45.404	0.00749	0.00537	0.956	0.977

Table 5.24: Mountain dataset image similarities to reference image for different times of day - Adaptive to reference (A) and error-aware to reference (EA)

In terms of quality, the gap between the two renderers is once again quite small depending on the time of day. For Time 1, one could argue that the adaptive renderer even matches the error-aware one, as the difference in quality is very small while the adaptive renderer manages to render the image at lower cost.

5.3.4 Biomes Dataset


(a) Error-aware rendering results


(b) Adaptive rendering ray counts

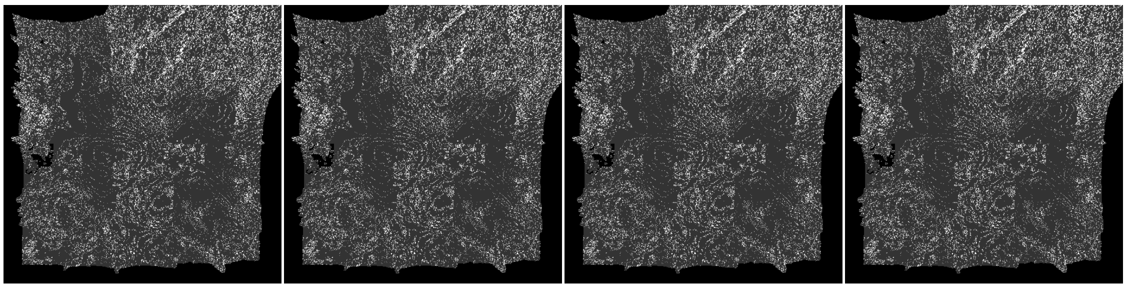

(c) Error-aware rendering maximum LOD levels

Figure 5.10: Biomes dataset results for different times of day

The adaptive renderings of the biomes dataset are profiting the most from the lighting conditions. The forest at the top left, which was difficult to render before, is now often in shade (as seen in 5.10(a)). In addition to this, the noisy, partially snow-covered terrain at the center of the area is also often dark enough to not be above the variance thresholds.

5.10(b) visualized the vast regions of the area that do not have to be refined by the renderer. In the second image, almost the entire mountain is even in the shadow of the surrounding terrain.

Time of Day	Ray Count (A)	Ray Count (EA)	Traversal Steps (A)	Traversal Steps (EA)
Time 0	391,904	456,034	3,213,830	3,992,801
Time 1	299,252	456,034	2,449,060	3,992,801
Time 2	500,552	456,034	4,139,100	3,992,801
Time 3	447,968	456,034	3,676,840	3,992,801

Table 5.25: Biomes dataset ray counts and traversal step counts for different times of day

This results in the best performance numbers for the adaptive renderer out of all datasets. The ray count from the albedo image (616,187) is drastically reduced here and the traversal counts manage to be below the ones for the error-aware renderer in all cases but one.

This highlights the need for lighting information to be included in the error calculations of the error-aware approach. As is clearly visible in 5.10(c), especially in the second image, the renderer wastes a lot of performance on areas that are barely visible in the final image and thus do not affect image quality as much.

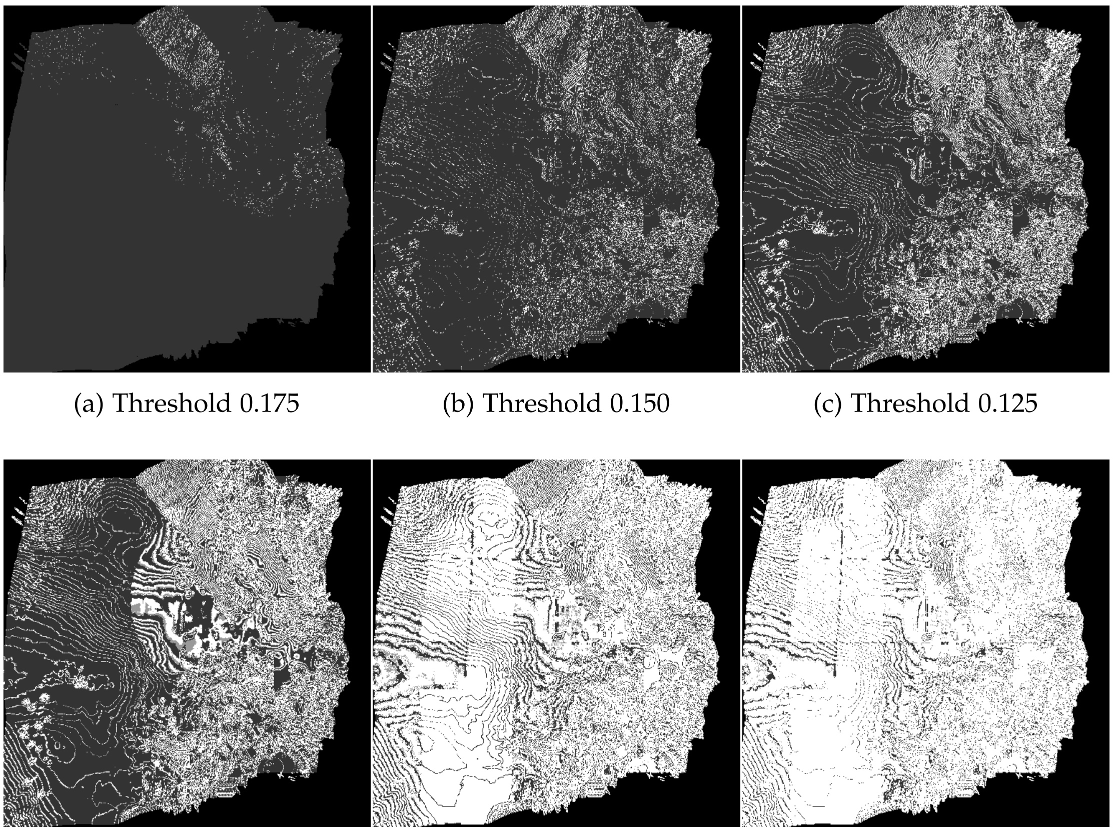
Time of Day	PSNR (A)	PSNR (EA)	RMSE (A)	RMSE (EA)	SSIM (A)	SSIM (EA)
Time 0	43.624	46.472	0.00659	0.00475	0.959	0.978
Time 1	49.561	51.263	0.00333	0.00273	0.988	0.992
Time 2	40.966	43.755	0.00895	0.00649	0.935	0.964
Time 3	41.757	45.336	0.00817	0.00541	0.944	0.974

Table 5.26: Biomes dataset image similarities to reference image for different times of day - Adaptive to reference (A) and error-aware to reference (EA)

Even though the performance metrics indicate that the error-aware approach still produces higher quality images, the cost difference is large enough to indicate a clear advantage for the adaptive renderer. For Time 1, for example, it would be possible to run multiple iterations of the renderer instead of one and still end up with a lower cost.

5.4 Error Threshold

This section shows are more detailed breakdown of the effect the error threshold has on rendering performance and quality. The desert dataset is used for testing and the images are being rendered at a resolution of 512x512.



(a) Threshold 0.175 (b) Threshold 0.150 (c) Threshold 0.125

(d) Threshold 0.100 (e) Threshold 0.075 (f) Threshold 0.050

Figure 5.11: Desert dataset ray counts and traversal step counts for different error thresholds

A short glimpse at 5.11 shows the effects of the error threshold on performance. It has to be set on a data by data basis, together with the decay factor.

If a target performance is given (e.g. the ray count of the adaptive rendering of the desert dataset as is used in this thesis), those values can be determined by solving a simple optimization problem (see 5.1). Alternatively, a visual quality target could be given instead. The goal of the optimization problem would then not be to find the highest visual quality possible, but rather to find the lowest performance required to reach that target.

Error Threshold	Ray Count (A)	Traversal Steps (A)	Ray Count (EA)	Traversal Steps (EA)
0.175	395,162	3,557,270	180,622	1,952,953
0.15	395,162	3,557,270	418,090	3,756,873
0.125	395,162	3,557,270	652,518	6,172,597
0.1	395,162	3,557,270	1,083,927	9,708,443
0.075	395,162	3,557,270	2,744,319	23,598,582
0.05	395,162	3,557,270	3,208,818	27,444,396

Table 5.27: Desert dataset ray counts and traversal step counts for different error thresholds

A comparison of performance between both renderer across a variety of error thresholds further underlines the need for a good selection of the value for it. For comparisons between the two renderers, it would be possible to select the error threshold first and set the variance thresholds for the adaptive renderer based on it instead of the way it is done here.

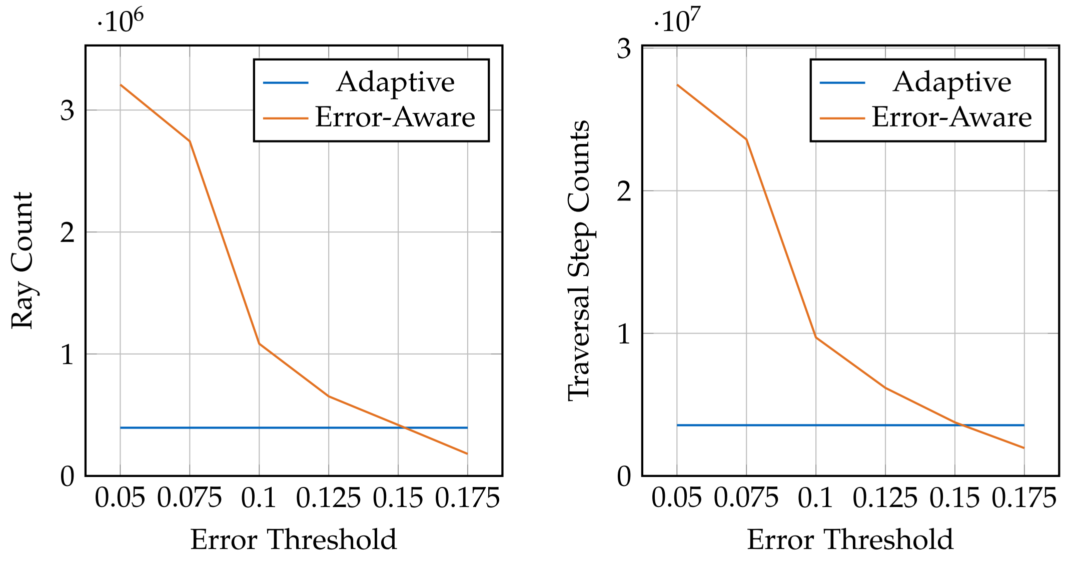


Figure 5.12: Desert dataset ray counts and traversal step counts for different error thresholds

Plotting the ray count and traversal step count over the error threshold shows that the performance scaling is proportional to 1/*error threshold*. This makes it difficult to set a value for it manually while still being able to accurately determine the performance impact. Setting the value by solving an optimization problem (as described above) might be the preferred way.

Error Threshold	PSNR (A)	PSNR (EA)	RMSE (A)	RMSE (EA)	SSIM (A)	SSIM (EA)
0.175	50.913	52.710	0.00285	0.00231	0.991	0.993
0.15	50.913	54.081	0.00285	0.00197	0.991	0.995
0.125	50.913	55.443	0.00285	0.00169	0.991	0.997
0.1	50.913	56.513	0.00285	0.00149	0.991	0.997
0.075	50.913	59.718	0.00285	0.00103	0.991	0.999
0.05	50.913	61.634	0.00285	0.00083	0.991	0.999

Table 5.28: Desert dataset image similarity metrics for different error thresholds

The important aspect of the threshold is that the quality of the image scales as well. This allows for quality-speed trade-offs, an important aspect of the error-aware algorithm introduced in this thesis. Traditionally, one could either use a level of detail system for rendering or use an adaptive renderer. The latter allows for adjustments of quality at the cost of rendering performance by changing the variance thresholds or the iteration count.

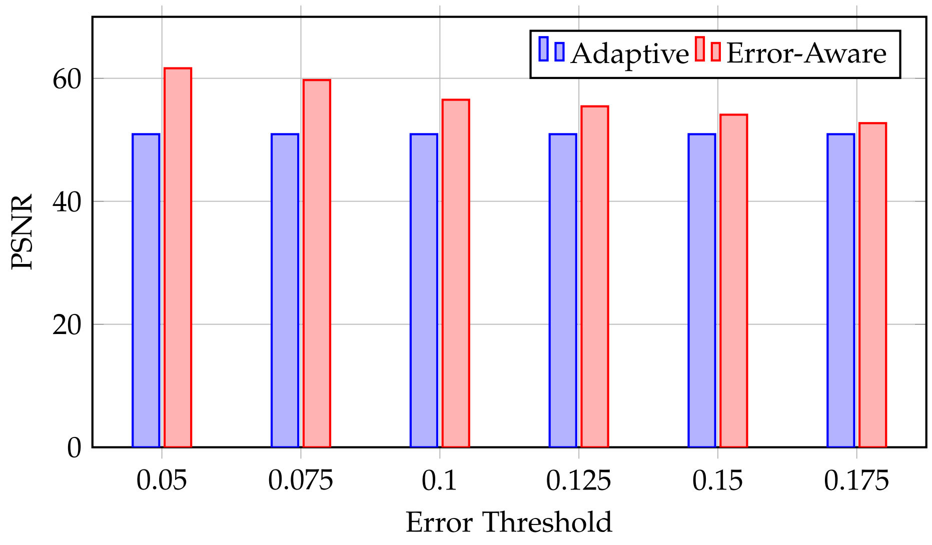


Figure 5.13: Desert dataset peak signal-to-noise ratio (PSNR) across different error thresholds

When plotting the PSNRs over different error thresholds, the scaling is clearly visible. Due to the non-linearity of the PSNR, it is only possible to judge quality and not quantity.

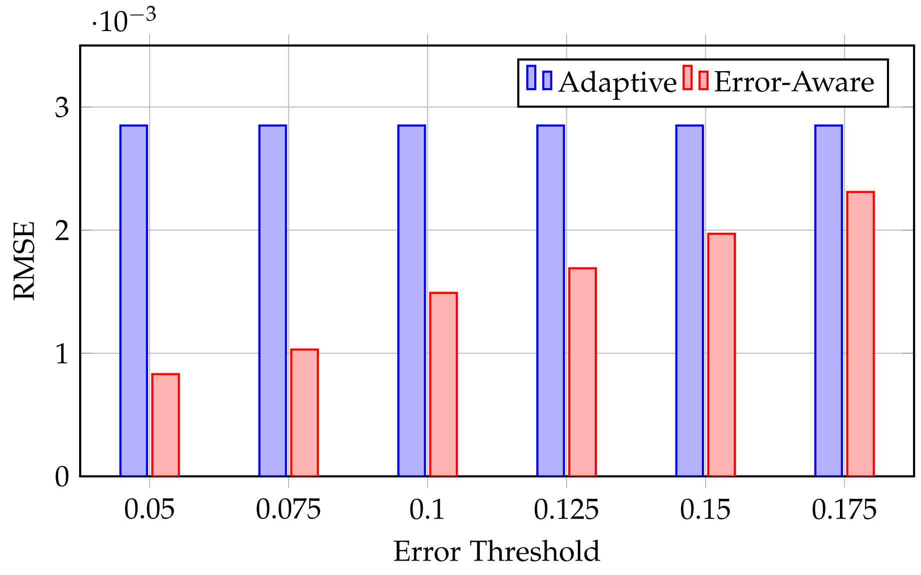


Figure 5.14: Desert dataset root-mean-square error (RMSE) across different error thresholds

From the RMSE, it becomes clear that the image quality of the error-aware renderer covers the entire scale between a rendering done using only the LOD system and the reference rendering. It also shows that while performance scales exponentially, image quality when measured using RMSE does not.

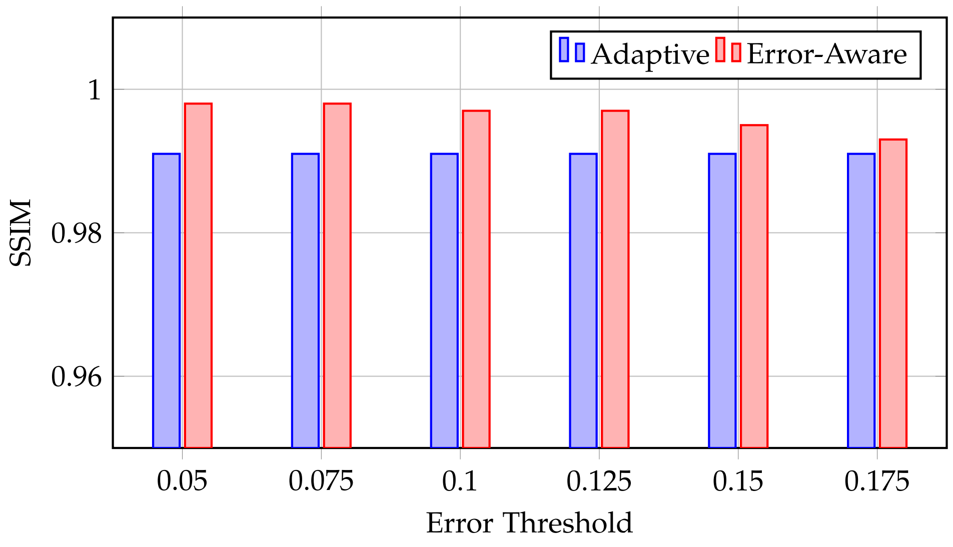
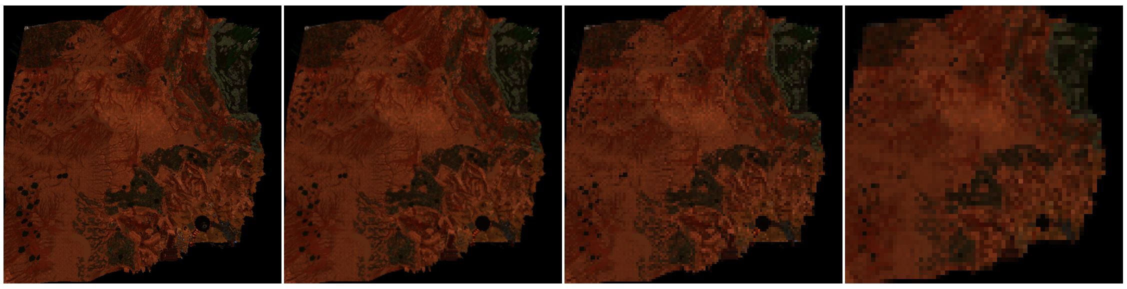
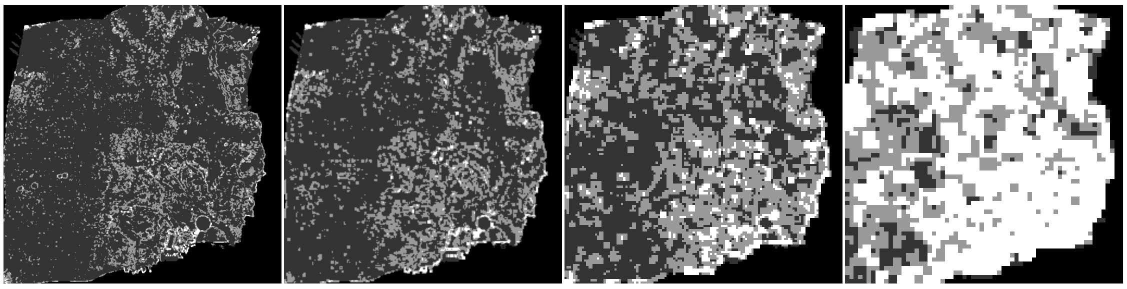


Figure 5.15: Desert dataset structural similarity index measure (SSIM) across different error thresholds

5.5 Resolution


(a) Error-aware rendering results


(b) Adaptive rendering ray counts

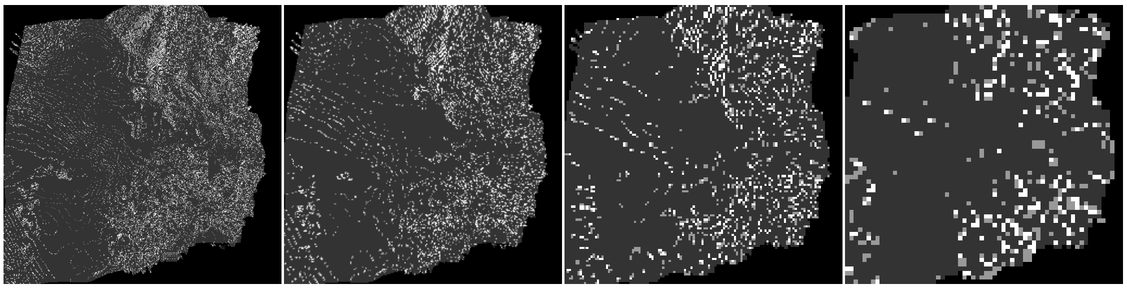

(c) Error-aware rendering maximum LOD levels

Figure 5.16: Desert dataset rendering at different resolutions

5.16(b) shows a shortcoming of the adaptive renderer. Due to the kernel size being at least 3x3 pixels when calculating the variance, this kernel covers more area of the data for smaller images. In this production dataset, this often means a change in vegetation, a change in biomes or structures being part of the kernel. The average variance therefore tends to be higher for smaller images.

The error-aware renderer in 5.16(c) does not have this issue, as its output is not related to voxel size or area coverage. Visually it appears to be scaling perfectly across different image resolutions.

Resolution	Ray Count (A)	Traversal Steps (A)	Ray Count (EA)	Traversal Steps (EA)
512x512	395,162	3,557,270	394,242	3,582,041
256x256	135,032	1,076,390	93,899	799,302
128x128	58,016	481,370	23,894	202,114
64x64	6,452	52,310	6,155	51,834

Table 5.29: Desert dataset ray counts and traversal step counts for different render resolutions

The ray counts support this theory, as it scales almost perfectly with resolution. A doubling of the resolution (quadrupling of the pixel count) leads to the ray count being 4 times as large. The same holds for the traversal step count.

The adaptive renderer does not scale as reliably, as it seems to struggle with ray counts for resolutions of 128^2 and 256^2. This is probably due to the frequency of image variance in relation to the resolution.

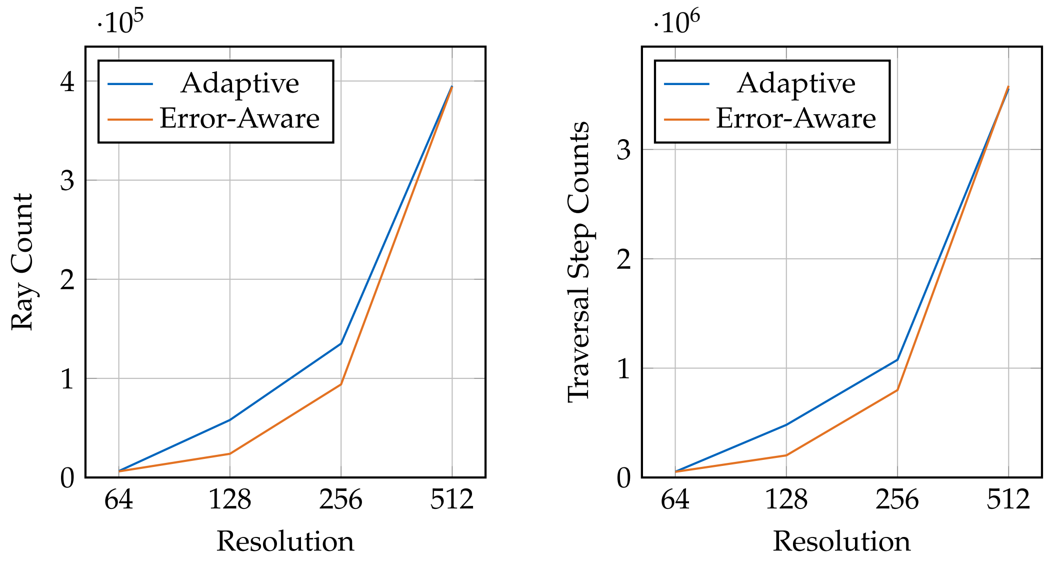


Figure 5.17: Desert dataset ray counts and traversal step counts for different render resolutions

Since light probe baking is usually done at small resolutions like 64x64, the values here are of special interest. Performance-wise, both renderers are comparable here as well, as the traversal step counts lie within 0.92%.

Resolution	PSNR (A)	PSNR (EA)	RMSE (A)	RMSE (EA)	SSIM (A)	SSIM (EA)
512x512	50.913	53.937	0.00285	0.00201	0.991	0.995
256x256	50.436	52.700	0.00301	0.00232	0.989	0.993
128x128	50.433	51.553	0.00301	0.00264	0.989	0.991
64x64	48.638	50.578	0.00370	0.00296	0.983	0.988

Table 5.30: Desert dataset image similarity metrics for different render resolutions

The similarity metrics show the same scaling as the performance metrics. While the adaptive renderer can maintain a high quality for 256^2 and 128^2 due to the large amounts of additional rays, it still produces results at lower quality than the error-aware renderer.

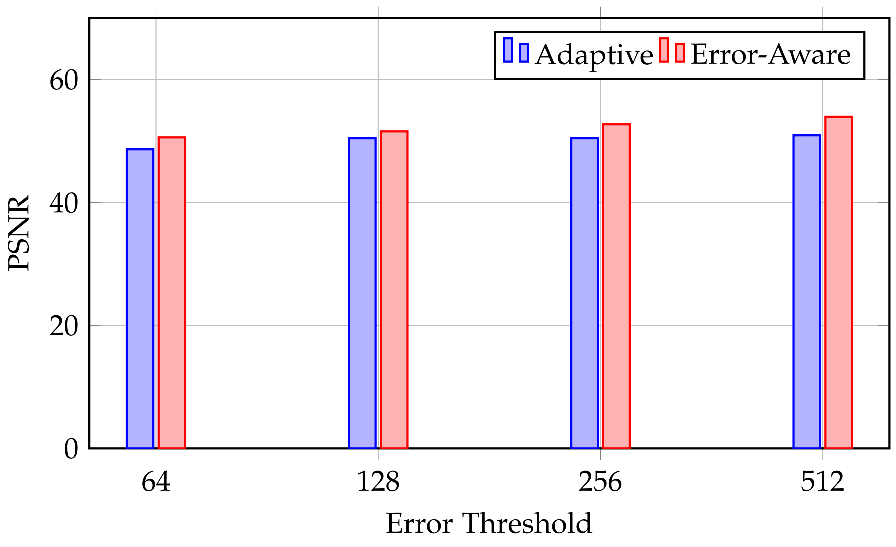


Figure 5.18: Desert dataset peak signal-to-noise ratio (PSNR) for different render resolutions

Overall, the performance of both rendering approaches declines with smaller resolutions. This is the expected behaviour, as the frequency of noise relative to the resolution of the image increases.

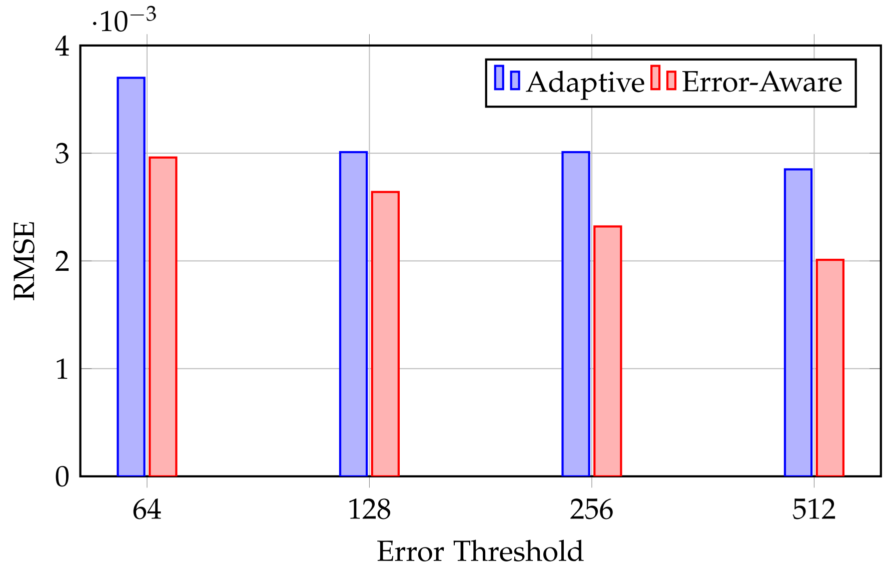


Figure 5.19: Desert dataset root-mean-square error (RMSE) for different render resolutions

The RMSEs show almost perfect linear scaling of the error-aware images across resolutions. This is not the case for the adaptive ones. In order to scale more reliably, changes to the variance threshold or iteration count would be needed.

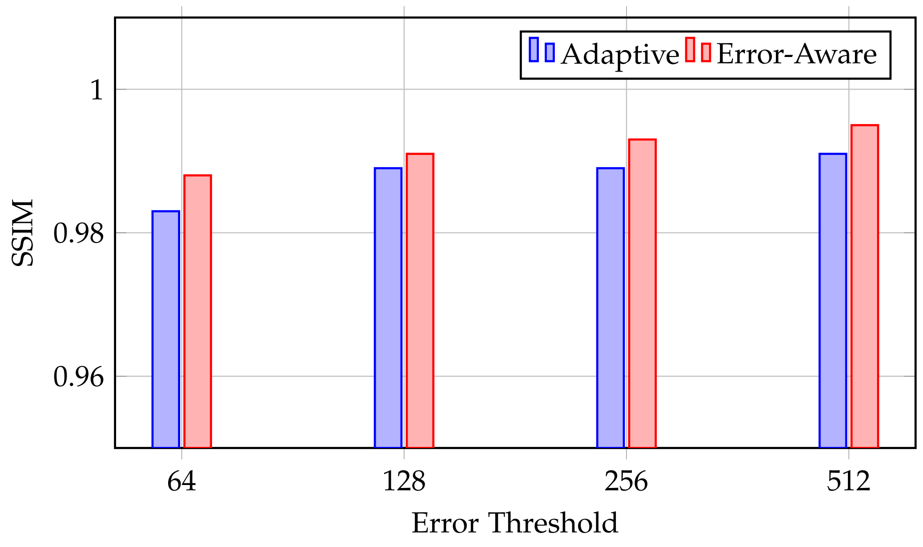


Figure 5.20: Desert dataset structural similarity index measure (SSIM) for different render resolutions

The SSIMs mirror this behaviour. For a resolution of 128^2 the adaptive renderer almost produces a qualitatively comparable result, given the much higher ray count it is however expected to do this.

The error-aware algorithm is therefore a great fit for rendering algorithms that are supposed to produce images with vastly different resolutions.

6 Future Work

Since the algorithm itself is not extremely specialized, there is a lot of room for future work. This includes the algorithm itself, the error value pre-computation and the integration into runtime systems.

6.1 Error Metrics

As the results clearly show, the algorithm needs to be extended in order to support additional rendering scenarios like e.g. the lighting of the different times of day. Even though it still mostly manages to outperform the adaptive renderer, there is a lot of performance wasted on pixels with low impact on the image quality.

The most difficult part about this is to find metrics that can be used in the offline error calculations. For visibility, there are only 256 possible combinations for the layout of the children. This is still an easily manageable number, calculating those values doesn't take an exorbitant amount of time. If color values would be used in addition to that (8 bits per RGB channel) there are an additional $\tilde{1}6.8$ million possibilities per child node. Obviously, this is not an option as all possible combinations would have to be computed.

Instead, one could think about using the luminance metric that is used in the adaptive renderer instead. By calculating the luminance for the child colors during runtime, a simple boolean could be used to mark a child voxel as containing high luminance. When calculating the errors, this would only add this individual boolean as a possibility to each child which would keep the number of possible combinations manageable.

It might be possible to go beyond that by using the symmetry of an octree node and delete duplicates within the calculations (e.g. there is no need to calculate errors for all 8 possible configurations with a single child, using just one in combination with rotations and mirror operations is enough).

6.2 Optimization

While the current spherical harmonics setup to store error values works well in the tests, it might not be the optimal solution. On one hand it is fairly cheap to calculate a value for an arbitrary vector, on the other hand this comes at some additional memory cost. Calculating one value for each direction instead of the 9 coefficients and using the axis-alignment of the grid to determine which value should be used might be a more effective solution. It might also be possible to reuse the single ray casted at the parent more effectively and not having to add 4, but only 3 additional rays if the error is above the threshold. This would massively decrease the traversal step count and improve performance quite a bit.

7 Conclusion

The goal of this thesis was to improve well know rendering algorithms for voxel raytracing by re-evaluating if the approaches used by those are as good as they can be or whether another solution would achieve better results.

The error-aware adaptive raytracing algorithm introduced in this paper achieves higher performance in almost every test-case while at the same time reaching higher quality levels. This was proven using an actual production dataset rather than purpose-build data. While there were some shortcomings in terms of handling lighting those are attributable to the way the algorithm handles its error calculations. Some small adjustments should help to easily outperform the adaptive rendering approach in those areas as well.

Given the increasing popularity of voxel raytracing, as well as bounding volume hierarchies for real-time raytracing in computer graphics, it is important to keep pushing this technology and to keep improving it.

Instead of simply improving established rendering approaches, they were fundamentally challenged by re-evaluating the problem they were trying to solve in the first place. However, there is still plenty of work possible in this area and this first version of the error-aware algorithm does most likely not reach its actual potential.

Looking back at the goals for the algorithm as described in 1, the error-aware adaptive raytracing achieves all of them.

List of Figures

3.1 Octree node example 5
3.2 Sample error values 7
3.3 Sample jittering from [Ken13] 9
3.4 Test scene setup 10
3.5 Test scene shadow ray quality comparison 10
3.6 Test scene level of detail visualization 11
3.7 Slackline dataset example 12
3.8 Slackline error-aware rendering test results 12

4.1 Lighting scenarios in dataset 14
4.2 Desert area overview 15
4.3 Forest area overview 16
4.4 Mountain area overview 17
4.5 Biomes area overview 18

5.1 Adaptive rendering algorithm and results 21
5.2 Error-aware rendering algorithm and results 22
5.3 Desert dataset results 25
5.4 Forest dataset results 27
5.5 Mountain dataset results 29
5.6 Biomes dataset results 31
5.7 Desert dataset results for different times of day 33
5.8 Forest dataset results for different times of day 36
5.9 Mountain dataset results for different times of day 38
5.10 Biomes dataset results for different times of day 40
5.11 Desert dataset ray counts and traversal step counts for different error thresholds 42
5.12 Desert dataset ray counts and traversal step counts for different error thresholds 44
5.13 Desert dataset peak signal-to-noise ratio (PSNR) across different error thresholds 45
5.14 Desert dataset root-mean-square error (RMSE) across different error thresholds 46

5.15 Desert dataset structural similarity index measure (SSIM) across different error thresholds 47
5.16 Desert dataset rendering at different resolutions 48
5.17 Desert dataset ray counts and traversal step counts for different render resolutions 49
5.18 Desert dataset peak signal-to-noise ratio (PSNR) for different render resolutions 50
5.19 Desert dataset root-mean-square error (RMSE) for different render resolutions 51
5.20 Desert dataset structural similarity index measure (SSIM) for different render resolutions 52

List of Tables

3.1 Error introduced through raycasting approximation 7

4.1 Voxel and octree node counts in dataset 15

5.1 Desert dataset ray counts for different decay factors and error thresholds 23
5.2 Desert dataset ray counts and traversal step counts 25
5.3 Desert dataset adaptive rendering pixel counts 26
5.4 Desert dataset error-aware rendering maximum LOD level used 26
5.5 Desert dataset image similarity to reference image 26
5.6 Forest dataset ray counts and traversal step counts 28
5.7 Forest dataset adaptive rendering pixel counts 28
5.8 Forest dataset error-aware rendering maximum LOD level used 28
5.9 Forest dataset image similarity to reference image 28
5.10 Mountain dataset ray counts and traversal step counts 29
5.11 Mountain dataset adaptive rendering pixel count used 30
5.12 Mountain dataset error-aware rendering maximum LOD levels 30
5.13 Mountain dataset image similarity to reference image 30
5.14 Biomes dataset ray counts and traversal step counts 31
5.15 Biomes dataset adaptive rendering pixel counts 32
5.16 Biomes dataset error-aware rendering maximum LOD level used 32
5.17 Mountain dataset image similarity to reference image 32
5.18 Desert dataset ray counts and traversal step counts for different times of day 34
5.19 Desert dataset average image luminance for different times of day . . . 34
5.20 Desert dataset image similarities to reference image for different times of day - Adaptive to reference (A) and error-aware to reference (EA) 35
5.21 Forest dataset ray counts and traversal step counts for different times of day 37
5.22 Forest dataset image similarities to reference image for different times of day - Adaptive to reference (A) and error-aware to reference (EA) 37
5.23 Mountain dataset ray counts and traversal step counts for different times of day 39

5.24 Mountain dataset image similarities to reference image for different times of day - Adaptive to reference (A) and error-aware to reference (EA) . . 39
5.25 Biomes dataset ray counts and traversal step counts for different times of day 41
5.26 Biomes dataset image similarities to reference image for different times of day - Adaptive to reference (A) and error-aware to reference (EA) . . 41
5.27 Desert dataset ray counts and traversal step counts for different error thresholds 43
5.28 Desert dataset image similarity metrics for different error thresholds . . 44
5.29 Desert dataset ray counts and traversal step counts for different render resolutions 49
5.30 Desert dataset image similarity metrics for different render resolutions 50

Bibliography

[AGL91] M. Agate, R. L. Grimsdale, and P. F. Lister. "The HERO algorithm for ray-tracing octrees." In: *Advances in Computer Graphics Hardware IV*. Springer, 1991, pp. 61–73.

[Cra+11] C. Crassin, F. Neyret, M. Sainz, S. Green, and E. Eisemann. "Interactive indirect illumination using voxel cone tracing." In: *Computer Graphics Forum*. Vol. 30. 7. Wiley Online Library. 2011, pp. 1921–1930.

[HB12] M. Hassan and C. Bhagvati. "Structural similarity measure for color images." In: *International Journal of Computer Applications* 43.14 (2012), pp. 7–12.

[HZ13] A. Horé and D. Ziou. "Is there a relationship between peak-signal-to-noise ratio and structural similarity index measure?" In: *IET Image Processing* 7.1 (2013), pp. 12–24.

[JW88] D. Jevans and B. Wyvill. "Adaptive voxel subdivision for ray tracing." In: (1988).

[Ken13] A. Kensler. "Correlated multi-jittered sampling." In: *Pixar Technical Memo* (2013).

[KY12] J. Korhonen and J. You. "Peak signal-to-noise ratio revisited: Is simple beautiful?" In: *2012 Fourth International Workshop on Quality of Multimedia Experience*. 2012, pp. 37–38. DOI: `10.1109/QoMEX.2012.6263880`.

[LK10] S. Laine and T. Karras. "Efficient sparse voxel octrees." In: *IEEE Transactions on Visualization and Computer Graphics* 17.8 (2010), pp. 1048–1059.

[Mül+20] M. U. Müller, N. Ekhtiari, R. M. Almeida, and C. Rieke. "SUPER-RESOLUTION OF MULTISPECTRAL SATELLITE IMAGES USING CONVOLUTIONAL NEURAL NETWORKS." In: *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci.* V-1-2020 (2020), pp. 33–40. DOI: `https://doi.org/10.5194/isprs-annals-V-1-2020-33-2020`.

[RUL00] J. Revelles, C. Urena, and M. Lastra. "An efficient parametric algorithm for octree traversal." In: (2000).

[Vel10] T. Veldhuizen. "Measures of image quality." In: *CVonline: The Evolving, Distributed, Non-Proprietary, On-Line Compendium of Computer Vision* (2010).

[Wid+15] S. Widmer, D. Pająk, A. Schulz, K. Pulli, J. Kautz, M. Goesele, and D. Luebke. "An adaptive acceleration structure for screen-space ray tracing." In: *Proceedings of the 7th Conference on High-Performance Graphics*. 2015, pp. 67–76.