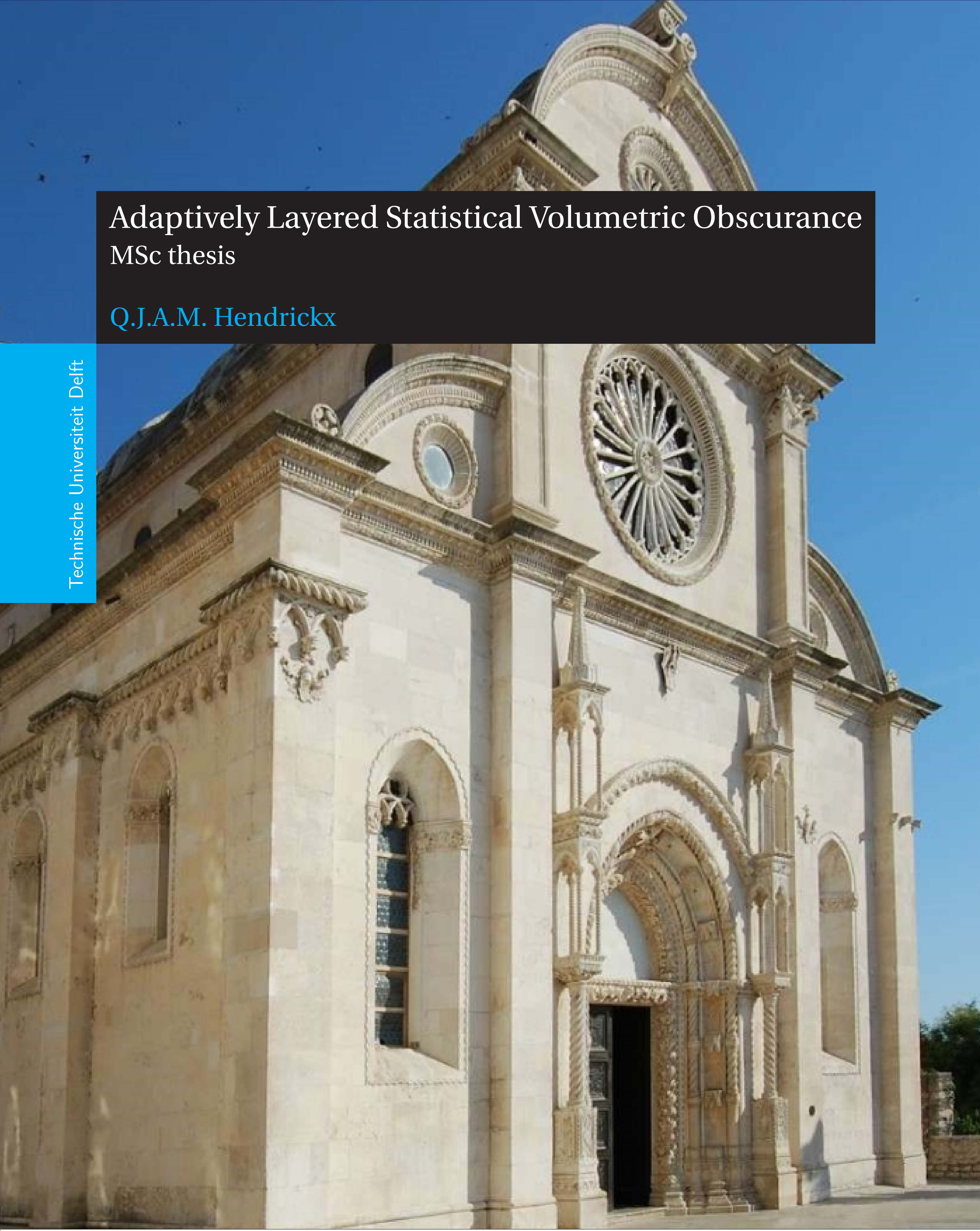


Adaptively Layered Statistical Volumetric Obscurance

MSc thesis

Q.J.A.M. Hendrickx

Technische Universiteit Delft

TU Delft Delft University of Technology

ADAPTIVELY LAYERED STATISTICAL VOLUMETRIC OBSCURANCE

MSc THESIS

by

Q.J.A.M. Hendrickx

in partial fulfillment of the requirements for the degree of

Master of Science
in Computer Science

at the Delft University of Technology,
to be defended publicly on Thursday February 5, 2015 at 2:00 PM.

Supervisor:	Prof. dr. Elmar Eisemann	
Thesis committee:	Prof. dr. Elmar Eisemann	TU Delft, Computer Graphics and Visualization
	Dr. ir. Rafael Bidarra	TU Delft, Game Technology
	Dr. ir. Willem-Paul Brinkman	TU Delft, Interactive Intelligence
	Dr.-Ing. Martin Eisemann	TU Delft, Computer Graphics and Visualization

TU Delft Delft University of Technology

ABSTRACT

We accelerate volumetric obscurance, a variant of ambient occlusion, and solve undersampling artifacts, such as banding, noise or blurring, that screen-space techniques traditionally suffer from. We make use of an efficient statistical model to evaluate the occlusion factor in screen-space using a single sample. Overestimations and halos are reduced by an adaptive layering of the visible geometry. Bias at tilted surfaces is avoided by projecting and evaluating the volumetric obscurance in tangent space of each surface point. We compare our approach to several traditional screen-space volumetric obscurance techniques and show its competitive qualitative and quantitative performance. Our algorithm maps well to graphics hardware, does not require the traditional bilateral blur step of previous approaches, and avoids typical screen-space related artifacts such as temporal instability due to undersampling.

Preface

This research is part of the proof competence for obtaining the Master of Science degree in Computer Science at the Delft University of Technology.

My interest for the process of creating good software had initially led me to chose the Software Technology (ST) track over the Media and Knowledge Engineering (MKE) track. This caused me to miss out on many usefull courses relating to the field of Computer Graphics. Fortunately, I was given the opportunity to make up for this loss by working on several projects at the Computer Graphics & Visualizations group during my bachelor. In my second year at the TU Delft I was given the opportunity to become acquainted with the scientific process during my work on rendering real-time river networks. For my final bachelor project I worked in a group of four students to develop a rendering engine for the SketchaWorld program. This project first introduced me to the concept of Ambient Occlusion (AO). In particular, I implemented some Screen-space Ambient Occlusion (SSAO) techniques that were the predecessors of many recent SSAO approaches, including the one presented in this thesis.

From a young age I have always been interested in the field of Computer Graphics. Receiving direct visual feedback while expressing your ideas through written code is a beautiful ability that I believe few other fields in Computer Science can match. The never ending search for smarter techniques that require less computational power to achieve better looking images has always provided me with a challenging environment during my work on this MSc thesis.

Q.J.A.M. Hendrickx
Delft, February 2015

Contents

1

Introduction

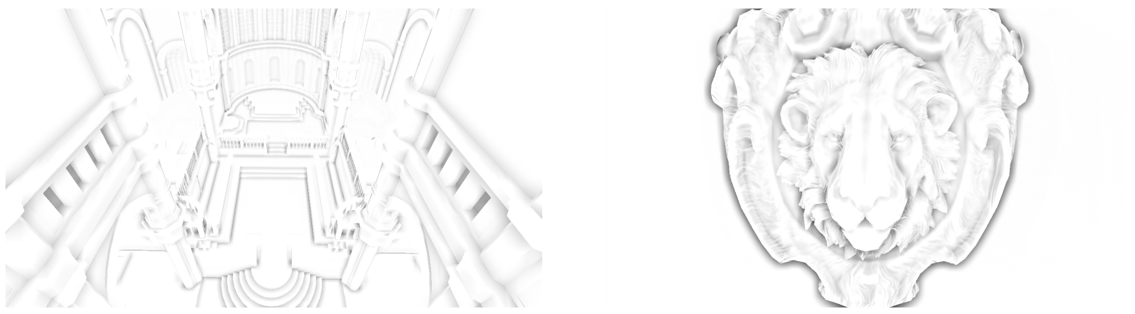


Figure 1.1: Ambient Occlusion without shading. Images rendered with the method proposed in this report (SVO). We can render at a resolution of 1280x720 pixels with 320 fps (294 MPixels/s) on an NVIDIA GTX 770 graphics card.

Efficient computation of global illumination is still one of the hardest problems in computer graphics. In consequence, real-time approximations often make very simplifying assumptions. *Ambient Occlusion* (AO) is an example and focuses on the evaluation of ambient light reaching a point on a surface [1] by considering only local geometry as occluders in the scene. Attenuating the ambient light term based on local occlusion generates realistically looking shadows and creates important contact cues improving overall depth perception (see Figure 1.1).

Historically, AO was first applied in static scenes where its effect could be baked into occlusion maps [1]. However, because AO is a global effect, this approach does not work well for dynamic scenes and would need to be recomputed per frame. In recent years, advances in graphics hardware and the development of screen-space approximations have led to real-time implementations of AO [2, 3]. These screen-space ambient occlusion (SSAO) techniques compute the amount of occlusion as a postprocessing pass based on a depth image from the camera's point of view. Traditionally, the occlusion factor is approximately estimated per pixel using a few samples and smoothed using a subsequent bilateral blur step. Most current rendering engines incorporate such solutions.

1.1. Integration in Graphics Drivers

It's worth noting that in 2009 NVIDIA started directly supporting (Horizon-Based) Ambient Occlusion (HBAO) in theirs drivers [4], in 2013 this support was updated with a higher quality implementation (HBAO+). The specifics of these particular approaches are discussed in more detail in Section 2.4. While most modern games have implemented native support for AO themselves, NVIDIA allows users to enable AO on a driver level for those that do not. This step from one of the leading manufactures of graphics hardware shows the importance of AO in the computer graphics industry today.

1.2. Example of AO in Lighting Models

To demonstrate how Ambient Occlusion effects the rendering of a scene we show how the appearance of a simple 3D rendered scene changes when applying different lighting models (see Figure 1.2). The scene consists of a statue standing on a plane, all geometry has a white diffuse material applied.

The shown light models in Figure 5.2 make a distinction between ambient light (i.e. environment light, without any particular direction, Figure 1.2a and 1.2d) and directional light (i.e. light coming from the direction of a particular light source, Figure 1.2b and 1.2e). Typically, both light sources are used and combined based on an artist defined distribution. In this case we have used 75% ambient light and 25% directional light for our combined images shown in Figures 1.2c and 1.2f.

By comparing a constant ambient light source as shown in Figure 1.2a with an ambient occlusion light source as shown in Figure 1.2d, it becomes clear that AO can increase our depth perception of a 3D scene significantly. This improvement in realism and depth perception also translates into the combined image. For example, in 1.2f we can see how the creases in the gown are much more distinctive than in 1.2c.

1.3. Problem Statement

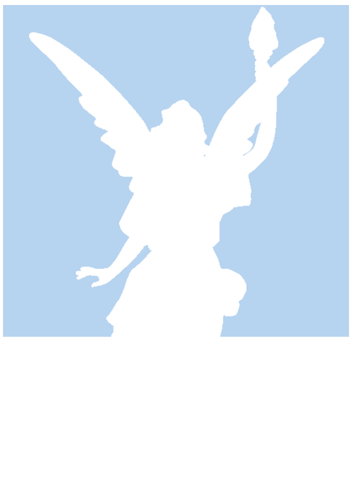
We aim at developing an approach that has reduced complexity of screen-space ambient occlusion (SSAO) approaches, but avoids the usual drawbacks, such as banding, noise or blurriness caused by undersampling. In order to eliminate these artifacts, we have to account for *all* of the local geometry visible in screen-space. To this extent, we reverse the typical order of operations applied in existing SSAO approaches. Instead of taking samples from the AO function and blurring the result afterwards, we compute a statistical model of the surrounding geometry at a pixel's world position and use this directly for AO computation. Because we do not use traditional sampling there is no need for randomization or blurring of the result [2].

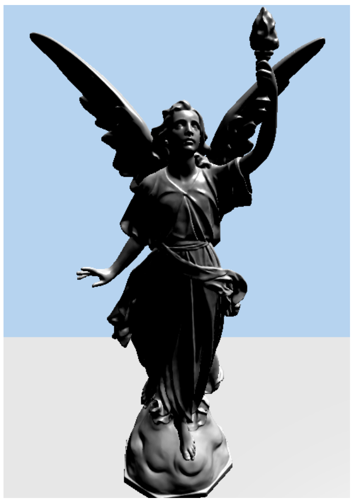
1.4. Contribution

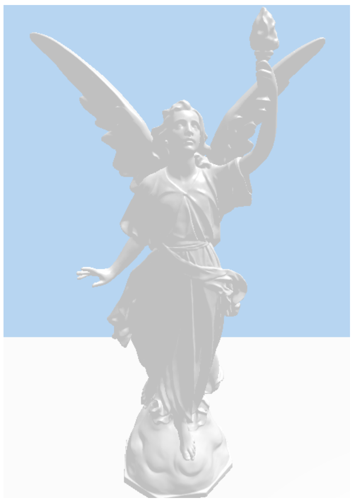
In particular, our contributions are as follows:

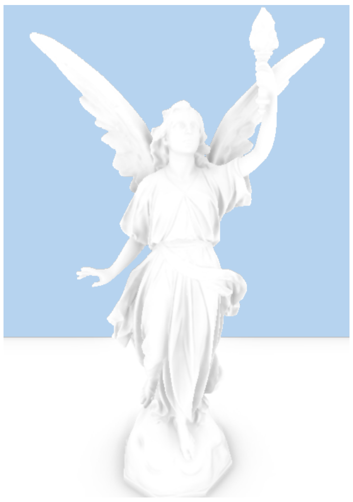
- A screen-space ambient-occlusion model, which can be evaluated using a single sample;
- An adaptive depth-slicing technique to efficiently compute this model;
- A GPU-friendly and highly-parallel implementation.

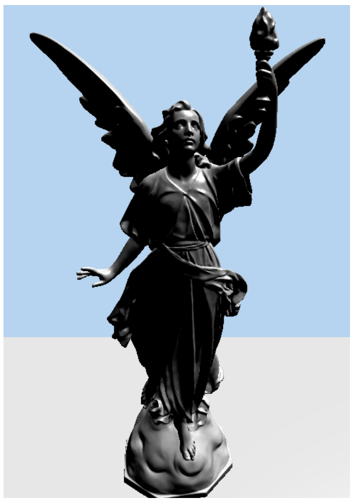
We start by discussing previous work (Section 2) and providing some necessary background information for readers less familiar with the field of computer graphics (Section 3). After this, we describe our algorithm in gradual steps. First, we introduce an approximation for volumetric obscurance, a variant of ambient occlusion, and how to compute it efficiently (Section 4). We will then describe how to improve quality by using depth layering (Section 4.3). For acceleration, we introduce an adaptive depth-slicing technique (Section 4.4) and remove bias in the result by incorporating the surface normal into the computation (Section 4.5). We introduce important optimizations for efficient implementation, like approximate summed-area tables (SAT) and differential SAT computation (Section 4.6). We evaluate and compare our approach to common screen-space ambient occlusion and volumetric-obscurance techniques (Section 5), before concluding (Section 6).


(a) ambient light


(b) directional light


(c) ambient + directional light


(d) ambient occlusion light


(e) directional light

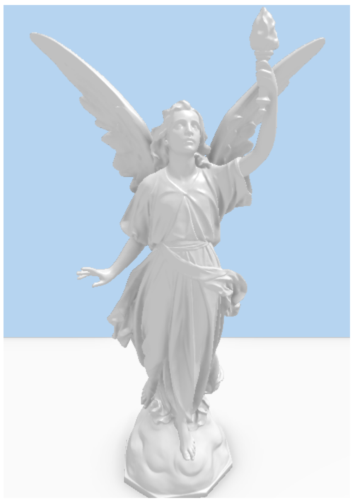
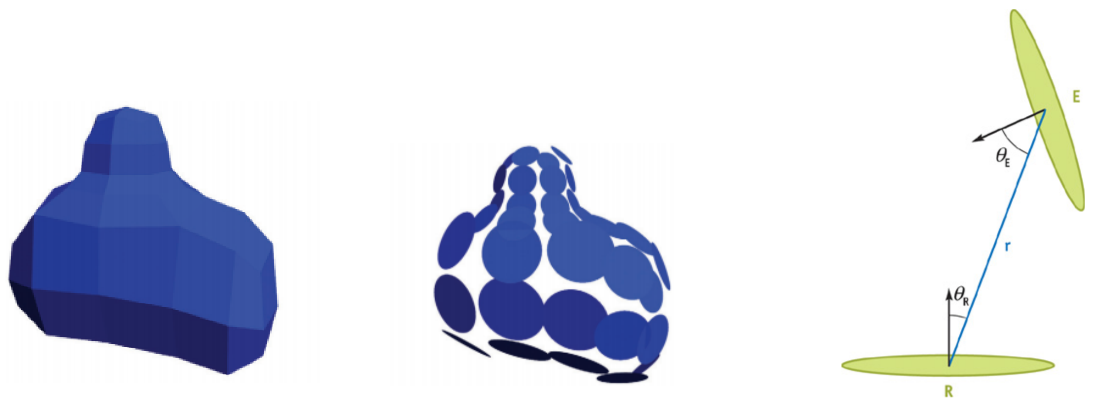

(f) ambient occlusion + directional light

Figure 1.2: The same scene shaded with different light models. In the upper row we see a traditional lighting setup with an ambient term (a) and a directional light (b), by combining both components we get the result shown in (c). A setup that incorporates the effect of ambient occlusion is shown in the bottom row. In (d) we see the raw ambient occlusion output that is combined with a directional light (e) to achieve the output shown in (f). Note how depth perception is noticeably improved in (f) as compared to (c).

2 Related Work

The concept of ambient occlusion (AO) was first described by Landis [1], who showed the importance of AO in improving depth perception through contact cues and soft shadows. AO has since gathered a significant amount of interest and numerous new techniques were developed over time. Most work can be divided in roughly two categories, geometry-based ambient occlusion and screen-space ambient occlusion. Where geometry-based AO operates on a three dimensional description of the scene, screen-space AO uses the rasterized depth buffer as input.

Because geometry-based methods incorporate all available geometry into the AO computation they have the potential of generating more accurate results. However, their performance usually depends heavily on the scene's geometrical complexity. In today's highly complex scenes they usually do not scale as well as screen-space methods.



(a) Original geometry (b) Geometry represented using disk elements (c) Shadowing evaluation

Figure 2.1: Geometry-based AO method as described by Bunnel [5]. In (a) we see the original geometry, each vertex is mapped to a proxy disk element as shown in (b). Occlusion shadows are then evaluated between disks as can be see in (c).

2.1. Geometry-based Ambient Occlusion

Bunnel [5] first presents a hardware accelerated geometry-based AO solution in 2005. Geometrical data is represented using surface elements in the form of disks. Each disk corresponds to a vertex in the original mesh, its size depends on the size of connected triangles.

Occlusion for each surface element is calculated by summing the contribution of each other element. A straightforward implementation would have a time complexity of $O(n^2)$ which is not practical for real-time applications. To solve this issue they propose grouping surface elements together in a hierarchy based on their distance. This hierarchy is traversed untill the desired level of detail is achieved, resulting in a much more manageable expected running time of $O(n \log n)$.

Interestingly, Bunnel [5] also shows how his technique can be extended to account for indirect illumination. The disk-to-disk evaluation function can be adapted to account for diffuse lighting with only little extra cost. They further extend the realism of their approach by evaluating light transfer over multiple iterations.

Because occlusion is evaluated per vertex the accuracy of this technique depends on the tesselation level of the geometry. Pixels between vertices are interpolated linearly, which can results in artifacts. Increasing the geometrical density can get rid of the interpolation artifacts but gives rise to disk shaped artifacts instead [6].

Hoberock and Jia [6] improves upon the original method in 2007 by smoothing the result. While this significantly improves the quality of the result, its performance for even moderately complex scenes remains an issue.

An alternative geometry-based approach was presented by McGuire in 2010 [7]. The technique is inspired by Shadow Volumes [8] and evaluates AO through a scattering process. The occlusion caused by each polygon is represented by an occlusion volume (see Figure 2.2). These occlusion volumes are rasterized and their occlusion contribution is weighted into the result.

While this approach achieves high quality results, its performance still depends heavily on the geometrical complexity of the scene. Additionally, it is important to note that if a large AO radius is chosen, the occlusion volumes will grow in size. Larger occlusion volumes will cause an increasing amount of pixels that are rasterized and evaluated for AO, this has a significant impact on performance.

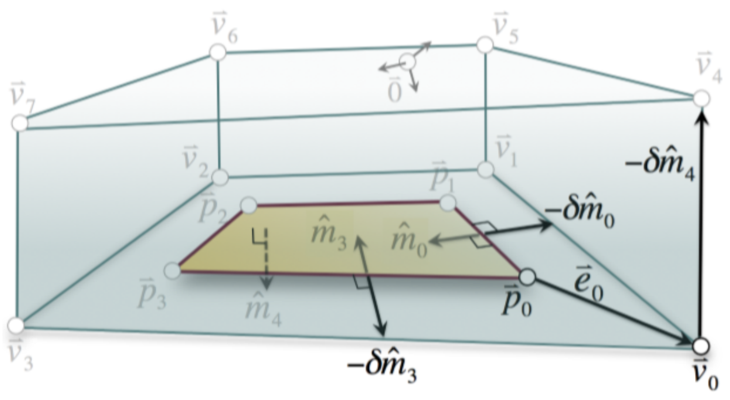


Figure 2.2: In Ambient Occlusion Volumes [7] the occlusion shadows cast by each polygon are represented with an occlusion volume. This volume is oriented with the surface normal and sized according to the occlusion radius.

2.2. SCREEN-SPACE AMBIENT OCCLUSION

Screen-space ambient occlusion (SSAO) techniques compute occlusion based on information in the depth buffer making occlusion evaluations (almost) independent of geometrical scene complexity.

The basic idea for SSAO was introduced by Crytek [2] in 2007. They were aware that the depth buffer could be used to serve as a coarse approximation of the scene. This approximation will always contain less information than the original scene because a depth buffer has a limited precision (usually 24 bits) and cannot store overlapping geometry. However, a depth buffer is almost always readily available without any additional rendering cost, making it a cheap resource to use. By evaluating AO based on this depth buffer it is no longer depended on the geometrical complexity of the scene, instead it will only depend on the chosen image resolution.

The method proposed by Crytek was based on a volumetric obscurance definition of ambient occlusion. The assumption here is that if a volume around a sample point is dense (i.e. it contains a high amount of geometry), it will be more occluded. While this definition doesn't directly relate to any physical process, it often produces similar results.

In particular, they define a sphere around the sample point. The geometrical density inside this sphere is estimated by taking randomized samples wihtin this sphere from the depth buffer (see Figure 4.1b). A simple depth comparison determines if each sample is inside or outside of the geometry. By combining all samples they determine an average density, this density is used as an indicator for occlusion.

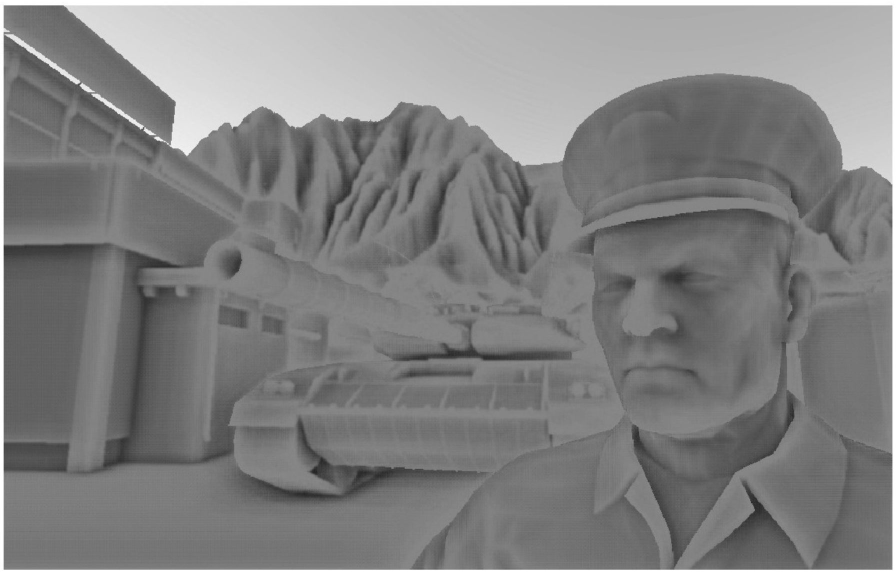


Figure 2.3: SSAO as originally introduced by Cytek. Note how even flat surfaces without any occlusion appear to be 50 % occluded (grey). [2]

As we can see in Figure 2.3 the overal result appears greyish, indicating a severe amount of overocclusion. In reality, one would expect the flat surfaces (e.g. the ground and the walls of the buildings) not to be occluded at all. This result can be traced back to the definition of occlusion as a volumetric integral over a sphere (see Figure 4.1b). As we will see in the following, more advanced methods, occlusion is better represented as a volumetric integral over a normal oriented hemisphere [9] to eliminate this self occlusion.

It is important to note that while the approach introduced by Crytek was relatively basic it still required an aggressive amount of undersampling to achieve real-time performance. In particular, it was not viable to take more than 8 samples per pixel with the GPUs of that time. This restriction led to significant banding artifacts in the result.

By applying a randomization kernel to the sample positions, it is possible to trade the banding artifacts for noise. For example, a 4x4 randomization kernel with 8 samples per pixel would result in 128 unique sample positions within every block of 4x4 pixels.

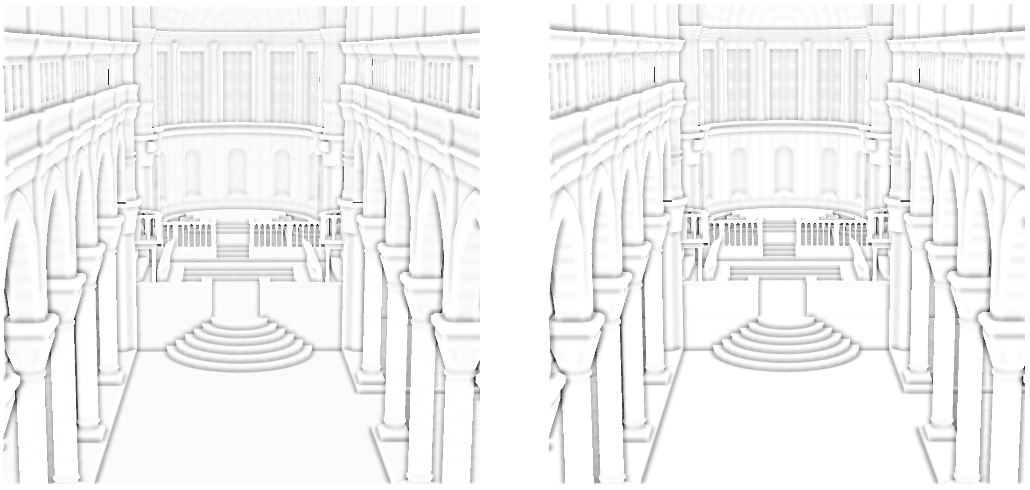
To achieve a good looking effect it is necessary to apply a blur filter over the noisy result. A simple gaussian blur filter is a relatively cheap solution but will blur occlusion over depth edges. Instead, a more expensive cross-bilateral blur [10] [11] step is required that preserves sharpness across depth and/or normal boundaries.

2.3. LINE AAMPLED AMBIENT OCCLUSION

Line sampling was introduced by Loos and Sloan in 2010 [12] and was in many ways a direct improvement over Crytek's SSAO. They identify the technique used by Crytek as a point sampling algorithm for volumetric integration. By integrating over the volume of the sphere using line segments instead of points (see Figure 4.1c), the algorithm is able to achieve smoother results while using less samples.

The key observation to make here is that by evaluating line segments, it is possible to extract more information from each sample. Whereas a point sample only evaluates into a binary value (occluded or unoccluded), a line sample evalutes into a percentage of occlusion along the line segment.

Loos and Sloan also state that if information about the normal is available (e.g. in a normal buffer), it can be used to restrict the sphere to a normal-oriented hemisphere. This method eliminates all self occlusion artifacts that were present in Crytek's original SSAO implementation. The additional cost of sampling a normal buffer is relatively low compared to the quality improvement of the result.



(a) 33 Point samples at 156 fps (b) 9 Line samples at 210 fps

Figure 2.4: Loos and Sloan [12] show how their line sampling approaches achieves similar results using less samples. While each line sample is slightly more expensive than a point sample, the overall performance increase is considerable.

2.4. HORIZON-BASED AMBIENT OCCLUSION

Horizon-based AO was proposed by Bavoil et al. in 2008 [4] and aims at finding the maximum horizon angle at which light can reach a sample point. It works by marching over rays in randomized directions and keeping track of the maximum elevation angle (see Figure 4.1d).

Because we march over the rays with fixed step sizes, it is possible that we miss the actual maximum horizon angle if the step size is too big. The authors suggest to introduce a randomized jitter in the step sizes, this will avoid systematic noise in the result.

Compared to both approaches discussed before, Horizon-based AO is not based on volumetric obscurance. Instead, it more closely resembles the original defintion of SSAO as the percentage of rays that can escape the scene without colliding with geometry. As a consequence, this technique has the potential of more closely resembling a physically correct solution. Of course, the quality of the result also depends heavily on the number of rays, the step size and the blur filter.

2.5. SCREEN-SPACE DIRECTIONAL OCCLUSION

Screen-space directional occlusion (SSDO) was introduced in 2009 by Ritschel et al. [13] and shows how SSAO can be extended with an additional diffuse indirect bounce of light. While the paper applies their ideas on a normal-oriented hemisphere point-sampling approach, the same ideas are also valid for many other screen-space methods.

The key observation that they describe is that for a relatively small cost, it is possible to compute diffuse light for each (unoccluded) point sample. This effectively removes the traditional decoupling between occlusion and illumination in a scene. They show how this can significantly improve the result, especially in case of incoming illumination with different colors from different directions.

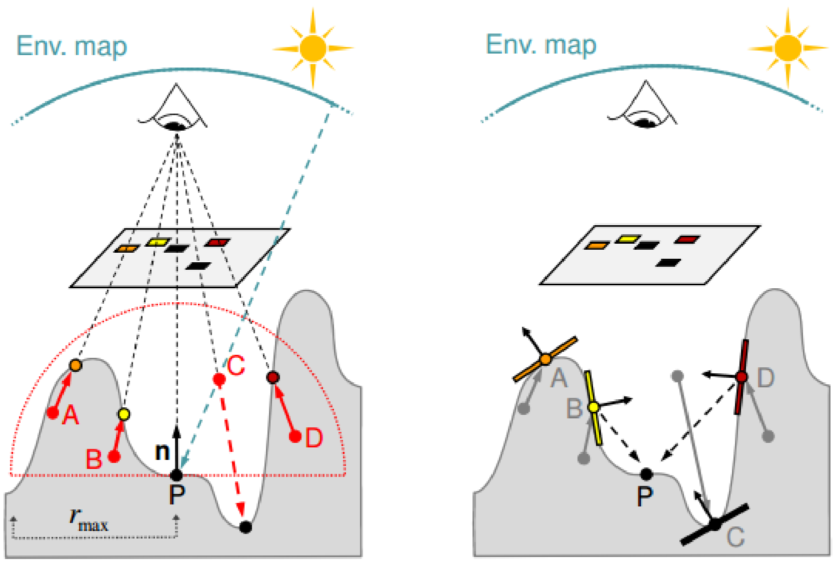


Figure 2.5: SSDO as described by Ritschel et al. in 2009 [13]. Note how each sample contains light information for directional light (left) and indirect light (right).

2.6. STATISTICAL APPROACHES

Statistical approaches aim at improving undersampling issues (e.g. see Figure 5.4a), which are still the primary problem when balancing performance and quality in SSAO-oriented methods. The general idea is to utilize some form of data structure that allows for efficient filtering over larger areas of the depth map.

As described by Slomp et al. in 2010 [14], Summed-area tables (SAT) are an efficient data structure to compute local averages of depth values per pixel. This local depth average can then be used as indicator for ambient occlusion. They show how the sampled area can be shifted by incorporating the surface normal. Additionally, they discuss how depth refinement can be used to subdivide the sampling area and achieve better looking results. The paper does not mention how to handle depth discontinuities, i.e. depth values that are out of range (either far in front or far behind) and significantly skew the average depth.

Around the same time Diaz et al. presented a similar technique [15]. They also utilize SATs and are able to achieve smooth-looking results with high performance.

They acknowledge that naively applying SATs leads to strong artifacts at depth discontinuities in the form of halos or overestimations. Because they operate on the statistical mean of a group of depth values they are unable to exclude samples that are outside the occlusion radius. These samples will strongly impact the mean value and result in overocclusion.

Diaz et al. [15] show how the halos can be benificial for identification purposes in, for example, medical applications. However, in most rendering scenarios that aim at achieving physically plausible results, the halos should be considered unwanted artifacts. In this thesis, we build upon the previously discussed approaches and show how to remove such artifacts using adaptive depth layers.

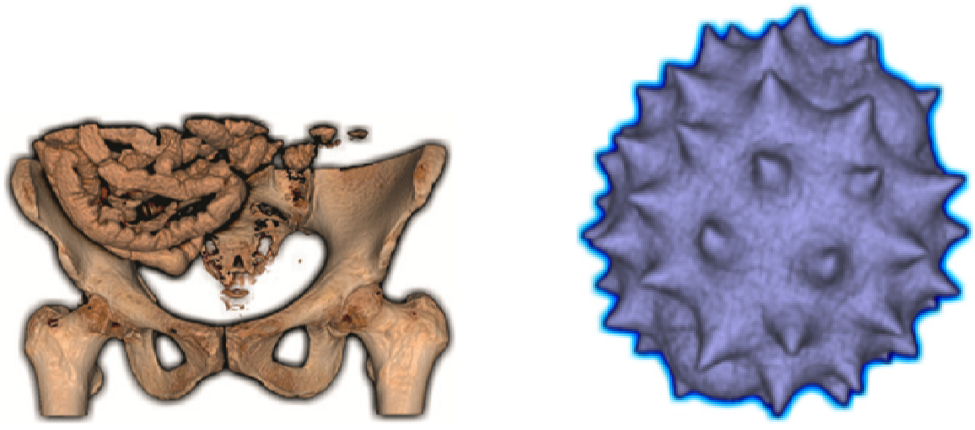

(a) AO with SATs (b) Stylized rendering of halos

Figure 2.6: As shown by Diaz [15], SATs can be used to efficiently evaluate smooth AO without the usual noise artifacts. Halos caused by overestimation are clearly visible and are used to improve object identification.

3 BACKGROUND

3.1. GPU ARCHITECTURE OVERVIEW

A GPU (or graphics processing unit) is an electronic component specialized at rapidly processing parallel work loads. Originally, the GPU was highly geared towards real-time rendering of 3D scenes using a fixed-function pipeline. This pipeline consisted of multiple stages that each performed a dedicated task, the most important once were vertex transformation, primitive assembly, texturing, blending and writing the result to a framebuffer.

The last couple of years the GPU has become increasingly powerful and flexible. Through the introduction of vertex and fragment shaders in 2001 the programmer has a much greater amount of control over the types of computation a GPU can perform. This trend continued with the introduction of unified shader architectures in 2006 and is still ongoing today while GPUs become increasingly capable of processing general purpose parallel workloads (GPGPU computing).

In this thesis we will show how we can take advantage of the flexibility of modern GPUs to efficiently evaluate the problem of Ambient Occlusion (AO). In order to do this we will frequently utilize programmable shaders and GPGPU programming APIs.

3.2. DEFERRED SHADING AND SCREEN-SPACE ALGORITHMS

Before introduction of programmable shaders GPUs utilized a fixed-function pipeline. This pipeline imposed the use of a forward rendering context. A forward renderer can be thought of as a fairly linear process in wich each piece of geometry is transformed, rasterized and shaded independently before being added to the result.

A deferred rendering context [16] [17] [18] works differently because it delays the shading untill the very end. After geometry is transformed and rasterized it is not shaded but instead its properties are written to a Geometry-Buffer (G-Buffer). A G-Buffer usually consists of at least depth, normal and color attributes. In a final step this G-Buffer is processed in a lighting stage that combines all contributions with a lighting model to produce the final shaded output.

Both forward- and deferred rendering are still in use today and each have their own respective benefits and drawbacks. Deferred rendering usually achieves better performance when there are many light sources but uses more memory because of the G-Buffer. Forward rendering combines better with existing fixed-function functionality such as built in anti-aliasing support.

A big advantage of deferred rendering contexts is the seamless integration with many screen-space techniques. Screen-space algorithms operate on rasterized images (such as the G-Buffer) instead of the original 3D geometry.

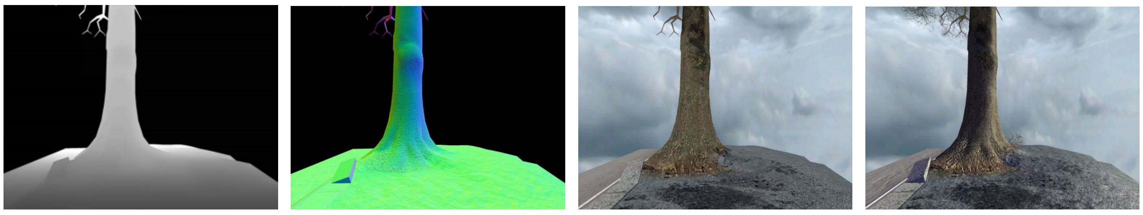

(a) Depth (b) Normals (c) Diffuse (d) Final

Figure 3.1: A scene rendered in a deferred context with depth (a), normals (b) and diffuse (c) components. The final results (d) is constructed from these inputs in the lighting phase.

In today's increasingly complex scenes the benefits of screen-space algorithms are becoming more pronounced. If it is possible to implement a technique as a screen-space effect than the amount of vertices or geometrical detail is no longer relevant for determining its performance.

Note that while screen-space algorithms are generally considered fast as compared to traditional methods, for best performance it is important to consider cache coherency. A GPUs processing power is most efficiently utilized if cache misses are minimized. Because GPUs optimize texture memory for 2D spatial locality, this means that performance is best when neighbouring pixels access texture memory from the same region. For many use cases this is indeed true. Ambient Occlusion is a good example because it is primarily concerned with data about local geometry.

3.3. Mip maps, N-Buffers and Y-maps

One of the most common datastructures in computer graphics is the Mip (multum in parvo) map [19]. A full mipmap chain consists of a full resolution main texture and a list of progressively lower resolution representations of the same image. Each image in the mipmap chain is always half the width and height of the previous image. This results in a total memory requirement of $\frac{4}{3}n^2$, where n equals the width and height of the image. As we can see in Figure 3.2a the final image in the mip map chain consists of a single pixel that covers the entire original image.

More recently the N-Buffer was proposed as an alternative datastructure by Décoret [20]. Where a Mip map chain useses progressively lower resolution images, an N-Buffer remains its original resolution throughout the chain. As we can see in Figure 3.2b this allows us to query any rectangle with power of two dimensions with a single query. The obvious downside is the higher memory consumption of $n^2 \log 2$.

Y-maps [21] promise to offer the best of both worlds. They consist of one (or more) mip map steps, followed by a full N-Buffer chain. While this somewhat limits the precision at which we can sample, it also significantly reduces the amount of memory. This is illustrated in the graph shown in Figure 3.3.

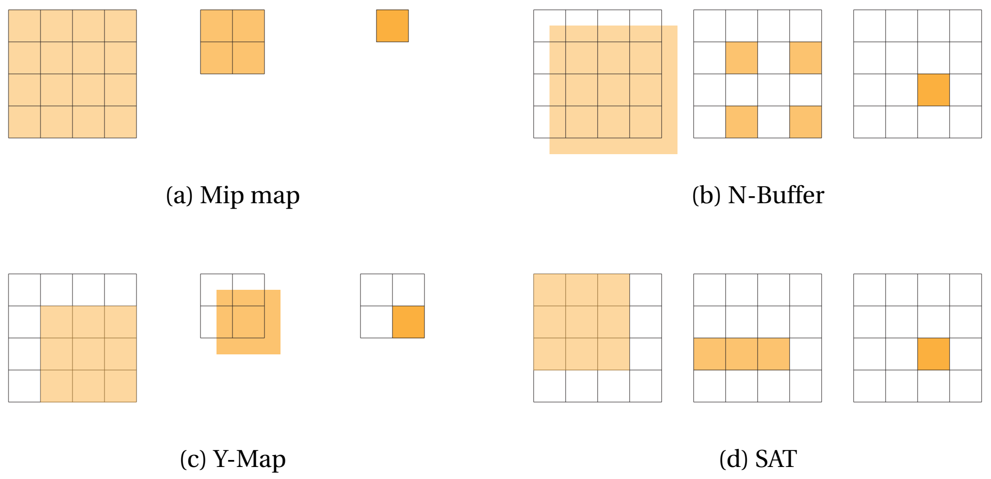


Figure 3.2: Comparison of sampling patterns for constructing MipMaps, N-Buffers, Y-Maps and SATs. Areas indicated in orange represent the same regions across levels in the data hierarchy.

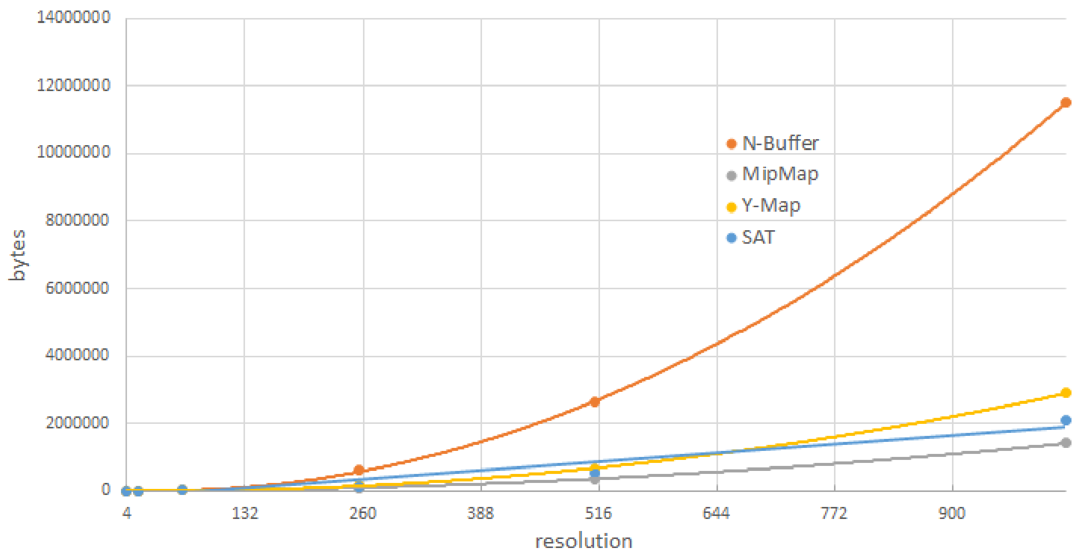


Figure 3.3: Memory requirements for storing different datastructures. Memory usage on the vertical axis in bytes, resolution in pixels on the horizontal axis.

3.4. SUMMED AREA TABLES

Summed Area Tables (SAT) were first introduced in 1984 by Crow [22]. It is possible to think of an SAT as an integral of a texture. Informally, every value in an SAT is equal to the sum of all values in the original texture that are above and left of it. More formally, we can describe an SAT as the result of a two dimensional prefix sum operation.

An SAT only requires a single n^2 texture to store its result. However, it should be noted that in practice an SAT is often stored in a higher precision format because it has to handle a much higher range of values.

The real benefit in using SATs is that we can sample any rectangular area using only four samples (see Figure 3.4). This makes SATs very usefull in applications where this kind of flexibility is required.

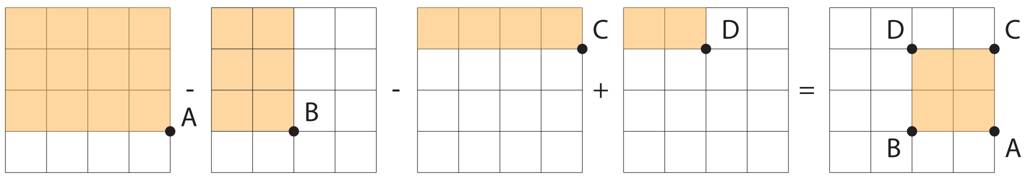


Figure 3.4: Any rectangular region can be sampled with an SAT using only four samples. We start with A and substract parts B and C, because the area where B and C is removed twice we correct this by adding D to the result.

SATs are usually constructed in two passes, a horizontal pass and a vertical pass. In a sequential execution environment the optimal way for constructing an SAT is fairly straightforward. As shown in Algorithm 1, we iterate over all values from start to finish while keeping track of the total sum. At every position we write the current sum value to construct the SAT. For the vertical pass we simply translate our buffers and repeat the same process.

Algorithm 1 SAT Generation

```
1: Given: original input buffer $I$, creates the corresponding SAT $S$.
2: for $x \leftarrow 0, x < n, x++$ do
3:     $sum \leftarrow 0$
4:     for $y \leftarrow 0, y < n, y++$ do
5:         $sum \leftarrow sum + I[x, y]$
6:         $S[x, y] \leftarrow sum$
7:     end for
8: end for
9: $I \leftarrow I^T, S \leftarrow S^T$
10: Repeat once more for the vertical pass.
```

In a parallel environment the problem of finding the most efficient SAT generation algorithm is more difficult. As we can see from Algorithm 1, it is possible to parallelize the outer for-loop. However, we cannot parallelize the inner for loop because of the dependency on sum.

Several approaches were proposed to accelerate SAT generation on a GPU [23]. One of the first was a recursive doubling approach that used $\log n$ passes. In the first pass, each element is summed with the element to its left. During the second pass the value two elements to the left is added. This process is repeated, doubling the stride after each pass. We can see how this results in a full SAT after $\log n$ passes in Figure 3.5.

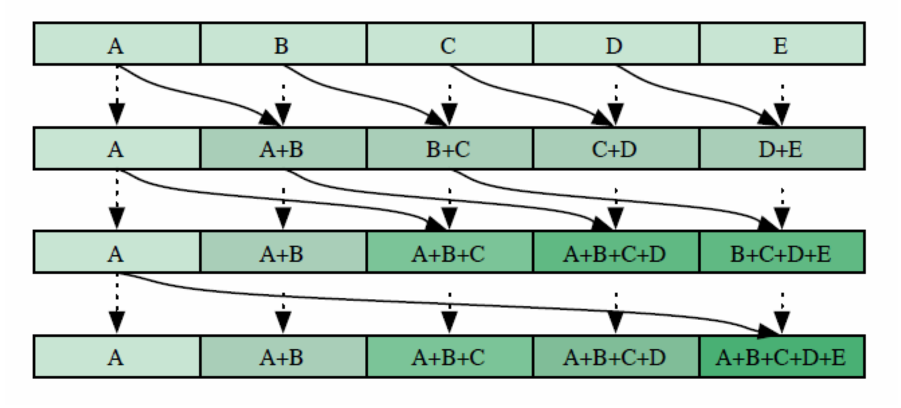
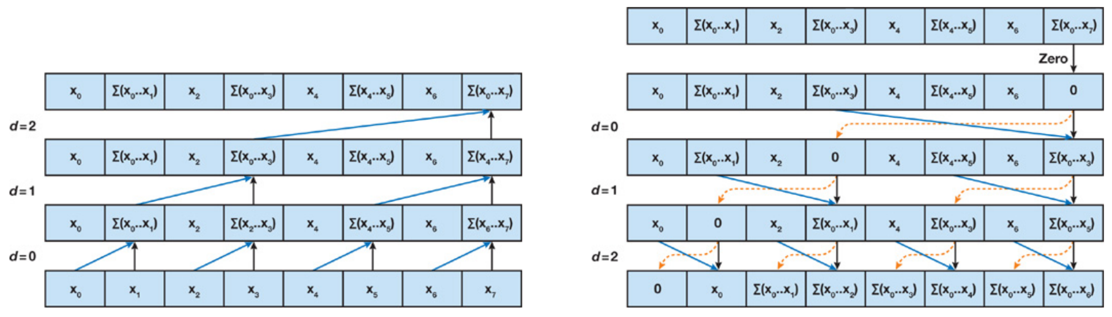


Figure 3.5: SAT generation through recursive doubling algorithm.

More recently, a work-efficient parallel scan was proposed by Harris et al. in 2007 [24]. The presented method aims to achieve the work efficiency of the sequential approach while still taking advantage of the parallel architecture of the GPU. They model the input data as a balanced tree on which they perform an up-sweep and down-sweep phase.



(a) Up-sweep phase (b) Down-sweep phase

Figure 3.6: The work-efficient parallel scan proposed by Harris et al. in 2007 works by firsting performing an up-sweep (a) followed by a down-sweep (b).

3.5. DATASTRUCTURE USE CASES

While all previosuly discussed datastructures have properties that make them applicable in a wide range situations, most have seen frequent use in specific scenarios wich we will quickly discuss.

Mip maps have seen a lot of use in (trilinear) texture filtering that is used to reduce noise and moiré patterns caused by sampling scaled textures during rasterization. Other use cases include LOD algorithms such as geometry mipmaps for efficient terrain rendering [25].

N-Buffers were introduced as a tool to efficiently perform culling operations directly on depth buffers. By using N-Buffers to store minimum and maximum depth values they show how objects can be culled in constant time with only four depth lookups [20].

A prominent use case for SATs is an effect called Depth of Field [23]. This effect simulates the blurriness of objects that are not within a lenses focus. It is implemented by varying the sampling size (i.e. blur radius) from the SATs based on an objects distance to the focus depth. Note that usually, many pixel samples are necessary if sampling is performed from arbitrarily-sized areas. A different scenario that benefits from SATs is glossy reflections. For this effect the amount of blur depends on the distance between an object and the reflector.

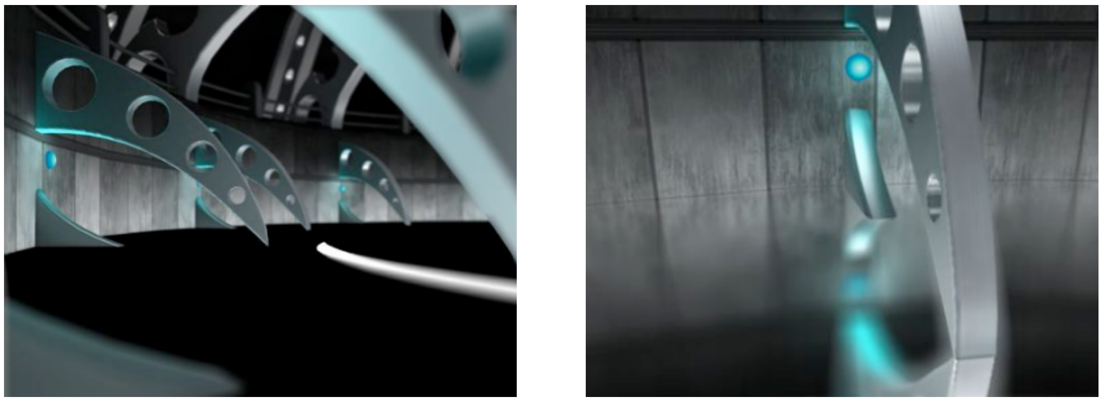

(a) Depth of Field (b) Glossy reflections

Figure 3.7: Examples of problems in the field of Computer Graphics where SATs are commonly applied [23]. In (a) we see how objects at the camera's focus depth are sharp whereas objects in front or behind this depth become increasingly blurry. Glossy reflections (b) show how objects reflected in a glossy surface become more blurred as the distance to the surface increases.

4

Statistical Volumetric Obscurance

4.1. Background

The amount of Ambient Occlusion (AO) at a point $\mathbf{x}$ on a surface is related to the ratio of outgoing rays that are able to escape the scene as opposed to rays that are being blocked by surrounding geometry (Figure 4.1a) [12]. Essentially, AO improves upon standard ambient terms in popular shading models and captures shadows caused by light coming equally from all directions. Formally, we can define AO as:

$$AO(\mathbf{x},\vec{n}) = \frac{1}{\pi}\int_{\Omega} \rho(d(\mathbf{x},\vec{\omega}))\vec{n}\cdot\vec{\omega}\mathrm{d}\vec{\omega} \ ,$$

where $\mathbf{x}$ is the position in the scene, and $\vec{n}$ the normal at $\mathbf{x}$. Here, Ω is the sample volume, usually a surface aligned hemisphere, d is the distance to the first intersection. The fall off function ρ is used to simulate rays with a limited extent, to model local occlusion. In practice a constant, linear or quadratic fall-off function is used.

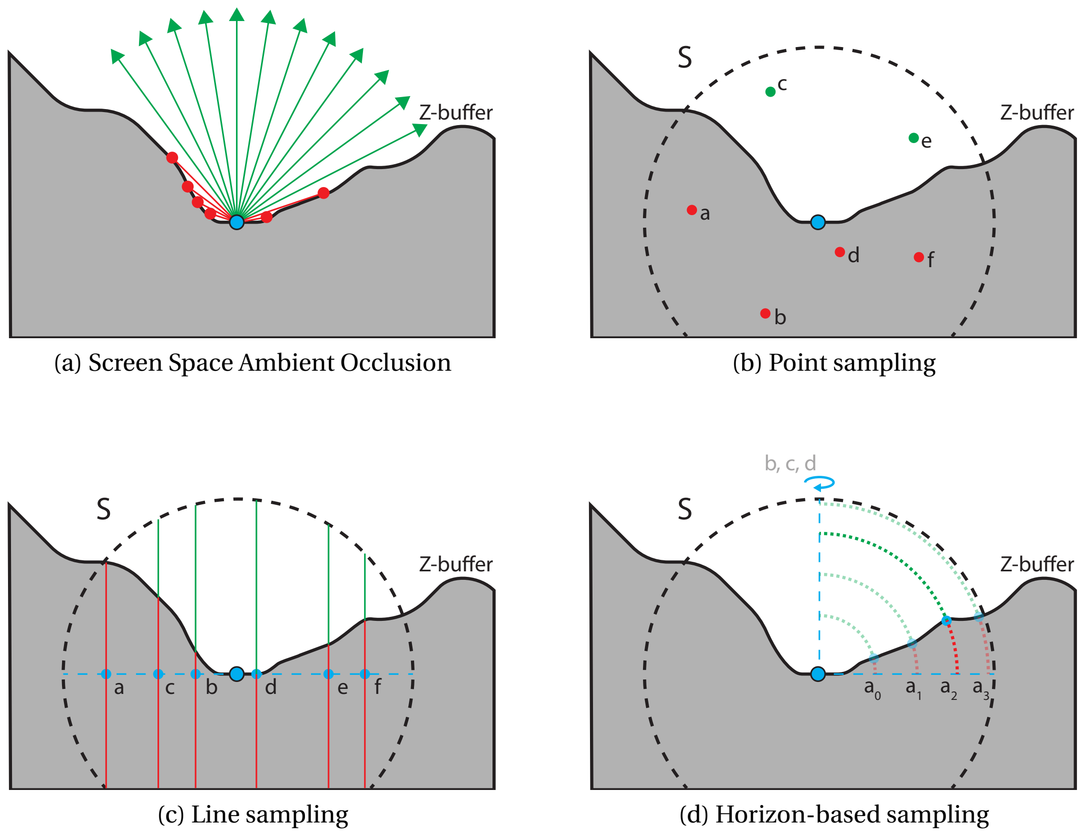
While an exact evaluation of AO based on this definition can be computationally costly, different models exist that achieve similar results. In particular, Volumetric Obscurance (VO) [12] determines the amount of occlusion by approximating the geometric density within a surrounding sample sphere S:

$$VO(\mathbf{x}) = \int_{S} \rho(d(\mathbf{x},s))O(s)\mathrm{d}s \ , \tag{4.1}$$

where O is an occupancy function that is defined to be 0 if s is inside of the geometry and 1 otherwise.

The general assumption used by the VO model is that if a large portion of the sample sphere is filled with geometry, it will be harder for ambient light rays to reach $\mathbf{x}$. While this intuition does not directly relate to any physical process, it has proven to provide similar results to AO in practice.

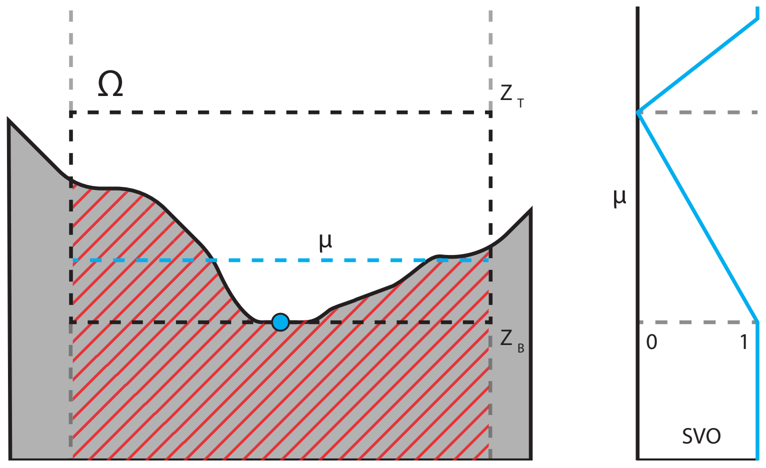
If accounting for the normal $\vec{n}$, one can improve upon the VO model by restricting S to the hemisphere in the direction of the normal, hereby eliminating self-occlusion effects caused by geometry behind the sample.



(a) Screen Space Ambient Occlusion (b) Point sampling

(c) Line sampling (d) Horizon-based sampling

Figure 4.1: **Screen Space Ambient Occlusion:** (a) SSAO at a sample point is defined by the ratio of rays that can escape the scene. Point sampling (b) and Line sampling (c) approximate local ambient occlusion with a volumetric obscurance model that solves the volume integral with randomized samples. Horizon-based sampling (d) marches in randomized directions to compute the maximum angle at which rays can escape the scene.



(a) Statistical Volumetric Obscurance

Figure 4.2: Our statistical volumetric obscurance (a) approach computes the volume integral of a box as an approximation of ambient occlusion. The graph at the right shows how occlusion increases as the average μ rises, after the average leaves the sample box, occlusion falls-off back to zero.

SCREEN-SPACE AMBIENT OCCLUSION

Screen-Space Ambient Occlusion techniques, such as the one presented in this paper, work on the depth buffer of the rendered image instead of the original geometry. As such, the input consists solely of the depth values given in view space and optionally a normal map of the rendered scene. Computing VO on the depth map becomes algorithmically very simple as the simplified geometry is readily available in the depth map and the integral in Equation (4.1) can be solved using Monte-Carlo sampling techniques or other approximations.

4.2. OUR MODEL

Our method builds upon the VO model but introduces some important changes that make it more suitable for current graphics hardware and avoids sampling artifacts. First, instead of a sample sphere, we use a sample box (Figure 4.3a). The general assumption used by VO stays the same, i.e. if we determine that our sample box is largely filled with geometry, we assume that there will be a high amount of occlusion and vice versa. Second, we replace the 3D obscurance function in Equation 4.1 with a 2D version as follows. We assume the depth values in our Z-Buffer form a continuous function $G : \mathbb{R}^2 \to \mathbb{R}$ with $G(x, y) = d_{x,y}$ being the depth value at pixel position x, y. The *first mean value theorem for integration* states that the mean value μ of G over a domain Ω is exactly:

$$\mu = \frac{1}{A_\Omega} \int_\Omega G(x)dx \ , \tag{4.2}$$

where A_Ω is the area of the integration domain. Applied to our case this means that the occupancy within the sample box can be expressed as the mean value $\mu(\mathbf{x})$ over the rectangular area of the bottom face of the sample box centered at $\mathbf{x}$ (Figure 4.3a) with respect to its z-value. Let, $z_B(\mathbf{x})$ be the depth value of the bottom face of the sample box and $z_T(\mathbf{x})$ of the top face, as we are only interested in the relative amount of occupancy we can cancel A_Ω out and define our statistical volumetric obscurance model as:

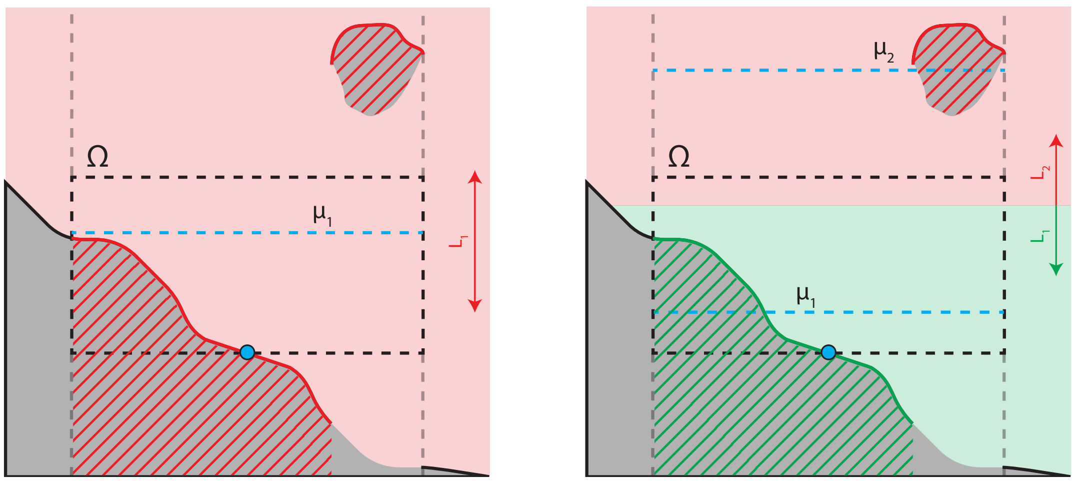
$$SVO(\mathbf{x}) = \rho\left(\frac{\mu(\mathbf{x}) - z_T(\mathbf{x})}{z_B(\mathbf{x}) - z_T(\mathbf{x})}\right). \tag{4.3}$$

The extent of the sample box is directly given by the pixel coordinates and the according depth value. Therefore, all we need for the evaluation of the obscurance is the mean value $\mu(\mathbf{x})$. As the size of the sample box varies with the depth at each pixel we make use of *Summed-Area Tables* [22] as they provide the flexibility to adjust the sample area per pixel over which we compute the average.

Note that in Equation 4.3 geometry outside of the sample box influences μ as well. To account for this fact, we would want to weigh the influence of each sample via a fall-off function $\rho(x)$, which is 1 (no occlusion) if $x \le 0$ and $(1-x)$ if $0 < x < 1$ and to rise to one with a user-defined slope as the distance to our sample point increases further (Figure 4.3a). In the next section, we explain how to incorporate this idea.

4.3. DEPTH LAYERING

Because we only posses information on the average depth of the geometry inside the sample area, we can no longer apply the fall-off function to every depth value within the sample box separately. The result are visible artifacts around strong depth discontinuities in the form of halos; overestimations of the true obscurance (Figure 4.5a).



(a) single depth layer (b) two seperate depth layers

Figure 4.3: **Depth Layering:** While a single layer will result in a single average over all geometry (a), we can slice our scene in multiple depth layers to obtain averages for each layer seperately (b).

To counteract overestimation, we divide the depth map into m uniformly arranged layers orthogonal to the viewing direction based on the maximum and minimum view space depth. Each depth pixel is assigned to the layer which overlaps with the according depth value (Figure 4.3). Non-assigned pixels are set to 0. The depth buffer is processed and split among these multiple layers. During the splitting operation, we use an additional (color) channel to keep track, which samples/pixels are valid, i.e., the channel is one if a depth sample was assigned to the according pixel and zero otherwise. We then generate SATs for each layer and channel separately. Contributions from each layer L_i overlapping with the sample volume Ω are weighted by the according sample count n_i (the number of samples assigned to L_i), to account for the missing values, and combined into a total obscurance value $SVO_{\text{Layered}}(\mathbf{x})$ as follows:

$$SVO_{\text{Layered}}(\mathbf{x}) = \frac{1}{\sum_{i\in\Omega} n_i(\mathbf{x})} \sum_{i\in\Omega} SVO_i(\mathbf{x}) n_i(\mathbf{x}) \ , \tag{4.4}$$

where SVO_i is the statistical volumetric obscurance defined in Equation (4.3) computed on layer L_i.

With $m \to \infty$ Equation (4.4) converges to an accurate VO value. However, the computational effort is linear in the number of layers m and larger scenes require a high number of depth layers so that computation times would no longer be competitive compared to other real-time AO techniques. For example, we found that the Sibenik cathedral scene would require around 64 layers for good results (Figure 4.5d).

4.4. ADAPTIVE DEPTH SLICING

To reduce the computational effort of the linear depth-slicing approach, we propose to use depth layers which adapt to the local geometry. We drew inspiration from higher-dimensional filtering approaches [26] as our technique also builds on a recursive process that partitions the current depth map of a layer into two disjoint sets in each recursion. The intuition behind this step is that as long as pixels with very different depth values are further apart than the screen-space size from the according sample area then they do not influence each other during the obscurance computation.

Our algorithm, depicted in Figure 4.4, works as follows: Starting with the original depth map, we create a smoothed version of this depth map using the according SAT. For this, we determine the sample area at each pixel and compute the mean value μ from the contained depth values as described in Section 4. Each depth value is then either assigned to the upper or lower layer based on its relative depth value compared to μ, hereby, often successfully separating far and near samples. As in the uniform depth slicing approach (Section 4.3), we keep track of the active pixels using an additional channel. The process is repeated for the newly created layers if desired.

During rendering we simply evaluate all layers using Equation 4.4. The fall-off function ρ automatically adjusts the influence of each layer correctly; reducing the influence of samples outside the sample box.

The adaptive depth slicing dramatically reduces the number of required layers. As little as four adaptive layers can achieve results that are comparable to the naive 64 uniform layer implementation (Figure 4.5d and 4.5f).

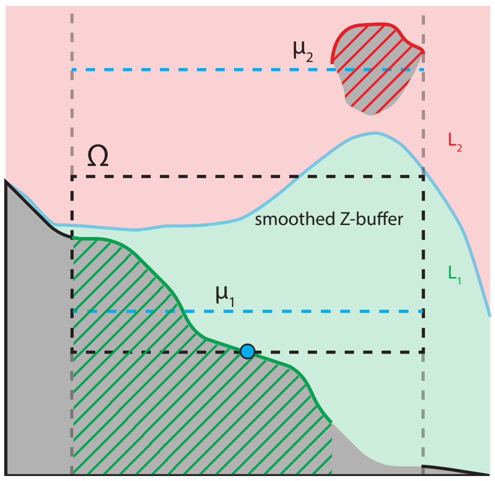


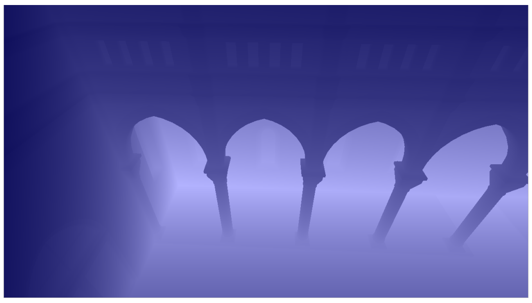
Figure 4.4: **Adaptive Depth Slicing:** In each recursion a smoothed Z-buffer is constructed, each depth sample is either assigned to the upper (red) or lower layer (green).

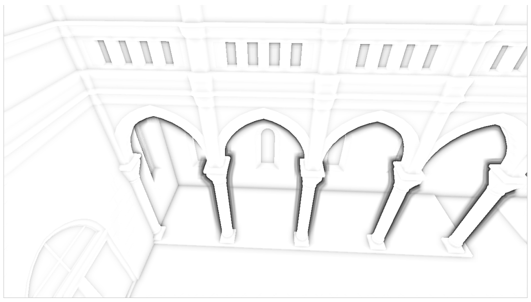
4.5. SURFACE NORMAL INCORPORATION

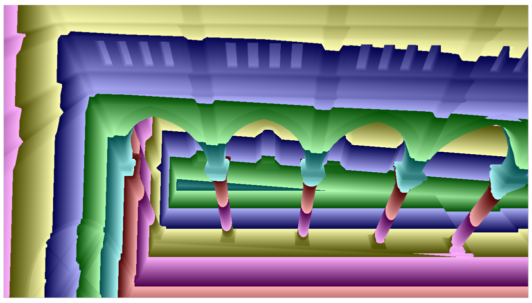
Up until now, we have not taken the surface normal into account. The sample box was always aligned with the viewing direction.

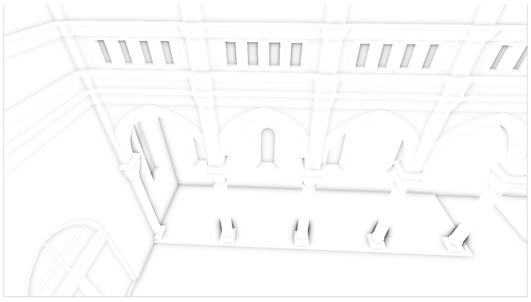
With an increasingly sloped surface, however, (Figure 4.6a and 4.6b), a bias is introduced if parts of this surface are hidden in the depth map by other objects closer to the camera. In Figure 4.6a the mean value of layer L_1 (green) is only slightly above the sample position. In Figure 4.6b it is raised as the object in layer L_2 hides a part of the surface underneath. Performing VO computation only for the positive half-space in the direction of the surface normal can enhance the perception of finer scale details [12]. To benefit from these advantages, we here extend our approach to 3D by orienting our sample box accordingly (Figure 4.6c).

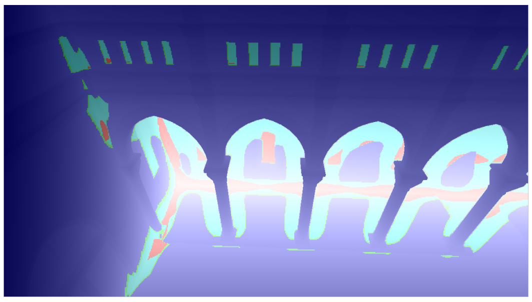
After the adaptive layer computation from Section 4.4, we reproject each depth value in each layer into view space to acquire its 3D position. We save the results in RGB maps


(a) 1 layer


(b) 1 layer, 1 ms


(c) 64 layer


(d) 64 layer, 55 ms


(e) 4 adaptive layer

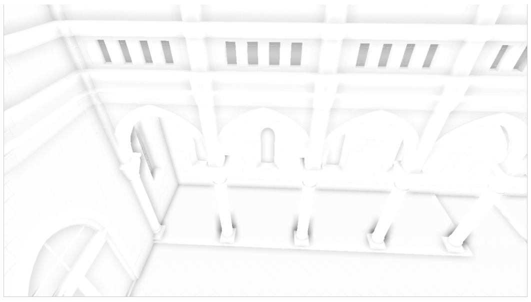

(f) 4 adaptive layer, 2-3 ms

Figure 4.5: **Comparison**: When using a single layer (a) we see that depth discontinuities result in dark halos (b). This problem can be solved by splitting the scene in multiple layers (c) however, this has a big impact on performance (d). By using adaptive slicing we only make use of additional layers at the depth discontinuities itself (e), this allows us to eliminate halos and still achieve good performance (f).

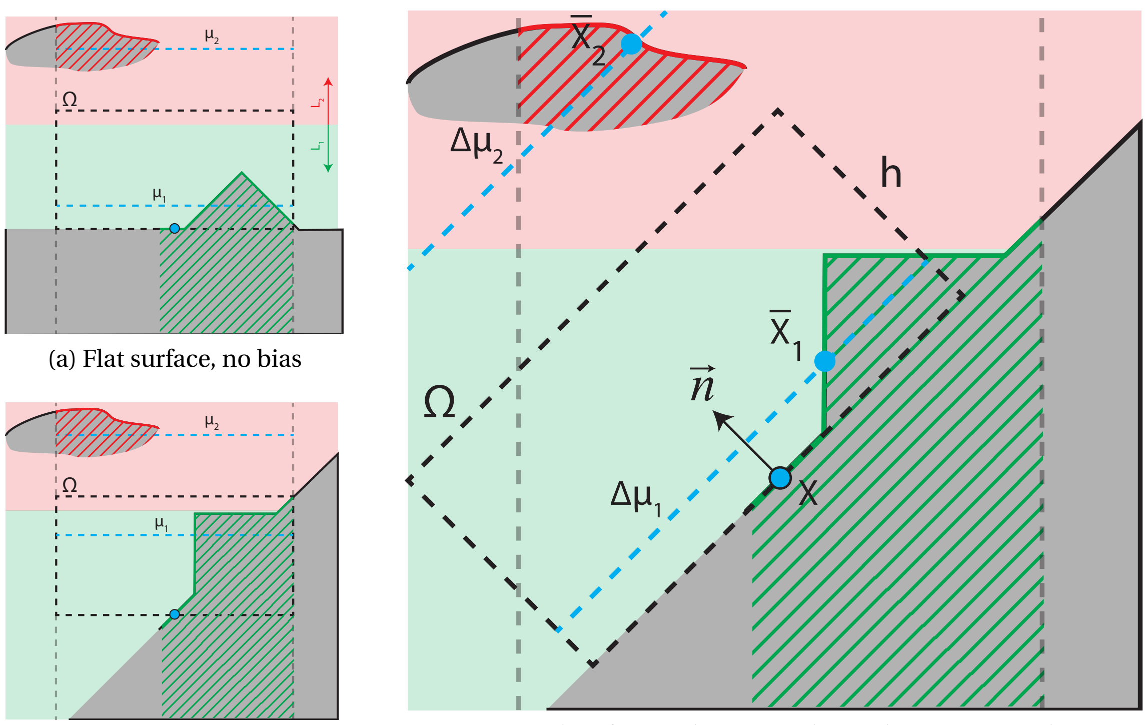
and compute the according SATs for each. For the SVO computation we still use the same samples inside the original sample area, but perform the computation in 3D now. We compute the average position $\bar{\mathbf{x}}$ of all reprojected depth samples and project it onto the surface normal $\vec{n}$, which conveniently reveals the average height of all samples along the surface normal:

$$\Delta\mu = (\bar{\mathbf{x}} - \mathbf{x}) \cdot \vec{n} \ , \tag{4.5}$$

where $\mathbf{x}$ is the surface position. The oriented statistical surface obscurance SVO_i per layer L_i is then:

$$SVO_i = \text{clamp}\left(\frac{\Delta\mu}{h}, 0, 1\right) \ , \tag{4.6}$$

where h is the height of the sample box. Equation 4.4 is used to compute the final obscurance.



(a) Flat surface, no bias

(b) Rotated surface, bias

(c) Rotated surface with projected samples, correct occlusion

Figure 4.6: **Normal Integration:** (a) and (b) Considering only the hemisphere/hemibox around a surface point in the viewing direction leads to different occlusion results depending on the slope of the surface. (c) Projection of the mean of all depth samples in 3D space onto the surface normal removes this bias.

4.6. OPTIMIZATIONS

We introduce two more important performance improvements for the presented technique; approximate SATs and differential SAT computation.

4.6.1. APPROXIMATE SATS

The computationally most costly part of our algorithm is the SAT computation. While we experimented with other prefiltering techniques such as Mipmaps, N-Buffers [20] or Y-Maps [27], SATs provided the highest quality. It turns out that a simple trick reduces the computation times by a factor of 4 to 16 with negligible impact on quality.

Instead of computing full-resolution SAT, we downsample the input by a factor of 2-4 in both width and height by averaging the depth values. We exploit that upscaling such an SAT with linear interpolation results in a similar SAT as for the full resolution input. Please note that our additional channel is important during downsampling to keep track of the sample count. As linear interpolation is hardware-accelerated, it does not induce an important performance hit. Figure 4.7 shows a quality comparison between using a full-resolution SAT and downsampled versions. Further, downsampling results in visible flickering artifacts during rendering.

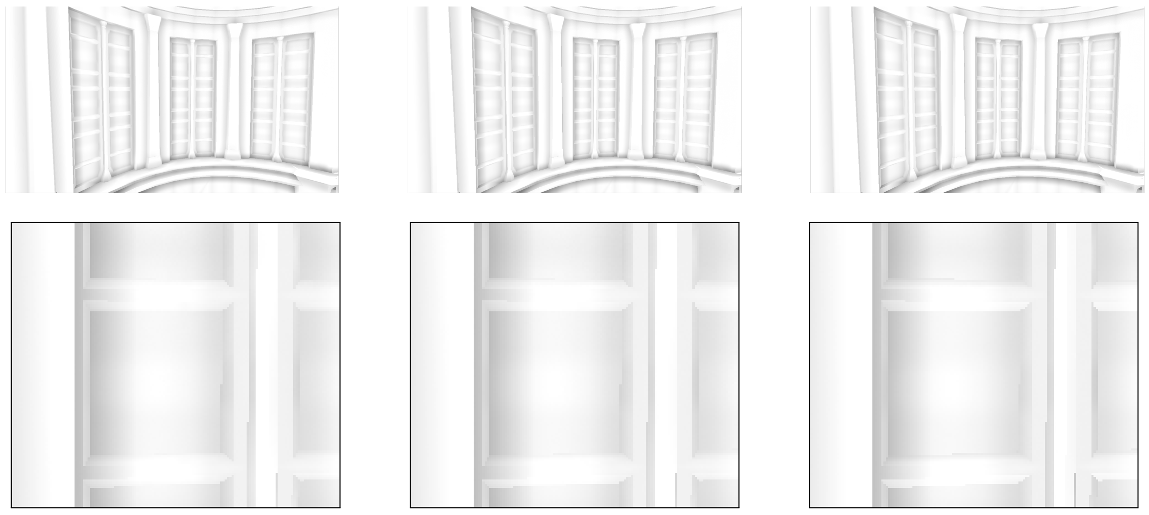


Figure 4.7: **Approximate SATs:** Constructing lower resolution SATs has a minimal impact on quality.

4.6.2. DIFFERENTIAL SAT COMPUTATION

Let L_0 be the (downsampled) original depth map, and $L_{1,0}$ and $L_{1,1}$ be the first two adaptive sublayers resulting from partitioning L_0. Each subdivision of a layer into its sublayers requires the computation of a corresponding SAT S. An important observation is that while the samples contained in L_0 are distributed among the sublayers $L_{1,0}$ and $L_{1,1}$, their total sum does not change. Thus, subtracting SAT $S_{1,0}$ from S_0 results in $S_{1,1}$ (Figure 4.8).

For four adaptive layers, we save generating and storing three out of seven SATs (for more layers the ratio approaches 1:2). During rendering, we compute them on-the-fly by subtracting all ancestral and the sibling layer from the root SAT S_0. The pseudo-code in Alg.2 shows how to query the SAT for all four leaf layers $S_{2,0}, S_{2,1}, S_{2,2}$ and $S_{2,3}$ given only $S_0, S_{1,0}, S_{2,0}$ and $S_{2,2}$.

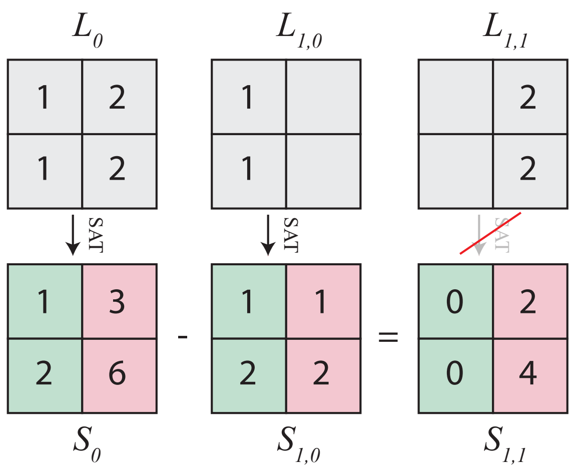


Figure 4.8: **Differential SAT Computation:** We can spare the computation of one SAT in each partitioning step as $S_{1,1} = S_0 - S_{1,0}$.

Algorithm 2 Adaptive Depth Slicing with Differential SAT Computation

```
1: Given: depth buffer $G_d$, and normal buffer $G_n$

2: $L_0 \leftarrow \text{downsample}(G_d)$
3: $S_0 \leftarrow \text{sat}(L_0)$

4: $L_{1,0} \leftarrow \text{partition}(L_0, S_0)$
5: $S_{1,0} \leftarrow \text{sat}(L_{1,0})$

6: $L_{1,1} \leftarrow L_0 - L_{1,0}$   // computed on-the-fly
7: $S_{1,1} \leftarrow S_0 - S_{1,0}$   // computed on-the-fly

8: $L_{2,0} \leftarrow \text{partition}(L_{1,0}, S_{1,0})$
9: $L_{2,2} \leftarrow \text{partition}(L_{1,1}, S_{1,1})$

10: $S_{2,0} \leftarrow \text{sat}(L_{2,0})$
11: $S_{2,2} \leftarrow \text{sat}(L_{2,2})$

12: $S_{2,1} \leftarrow S_{1,0} - S_{2,0}$         // computed on-the-fly
13: $S_{2,3} \leftarrow S_0 - S_{1,0} - S_{2,2}$   // computed on-the-fly

14: $SVO(G_d, G_n, S_{2,0}, S_{2,1}, S_{2,2}, S_{2,3})$
```

5 Results

We have implemented our technique using OpenGL/C++. All statistics were measured for an image resolution of 1280×720 pixels on an Intel Core i5 4590 with 8GB of RAM, equipped with an NVIDIA GTX 770 graphics card. We implemented the SAT generation algorithm as an OpenGL compute shader [28].

5.1. Performance

In Table 5.1, we show a detailed performance analysis of our algorithm using full resolution SATs and half resolution in width and height. The numbers refer to the according steps in Alg. 2. As the SAT computation is the most costly part of our algorithm, performance increases by a factor of four if width and height are halved. We found that in many scenes, the overall quality loss was minimal even when reducing the resolution along each axis by a factor of four (Figure 4.7). An even more aggressive downsampling results in temporal flickering around depth discontinuities when the camera moves.

Step (see alg. 2)	T_{full} (ms)	T_{half} (ms)	Speed-up
2	0.13	0.05	x2.6
3	1.40	0.32	x4.4
4	0.57	0.11	x5.2
5	1.39	0.32	x4.4
8, 9	0.92	0.21	x4.3
10, 11	2.80	0.63	x4.4
14	1.65	0.50	x3.3
Total	8.86	2.14	x4.1

Table 5.1: **Performance Evaluation:** Breakdown of computational cost of our algorithm for full resolution SATs and half resolution SATs. The numbers refer to Algorithm 2.

5.2. Memory Requirements

For our four final SATs, we use textures that contain four 32bit floating point channels. The first three channels are used to store view-space coordinates and the last one is used as a mask to mark valid samples in each layer. When computing SATs at half resolution in width and height we require a total of 14MB of memory.

5.3. Technical Implementation

To be able to quickly compare different AO techniques we set up a deferred rendering framework (OpenGL, C++) that allowed us to quickly prototype different implementations. This framework was designed to have an interface that provided an abstraction layer for the implementation details of different AO approaches.

While most SSAO approaches are relatively straightforward to implement, some of our ideas were harder to evaluate efficiently on a GPU. In particular, a significant amount of time was spend on finding the fastest SAT construction algorithm for current graphics hardware.

Several different GPGPU APIs such as NVIDIA CUDA, OpenCL and OpenGL Compute Shaders were studied and evaluated. By profiling several implementations we were able to arrive at a solution that was both fast and did not introduce a significant amount of API overhead (see Table 5.2).

Our original implementation used the standard OpenGL rasterization API with fragment shaders and was relatively slow. By switching to a GPGPU language such as CUDA we were able to achieve significantly better performance. As we can see in Table 5.1, the most computationally intensive part of our method is the computation of the SATs. CUDA exposed functionality that allowed us to implement more efficient algorithms for computing these SATs on a GPU (see Section 3.4).

However, the relative slow interop between CUDA and OpenGL pushed us to switch to OpenGL Compute Shaders. CUDA is generally known as being a more flexible and matured environment as compared to OpenGL Compute Shaders. However, for our specific requirements we found that OpenGL Compute Shaders were sufficiently capable as well. Additionally, because Compute Shaders can operate on the same OpenGL databuffers, there was no need for API interop code. In our final implementation we are able to compute our 4 adaptive layers with SATs in under 8.9 ms.

API	Performance	API interop
OpenGL (Fragment Shaders)	65.4ms	n/a
CUDA	9.7ms	3.5ms
OpenGL (Compute Shaders)	8.9ms	n/a

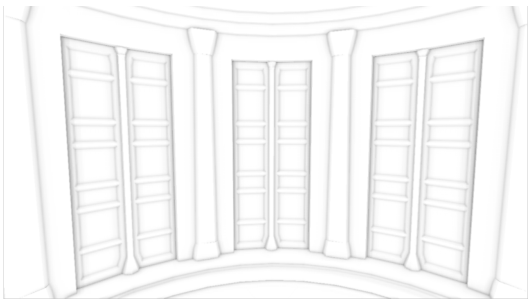
Table 5.2: **Performance Evaluation:** Breakdown of computational cost of our algorithm for full resolution SATs and half resolution SATs. The numbers refer to Algorithm 2.

5.4. COMPARISON TO OTHER TECHNIQUES

We compare our technique to the classic point and line sampling SSAO techniques, which are the most commonly used [2, 12]. By choosing a very high sample count (256 point samples per pixel), we additionally generated a reference image for volumetric obscurance. In Figure 5.2, we show that our technique can generate results that are comparable in quality. In Figure 5.3 we compare our technique to point and line sampling. We chose the number of samples so that all approaches produce visually similar quality. Using an SAT with a quarter of the width and height resolution our approach is slightly faster than both.

Increasing the AO radius our performance stays the same, whereas the performance of line and point sampling decrease, due to an increase of the bilateral blur radius, which is mandatory to diminish the increasing undersampling artifacts. That means, our algorithm scales well to higher resolutions compared to point and line sampling.

If only few samples are computed for the point or line-sampling (e.g., for very high performance), visible undersampling artifacts appear (Figure 5.4). Our approach does not suffer from undersampling and leads to more details, e.g. at the wall.


(a) 11 line samples (1.47ms)

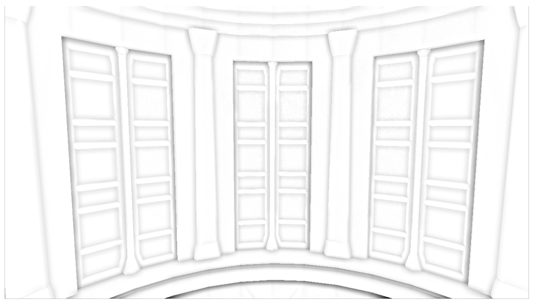
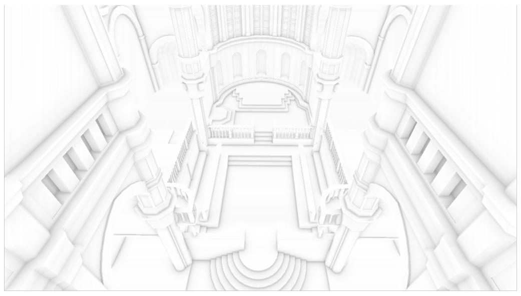

(b) SVO (0.93ms)

Figure 5.1: **Comparison to Line Sampling:** Both figures evaluate the occlusion at full resolution of 1280x720. Line sampling (a) using 8 samples with a 4x4 randomization kernel with an 8x8 bilateral blur applied. SVO (b) with 4 adaptive layers with quarter resolution SATs.


(a) VO reference

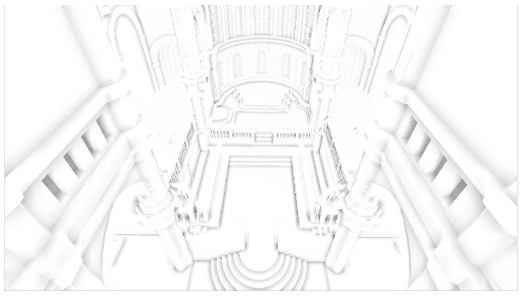
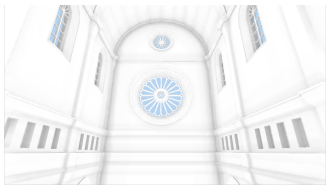
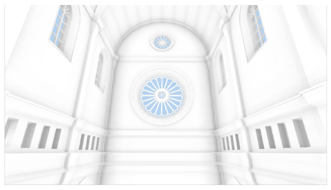

(b) SVO

Figure 5.2: **Comparison to a Reference VO:** (a) Reference from 256 point samples without a randomization kernel or a blur filter. Our method (b) shows comparable results using 4 adaptive layers and full resolution SATs.


(a) 17 point samples (2.0ms)


(b) 12 line samples (1.7ms)

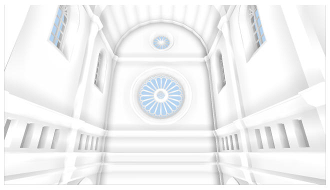
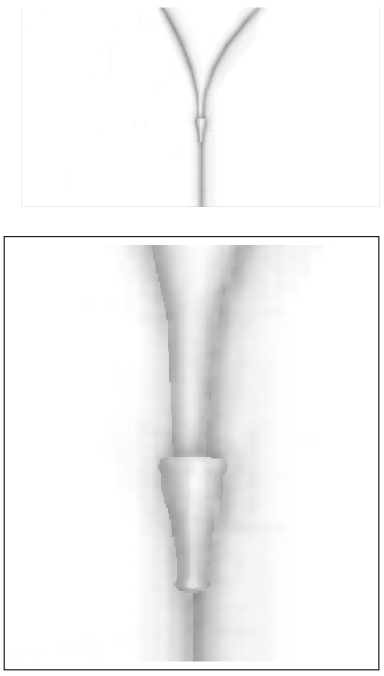

(c) SVO (1.6ms)

Figure 5.3: **Comparison to Point and Line Sampling:** (a) uses a point sampling approach with 17 samples and a 4x4 randomization kernel, occlusion is evaluated at half resolution and a bilateral upsampling with 7x7 blur kernel is applied. (b) achieves similar results but only uses 12 line samples. In (c) we evaluate occlusion at the full resolution using 4 adaptive layers with quarter resolution SATs.


(a) 16 point samples (0.77ms)

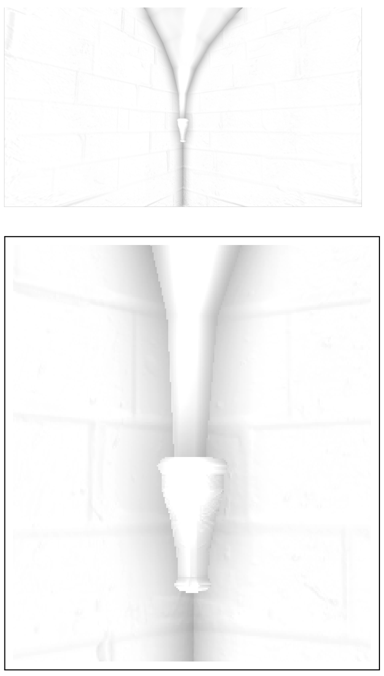

(b) SVO (0.65 ms)

Figure 5.4: **Comparison to Point Sampling:** (a) shows a close-up view of typical point sampling configuration were occlusion is evaluated at half resolution, using 16 samples with a 4x4 randomization kernel. The result is upsampled and an 8x8 bilateral blur filter is applied. (b) SVO evaluated at full resolution with 4 adaptive layers and quarter resolution SATs.

6 Conclusion

Statistical Volumetric Obscurance is an alternative to traditional screen-space ambient occlusion, which does not rely on typical sampling techniques. Due to the approximation of the sample sphere by a box function, the evaluation of the local obscurance is reduced to a simple mean-value computation over the sample area, which is efficiently evaluated on the GPU using specialized summed-area tables. Previous approaches in this direction suffered from artifacts such as halos, or very approximate solutions. The adaptive depth slicing avoids these and preserves fine-scale features, leading to a quality similar to previous approaches with many more samples.

6.1. Limitations

For best results the amount of nearby depth discontinuities should be limited. An extreme case, like looking along a row of aligned pillars, breaks this assumption and small halos and dark creases are introduced. Locally adaptive layering would be an interesting future work to address such issues.

As with any screen-space ambient-occlusion technique, the depth map represents only the visible geometry, whereas important information of the overall scene is lost. Rendering the occluded geometry into the depth layers after they have been created would allow us to incorporate even these occluded parts for more precise results beyond the capabilities of traditional SSAO techniques.

6.2. Future Work

As a future exercise it might be interesting to examine the performance/quality tradeoff when using less (2) or more (4+) adaptive manifold layers. Based on intuiton one would expect the performance to scale linearly whereas performance benefits would show diminishing returns.

Moving forward, it would be interesting to investigate using a dynamic number of manifolds based on the rendered scene. As more depth discontinuities are introduced into the rendered scene the number of manifolds could rise for better quality, if there are less depth discontinuities the number of manifolds could be lowered for better performance.

In a next step the idea of a dynamic number of layers could be extended by filtering parts of the scene independently. This would allow us to use a low number of manifolds for the majority of the image. However, if needed, specifically identified parts of the scene could use a high(er) number of manifolds. While this requires a preprocessing step that identifies regions with many depth discontinuities, we have found that this can be achieved relatively cheap by using an N-Buffer with min/max components.

7 Personal Reflection

While the problem statement for this MSc thesis was well defined from the beginning, it has taken a significant amount of exploration of different ideas before arriving at the solutions presented in this report. In this section I will reflect on my work progress and provide an overview of attempts that failed to achieve the desired results but have contributed in other ways to arrive at the work presented in this report.

7.1. Process

The first stage of the MSc thesis was dedicated to a literature study of the most notable papers on AO, Shadow Mapping and Global Illumination. This allowed me to become familiar with existing approaches to AO, as well as positioning it within a wider context of GI.

Originally, the work in this paper was focussed on N-Buffers as the primary datastructure. We experimented with using different pixel formats (integral, floating point) and varying the precision (halfs/floats). Additionally, we looked at how different kernel sizes and the amount of layers affected the performance and quality of the result. To reduce memory requirements without impacting quality we investigated reducing the resolution of the first layers of the N-Buffer (i.e. Y-maps).

The first efforts to eliminate the halos around depth discontinuities were focussed on *detection*. By extending our N-Buffers with channels for minimum and maximum depth values we were able to detect and remove occlusion from the affected regions. However, this resulted in under occlusion artifacts if halos were overlapping actually occluded areas. We experimented with constructing a probabilistic model of the depth map by deriving the variance. This turned out to be less effective than the simpler min/max approach.

In a next step, we experimented with sampling different shapes from our N-Buffer such as lines (instead of squares or rings). Also, we tried several approaches were the depth map was filtered to flatten depth discontinuities before constructing our N-Buffers. Unfortunately, neither of these approaches were able to achieve smooth looking results without introducing different artifacts.

A more promising approach we investigated was based on depth layering. While distributing depth values between multiple layers caused some issues by itself, we were able to arrive at a solution that would generate good looking results as longs as the number of layers was high enough. Because, our solution should have competitive performance characteristics compared to existing techniques we had to siginificantly reduce the required number of layers. Here, we were inspired by the Adaptive-Manifolds paper [26]. While the ideas presented in this paper were focussed on high dimensional filtering algorithms, we were able to introduce some modifications that allowed us to apply them in our depth layering scenario.

Acknowledgements

I wish to thank the following individuals for the help they provided me during my thesis research:

My supervisor, Prof. Elmar Eisemann, for his insights and helpful feedback in our meetings while developing the work presented in this master thesis.

Dr.-Ing. Martin Eisemann, for contributing some new and fresh ideas halfway during my thesis project. Also, I would like to thank him for his helpful feedback during my (continuing) learning process of writing and presentic scientific content.

Finally, I would like to thank Dr.ir. Rafael Bidarra for introducing me to the Computer Graphics and Visualization group during my bachelor. The good experiences during my previous projects at this group have been the primary reason for returning to do my MSc thesis at the CG&V group.

A
Code Listing

A.1. SAT Generation (GLSL compute shader)

```
// (defined at compile time)

// #define HALF_WIDTH
// #define WIDTH
// #define FORMAT_SOURCE
// #define FORMAT_OUTPUT
// #define TYPE
// #define LOAD(x)

layout (local_size_x = HALF_WIDTH) in;

layout (FORMAT_SOURCE) readonly uniform image2D source;
layout (FORMAT_OUTPUT) writeonly uniform image2D result;

shared TYPE shared_data[WIDTH];

uniform vec2 cameraClippingPlanes;
uniform vec2 cameraProjectionConstants;

vec4 toVec4(vec4 x) { return x; }
vec4 toVec4(vec2 x) { return vec4(x, 0, 0); }

void main(void)
{
  uint id = gl_LocalInvocationID.x;

  ivec2 P0 = ivec2(id * 2, gl_WorkGroupID.x);
  ivec2 P1 = ivec2(id * 2 + 1, gl_WorkGroupID.x);

  vec4 V0 = imageLoad(source, P0);
  vec4 V1 = imageLoad(source, P1);

  shared_data[P0.x] = LOAD(V0);
  shared_data[P1.x] = LOAD(V1);

  // (OpenGL SuperBible)
  for (uint step = 0; step < log2(WIDTH); step++)
  {
    barrier();

    uint mask = (1 << step) - 1;
    uint rd_id = ((id >> step) << (step + 1)) + mask;
    uint wr_id = rd_id + 1 + (id & mask);
    shared_data[wr_id] += shared_data[rd_id];
  }

  barrier();

  imageStore(result, P0.yx, toVec4(shared_data[P0.x]));
  imageStore(result, P1.yx, toVec4(shared_data[P1.x]));
}
```

A.2. AMBIENT OCCLUSION EVALUATION (GLSL FRAGMENT SHADER)

```
in vec2 position;
in vec3 viewRay;

// Sources
uniform sampler2D sourceDepth;
uniform sampler2D sourceNormals;

// SAT's
uniform sampler2D sourceSAT;
uniform sampler2D sourceSAT_1;
uniform sampler2D sourceSAT_11;
uniform sampler2D sourceSAT_21;

// Sample box, falloff
uniform float sampleBox_Size, sampleBox_Height;
uniform float falloff_Range;

//
uniform vec2 texelSize;

// Camera
uniform vec2 cameraClippingPlanes;
uniform vec2 cameraProjectionConstants;

// Result
out float result;

// Sample a SAT
vec4 sampleSAT(sampler2D sampler, vec2 position, vec2 texel, float area) {

  vec4 sum;

  sum  = texture(sampler, position + vec2(+texel.x, +texel.y) * area * 0.5);
  sum += texture(sampler, position + vec2(-texel.x, -texel.y) * area * 0.5);

  sum -= texture(sampler, position + vec2(+texel.x, -texel.y) * area * 0.5);
  sum -= texture(sampler, position + vec2(-texel.x, +texel.y) * area * 0.5);

  return sum;
}

void main() {

  // SAT ambient occlusion
  float viewDepth = texture(sourceDepth, position).x;
  vec3 viewPosition = viewDepth * viewRay.xyz;
  vec3 viewNormal = normalize(texture(sourceNormals, position).xyz * 2.0 - 1.0);

  // Sample the sat's
  float sampleBoxSize = sampleBox_Size / viewDepth;

  vec4 sat = sampleSAT(sourceSAT, position, texelSize,  sampleBoxSize);
  vec4 sat_1 = sampleSAT(sourceSAT_1, position, texelSize,  sampleBoxSize);
  vec4 sat_11 = sampleSAT(sourceSAT_11, position, texelSize,  sampleBoxSize);
  vec4 sat_21 = sampleSAT(sourceSAT_21, position, texelSize,  sampleBoxSize);
  // Compute remaining sat's through substraction
  vec4 sat_2 = sat - sat_1;
  vec4 sat_12 = sat_1 - sat_11;
  vec4 sat_22 = sat_2 - sat_21;

  // Compute AO
  vec2 aoNearNear = evaluateAO(sat_11, viewDepth, viewPosition, viewNormal, radius);
  vec2 aoNearFar = evaluateAO(sat_12, viewDepth, viewPosition, viewNormal, radius);
  vec2 aoFarNear = evaluateAO(sat_21, viewDepth, viewPosition, viewNormal, radius);
  vec2 aoFarFar = evaluateAO(sat_22, viewDepth, viewPosition, viewNormal, radius);

  result += (aoNearNear.x * aoNearNear.y +
             aoNearFar.x * aoNearFar.y +
             aoFarNear.x * aoFarNear.y +
             aoFarFar.x * aoFarFar.y) /
            (aoNearNear.y + aoNearFar.y + aoFarNear.y + aoFarFar.y);

  result = 1.0 - result;

  return;
}

// Evaluate ambient occlusion
vec2 evaluateAO(vec4 sat, float viewDepth, vec3 viewPosition, vec3 viewNormal, float radius) {

  vec3 samplePosition = sat.xyz / sat.w;

  float height = dot(samplePosition - viewPosition, viewNormal);

  float occlusion = clamp(height / (sampleBox_Height), 0.0, 1.0);

  float minHeight = sampleBox_Height;
  float maxHeight = sampleBox_Height + falloff_Range;

  // Falloff
  if  (height > minHeight) {
    occlusion *= 1.0 - min(1.0, (height - minHeight) / (maxHeight - minHeight));
  }

  return vec2(occlusion, sat.w);
}
```

A.3. AMBIENT OCCLUSION EVALUATION (C++/OPENGL)

```
void AdaptiveManifoldsAO:draw(DeferredFrameBuffer* frameBuffer, AmbientOcclusionBuffer* ↩
    ambientOcclusionBuffer, Camera* camera) {

  // 1. Compute sat of original depth map
  this->sat->dispatch(camera, frameBuffer->getDepthTexture());

  // 2. Split upper part from depth map into new layer
  this->splitter1->draw(camera, frameBuffer->getDepthTexture(), this->sat->getTexture());

  // 3. Compute sat of upper layer
  this->sat_1->dispatch(camera, this->splitter1->getTexture());

  // 4. Split upper part from upper and lower layer.
  this->splitter2->draw(camera, frameBuffer->getDepthTexture(), this->splitter1->getTexture(), this->sat↩
      ->getTexture(), this->sat_1->getTexture());

  // 5. Compute sat for upper layers
  this->sat_11->dispatch(camera, this->splitter2->getTexture(0));
  this->sat_21->dispatch(camera, this->splitter2->getTexture(1));

  // Ambient occlusion evaluation
  glBindFramebuffer(GL_DRAW_FRAMEBUFFER, ambientOcclusionBuffer->getIdentifier());

  glUseProgram(this->shader->getIdentifier());

  int unit = 0;

  this->shader->bindTexture("sourceDepth", frameBuffer->getDepthTexture(), unit++);
  this->shader->bindTexture("sourceNormals", frameBuffer->getNormalTexture(), unit++);

  this->shader->bindTexture("sourceSAT", this->sat->getTexture(), unit++);
  this->shader->bindTexture("sourceSAT_1", this->sat_1->getTexture(), unit++);
  this->shader->bindTexture("sourceSAT_11", this->sat_11->getTexture(), unit++);
  this->shader->bindTexture("sourceSAT_21", this->sat_21->getTexture(), unit++);

  this->shader->drawQuad();
}
```

Bibliography

[1] H. Landis, *Production-ready global illumination,* Siggraph course notes **16**, 11 (2002).

[2] M. Mittring, *Finding next gen: Cryengine 2,* in *ACM SIGGRAPH 2007 Courses*, SIGGRAPH '07 (ACM, New York, NY, USA, 2007) pp. 97–121.

[3] P. Shanmugam and O. Arikan, *Hardware accelerated ambient occlusion techniques on gpus,* in *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, I3D '07 (2007) pp. 73–80.

[4] L. Bavoil, M. Sainz, and R. Dimitrov, *Image-space horizon-based ambient occlusion,* in *ACM SIGGRAPH 2008 Talks* (2008) pp. 22:1–22:1.

[5] M. Bunnell, *Dynamic ambient occlusion and indirect lighting,* Gpu gems **2**, 223 (2005).

[6] J. Hoberock and Y. Jia, *High-quality ambient occlusion,* Gpu gems 3 (2007).

[7] M. McGuire, *Ambient occlusion volumes,* in *Proceedings of High Performance Graphics 2010* (2010).

[8] F. C. Crow, *Shadow algorithms for computer graphics,* in *ACM SIGGRAPH Computer Graphics*, Vol. 11 (ACM, 1977) pp. 242–248.

[9] D. Filion and R. McNaughton, *Effects & techniques,* in *ACM SIGGRAPH 2008 Games* (ACM, 2008) pp. 133–164.

[10] J. Kopf, M. F. Cohen, D. Lischinski, and M. Uyttendaele, *Joint bilateral upsampling,* in *ACM Transactions on Graphics (TOG)*, Vol. 26 (ACM, 2007) p. 96.

[11] C. Tomasi and R. Manduchi, *Bilateral filtering for gray and color images,* in *Computer Vision, 1998. Sixth International Conference on* (IEEE, 1998) pp. 839–846.

[12] B. J. Loos and P.-P. Sloan, *Volumetric obscurance,* in *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games* (ACM, 2010) pp. 151–156.

[13] T. Ritschel, T. Grosch, and H.-P. Seidel, *Approximating dynamic global illumination in image space,* in *Proceedings of the 2009 symposium on Interactive 3D graphics and games* (ACM, 2009) pp. 75–82.

[14] M. Slomp, T. Tamaki, and K. Kaneda, *Screen-space ambient occlusion through summed-area tables,* in *Networking and Computing (ICNC), 2010 First International Conference on* (IEEE, 2010) pp. 1–8.

[15] J. Díaz, P.-P. Vázquez, I. Navazo, and F. Duguet, *Real-time ambient occlusion and halos with summed area tables,* Computers & Graphics **34**, 337 (2010).

[16] M. Deering, S. Winner, B. Schediwy, C. Duffy, and N. Hunt, *The triangle processor and normal vector shader: a vlsi system for high performance graphics,* in *ACM SIGGRAPH Computer Graphics*, Vol. 22 (ACM, 1988) pp. 21–30.

[17] T. Saito and T. Takahashi, *Comprehensible rendering of 3-d shapes,* in *ACM SIGGRAPH Computer Graphics*, Vol. 24 (ACM, 1990) pp. 197–206.

[18] J. Klint, *Deferred rendering in leadwerks engine,* Copyright Leadwerks Corporation (2008).

[19] L. Williams, *Pyramidal parametrics,* in *ACM Siggraph Computer Graphics*, Vol. 17 (ACM, 1983) pp. 1–11.

[20] X. Décoret, *N-buffers for efficient depth map query,* in *Computer Graphics Forum*, Vol. 24 (Wiley Online Library, 2005) pp. 393–400.

[21] K. Selgrad, C. Dachsbacher, Q. Meyer, and M. Stamminger, *Filtering multi-layer shadow maps for accurate soft shadows,* in *Computer Graphics Forum* (Wiley Online Library, 2014).

[22] F. C. Crow, *Summed-area tables for texture mapping,* ACM SIGGRAPH computer graphics **18**, 207 (1984).

[23] J. Hensley, T. Scheuermann, G. Coombe, M. Singh, and A. Lastra, *Fast summed-area table generation and its applications,* in *Computer Graphics Forum*, Vol. 24 (Wiley Online Library, 2005) pp. 547–555.

[24] M. Harris, S. Sengupta, and J. D. Owens, *Parallel prefix sum (scan) with cuda,* Gpu gems 3 (2007).

[25] W. H. De Boer, *Fast terrain rendering using geometrical mipmapping,* Unpublished paper, available at http://www. flipcode. com/articles/article geomipmaps. pdf (2000).

[26] E. S. Gastal and M. M. Oliveira, *Adaptive manifolds for real-time high-dimensional filtering,* ACM Transactions on Graphics (TOG) **31**, 33 (2012).

[27] M. Schwarz and M. Stamminger, *Quality scalability of soft shadow mapping,* in *Graphics Interface 2008* (2008) pp. 147–154.

[28] G. Sellers, R. Wright, and N. Haemel, *OpenGL SuperBible: Comprehensive Tutorial and Reference*, sixth ed. (Addison-Wesley Professional, 2013).