Smooth Probabilistic Ambient Occlusion for Volume Rendering

Thomas Kroes, Dirk Schut, and Elmar Eisemann

1.1 Introduction

Ambient occlusion [Zhukov et al. 98] is a compelling approach to improve depth and shape perception [Lindemann and Ropinski 11, Langer and Bülthoff 99], to give the illusion of global illumination, and it is an efficient way to approximate low-frequency outdoor lighting. In principle, ambient occlusion computes the light accessibility of a point, i.e., it measures how much a point is exposed to its surrounding environment.

An efficient and often used version of ambient occlusion is Screen Space Ambient Occlusion [Kajalin 09]. It uses the depth buffer to compute an approximate visibility. This method is very appealing, since the computational overhead of this method is minimal. However it cannot be applied to direct volume rendering (DVR) because voxels are typically semi-transparent (defined via a transfer function). Consequently, a depth buffer would be ambiguous and is not useful in this context.

The first method to compute ambient occlusion in direct DVR, called *Vicinity Shading*, was developed by Steward [Stewart 03]. This method computes the ambient occlusion in each voxel by taking into account how much the neighboring voxels obscure it. The resulting illumination is stored in an additional volume, which needs to be recomputed after each scene modification. Similarly, Hernell et al. [Hernell et al. 10] compute ambient occlusion by raytracing inside a small neighborhood around the voxel. Kroes et al. extend this method by taking the entire volume into account [Kroes et al. 12].

Our approach tries to avoid costly ray tracing and casts the problem into a filtering process. In this sense, it is similar in spirit to Penner et al. [Penner and Mitchell 08], who use statistical information about the neighborhood of the voxels to estimate ambient occlusion, as well as the method by Ropinski et al. which is similar and also adds color bleed-

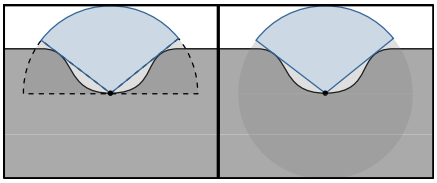


Figure 1.1: Left: The hemisphere around a point which determines ambient occlusion. The blue part is unoccluded. Right: Volumetric obscurance relies on a full sphere.

ing [Ropinski et al. 08]. Furthermore, our approach relates to Crassin et al. [Crassin et al. 10] who propose to use filtering for shadow and out-of-focus computations.

Our Smooth Probabilistic Ambient Occlusion (SPAO) is a novel and easy-to-implement solution for ambient occlusion in DVR. Instead of applying costly ray casting to determine the accessibility of a voxel, this technique employs a probabilistic heuristic in concert with 3D image filtering. In this way, ambient occlusion can be efficiently approximated and it is possible to interactively modify the transfer function, which is critical in many applications, such as medical and scientific DVR. Furthermore, our method offers various quality trade-offs regarding memory, performance, and visual quality. Only very few texture look-ups are needed in comparison to ray-casting solutions and the interpretation as a filtering process ensures a noise-free and smooth appearance.

1.2 Smooth Probabilistic Ambient Occlusion

There are various definitions for ambient occlusion. Here, we define it as the part of a point that is accessible from the outside world. A 2D example is given in Figure 1.1 and illustrates the ambient occlusion computation. More formally the ambient-occlusion value $\mathcal{A}(p, n)$ is given by the integral of the visibility function over the hemisphere Ω centered around a point p in the direction of the normal n of that point:

$$\mathcal{A}(p, n) := \frac{1}{\pi} \int_{\Omega(n)} V(p, \omega) d\omega,$$

where V is the visibility function. In other words, V stems from the volume data itself after it was transformed by the transfer function. $V(p, \omega)$ is 0 if the ray from point p in direction ω is blocked, 1 if it is unblocked, and an intermediate value attenuates the ray. To simplify the description, we will only use the notion of blocked and unblocked rays in the following. Please

notice that we can interpret intermediate values of V as a probability for a ray to be blocked. For example, if V returns a value of 0.5, there is a 50% chance for a ray to be blocked.

It is also possible to integrate the visibility function over the whole sphere around a point, making Ω a full sphere, instead of a hemisphere, and making it independent of n. The result is called obscurance and denoted $\mathcal{A}(p)$, and produces similar effects. Calculating obscurance instead of ambient occlusion has the advantage that it does not require a normal. However, this definition will lead to parts of the volume that are located behind the point to intervene in the computation. This property can be a disadvantage for standard scenes, as the result might become too dark, but in the context of DVR, it is sometimes even preferable, as it will unveil information below the surface, which is often desired.

Both, ambient occlusion and obscurance, only depend on the geometry of the volume. Therefore they can be stored in an additional volume that is then used to modulate the original volume's illumination. The occlusion values can be calculated directly from the opacity of the original volume. Nonetheless, the values have to be recomputed when the original volume changes, for example when the user changes the transfer function. This latter step can be very costly and makes it impossible to interact with transfer functions, while maintaining a high visual fidelity. Our approach is fast to compute and enables a user to quickly apply such modifications without having to wait a long time for the result.

Initially, our solution will be explained in the context of obscurance, but in Section 1.3, we will extend our algorithm to approach ambient occlusion by making use of the normals to reduce the influence of the part of the volume below the surface.

1.2.1 Overview

To approximate obscurance at a certain point in the volume, we avoid ray casting. Instead, we introduce an approximation, which is based on the probability of the rays being blocked by the volume. Instead of solving $\mathcal{A}(p)$ and its integral entirely, we consider a limited region around p, formed by volumes of increasing size. The volume between successive volumes forms a layer of voxels, a so-called shell (Figure 1.2). We will show how to derive the probability of a random ray to be blocked by a shell. From this result, we deduce an approximation of the integral $\mathcal{A}(p)$ assuming that the entire volume is represented by a single shell. Finally, the results for these various shells are combined heuristically to yield our occlusion approximation for the entire volume.

First, we consider shells being represented by a sphere with a one-voxel-wide boundary S. These shells are formed by a set of successive spheres,

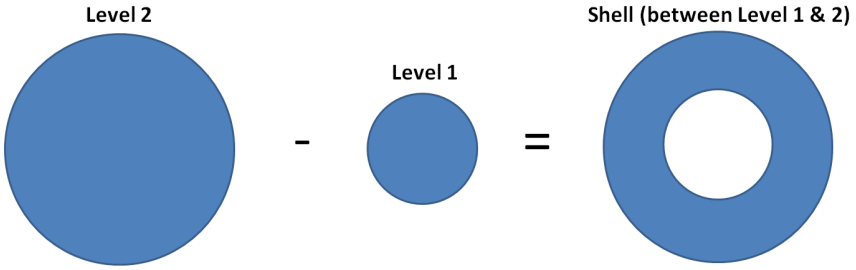


Figure 1.2: A shell is a layer of voxels formed by the difference between two differently-sized volumes. By creating a cascade of these volumes, a set of shells is formed. For each shell, we approximate the probability of a ray to be blocked and combine these probabilities heuristically to form the final obscurance value.

which each grow in radius by one voxel. In this situation, if we consider one independent shell, any random ray sent from its center will intersect exactly one voxel. If all directions are equally likely the probability for a ray to be blocked then boils down to an average of all voxel values in the shell $Average_S(p)$. Looking carefully at this definition, it turns out that this probability is equivalent to solving for $\mathcal{A}$ in the presence of a single shell.

If we now decompose the volume into such a set of shells around a point, we can compute the probability of the rays to be blocked by each shell, but still need to combine all these blocking contributions together. In order to do so, we make use of a heuristic. We assume a statistical independence between the value distributions in the various shells. The probability of rays originating at p to be blocked by a set of n englobing shells $\{S_i\}_{i=1}^n$ ordered from small to large is then given by:

$$\prod_{i=1}^{n}(1 - Average_{S_i}(p)).$$

To understand this formula, it helps considering only two layers $\{S_1, S_2\}$. A random ray from p traverses S_1 with probability $(1 - Average_{S_1}(p))$. If this ray passed S_1, it is again, potentially, stopped by S_2, this time with probability $(1 - Average_{S_2}(p))$, yielding a total probability of $(1 - Average_{S_1}(p))(1 - Average_{S_2}(p))$. In the following, we will describe an efficient and GPU-friendly approach to compute an approximation of this solution.

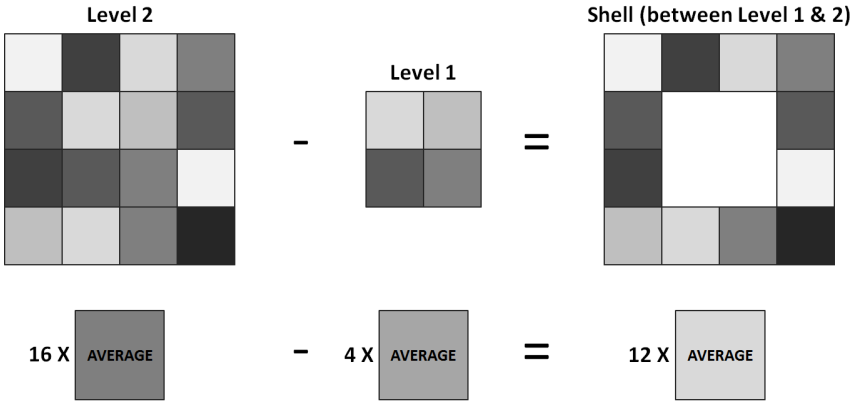


Figure 1.3: In this 2D illustration, the shell on the right is a one voxel thick hull that is formed by subtracting the average opacity from level 1 in the middle from level 2 on the left.

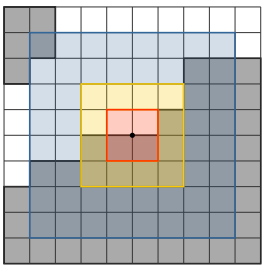


Figure 1.4: Cube shells used to approximate obscurance.

1.2.2 Approximating Obscurance for Cube Shells

In practice, we will use box-shaped shells instead of spheres (Figure 1.3). We will show in the next section that this choice will allow us to benefit from GPU texture filtering to compute $Average_{S_i}$, making the algorithm very efficient. The cubes are chosen to be of increasing size and centered at each point p of the volume. The shells are then defined by hollowing out these cubes by subtracting the next-smaller cube from its successor. In reality, these cubes will never have to be explicitly constructed, but it is helpful to think of them for illustrative purposes. The process is illustrated in Figure 1.4.

Following the previously described steps, we need to deduce $Average_{S_i}$ for each of the shells, which in our new situation corresponds to the average

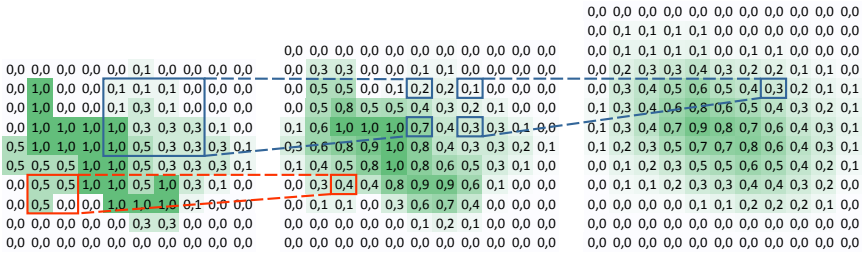


Figure 1.5: 2D Example of how N-buffers are calculated. A dataset is shown on the left, with the first two n buffer levels next to it. In each level the average of four values of the previous level is combined into one value.

of all voxel values between two successive cubes. If we assume for now that we have a quick way of determining the average inside of a complete cube, we can rapidly determine $Average_{S_i}$. To illustrate this computation, we will assume that we want to determine $Average_S$ of a shell S defined two cubes C_1 and C_2, with voxel-value averages A_1 and A_2 and number of voxels S_1, S_2 ($S_1 < S_2$), respectively. The solution is then given by:

$$Average_S = (S_2A_2 - S_1A_1)/(S_2 - S_1)$$

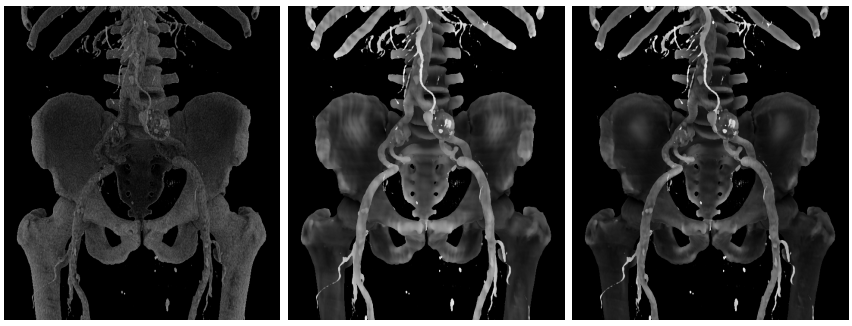
In other words, we can subtract from the total voxel sum of one cube S_2A_2, the total voxel sum of the next-smaller one (S_2A_2) and normalize the result by the number of voxels in the shell between both (Figure 1.3, lower row).

Please notice, that the above formula can be rewritten to: $Average_S = 1/(1 - S_1/S_2)(A_2 - (S_1/S_2)A1)$ - in consequence, only the average and the relative change in size (S_1/S_2) is needed to deduce $Average(S)$, which facilitates computations further. Imagine, each cube is obtained by doubling the length of each edge of the predecessor. Then the ratio would be $1 : 8$, resulting in $Average_S = 8/7(A_2 - 1/8\ A_1)$.

1.2.3 Fast Cube Averages

In the previous section, we assumed to have a quick method to determine the average inside of a cube. Here, we will propose two possible solutions to this problem. Our observation is that, for a given cube size, the averages are equivalent to a box-filtering of the volume.

Determining averages of various kernel sizes is a common problem in computer graphics in the context of texture mapping. These techniques translate to corresponding operations in a 3D volume. The most common such approximation is mipmapping, but we will also present N-Buffers [Décoret 05], which deliver higher quality filtering at an additional cost.



(a) (b) (c)

Figure 1.6: Volumetric obscurance using a) raytacing (256 rays/voxel) b) Mipmap filtering c) N-buffer filtering

As mipmaps are rather standard, we will only focus on N-buffers here. Like mipmaps, they consist of multiple levels l, each representing the average values of the original volume inside cubes of width 2^l. Unlike mipmaps, the resolution of N-buffers is not reduced in each level. In consequence, it is possible to retrieve the exact filled part of a cube at every position in the volume, whereas for a mipmap linear interpolation can provide only an approximation based on the eight closest voxels, which reduces the quality (Figure 1.6).

The N-Buffer construction is efficient, as each new level can be computed from the previous using only 8 lookups. A 2D example of the calculation is shown in Figure 1.5. Nonetheless, N-Buffers result in higher memory consumption, so it can be useful to apply a few mipmap levels before processing the rest using N-Buffers.

1.3 Approximating Ambient Occlusion

In Section 1.2, we explained that ambient occlusion in comparison with obscurance can provide cues that are closer to realistic lighting because voxels behind the point of interest are not taken into account. To reduce this effect, we can offset the lookup operations in the direction of the normal. When choosing the offset carefully, the increase in size of the cubes and the offset can be correlated to obtain shells that correspond now to hemispheres. This goal can be achieved by multiplying the normal vector by half the size of the box. An example with a shorter vector is illustrated in Figure 1.7.

However, in DVR a normal is not always clearly defined, e.g., inside

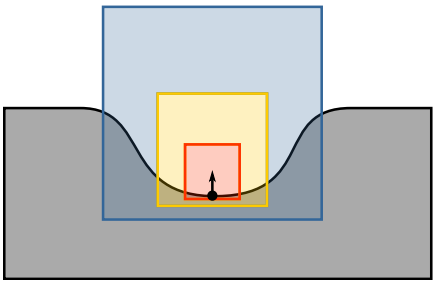


Figure 1.7: The lookups of the cubes from a point with a normal of length 0.75 in the upwards direction.

(a) Factor 0.0 (No normal correction) (b) Factor 0.5 (c) Factor 1.0 (d) Factor 2.0

Figure 1.8: Effect of the normal factor

a homogeneous semi-transparent volume, e.g., a jelly pudding. Similarly, between two different semi-transparent voxels it might be less clear how to define a normal at the interface between opaque and transparent materials. In consequence, we propose to scale the cube offset based on how strong the gradient is. Interestingly, while most techniques derive normals from the normalized gradient via central differences, we can use the gradient magnitude to determine if a normal is clearly defined. Hence, we propose to remove the normalization operation and instead normalize the voxel values themselves to the range [0,1], which will lead to the gradient becoming an appropriately-scaled normal. Additionally, we allow the user to specify a global scale to either pronounce or reduce the impact of this ambient-occlusion approximation (Figure 1.8).

	N-buffers	Mipmaps	Raytaced, 512 rays
Level 0	30.93 ms	33.00 ms	-
Level 1	33.99 ms	4.58 ms	-
Level 2	40.13 ms	0.66 ms	-
Level 3	41.16 ms	0.17 ms	-
Level 4	42.69 ms	0.14 ms	-
Level 5	38.09 ms	0.13 ms	-
Level 6	41.91 ms	0.12 ms	-
Levels Total	268.90 ms	38.8 ms	-
AO Computation	63.24 ms	110.39 ms	425.36 sec
Total	332.14 ms	149.19 ms	425.36 sec

Table 1.1: Performance measurements for the Macoessix data set (512 x 512 x 512) for N-buffers and mipmaps based SPAO. For each technique we show the time it takes to compute the individual levels and to combine them into an ambient occlusion volume.

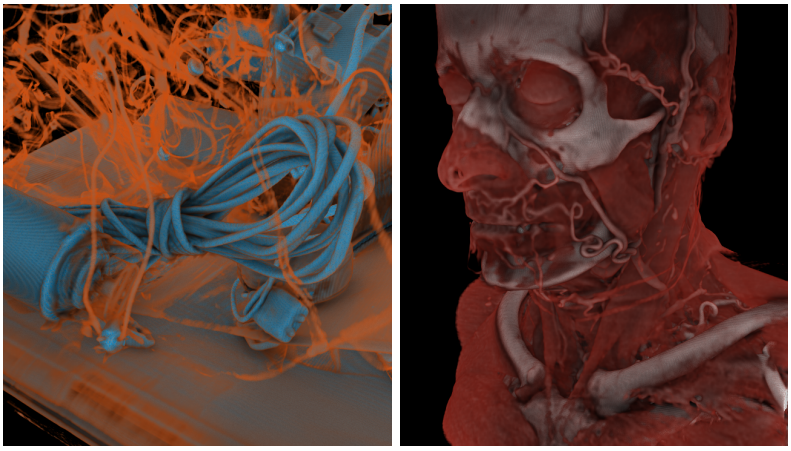
1.4 Results

Our method has been implemented in a CUDA-based stand-alone software program for direct DVR. The program and its source code are available under the original BSD license. It is shipped with sample data sets. The transfer function and, thus, the visual representation can be changed on-the-fly. Also, the user can select from three different methods of ambient occlusion computation: mipmaps, N-buffers, and ray tracing. Our program makes use of CUDA 3.0 texture objects and will not support lower CUDA versions.

We tested the performance of our technique using the publicly available Macoessix data set from the Osirix Website [1], see Table 1.1. All tests were peformed on an Intel(r) Xeon(r) W3530 (2.80 GHz) workstation with 12 GB RAM and a GeForce GTX TITAN Graphics Card with 4 GB of RAM. N-buffers are slightly more costly than Mipmaps, but both are orders of magnitude faster than a volumetric ambient-occlusion ray tracer. The latter takes more than four minutes, see Table 1.1.

Figure 1.9 shows some results of our approach on the Backpack and Manix data sets.

[1] http://www.osirix-viewer.com/datasets/



(a) Backpack data set (b) Manix data set

Figure 1.9: SPAO applied to the Backpack (512 x 512 x 461) and Manix (512 x 512 x 460) data sets.

1.5 Conclusion

This chapter presents a novel approach to compute ambient occlusion for direct DVR. We demonstrate that, by considering the ambient-occlusion computation as a filtering process, we can significantly improve efficiency and make it usable in a real-time DVR application. Such an approach is useful for medical visualization applications were transfer functions are very often subject to change.

Our approach is simple to implement and efficient and leads to a very good quality/performance tradeoff. Nonetheless, we also experimented with more complex combinations of the shells, especially, as the assumption of independence of the occlusion probabilities is usually not true in most data sets. In practice, it turns out that our solution seems to be a good choice and any increase in complexity also led to a significant performance impact. Nonetheless, this topic remains interesting future work. Further, we would like to investigate to approximate physically-plausible light transport, such as global illumination, with our filtering technique, which could further enhance the volume depiction.

Bibliography

[Crassin et al. 10] Cyril Crassin, Fabrice Neyret, Miguel Sainz, and Elmar Eisemann. *GPU Pro*, Chapter X.3 Efficient Rendering of Highly Detailed Volumetric Scenes with GigaVoxels, pp. 643–676. AK Peters, 2010.

[Décoret 05] Xavier Décoret. "N-Buffers for efficient depth map query.", 2005. Available online (http://maverick.inria.fr/Publications/2005/Dec05).

[Hernell et al. 10] Frida Hernell, Patric Ljung, and Anders Ynnerman. "Local ambient occlusion in direct volume rendering.", 2010.

[Kajalin 09] Vladimir Kajalin. "Screen Space Ambient Occlusion." In *ShaderX7*, edited by Wolfgang Engel. Boston: Cengage Learning, 2009.

[Kroes et al. 12] Thomas Kroes, Frits H Post, and Charl P Botha. "Exposure render: An interactive photo-realistic volume rendering framework.", 2012.

[Langer and Bülthoff 99] Michael S Langer and Heinrich H Bülthoff. "Depth discrimination from shading under diffuse lighting.", 1999.

[Lindemann and Ropinski 11] Florian Lindemann and Timo Ropinski. "About the Influence of Illumination Models on Image Comprehension in Direct Volume Rendering.", 2011.

[Penner and Mitchell 08] Eric Penner and Ross Mitchell. "Isosurface ambient occlusion and soft shadows with filterable occlusion maps.", 2008.

[Ropinski et al. 08] Timo Ropinski, Jennis Meyer-Spradow, Stefan Diepenbrock, Jörg Mensmann, and Klaus Hinrichs. "Interactive volume rendering with dynamic ambient occlusion and color bleeding.", 2008.

[Stewart 03] A James Stewart. "Vicinity shading for enhanced perception of volumetric data.", 2003.

[Zhukov et al. 98] Sergey Zhukov, Andrei Iones, and Grigorij Kronin. "An ambient light illumination model.", 1998.