TUCS

Ville Timonen

Scalable Algorithms for Height Field Illumination

TURKU CENTRE *for* COMPUTER SCIENCE

TUCS Dissertations

No 173, April 2014

Scalable Algorithms for Height Field Illumination

Ville Timonen

To be presented, with the permission of the Department of Information Technologies at Åbo Akademi University, for public criticism in Auditorium Lambda on April 11, 2014, at 12 noon.

Åbo Akademi University
Department of Information Technologies
Joukahaisenkatu 3-5 A, 20520 Turku

2014

Supervisors

Jan Westerholm
Department of Information Technologies
Åbo Akademi University
Joukahaisenkatu 3-5 A, 20520 Turku
Finland

Jukka Arvo
Unity Technologies
Vendersgade 28, DK-1363 Copenhagen
Denmark

Reviewers

Elmar Eisemann
Department of Intelligent Systems
Delft University of Technology
Mekelweg 4, 2628 CD, Delft
Netherlands

Ulf Assarsson
Department of Computer Science and Engineering
Chalmers University of Technology
S-412 96, Gothenburg
Sweden

Opponent

Ulf Assarsson
Department of Computer Science and Engineering
Chalmers University of Technology
S-412 96, Gothenburg
Sweden

ISBN 978-952-12-3036-3
ISSN 1239-1883

Abstract

Global illumination algorithms are at the center of realistic image synthesis and account for non-trivial light transport and occlusion within scenes, such as indirect illumination, ambient occlusion, and environment lighting. Their computationally most difficult part is determining light source visibility at each visible scene point. Height fields, on the other hand, constitute an important special case of geometry and are mainly used to describe certain types of objects such as terrains and to map detailed geometry onto object surfaces. The geometry of an entire scene can also be approximated by treating the distance values of its camera projection as a screen-space height field.

In order to shadow height fields from environment lights a horizon map is usually used to occlude incident light. We reduce the per-receiver time complexity of generating the horizon map on $N \times N$ height fields from $O(N)$ of the previous work to $O(1)$ by using an algorithm that incrementally traverses the height field and reuses the information already gathered along the path of traversal. We also propose an accurate method to integrate the incident light within the limits given by the horizon map. Indirect illumination in height fields requires information about which other points are visible to each height field point. We present an algorithm to determine this intervisibility in a time complexity that matches the space complexity of the produced visibility information, which is in contrast to previous methods which scale in the height field size. As a result the amount of computation is reduced by two orders of magnitude in common use cases.

Screen-space ambient obscurance methods approximate ambient obscurance from the depth buffer geometry and have been widely adopted by contemporary real-time applications. They work by sampling the screen-space geometry around each receiver point but have been previously limited to near-field effects because sampling a large radius quickly exceeds the render time budget. We present an algorithm that reduces the quadratic per-pixel complexity of previous methods to a linear complexity by line sweeping over the depth buffer and maintaining an internal representation of the processed geometry from which occluders can be efficiently queried. Another algorithm is presented to determine ambient obscurance from the entire depth buffer at each screen pixel. The algorithm scans the depth buffer in a quick pre-pass and locates important features in it, which are then used to evaluate the ambient obscurance integral accurately. We also propose an evaluation of the integral such that results within a few percent of the ray traced screen-space reference are obtained at real-time render times.

Sammanfattning

Globala belysningsalgoritmer beskriver hur ljuset framskrider bland datorgenererade objekt i en scen, omfattande bland annat indirekt belysning, skuggor, blockering av ambient ljus samt belysning från omgivningen. Dessa algoritmer spelar en central roll då man önskar generera realistiska syntetiska bilder. Den beräkningsmässigt tyngsta delen i dessa algoritmer består av att man skilt för varje synlig punkt i scenen måste avgöra om en ljuskälla är synlig i denna punkt eller inte. Höjdkartor utgör å andra sidan ett viktigt specialfall av geometrier som huvudsakligen används för att beskriva vissa typer av objekt såsom terräng eller för att avbilda detaljerade ytor på objekt. Geometrin för en hel scen kan också approximeras genom att behandla avstånden inom kameraprojektionen som en höjdkarta i skärmrummet.

För att beräkna skuggor i höjdkartor använder man ofta en horisontsilhuett för att blockera inkommande direkt ljus. I detta arbete minskar vi komplexiteten för att beräkna horisontsilhuetten per mottagande punkt i en höjdkarta med dimensionen $N \times N$ från $O(N)$ i tidigare arbeten till $O(1)$ genom att utnyttja en algoritm som går igenom höjdkartan inkrementellt längs en linje och återanvänder den redan insamlade informationen längs linjen. Vi förslår också en exakt metod för att integrera den inkommande belysningen för horisontsilhuetter. Den indirekta belysningen i höjdkartor kräver information om vilka punkter i höjdkartan som syns från en given punkt. Vi presenterar en algoritm för att bestämma denna intervisibilitet med en tidskomplexitet som sammanfaller med rumskomplexiteten för visibilitetsinformationen, i kontrast till tidigare metoder vars tidskomplexitet växer enligt antaler punkter i höjdkartan. Detta resulterar i att mängden beräkningar i normala tillämpningar minskar hundrafalt.

Kända metoder för att beräkna ambient blockering i skärmrummet approximerar skuggningen från djupkartans geometri och de används allmänt inom moderna realtidstillämpningar. Metoderna baserar sig på att sampla skärmrummets geometri runt varje mottagarpunkt men de har hittills begränsats till närområden eftersom sampling från större områden snabbt blir för tidskrävande. Vi presenterar en algoritm som minskar den kvadratiska tidskomplexiteten per punkt hos tidigare metoder till en linjär tidskomplexitet genom att gå över djupkartan längs räta linjer och upprätthålla en intern representation av den redan behandlade geometrin ur vilken blockerande geometri kan bestämmas effektivt. Vi presenterar också en algoritm för att beräkna ambient blockering från hela djupkartan i varje skärmpunkt. Denna algoritm går igenom djupkartan via en snabb förbehandling som identifierar viktiga drag i djupkartan, och dessa drag används sedan för att beräkna den ambienta blockeringens integral exakt. Vi framlägger också en metod att beräkna integralen i realtid vars resultat sammanfaller med några procents noggrannhet med resultaten från den kända strålgångsmetoden tillämpad på skärmrummet.

Acknowledgements

First and foremost I would like to express my gratitude to my thesis supervisor Prof. Jan Westerholm for teaching me academic discourse and for his inspiring support. I would like to thank Dr. Jukka Arvo for helping me with the manuscripts and Prof. Elmar Eisemann and Assoc. Prof. Ulf Assarsson for reviewing this thesis and for providing me with positive and constructive feedback. I would also like to extend my thanks to Dr. Samuli Laine for helping me with the final list of reviewers.

Finally I am grateful to my family for their reassuring support and I would like to thank my friends and #muropaketti for their encouragement during my studies.

List of original publications

[P1] Ville Timonen and Jan Westerholm. Scalable Height Field Self-Shadowing. *Computer Graphics Forum*, 29(2), pages 723–731, 2010. *Eurographics Conference 2010.* 3rd Best Paper.

[P2] Ville Timonen. Low-Complexity Intervisibility in Height Fields. *Computer Graphics Forum*, 31(8), pages 2348–2362, 2012. Invited to *Eurographics Conference 2013.*

[P3] Ville Timonen. Line-Sweep Ambient Obscurance. *Computer Graphics Forum*, 32(4), pages 97–105, 2013. *Eurographics Symposium on Rendering 2013.* Best Student Paper.

[P4] Ville Timonen. Screen-Space Far-Field Ambient Obscurance. In Proceedings of the *High Performance Graphics 2013*, pages 33–43, ACM.

List of abbreviations

2D	Two-dimensional
3D	Three-dimensional
CPU	Central processing unit
GPU	Graphics processing unit
GPGPU	General-purpose computing on graphics processing units
GI	Global illumination
AO	Ambient obscurance
SSAO	Screen-space ambient obscurance
HBAO	Horizon-based ambient occlusion

Contents

Part I

Research summary

Chapter 1

Introduction

Computer graphics as a field of computer science, and in particular *rendering*, is about image synthesis—creating images using computers. These computer generated images (CGI) are commonplace in our lives: Most features films today incorporate CGI effects as parts of scenes or have entire scenes digitally rendered; video games on smartphones, gaming consoles, and personal computers boast increasingly realistic graphics; visualization of medical and scientific data and complex datasets is important as the human brain is adjusted to understanding data through visualization; architecture and interior design require physically accurate visual previewing; and even still images found in magazines and newspapers are often rendered or manipulated digitally. New fields that increase in popularity, such as virtual and augmented reality, also depend heavily on high quality rendering to make the user experience plausible. And the future will most probably see many new applications of computer graphics.

The main challenges in computer graphics have been in *photorealistic rendering* where the goal is to is to synthesize graphics that are indistinguishable from the real world. Photorealistic rendering is important for feature films and video games as their immersiveness depends on plausibly convincing the user that the created environment could be real. An even higher level of realism is required for computer-aided architecture and interior design because the pre-visualization needs not only be plausible, but also physically correct to accurately represent the actual deliverable. The same requirements also apply, to some extent, to augmented reality type applications where one places virtual objects, such as furniture, into an image of a real scene. The appearance of the virtual objects needs to adjust to the environment in a physically correct fashion.

Photorealistic rendering involves modeling at least the geometry of the rendered objects, the material of the object surfaces, and light sources. After light has been emitted by the light sources, it is transported and bounces around in the scene before finally reaching the eye and forming an image. Upon contact with an object, the object's surface properties define how the light is modulated before being transmitted forward. As a function of wavelength (i.e. color of the light), light can be absorbed, reflected, or refracted. Although many more seemingly distinct effects of illumination can be observed in real life, they are for the most part result of these three basic building blocks. While there are other optical phenomena—such as diffraction, Rayleigh scattering, and quantum effects—they do not play a large role in how the everyday world looks like apart from special situations. Therefore the behavior of light is very well understood.

However, synthesizing photorealistic scenes is far from a solved problem for the simple reason that an accurate simulation of light is computationally extremely complex. For example, rendering a single frame in a feature film may take up to a day on one computer, and still several levels of approximations for speeding up rendering have to be made, such as tailored algorithms and simplified models for surface materials and object geometry. The amount of computation required per frame is also steadily increasing because still not all artistic visions can be realized: there is a demand for larger scenes, higher detail, and more exotic environments.

Real-time and interactive applications are much more computationally constrained as up to 60 images have to be rendered each second on commodity hardware. Therefore, there is a lot of work left to be done in the field of computer graphics. Currently, one major part in rendering research is to determine fast and robust methods to simulate light transport throughout the scene. In order to determine the appearance of a point in a scene, incoming light from primary light sources (*direct illumination*) and reflected and refracted light from other objects (*indirect illumination*) has to be accounted for. *Global illumination* (GI) [23] is the term used to describe a system that attempts to account for the full scene lighting, or at the very least both direct and reflected indirect light. The most computationally complex part in global illumination is visibility determination of both the primary light sources and other objects in the scene.

This work, overviewed at the end of this chapter in Section 1.5 and presented in more detail in Chapters 3 and 4, contributes to efficient and scalable visibility and illumination algorithms for height field and screen-space geometry. This geometry is described and defined in Section 1.3. An introduction to global illumination is given in Section 1.1 followed by a focused introduction to one GI effect in particular: ambient occlusion in Section 1.2. General-purpose languages for GPUs have allowed an efficient implementation of our algorithms and these are described in Section 1.4. Chapter 5 concludes this thesis with a summary and a discussion about future work.

1.1 Global illumination

As described in the previous section, upon contact with a surface light can be absorbed, reflected, or refracted. While the interaction might sound simple, complex illumination effects emerge. The basic reflection effects include hard shadows, which appear when a surface is not in direct view of a point light source, and soft shadows, which appear when a surface is partially visible from an area light source. Scattering effects such as crepuscular rays and subsurface scattering occur when an otherwise transparent volume, such as air, is occupied by tiny reflective particles which cause light to unpredictably change direction and eventually end up in the eye even if the light source is not directly visible.

Practical objects are never fully absorptive but instead always reflect some light. Therefore objects themselves become light sources when struck by light. This type of light is called secondary or indirect light, and reflection effects for this type of light include ambient and indirect illumination (from diffuse surfaces) and reflections (from glossy surfaces). Sometimes indirect illumination is further divided into global effects and local effects such as color bleeding.

Refraction changes the direction of light, and refraction effects include morphing of the view as seen through a medium with changes in its optical density. This effect can be seen for instance in glassware, water, and hot air leaving the surface of a highway. Another type of refraction effects are *caustics* which is the focusing of light that passes through an optically varying object, such as a glass, into patterns on a surface.

Using a single rendering strategy to simulate light transport with high enough fidelity such that *all* of the above illumination effects can be faithfully produced, would currently be computationally too expensive. Therefore, different rendering methods are generally used to approximate different illumination effects, especially in real-time applications where each rendering method is strongly tailored to reproduce a small subset of illumination effects due to strict execution time constraints.

In this thesis, we focus on direct and indirect reflected illumination on solid—mostly diffuse—objects. This type of light constitutes the significant majority of the visible world and is therefore very important to solve accurately. Shadowing of direct light is important for creating scenes that look realistic and to convey the shape, location, and movement of objects [88] [44] [51]. Specifically, hard shadows convey the shape of an object whereas soft shadows [34] convey distance: the shadow from an area light source gradually spreads and softens as the distance from the caster to the receiver increases. Shadows also convey information about the environment: where the light sources are located and how large and intensive they are. Indirect illumination further adds to realism, without which scenes often look artificial or dull. It also allows shadowed parts of the scene to be lit that otherwise would be dark. Indirect illumination also conveys information about the neighborhood of a given point indicating whether it is mostly occluded by nearby objects or in the open. Figure 1.1 demonstrates direct and indirect illumination on a height field lit by a point light source.

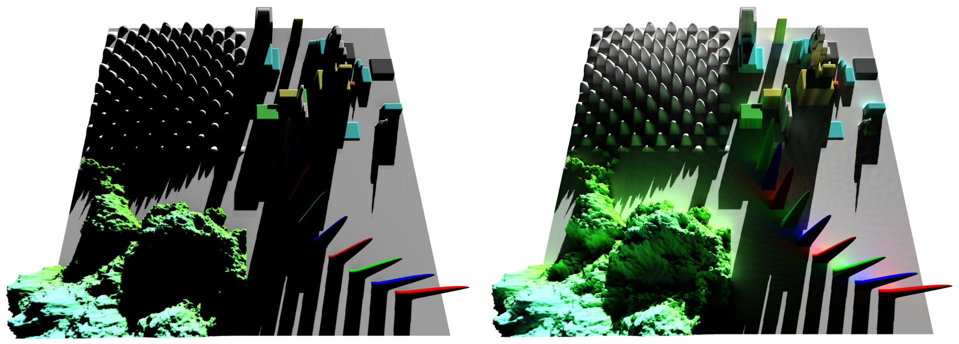


Figure 1.1: Left: direct illumination from a point light source on a 1024^2 diffuse height field rendered using the algorithm in [P1]. Right: an additional indirect light bounce rendered using the algorithm in [P2].

Most illumination effects can be described using a 2-dimensional integral at the receiver point. This integral is called *the rendering equation* [40] and it describes how a point looks like when viewed from a given direction ω_o. The equation accounts for light reflected from and emitted by the surface point $\mathbf{x}$ with a normal $\vec{n}$ as a function of the wavelength of light λ:

$$L_o(\mathbf{x}, \vec{n}, \omega_o, \lambda) = L_e(\mathbf{x}, \omega_o, \lambda) + \int_\Omega L_i(\mathbf{x}, \omega_i, \lambda) f_r(\mathbf{x}, \omega_i, \omega_o)(\vec{n} \cdot \omega_i) d\omega_i, \quad (1.1)$$

where L_e is the emitted radiance of point $\mathbf{x}$, Ω is the normal-oriented unit hemisphere around $\mathbf{x}$, L_i is the incident light at point $\mathbf{x}$ from direction ω_i and f_r is the *bi-directional reflectance distribution function* (BRDF). BRDF describes how the surface reflects light coming from direction ω_i into the viewing direction ω_o. For perfectly diffuse surfaces the value of BRDF is constant as a function of λ. This formulation is very useful because it implies a straight-forward way to solve a point's appearance: The integral can be numerically evaluated by casting rays from the receiver point $\mathbf{x}$ in the direction ω_i and finding the nearest intersection of this ray with the scene geometry—or a light source—and sampling the emitted radiance at the intersection point. This approach is called *path tracing* and it was published simultaneously with the rendering equation in [40]. Visibility determination, i.e. determining which point is the nearest along the ray direction is the most computationally challenging part of the evaluation of the rendering equation.

When only direct light is accounted for, it is not necessary to integrate across the full hemisphere but only solve the visibility of the light sources. The simplest and easiest case of this type is the point light source, where one visibility query is sufficient. Area light sources can be partially visible and require a more exhaustive visibility search to determine the visible area of the light source. As the most general case direct light can also be integrated from an environment light map where the incident light is defined as a function of direction. Obtaining the integration limits for this integral requires determining the full visibility of the environment, making it the most computationally difficult form of direct lighting. In case the geometry can be expressed as a height field, visibility can be described as *a horizon map.* A horizon map stores the horizon angle as a function of azimuthal direction for each height field point. In order to generate this data previous state-of-the-art methods require, at each height field point, an exhaustive search that is dependent on the height field size. In [P1] we propose a method to generate the horizon map in constant time at each receiver point, regardless of the size of the height field.

For perfectly diffuse surfaces, Equation 1.1 can alternatively be written as:

$$L_o(\mathbf{x},\lambda) = L_e(\mathbf{x},\lambda) + f_r(\mathbf{x},\lambda)\int_S L_o(\mathbf{x}',\lambda)G(\mathbf{x},\mathbf{x}')V(\mathbf{x},\mathbf{x}')dA' \qquad (1.2)$$

where instead of integrating over the hemisphere at each point, the integral is taken over all scene surfaces A' and the binary visibility function V determines whether the surface point $\mathbf{x}'$ is visible to $\mathbf{x}$ or not. This integral can be numerically evaluated by dividing the scene surfaces into finite size patches and iterating over the patches and weighing them according to their size as seen from the receiver point according to the geometrical form factor G. When each patch is considered as a receiver and the incoming radiance is assumed to be transferred uniformly in every direction (which is to say that surfaces are assumed to be perfectly diffuse), the approach is called *radiosity* [30]. The significant downside of this approach is that all surface patches are traversed through instead of only the visible ones. We show in [P2] that for height fields, on average, only a small fraction of geometry is generally visible. Furthermore visibility scales strikingly sublinearly when the geometry is expanded. The method we propose in [P2] improves the previous state-of-the-art, where all height field points need to be traversed to find the visible ones, by allowing to traverse only the visible parts of the height field, which reduces the amount of computation required by two orders of magnitude in common use cases.

1.2 Ambient occlusion

There is one method relevant to our work which tries to simulate an important subset of the difficult-to-calculate indirect diffuse illumination effects: *ambient occlusion.* Ambient occlusion assumes that there is a uniform environment light source in the scene, and blocks the light reaching the receiving point by considering the surrounding scene geometry.

The strong visual cues about the shape and surroundings of an object that arise from ambient lighting were first noted by [49]. Then a few years later *obscuring* ambient light was used as a shading method to render computer graphics in [92]. *Ambient obscurance* generalizes ambient occlusion by introducing a falloff function $\rho(d)$ which weighs occlusion according to the distance d to the occluding geometry. The idea is to diminish the amount of occlusion that an occluder casts when it is farther from the receiver, which simulates the behavior of indirect light. Therefore the falloff function is a monotonically decreasing function and it generally applies that $\rho(0) = 1, \rho(\infty) = 0$. Additionally, the falloff function takes care of an issue when ambient occlusion is applied to indoor scenes: Unless a falloff function is used, indoor scenes appear pitch black as generally every direction from a receiver is blocked by some geometry. In terms of the rendering equation Eq. 1.1, ambient obscurance AO integrates over the hemisphere of the receiver and weighs the integral with the falloff function and the geometric term:

$$AO(\mathbf{x}, \vec{n}) = 1 - \frac{1}{\pi} \int_{\Omega} \rho(D(\mathbf{x}, \omega))(\vec{n} \cdot \omega) d\omega \tag{1.3}$$

where D returns the distance to the nearest geometry in direction ω from the receiver $\mathbf{x}$. It should be noted that if the scene is assumed to be evenly populated by "light emitting fog", the falloff function ρ becomes an inverse exponential function, and 1.3 becomes an accurate description of direct illumination from this lit fog.

However, ambient obscurance is most often used in conjunction with a direct illumination method to approximate indirect lighting, and the falloff function is empirically selected to convey a plausible global illumination look. Ambient obscurance is not physically based as it does not sample actual surface illumination from the blocking geometry, and it ignores surface properties such as glossiness and varying reflectivity. Despite of AO not being physically based, it has become extremely popular in feature film effects and computer games because it produces visually plausible results but executes significantly faster than physically accurate solutions such as radiosity or recursive path tracing. Figure 1.2 shows the ambient occlusion component in a patio scene. Applications where physical accuracy is of utmost importance, such as architectural visualization, ambient occlusion or obscurance is not the best option as its results generally do not converge to

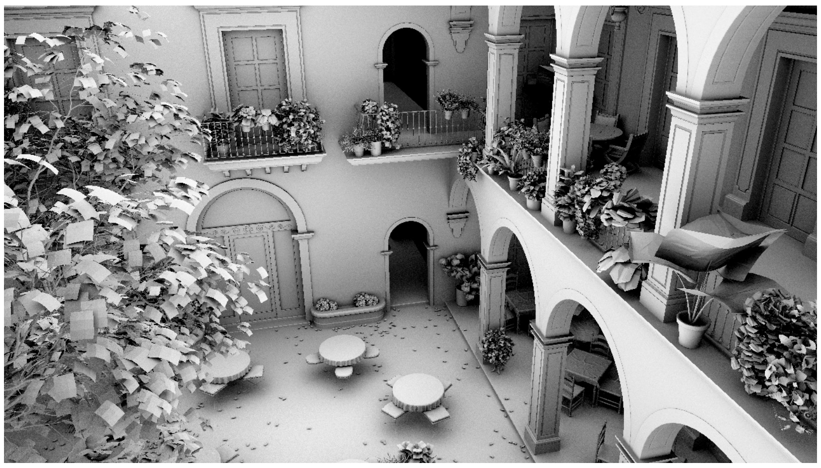


Figure 1.2: Ambient occlusion in the San Miguel scene, courtesy of Guillermo M. Leal Llaguno, rendered by the Blender renderer [27] in 65 seconds.

what is seen in reality.

AO has the additional benefit that it is not dependent on light sources or other illumination methods which makes it very easy to integrate into a renderer, and it can be used in previsualization situations where light sources are not yet present. The occlusion of ambient light is also a purely geometric property of a receiver point, and it does not need to be recomputed when light sources or materials change while geometry does not. Additionally, the obscurance effect is often limited to a certain neighborhood d_{max} around the receiver pixel by having $\rho(d) = 0, d > d_{max}$, which also makes ambient obscurance evaluation computationally cheaper as not all scene geometry has to be considered for each receiver.

1.3 Height fields as geometry

Height fields constitute an important special case of geometry. They are scalar functions on a plane, and define a surface of points $P = (x, y, HF(x, y))$ where $HF(x, y)$ is the function returning the height at surface coordinate (x, y). The main limitation of height fields as geometry is that one surface coordinate cannot have more than one defining point. It is possible to classify objects that can be described *fully* using a height field as follows: When the object is projected onto a 2-dimensional plane, there can only be at most one surface point in the object that maps to each point on the projection plane. The simplest case that does not meet this requirement is a solid object that has different front and back surfaces with respect to any viewing direction. It is possible, however, to represent arbitrary objects by multi-layer height fields [65] where multiple height fields are stacked on top of each other.

From now on we assume height fields to be defined at $x \in [0, W[, y \in [0, H[$ discrete coordinates on a regular grid. We also assume height fields to be linearly continuous between height field points, roughly as if the $W \times H$ grid was split into $W \cdot H \cdot 2$ triangles whose Z coordinate is displaced according to $HF(x, y)$. As no explicit information is available on how the surface should behave between the discrete coordinates, the linearity assumption is practical and maps efficiently to current GPU features and it is therefore usually made in graphics applications. It should be noted that this type of a height field can be trivially turned into a triangle mesh, and therefore algorithms that work on arbitrary triangle meshes can be used on them as well. The main benefit of algorithms tailored for height fields is that they are faster and sometimes of significantly lower time complexity than methods for generic geometry.

Height fields are a simple geometry representation that can be obtained by scanning real life objects or by “painting” onto an image using image manipulation tools. GPUs are efficient at handling height fields because they can be stored as a 2D texture for which dedicated sampling and caching hardware exists in virtually every GPU architecture. In computer graphics, height fields are most often used as follows:

- to represent standalone objects such as terrains (Figure 1.3, right)
- to describe micro-geometry on top of larger scale polygons by using a displacement texture (Figure 1.3, left), and
- as an approximation for scene geometry by treating the depth buffer as an inverse height map (Figure 1.4).

Displacement textures can be used by *displacement mapping* techniques [81] whereby new triangles are generated from the height data, or by *relief mapping* techniques [84] where only one polygon is rendered and the visible point is searched by ray marching in the fragment shader.

The third use case has gained widespread popularity in real-time applications recently, especially by *screen-space ambient occlusion and obscurance* (SSAO) methods which determine ambient occlusion from the depth buffer geometry. As most real-time renderers sort fragments according to their depth by updating and testing against a depth buffer, depth buffers are readily available as a by-product of the rendering pipeline. GPUs are extremely efficient at rasterizing a scene into a depth buffer, whereas classic meshing is complex and error-prone. The depth buffer also inherently adapts the level of detail to the relevant parts of the scene. The downside

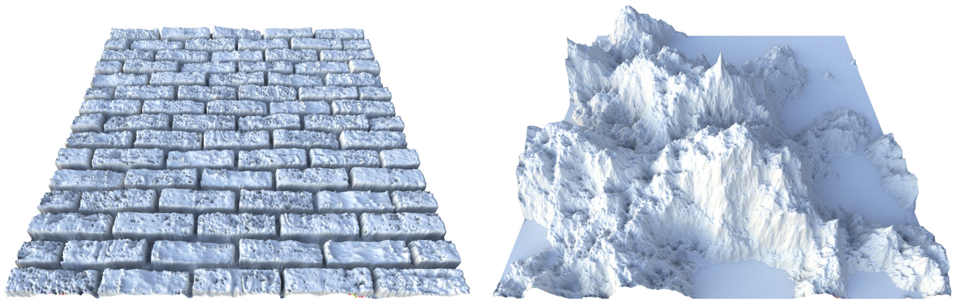


Figure 1.3: Left: a height field representing a brick surface. Right: a height field representing a fractal terrain. Images are rendered using the methods proposed by [P1] and [P2].

is that the depth buffer is not a complete description of the scene geometry. Specifically, geometry behind the first depth layer (due to occlusion) and geometry outside the view frustum (due to clipping) is unknown. Figure 1.4 shows an example SSAO rendering.

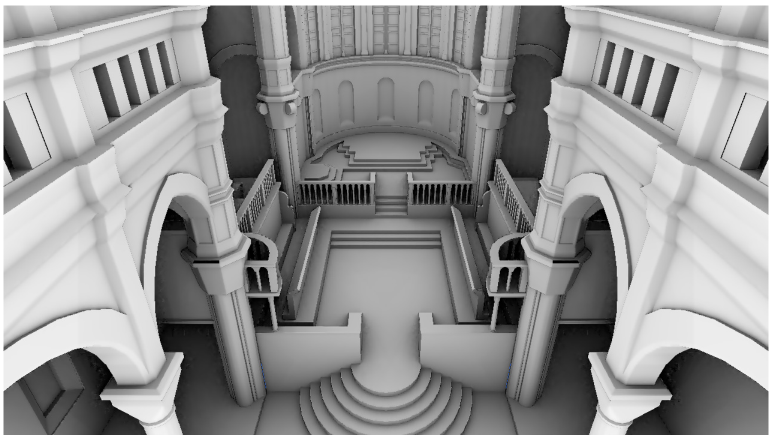


Figure 1.4: Screen-space ambient obscurance rendered for a model of the the Šibenik Cathedral, courtesy of Marko Dabrovic, by our method [P4] in under 5 ms.

Height fields have many uses beyond graphics as well, mainly because relief maps are usually a sufficient description for a terrain. For example, height field algorithms are used for siting problems [28], finding good-coverage locations for radio towers, missile defense systems [29], et cetera. They can also be used to plan routes for camera-equipped vehicles such as Mars rovers [71]. Nagy [59] gives a review of non-CG-related height field algorithms and their applications.

1.4 General-purpose computation on GPUs

Most real-time applications use a graphics library to employ the computational power of GPUs. Two graphics libraries dominate today: Khronos Group's OpenGL [73] and Microsoft's DirectX [31]. They allow the programmer to replace only certain parts of the pipeline by user-specified programs. These parts include, for example; vertex shaders which transform vertices from one coordinate system to another; fragment shaders which define the final color value of a pixel or a fragment; geometry shaders which modify geometric primitives; and tessellation shaders that specialize in efficiently generating new geometry. Parts of the graphics pipeline therefore remain non-programmable (*fixed-function*) and are carefully optimized by the GPU vendors into device drivers. Also, some computational features that are found in the hardware are restricted in the user-programmable shaders because not all GPU architectures support them and graphics libraries are intended to be as portable as possible. Additionally, some functionality which the hardware is technically capable of is prohibited because GPUs are generally not optimized for them. One example for this are scatter writes.

GPUs are a specialized processor architecture that is very efficient at highly data-parallel tasks, and therefore many non-graphics applications such as physics simulations are taking advantage of them as well. As general-purpose programming on GPUs gained popularity and GPU architectures became increasingly programmable, languages for *general purpose computation on graphics processing units* (GPGPU) started to emerge, which dropped most of the unnecessary graphics abstractions and allowed a more transparent access to hardware. Prior to GPGPU languages this was done by mapping generic programs as rendering tasks through the programmable shaders. The first GPGPU language to become popular was the NVIDIA's vendor-specific CUDA [33] and since then the vendor-agnostic OpenCL [32] from the Khronos Group has also become widely adopted. When GPGPU languages were introduced they were not only useful for non-graphics applications: Because they allow a more direct access to graphics hardware, also some graphics algorithms are now possible to implement on GPUs that would not otherwise be due to the limitations that the shaders impose. On the other hand, the restrictions in shaders of graphics libraries take care of many aspects that are crucial for performance, such as coalesced writes and efficient scheduling, which the developer becomes responsible for in GPGPU languages. When using a GPGPU language the developer has to adhere to careful code and memory access guidelines to maintain good performance.

As algorithms in this thesis have phases that do not map to traditional rasterization used by graphics libraries and therefore exhibit what the shaders classify as scatter writes, we, too, rely on GPGPU languages:

papers [P1] and [P2] are partially implemented in CUDA and papers [P3] and [P4] are implemented entirely in both OpenCL and CUDA. Additionally, some phases in our algorithms can be accelerated by using the direct access to the on-chip *shared memory* [91] (NVIDIA) or the equivalent *local data share* [1] (AMD) that is exposed by GPGPU languages but unavailable in shaders.

It is very important to note that during the development of the algorithms presented in this thesis, *compute shaders* were introduced to both DirectX [8] and OpenGL [74] which allow GPGPU programs to be executed in the graphics libraries. Therefore all of our algorithms can be entirely implemented in both OpenGL and DirectX as of 2012. CUDA, being specific to NVIDIA hardware, still offers some minor performance advantages on their hardware, such as the ability to control the shared memory-L1 split and access to NVIDIA specific instructions such as vote functions. Our algorithms make use of these features when they are available. Generally, GPU architectures are well suited to handle height field geometry because height fields can be expressed as floating point textures for which efficient sampling and filtering functionalities exist in current hardware.

Any microchip design is constrained by transistor, heat, and power budgets and increasing resources somewhere has to be offset by compromising elsewhere. Despite of this, there are some bottlenecks in current GPU architectures for our algorithms, and a different resource balance would further improve the performance of our implementations. Namely, [P1], [P2], and [P3] all suffer from load imbalance as our implementations due to dynamic branching leave, on average, a significant portion of the *single instruction multiple thread* (SIMT) lanes idling. Battling this with a software load balancer, on the other hand, becomes easily very costly due to the extra instruction overhead. Therefore allocating some transistors to hardware load balancing features would likely improve the overall efficiency of our algorithms. Additionally, the algorithm in [P2] uses a relatively large data set per thread, and larger on-chip caches would cause less DRAM traffic which is currently the limiting factor.

1.5 Our contribution

Our publications target height field and screen-space geometry and are aimed to be used on dynamic geometry where computational efficiency is a key. We use the concept of *time complexity* [75], denoted by the complexity notation $O()$, to characterize the scaling of our algorithms. Time complexity essentially tells how much work is required, i.e. how many algorithm iterations needs to be taken, to solve a problem with respect to the input size which in our case is the amount of height field geometry. For example, an algorithm with a time complexity of $O(N^2)$ performs four times the work when the input size is doubled, or a hundred times the work when the input is increased by ten-fold. Our algorithms are *scalable*, by which we mean that they show lower time complexity than previous work and scale well in the height field size allowing for accurate illumination effects to be produced on very large height fields. In practice, this scalability also shows as short execution times in common height field sizes. Our algorithms also scale well with respect to image quality: they allow high quality effects to span the entire height field unlike previous methods which are limited to short-range effects or low-quality long-range effects.

The method proposed in [P1] reduces the time complexity for generating a *horizon map*, i.e. determining visibility of a light environment from each height field point. Previous work takes $O(N)$ time per height field point on N^2 height fields, whereas our method produces the same results in an amortized $O(1)$ time per point by incrementally traversing the height field. A method to accurately and efficiently integrate Equation 1.1 over the visible light environment on diffuse surfaces is also proposed in [P1].

In [P2] a method is proposed for reducing the time complexity for determining *intervisibility* in a height field, which is necessary for accurate indirect illumination. Our algorithm scales in the actual visibility for each height field point and not in the entire height field size as previous methods. We show that visibility scales strongly sublinearly in the height field size and therefore our method reaches substantial savings in the number of algorithm iterations—two orders of magnitude in common use cases. For the 1024^2 height fields shown in Figure 1.3, we measure the percentage of the average number of iterations in our algorithm as compared to the previous work to be 0.99 % for the brick surface and 2.4 % for the fractal terrain.

Publications [P3] and [P4] contribute to screen-space ambient obscurance (SSAO) family of real-time rendering methods. In [P3] the time complexity for approximate SSAO evaluation is reduced by using an approach similar to [P1] and issues are solved in sampling and AO evaluation that are specific to SSAO. An SSAO algorithm for rendering very high quality AO effects is proposed in [P4]. This is achieved by extracting samples from the depth field that are tailored to capture features that are important for AO. The sampling technique achieves results that show an order of magnitude smaller error than previous work. A strategy for integrating Equation 1.3 accurately and efficiently is also proposed in [P4]. These two contributions together allow real-time rendering of SSAO effects that are within a few percent of a ray traced screen-space reference result, only much faster.

Author's contribution: In the presented publications the author developed the ideas and the resulting algorithms and carried out their implementation and optimization. Analysis of the methods and the content of the papers were also produced by the author. Prof. Jan Westerholm helped in refining the manuscripts, especially [P1], to meet academic standards.

Chapter 2

Previous work

In this chapter we present prior work most relevant to ours with a focus on height field methods used in illumination algorithms. Before the height field methods we briefly cover the main visibility and ambient obscurance strategies for generic geometries in Sections 2.1 and 2.3 and show how these are less efficient than the height field specific methods. Height field visibility and screen-space ambient obscurance methods are then covered in detail in Sections 2.2 and 2.4.

2.1 Visibility

In non-real-time graphics, where high quality renderings are preferred over fast execution, the two main branches for solving visibility are based on path tracing [40] and radiosity [30]. Path tracing evaluates visibility according to Equation 1.1 by shooting rays into the scene from each receiver. The rays are traced to their first intersection with the scene geometry in order to determine intervisibility and produce indirect lighting effects. When producing direct lighting effects, it is necessary to query whether the ray hits any given light source instead. In case the direct light source is modeled as a light environment, it is sufficient to query whether *any* geometry exists along the ray path. When the incident light is roughly uniform like in the case of ambient obscurance, a few hundred rays are sufficient to produce results of adequately low noise. However, when incident light is uneven—as is the case of generic indirect illumination—thousands of rays need to be traced to produce good results. These cases can be helped by tracing light additionally from the light sources into the scene and connecting the paths—a technique called *bi-directional path tracing* [46]. Even more difficult light paths can be traversed efficiently using *metropolis light transport* [86].

By using hierarchical data structures for the scene geometry the computation of tracing one ray against the scene geometry can be achieved in

$O(log(M))$ time where M is the amount of geometry in the scene. Therefore, on a $W \times H$ height field, the time complexity of solving intervisibility by ray tracing for one height field point is $O(R \cdot log(W \cdot H))$ where R is the number of rays traced per pixel. This is significantly higher than the complexity of our height field specific methods [P1] and [P2]. Additionally, the complexity of reconstructing the hierarchical data structure for dynamic geometry can be high [87], which is a step not required by height field methods.

The second main approach to solving intervisibility of arbitrary geometry is radiosity, which solves Equation 1.2 by iterating over the scene in finite elements. If a $W \times H$ sized height field is split into one patch per height field point, $O(W^2 \cdot H^2)$ patch-to-patch operations have to be performed to determine full intervisibility. This, too, has significantly higher time complexity than our methods. Additionally, the binary visibility term has to be solved in the radiosity method, which further increases the time complexity. A common approach is to rasterize a view of the occluders onto a hemicube [13] or a cubic tetrahedron [6]. Using P rasterized pixels, the full time complexity for a radiosity method is $O(W^2 \cdot H^2 \cdot P)$.

In [10] a similar approach to radiosity is presented which treats each face of a polygon mesh as a disk. Visibility is ignored and the contribution of each disk occluder is accumulated. The resulting over-occlusion is removed layer by layer by performing incremental passes of the algorithm. The same approach to visibility is pursued in [18]. Overall, radiosity family methods are not suitable for dynamic geometry in real-time applications because they don't scale beyond a low amount of geometry [67], although they are efficient for precomputing radiance in diffuse scenes.

Finally, there are global illumination solutions that are not based on an efficient solution to the visibility problem, but rather simplify either lighting, occluders, or receivers enough such that the input to the visibility solver becomes tractably small. These methods include *photon mapping* [39] where a finite number of photons are cast from the light sources, and their intersections with the scene geometry are then stored into a data structure that is queried when evaluating lighting for a given receiver. *Instant radiosity* [43] similarly traces light paths and creates virtual point lights (VPLs) at light-surface bounces. Indirect lighting can then be rendered by lighting the scene from the VPLs by using *shadow maps* [89].

Reflective shadow maps [17] on the other hand store light intensities into the shadow maps and gather lighting for each receiver by sampling the shadow map textures. Several other methods reduce illumination to a relatively small number of virtual point lights, from which gathering becomes computationally feasible. In [2], for example, objects are directly illuminated from an environment light source by collapsing the environment into a tractable number of light sources and using a fast approximate soft-shadowing method to generate shadows from them. Additionally the screen-space receivers can be adaptively simplified making interactive global illumination feasible, as done in [60]. Finally, many methods exist that illuminate objects based on precomputed visibility information, such as the popular *precomputed radiance transfer* (PRT) [76] method, but as they do not offer a solution to the visibility determination we do not consider them any further.

2.2 Height field visibility

2.2.1 Horizon mapping

Most height field methods that compute self-occlusion in order to shadow the surface from a point, an area, or an environment light source are based on evaluating a *horizon map* [52]. For each receiver point, the horizon map stores the horizon angle as a function of azimuthal direction. The horizon angle is the angle at which the receiver no longer sees the height field, which is defined through the point in the full height field that has the highest slope when measured from the receiver. The horizon map is usually discretized into a finite number of azimuthal sectors (which we denote by K from now on), and has been used to self-shadow height mapped surfaces in real-time in [77] and [63]. These methods do not suggest a way for generating the horizon map efficiently and are thus aimed for static geometry.

Before our method in [P1], generating the horizon map for dynamic geometry has been computationally significantly more complex than evaluating lighting from the horizon map, which is an $O(K)$ time operation per receiver. The approach taken by previous real-time methods is to traverse the height field from each receiver point along the azimuthal directions by ray marching. In [83] this approach is implemented in a fragment shader and illumination from an area light source is determined. Also [5] takes the same sampling approach until a specified maximum distance from the receiver. These approaches sample the center line of each azimuthal direction, which is prone to produce discontinued shadows from thin occluders.

In [80] the highest horizon value is determined out of all points in each azimuthal sector for each point, trading the discontinuity artifact for overshadowing but also making the method biased. Their approach is however not targeted for real-time evaluation and due to the high execution time is only suitable for static geometry. Our approach in [P4] also determines occlusion across the entire azimuthal sector and therefore does not produce discontinuous shadows, but instead of finding the highest horizon, our method takes the average and is therefore not biased. Our method is also significantly faster and allows real-time evaluation.

Finally, there are some height field methods that do not base their visibility on horizon maps. In [62] the maximal unoccluded cone is found for each height field point and surface normals are bent towards the open area. This produces approximate self-shadowing under environment lighting and is fast to evaluate, but is significantly less accurate than horizon map based methods and cannot produce detailed self-shadowing effects.

2.2.2 Multi-resolution approaches

In order to make the ray marching approach feasible for generating horizon maps for dynamic geometry, a multi-resolution pyramid of the height data can be generated. Lower resolution levels are then used when sampling farther from the receiver, allowing the step size to be increased exponentially when traversing farther from the receiver. Assuming $N \times N$ height fields, this reduces the $O(K \cdot N)$ time complexity per height field point to $O(K \cdot log(N))$. Having pre-filtered lower resolutions of the height field removes many undersampling artifacts much like mipmapping [90] does and tri-linear filtering reduces texture aliasing artifacts. However, as visibility information becomes increasingly less accurate when distance from the receiver increases, multi-resolution approaches are suitable only for reasonably soft effects such as low frequency indirect illumination and ambient occlusion; detailed shadows that extend far from the caster, for example, are not possible. A recent effort to use multi-resolution height fields is presented in [78] where low-order spherical harmonics are used to soft-shadow height fields. Their approach achieves interactive and real-time execution times for moderately sized height fields. In contrast, our method computes accurate horizon maps in an amortized $O(K)$ time per receiver point, and allows all-frequency real-time self-shadowing for much larger height fields.

Multi-resolution depth buffers are used in screen-space ambient occlusion and obscurance methods as well: in [4] and [36] SSAO is evaluated on multiple resolutions of the depth buffer and combined into one full SSAO result. Mipmaps are also used in [54], but depth buffer values from the base resolution are used instead of the averaged values, and multiple resolutions are only utilized to improve texture cache utilization. When using multiple resolutions one needs to be careful when reconstructing results from different levels of detail in order to avoid transition artifacts. In [78] oversampling and B-splines are used to interpolate between different scales, and [36] uses temporal filtering as a post-process to smooth out flickering.

Mipmaps are also widely used to accelerate indirect illumination effects in screen-space by approximating far-field effects using lower resolution levels. For instance, [79] extends local indirect color bleeding effects of [68] to arbitrary distances.

2.2.3 Intervisibility

We have now covered methods which calculate the horizon map for a height field. The horizon map can be used to illuminate a height field from external direct light. The computationally more difficult problem to horizon mapping is to calculate intervisibility, i.e. to determine which other height field points are visible to each height field point. This information is necessary for producing indirect illumination effects where the height field itself acts as a light source, and height field points need to know from which other points they are able to receive light.

Several efficient methods exist to determine visibility of a height field (the *viewshed*) from a single point on the height field or slightly above it [42] [25]. In computer graphics, this information can be used to shadow a height field from a point light source, or to cull invisible height field regions to accelerate rendering [14]. Methods have also been developed to determine visibility of the height field from a line path [16] [26]. This information can be used to conservatively cull geometry for multiple camera points when the camera path is known in advance, or to evaluate the coverage that a transmitter or camera equipped vehicle will cover along its path. Finally, visibility of a height field can be determined from a region [7]. This allows lazy geometry culling for a camera even when its path is not known beforehand, but its maximum velocity is known to be bound. Only when the camera exits the region does the visibility have to be recomputed. A logical extension for this is the yet harder intervisibility problem, where exact visibility for each height field point individually is calculated. Storage requirements permitting, intervisibility information can also be used to cull geometry if the camera position remains sufficiently close to the surface.

Prior to our work, state-of-the-art methods for determining intervisibility in a height field were based on the same idea as horizon map generation: from each receiver point, the height field is traversed outwards separately for each receiver point for the K azimuthal directions [15] [72]. For each outwards step, the maximum slope from the receiver up till the latest step is tracked and each time it is exceeded, the height field point at the new step is known to be visible. This also has the same complexity as the horizon map generation: for K azimuthal directions $O(K \cdot N)$ time is spent for each height field point. Similarly to horizon map generation, this complexity can be reduced to a logarithmic complexity by multi-resolution approaches if the limitation to soft effects can be accepted. The multi-resolution approach to intervisibility is pursued in [61] where real-time performance is achieved for moderately sized height fields.

Methods that are based on static geometry and precomputed intervisibility also exist. In [35] intervisibility is pre-calculated and coordinates of the visible points along a small set of directions are stored into a texture for each receiving height field point. However our focus in this thesis is determining the intervisibility information efficiently enough to be suitable for use for dynamic geometry, and methods based on precomputed visibility are out of our scope.

2.3 Ambient obscurance

As described in Section 1.2, ambient occlusion or obscurance (AO) is an approximation for a subset of global illumination effects that are otherwise computationally difficult. A precursor to AO was *accessibility shading* [57] which fits as large a sphere as possible onto each surface point such that it does not intersect the surface. The size of the sphere determines the amount of light that can reach the surface. While this effect is extremely local, it gives plausible visual cues about the small scale structure of an object.

However, modern AO methods began with Zhukov et al. in 1998 [92] who formulated AO using the rendering equation. The incident light is replaced by a falloff function ρ, which is an empirically selected monotonically decreasing function of distance to the occluding geometry. AO can be limited to strictly local effects, by having $\rho(d) = 0, d > d_{max}$, and combined with another far-field method, an approach pursued in e.g. [3]. Ambient occlusion started to become a commercially popular shading method in 2002 when it was introduced in the RenderMan renderer [48] [11] [12] [9] and was used in two feature films that year. Since then, it has become a de-facto component in feature film rendering. A thorough survey of AO methods is given in [56].

AO methods that work on arbitrary geometry meshes have been used in real-time applications as well, but they generally assume a static scene geometry or target non-deformable objects. In [10], for example, a disk occluder is formed from each scene triangle and, in an approach similar to radiosity, pair-wise occlusion is gathered from them as a pre-pass. The algorithm needs to be run iteratively to determine visibility. This method is improved in [37] by gathering occlusion at each screen pixel instead of at each vertex, and the method is implemented using GPUs. Fast ray tracing based methods have also been proposed: e.g. [47] employs a hemispherical rasterization scheme to overcome the over-occlusion problem more efficiently. However, these methods are still not real-time ready for dynamic geometry and large scenes.

Inter-object AO is calculated in [45] by precalculating the occlusion cast by an object as a function of object direction and distance. At runtime, occlusion from an object is approximated as a spherical cap in the receiver's hemisphere. This method is fast but the final occlusion is a very rough approximation. In [53] AO is computed by expanding each scene triangle into a polygonal volume which is then rasterized efficiently using GPUs and occlusion from all volumes is accumulated in the framebuffer. Thinly stacked occluders are prone to cause over-occlusion, which is remedied in [47] by using a visibility mask on the hemisphere such that multiple occlusions from the same direction cannot accumulate. When the AO radius is large or the volumes overlap significantly, this approach requires substantial fillrate capacity from the graphics hardware and ultimately limits the method's suitability to real-time applications.

Methods that work on a 3D voxelization of the scene [64] [69] [66] can achieve real-time performance in some scenes but are highly sensitive to the amount of scene geometry, thus making them not scalable enough for large scenes in real-time applications.

2.4 Screen-space ambient obscurance

Even the fastest AO methods that work on generic geometry are not yet quite real-time ready for production-sized dynamic scenes. This is where screen-space AO (SSAO) evaluation is often considered and SSAO methods are indeed used by most high-end computer games today. SSAO methods evaluate AO using only the geometry found in the depth buffer, and are therefore insensitive to scene complexity and work on fully dynamic scenes. The depth buffer is a readily available by-product of most renderers, and essentially holds the distance from the camera to the nearest object at each screen pixel. Evaluating lighting as a separate post-process in the framebuffer, after the scene geometry has been rasterized, is a common approach outside ambient occlusion methods as well, and is referred to as *deferred shading* [21].

The idea to evaluate AO by sampling the depth buffer around each receiving pixel was simultaneously introduced in the industry in the CryEngine game engine [58] and in the academia [70]. Their approach is very simple: they sparsely populate a fixed-size 3D sphere volume with sample points that are tested for occupancy. Each point is projected onto screen-space to obtain a 2D sampling position where the depth buffer is sampled. If the 3D sample position is below the sampled depth value, the point is considered to be occupied by scene geometry and the point is treated as an occluder. The ratio of occluded points to all sampled points is then used as the amount of occlusion of the ambient light.

Another option is to take line or area samples around each receiver point to approximate how much of the sphere's volume is occluded [50] [41] [82]. While these methods can be fast, they, and the said point sampling methods, are not correct geometry-wise because they ignore occluder fusion: for each occluded sample there might be a nearer occluder along the same direction.

Only the nearest occluder in each direction, as per Equation 1.3, should be accounted for and all other occluders in the same direction should be ignored. These methods therefore exhibit occasional over- or under-occlusion, depending on how algorithm parameters have been tuned. *Horizon-based ambient occlusion* (HBAO) [5] [22] solves this problem by finding the largest horizon along a number of azimuthal directions by ray marching, and assumes geometry below the horizon to be blocked. This approach is geometrically correct but requires more depth buffer samples and thus high execution times.

Methods presented so far also ignore the falloff function and cut occlusion abruptly after a certain distance. The SSAO quality is further improved in [5] and [24] by taking the falloff function into account and result in properly attenuated and smoother occlusion. The falloff radius is still usually cut after a relatively short distance because larger sampling radii become prohibitively expensive due to increased sample counts. Unfortunately the falloff is defined as a function of *eye-space* distances which, in screen-space, depend on the camera's distance to scene geometry and may thus get arbitrarily large. Limiting the occlusion effect in screen-space causes objects to change appearance depending on their distance to the camera, but allowing arbitrarily large screen-space radii affects performance adversely.

A fundamental limitation to SSAO methods is that the scene geometry in the depth buffer is incomplete: only the first depth layer is known without any information what is behind it. This issue is tackled in [4] by taking multiple depth-peeled layers into account. This approach works in some scenes but fails in situations where depth complexity is too great and too many layers would have to be peeled, each layer increasing the execution time of the method. Furthermore, depth peeling is more difficult to adopt into graphics engines as otherwise unnecessary geometry passes to peel the depth layers have to be performed. Another approach is to use multiple views of the scene to fill in missing information. This is pursued in [85] and [68] by rasterizing additional views using phantom cameras around the real one. In order to avoid having to rasterize new views just for SSAO, shadow map [89] views can be used instead of the additional cameras at no extra cost.

The second limitation in the depth buffer geometry is that geometry outside the view frustum is unknown. Most methods mentioned in this section mitigate this problem by extending the depth buffer in each direction by 10 % or so to form a *guard band*. Geometry within the guard band is considered as potential occluders for the visible pixels which alleviates the problem of missing occlusion from occluders that are right outside the framebuffer.

High quality AO requires a prohibitively large number of samples around the receiver, especially for a large AO radius. To combat the cost of far-field

AO multi-scale methods have been proposed, much like the multi-resolution methods described in Section 2.2.2. In [4] and [36] ambient occlusion is evaluated on multiple resolutions of the depth buffer and the results are then upsampled and combined with the full resolution result. Regardless of the used low-pass filter, filtered lower resolutions of the depth buffer do not preserve the original geometry and can significantly corrupt occluders whose projection in the depth buffer is thin.

Finally, a series of refinements to the above approaches have been proposed, such as allowing for more artistic control and optimizing the evaluation of the AO integral [55]. In [54] sampling patterns and texture cache efficiency are improved as well as the performance of the commonly used blur phase which hides noise that arises from sparse sampling. AO evaluation can also be interpreted as a filter kernel and assumed separable [38]. While AO is not physically separable, this approach can significantly reduce execution times especially for large kernels without introducing very disturbing visual artifacts.

Despite the extensive research effort, SSAO methods still suffer from several issues. Firstly, they do not approximate the integral in Equation 1.3 accurately and therefore do not converge to a ray traced reference which can be obtained by evaluating the Equation 1.3 by casting many rays from each receiver point into the screen-space scene geometry. Secondly, high quality results require more samples per pixel than is practically affordable in real-time applications and therefore sparse sampling is resorted to. Sparse sampling is prone to miss thin occluders and cause under-occlusion as well as to produce noisy results. If noise is alleviated by a post-process blur, details are lost. Thirdly, to limit the sampling radius SSAO effects are often limited to very local effects which tend to look unrealistic and produce dark halos around objects. Finally, the problem of incomplete scene geometry has not yet been fully solved.

Our work targets the first three issues: in [P3] we reduce the computational complexity of the geometrically correct approach [5] and manage an order of magnitude reduction in its render times. In [P4] we propose a method which accurately captures occluders in the depth buffer within unlimited radius at constant real-time render times, and evaluates the AO integral accurately such that results very close to ray traced screen-space reference are obtained.

Chapter 3

Height field visibility

Two of the publications in this thesis propose methods which lower the time complexity for determining visibility in height fields. In [P1] the time complexity for determining the horizon map in K azimuthal directions for a point in $N \times N$ height fields is lowered from $O(K \cdot N)$ of previous work to $O(K)$, whereas [P2] lowers the time complexity for determining intervisibility from $O(K \cdot N)$ of previous work to between $O(K \cdot N^{0.01})$ and $O(K \cdot N^{0.65})$. In Section 3.1 we discuss the time complexity and the output space complexity of the horizon map and intervisibility computation. The way [P1] and [P2] traverse the height field is described in Section 3.2 and the algorithms used during the traversals are described in Sections 3.3 and 3.4, respectfully. Results are presented in Section 3.5.

3.1 Computational complexity

For simplicity, in this section we assume square $N \times N$ height fields. Current state-of-the-art height field methods determine visibility in a number of discrete azimuthal directions at each receiver point. In [P1] and [P2] we take the same approach and here denote the number of uniformly distributed azimuthal directions by K. K can be freely chosen, but for low values of K, say $K < 15$, banding artifacts in the illumination results may become visible.

Previous methods perform independent visibility searches for each receiver by sampling the height field along the azimuthal directions. For both the horizon map generation (Section 2.2.1) and the intervisibility determination (Section 2.2.3) $O(K \cdot N)$ iterations per receiver have to be performed in order to accurately cover the height field. If approximate results are sufficient, this cost can be reduced to $O(K \cdot log(N))$ through multi-resolution height fields (Section 2.2.2).

Traversal method	Time complexity	Accuracy
Horizon map generation, output complexity of $O(1)$:		
Linear [83]	$O(N)$	exact, sector centerline
Multi-resolution [78]	$O(log(N))$	approximate
Full sector [80]	$O(log^2(N))$	exact, max within sector
Our method [P1]	$O(1)$	exact, sector centerline
Intervisibility calculation, output complexity of $O(N^k), 0 < k < \frac{2}{3}$:		
Linear [68]	$O(N)$	exact, sector centerline
Multi-resolution [61]	$O(log(N))$	approximate
Our method [P2]	$O(N^k)$	exact, sector centerline

Table 3.1: The space complexities of exact visibility information at one receiver point along one azimuthal direction, and the corresponding time complexities for different height field visibility algorithms that calculate the information.

When the horizon map is generated for the purpose of shadowing direct external light, the output for each receiver is K horizon values. When intervisibility is determined, the output per receiver is a description of the visible points in the height field along each azimuthal direction. The amount of output data is therefore dependent on the height field content. While in the worst case the amount of data may be $O(K \cdot N)$ per receiver, in practical height fields visibility scales strongly sub-linearly in the height field size. In [P2] we measured visibility scaling to be between $O(K \cdot N^{0.01})$ and $O(K \cdot N^{0.65})$ where the lower limit is reached when the height field is grown such that new content is introduced, and the higher limit is reached when the resolution of the existing content is increased. We measured the average number of visible height field points to be 1 - 6 % of the total number of height field points for different types of 1024×1024 height fields.

Both of our methods, [P1] and [P2], determine visibility in time that is linear in the amount of output data. Table 3.1 compares the time complexities of the previous work and our methods for horizon map and intervisibility computation. It should be noted that the method in [80] has not been found suitable for real-time evaluation.

3.2 Height field traversal

Instead of performing independent visibility searches for each receiver point as done in prior work, we perform K sweeps over the height field and simultaneously determine visibility for all height field points along the direction of the sweep. Sweeps are performed in parallel lines (Figure 3.1, left), where each line is incrementally traversed by taking unit length steps along the

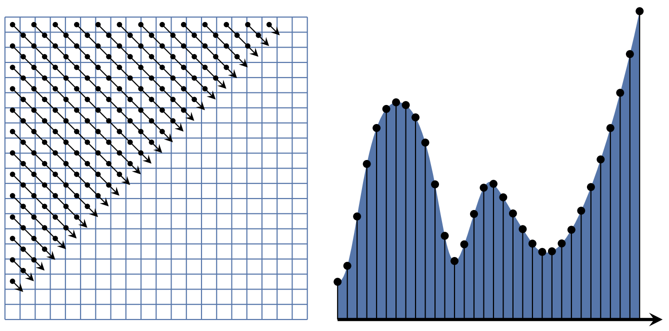


Figure 3.1: For one azimuthal direction, the height field is processed in parallel lines shown to the left. Each line is traversed one step at a time by sampling the height field at the corresponding coordinate as shown to the right.

line (Figure 3.1, right). After the sweeps have been performed and their results have been written to intermediate buffers, they are gathered in a post-process to obtain the full result as shown in Figure 3.2.

We assume the height field to be continuous and use bilinear interpolation to sample the height field at the corresponding floating point coordinate of each step. While this type of sampling does not exactly visit the original height field points it is sufficiently accurate in practice, especially when the height field represents sampled data itself. Once the height field has been sampled, the sampled point is inserted into an internal data structure. The data structures for determining the horizon map and intervisibility are different and are described in Sections 3.3 and 3.4, respectively.

After the internal data structure has been updated with the new height field point, visibility of the previously visited samples along the line as seen from the new point can be trivially read from the data structure. The visibility information can either be stored for later use, or illumination (or any function of visibility) can be directly evaluated and stored. In [P1] we also propose a novel way to accurately and efficiently integrate direct illumination according to Equation 1.1 from an environment light source given as a high dynamic range cube map [20].

3.3 Horizon map computation

When generating a horizon map in [P1], the internal data structure we use is logically a stack which holds, at any given point, the convex hull subset of the traversed height field points. The motivation for this is that for any height field point along the line of traversal the point casting the horizon is always part of the convex hull, and in fact the immediate neighbor of

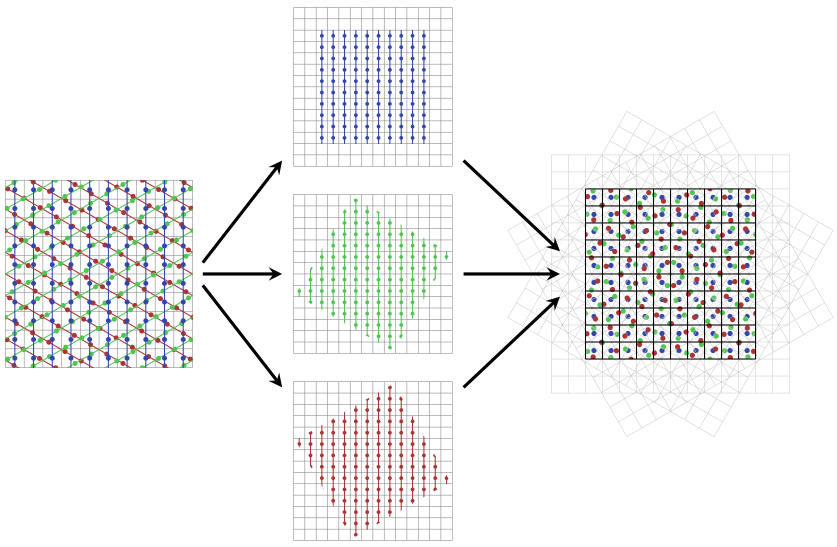


Figure 3.2: Three sweeps (denoted by different colors) are performed over the input height field (left), their results are written to axis-aligned intermediate buffers (center), and finally accumulated in the result buffer (right).

the receiver. When a new height field point along the line is processed, the convex hull stack is popped until the new point can be inserted into the stack such that the set remains convex. The point previous to the new point in the stack then is the point that casts the highest horizon onto the new point. The progression along one line is demonstrated in Figure 3.3, and the extracted horizon angles along a sweep are visualized in Figure 3.4.

Appendix A.1 lists the procedure that processes one line along the height field and outputs the global horizons for points along the line. In a threaded implementation, one thread processes one line and a total of approximately $(1 + \sqrt{2})/2 \cdot N \cdot K$ lines can be processed in parallel.

3.4 Intervisibility computation

The internal data structure used to determine intervisibility in [P2] is a convex hull *tree*. During traversal the line is split into convex and concave parts and visibility of each convex-concave pair is determined. From the beginning of each convex part, a convex hull is formed much like a convex hull is formed from the beginning of the line when generating horizon maps. Multiple convex hulls are therefore formed, and they are stored into a tree structure as listed in detail in Appendix A.2. This structure is useful because intervisibility becomes described in intervals by a series of visibility horizons that point towards the surface itself.

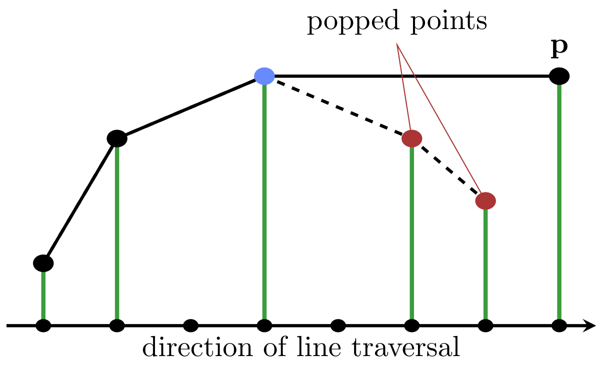


Figure 3.3: The convex hull subset of the height field points. When **p** is inserted, the red points are popped from the hull, making the blue point the highest horizon caster for point **p**.

Figure 3.5 visualizes a convex hull tree. The new height field point is always the root node of this tree. The vertices of the first-level children of the root form a series of local visibility horizons. Local visibility horizons were first introduced in [19] and they constitute a more compact description of height field visibility than a simple enumeration of the visible points. Local visibility horizons start from the receiver and always end at a point in the height field that is visible from the receiver but whose previous point is not. A pair of visibility horizons therefore encloses consecutive visible height field points. If needed, it is trivial to enumerate the visible points from the set of local visibility horizons.

3.5 Results

On 1024^2 height fields [P1] reduces the number of algorithm iterations required to generate a horizon map by more than two orders of magnitude, and [P2] reduces intervisibility determination by roughly two orders of magnitude. The source of this reduction is the reduced time complexity and therefore the difference to previous work gets greater with larger height fields. Practical execution times are also reduced significantly with respect to previous work. This is more prominent in [P1] due to its simpler implementation: compared to most relevant prior work [78] our algorithm calculates the horizon map roughly 15 times faster and at the same time more accurately. Due to its accuracy our method is able to produce both low- and high-frequency self-shadowing as shown in Figure 3.6. Figure 3.7 shows a height field rendered under environment lighting.

The implementation of [P2] on the other hand is more challenging due to algorithmic complexity, and its performance is dependent on the height

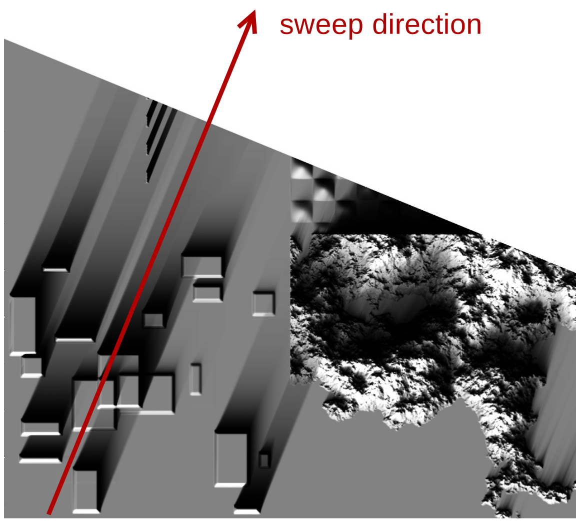


Figure 3.4: Visualization of the extracted horizons for one partially finished azimuthal sweep.

field content. However, we measured that our algorithm calculates exact intervisibility information faster than previous work in all of our test cases, and the speed-up varied between 2.4 and 41 in typical 1024^2 height fields. Intervisibility information is necessary in indirect illumination and to render surfaces that self-emit light. Figure 3.8 shows a height field that is directly and indirectly lit using our methods [P1] and [P2] under outdoor lighting. Figure 3.9 shows a height field which, in addition to direct and indirect illumination, also emits light.

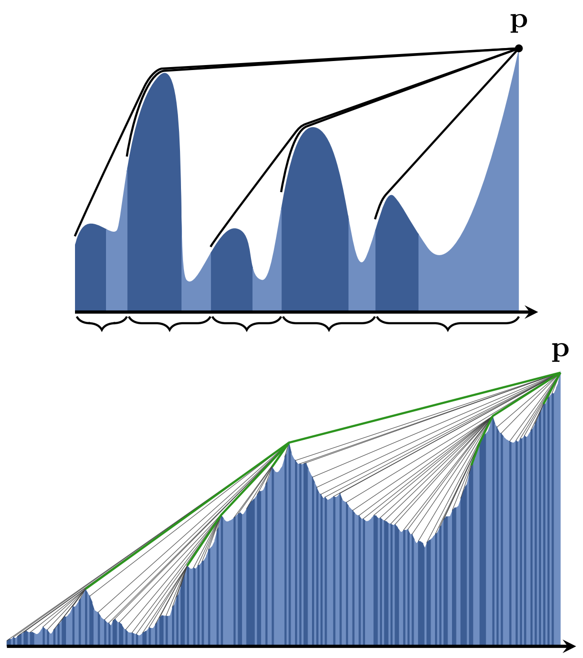


Figure 3.5: Top: convex hulls are formed from the beginning of each convex part (dark blue) to the latest height field sample **p** and organized into a tree structure. Bottom: visualization of a convex hull tree in a fractal terrain (Figure 1.3, right). Shared links are highlighted in green.

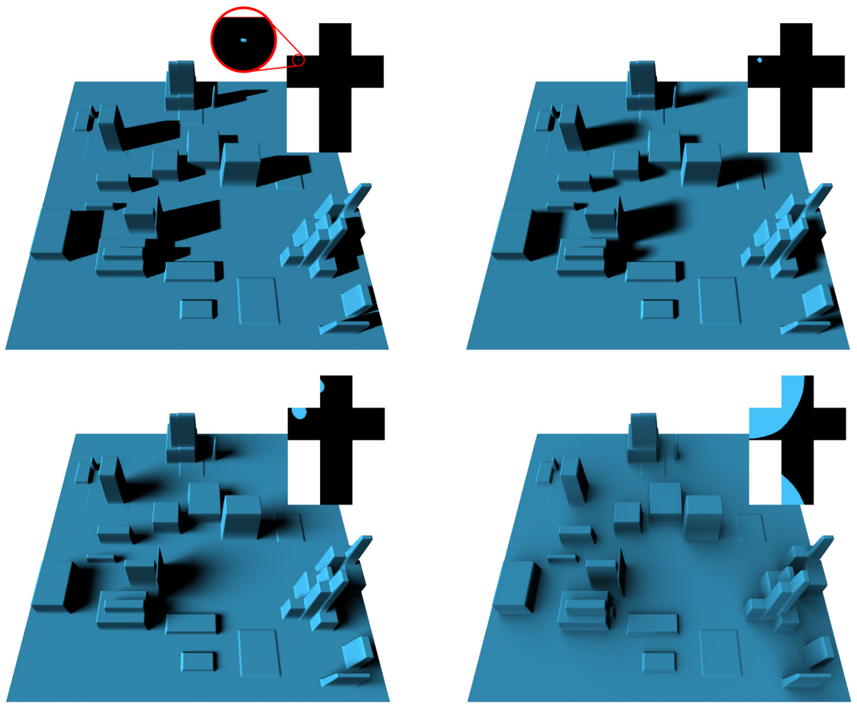


Figure 3.6: A 1024^2 height field illuminated using different light source sizes shown in the unfolded cube map to the top right of each image. Both low- and high-frequency shadows are produced by the method in [P1].

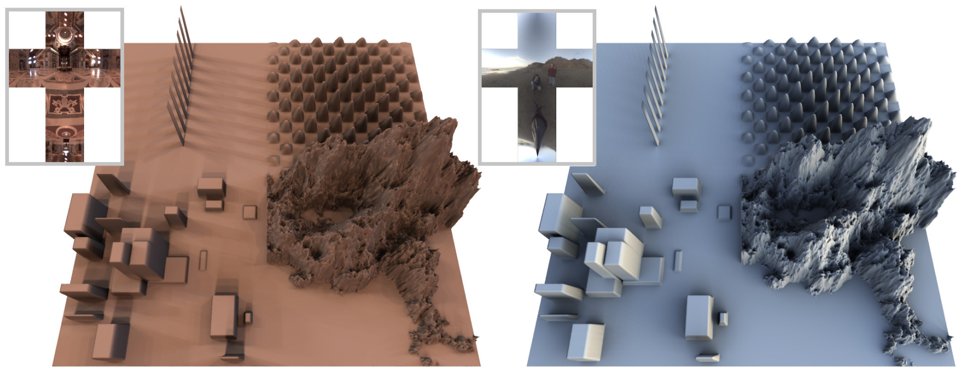


Figure 3.7: A height field illuminated using the method proposed in [P1] using environment light maps that are shown unfolded to the top left of each image. The horizon map information is generated in 0.30 ms per direction and illumination is rendered in 0.35 ms per direction for the full 1024^2 height field on an NVIDIA GeForce GTX 280.

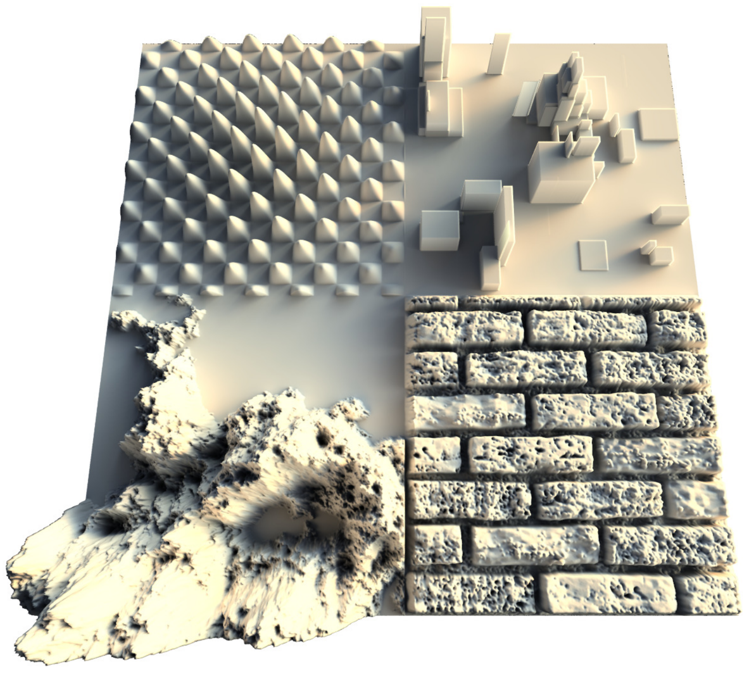


Figure 3.8: A diffuse height field indirectly illuminated using our method in [P2] under outdoor environment lighting. Intervisibility information is calculated in 3.14 ms per azimuthal direction for the full 1024^2 height field on an NVIDIA GeForce GTX 480.

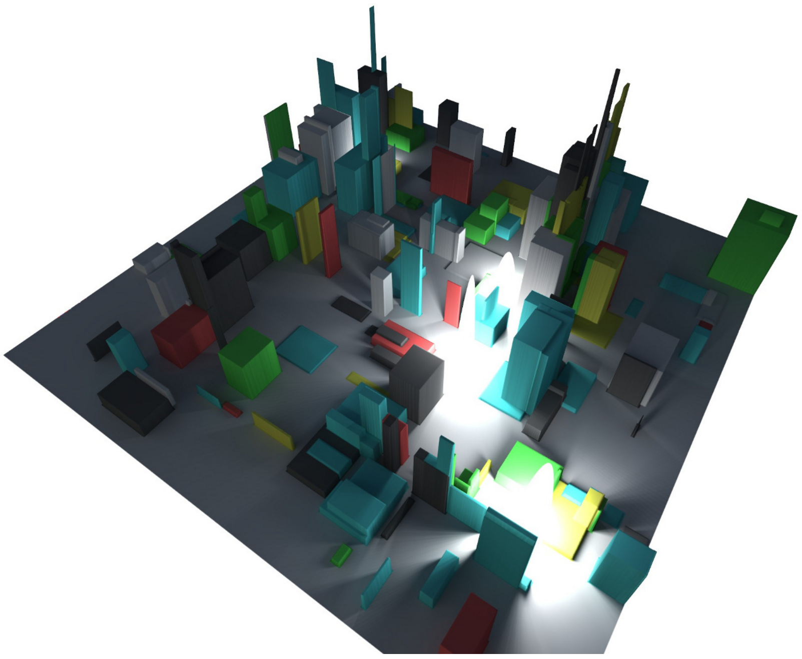


Figure 3.9: Indirectly illuminated 1024^2 height field using our method in [P2]. Additionally, parts of the height field emit light. Intervisibility information is calculated in 0.59 ms per direction on an NVIDIA GeForce GTX 480.

Chapter 4

Screen-space ambient obscurance

Two of the methods in this thesis improve quality, scaling, and performance of the popular screen-space ambient obscurance (SSAO) method. The method presented in [P3], described in more detail in Section 4.1, accelerates the rendering of approximate AO effects by reducing the time complexity over previous work. The method in [P4] on the other hand targets higher quality AO results than previous methods but at comparable render times, and is described in Section 4.2.

4.1 Line-Sweep Ambient Obscurance

The horizon-based ambient occlusion (HBAO) [5] method closely relates to our method. HBAO is an established SSAO method and often used as a reference for new SSAO methods. It takes a given number of samples, M, from each receiver point along K discrete azimuthal directions. The highest horizon from these samples is found along each azimuthal direction, and the occlusion cast by it is weighted according to a distance dependent falloff function. The physically based treatment of geometry in this method produces more accurate results and scales better quality-wise than methods which do not check whether the sampled occluders are visible (unoccluded by a nearer occluder along the same direction) to the receiver point. This approach, however, comes with a costly execution time because relatively many samples have to be taken. Specifically, $O(K \cdot M)$ operations are performed per pixel and the AO radius often has to be limited to keep M manageable.

The method in [P3] targets the time complexity of this approach and achieves essentially the same results in $O(K)$ operations per pixel. We start from the insight of [P1] that the global horizons for each height field point can be extracted in $O(K)$ time per receiver. The geometrical convex hull subset of points along lines on the height field are kept track of in [P1], which are only usable for determining the global horizon. For the purpose of calculating ambient obscurance, the point that casts the global horizon might be far away from the receiver, and as occlusion should be weighted according to a falloff function, it may cast an insignificant amount of occlusion.

Therefore, instead of generating a geometric convex hull, we generate an *obscurance hull* which essentially stores a set of points according to how much falloff attenuated occlusion they cast. From the obscurance hull the point casting the largest occlusion in each azimuthal direction can be extracted in order to approximate AO. Because the amount of occlusion any single point casts is dependent on the direction of the receiver's normal, defining the obscurance hull is non-trivial. We evaluate different strategies to define the hull and to evaluate AO and propose a method which produces results virtually identical to HBAO.

Furthermore, we suggest an acceleration strategy which employs sparse sampling and an edge-aware gather phase to produce slightly blurred results but at faster render times. Finally we demonstrate how, during line traversal, having to interpolate the depth data can be avoided by carefully selecting sampling patterns. Using the suggested sampling patterns the method becomes suitable for evaluating AO on depth fields which generally cannot be assumed continuous.

4.2 Far-Field Screen-Space Ambient Obscurance

The method [P4] has a $O(K \cdot log(N))$ time complexity on N^2 depth fields similarly to previous multi-resolution SSAO methods, but achieves higher quality results. Previous methods which rely on point sampling the depth field have trouble scaling to far-field effects because maintaining a certain sampling density quickly escalates the number of samples required to cover a large screen-space radius. Sparse sampling, on the other hand, is prone to produce various undersampling artifacts because occluders can be missed.

When mipmaps are used to cover a larger area of the depth field when sampling farther from the receiver, another problem arises: filtering the lower resolution mip levels corrupts the view-dependent features of the depth field which are important for computing accurate AO. We tackle this problem by pre-processing the depth field in two fast phases that are of $O(K)$ time complexity. These two phases and the final evaluation phase are illustrated in Figure 4.1. The first scanning phase locates local peaks in the depth field and stores an intermediate representation of them. Second is the prefix sum phase which prepares the intermediate data for fast occluder integration across the whole width of each azimuthal sector. After these two

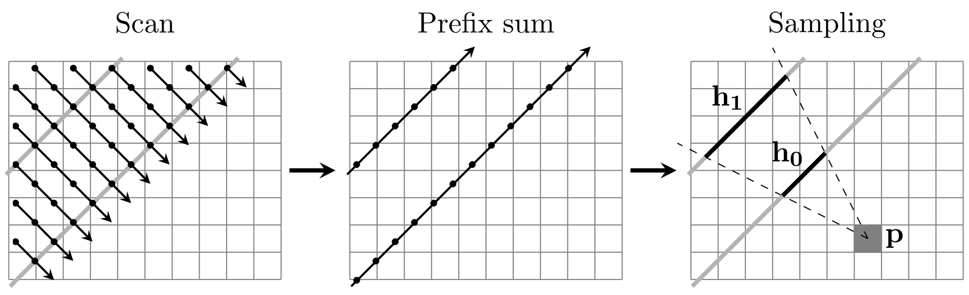


Figure 4.1: The three phases of [P4]: Firstly the height field is scanned and a view-dependent representation of local peaks of the height field is created. Secondly the representation is turned into a set of prefix sums. Thirdly points $\mathbf{h_i}$ are reconstructed that accurately represent features important for AO, averaged across the sector width.

pre-processing phases ambient occlusion is evaluated in the usual fashion where AO is calculated for each screen pixel separately. Instead of directly sampling the depth field or its mipmaps, scene points are reconstructed from the processed intermediate data. These scene points are tailored to capture depth field features that are specifically important for AO, and only a small number of these samples need to be used for an accurate representation of the surrounding geometry.

A resolution hierarchy for the intermediate representation can also be generated and features are still accurately represented without distortions that are typical of averaging used in previous state-of-the-art methods. Overall our method is able to integrate ambient occlusion effects accurately from the entire framebuffer for each receiver point using only a small number of reconstructed scene points. Our second contribution in [P4] is an obscurance estimator which accurately estimates Equation 1.3 and therefore converges to a ray traced reference result.

4.3 Results

The method proposed by [P3] achieves results similar to HBAO but an order of magnitude faster. HBAO's performance depends on the AO radius in screen-space whereas our method executes in constant time and quality regardless of the AO effect's radius. As the AO radius may become arbitrarily large in screen-space, previous methods often resort to undersampling to keep render times under control. Our method, on the other hand, avoids this and produces smooth ambient occlusion. Figure 4.2 shows the Šibenik Cathedral rendered by our method in [P3], and Figure 4.3 contrasts our

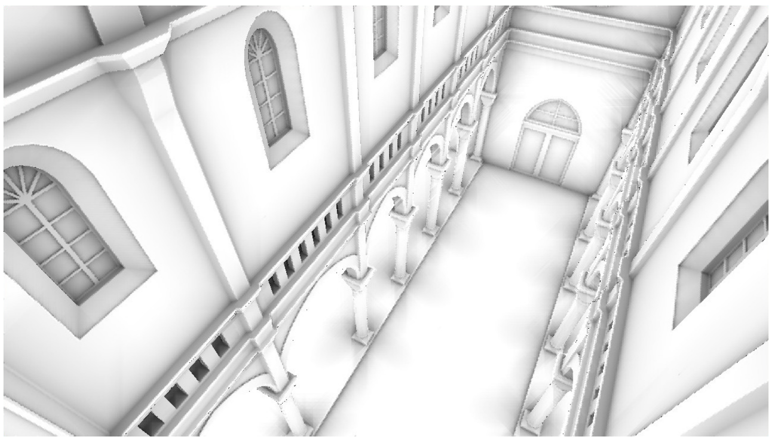


Figure 4.2: SSAO rendered using the method in [P3] in the Šibenik Cathedral model in 8 azimuthal directions ($K = 8$) at 1280×720 resolution in 0.7 ms on an NVIDIA GeForce GTX 480.

Screen resolution	[P3]	HBAO
800×600	1.49 ms	10.5 ms
1280×720	2.56 ms	24.2 ms
1920×1080	5.24 ms	92.5 ms
2560×1600	9.58 ms	249 ms

Table 4.1: Render times of our method [P3] and HBAO at different resolutions with a 20 % guard band. The scene used is shown to the bottom in Figure 4.3.

method with HBAO. Scaling with respect to screen resolution is listed in Table 4.1.

The method in [P4] evaluates AO using information from the entire framebuffer and due to the way the obscurance estimator is constructed it is able to use any falloff function such that the complexity of the falloff function does not affect execution time. At the same number of sample points the intermediate geometry representation achieves an order of magnitude smaller errors than average mipmaps that are used by the previous state-of-the-art methods. Figure 4.4 compares our sampling strategy against the mipmap sampling using the same number of scene samples. Overall [P4] achieves render results that are within a few percent of a ray traced screen-space reference result at real-time frame rates. Figure 4.5 shows another scene rendered by our method next to a screen-space ray traced reference.

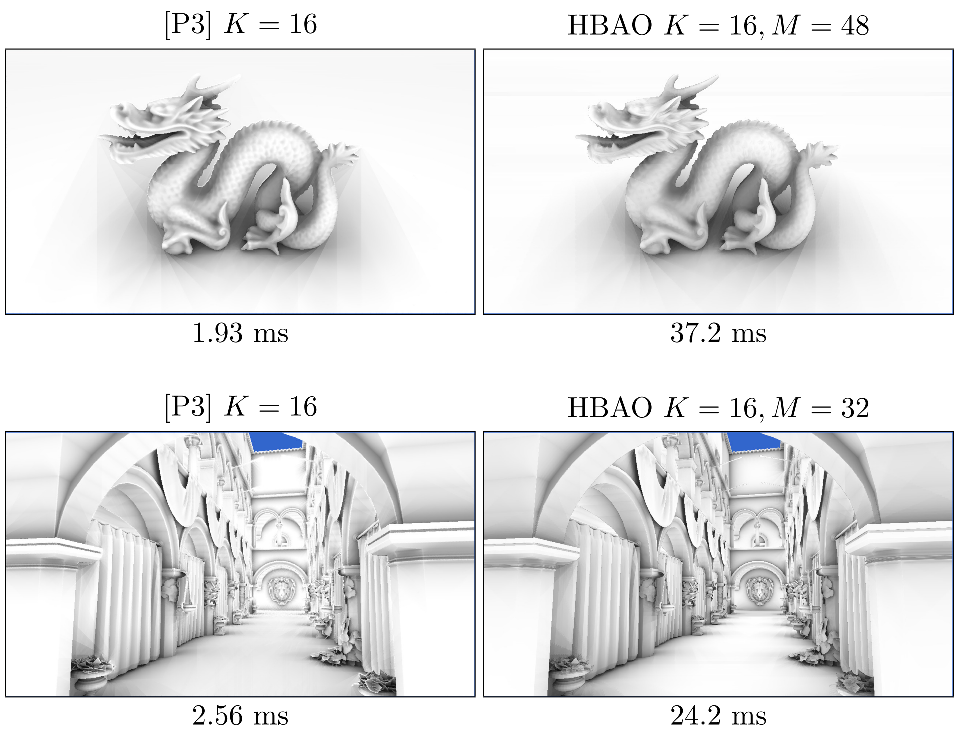


Figure 4.3: The Stanford Dragon, courtesy of Stanford Computer Graphics Laboratory, and the Sponza scene, courtesy of Frank Meinl, Crytek, rendered at 1280×720 by the method in [P3] and by HBAO with the render times shown below each image. For HBAO the number of steps along each of the K azimuthal directions is denoted by M whereas our method has a constant cost per direction.

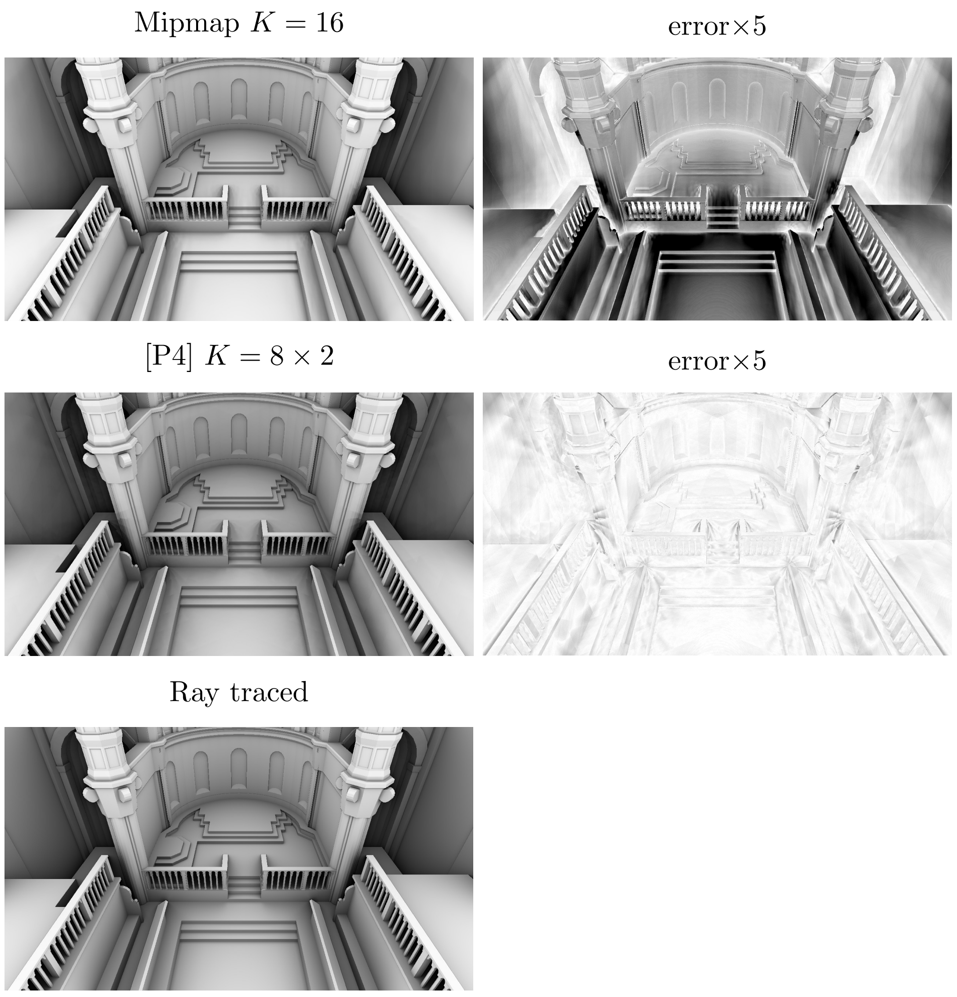


Figure 4.4: SSAO evaluated by our geometry [P4] and by mipmap geometry and their respective error images (white = 0 %, black ≥ 20 %, brighter is better) of the ray traced reference. Both methods use the obscurance estimator from [P4].

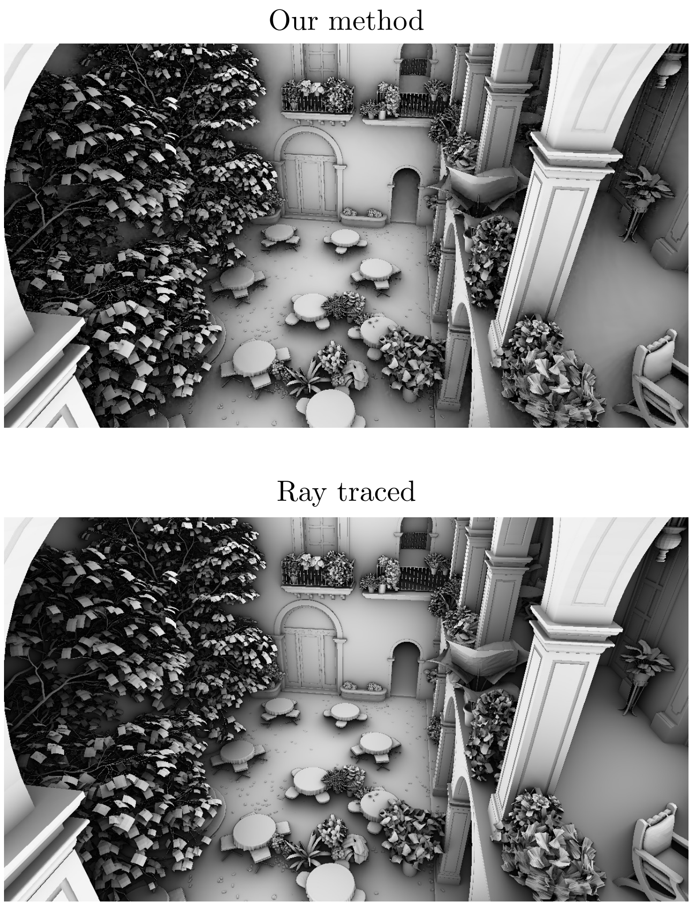


Figure 4.5: Top: far-field SSAO component rendered by our method [P4] in 4.6 ms in a 1280×720 framebuffer on an AMD Radeon HD 7970. Bottom: ray traced screen-space reference result.

Chapter 5

Conclusion and future work

5.1 Conclusion

Plausible illumination, both direct and indirect, is the most important part in photorealistic rendering. Determining visibility of the light sources and the scene geometry for each visible scene point is the most computationally challenging stage of illumination computation. In Chapter 3, we have presented two novel methods that lower the computational complexity of visibility calculations on height field geometry. The first method, [P1], lowers the time complexity of calculating the horizon map on $N \times N$ height fields to $O(1)$ per point compared to $O(N)$ in previous work. The horizon map can be used to calculate the visibility of arbitrary direct light sources outside of the height field. The second method, [P2], lowers the time complexity of calculating intervisibility from $O(N)$ of previous work to between $O(N^{0.01})$ and $O(N^{0.65})$ in common types of height fields, and is content dependent. In 1024^2 height fields two orders of magnitude less iterations are required as compared to previous work. The intervisibility information can be used to produce self-illumination and indirect illumination effects in height fields.

Ambient occlusion or obscurance is a very popular method to plausibly and efficiently reproduce difficult global illumination effects, and screen-space evaluation of it has been adopted by most current real-time renderers. In Chapter 4, we presented two methods that improve render times and render quality of SSAO. Previous SSAO methods sample the depth field around each receiver point whereas the method introduced by [P3] uses line sweeps to calculate occlusion for multiple points in one process. This reduces the time complexity to evaluate SSAO and achieves an order of magnitude reduction in render times. Finally, [P4] improves the descriptiveness of samples that are taken from the depth field by pre-processing the depth data into an intermediate form that is tailored to capture features that are important for AO. Combined with an improved way to evaluate the ambient occlusion integral, the method achieves higher quality renderings than previous SSAO methods but does not increase render times.

5.2 Future work

This thesis has introduced new visibility algorithms which show significantly lower time complexity than methods that are used for generic scene geometry today. As time complexity is the fundamental measure of efficiency and scalability, we hope that these ideas can be generalized to arbitrary geometries instead of being limited to height fields. This is not a trivial task, but worth investigating further as any reduction in time complexity has the potential to have a long-lasting impact on future algorithms.

Another way to apply these algorithms to more generic geometry is via the screen-space approach, which has gained recent interest in both academia and industry. While geometry available in the depth buffer is incomplete, there has been promising recent activity towards solving or at least mitigating this problem by either making assumptions about the missing regions of the scene or by completing geometry with multiple depth layers or auxiliary views. It remains to be seen whether the limitations of screen-space geometry can be overcome to a sufficient extent in the future or whether higher time complexity methods that work on generic geometries eventually replace them by becoming fast enough for real-time evaluation in practical scenes.

Bibliography

[1] AMD. *AMD Graphics Core Next (GCN) Architecture white paper*, June 2012.

[2] Thomas Annen, Zhao Dong, Tom Mertens, Philippe Bekaert, Hans-Peter Seidel, and Jan Kautz. Real-time, all-frequency shadows in dynamic scenes. In *ACM SIGGRAPH '08*, 2008.

[3] Okan Arikan, David A Forsyth, and James F O'Brien. Fast and detailed approximate global illumination by irradiance decomposition. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 1108–1114. ACM, 2005.

[4] Louis Bavoil and Miguel Sainz. Multi-layer dual-resolution screen-space ambient occlusion. In *SIGGRAPH '09 Talks*. ACM, 2009.

[5] Louis Bavoil, Miguel Sainz, and Rouslan Dimitrov. Image-space horizon-based ambient occlusion. In *SIGGRAPH '08 Talks*, 2008.

[6] Jeffrey C Beran-Koehn and Mark J Pavicic. A cubic tetrahedral adaptation of the hemi-cube algorithm, 1991.

[7] Jiří Bittner, Peter Wonka, and Michael Wimmer. Fast exact from-region visibility in urban scenes. In *Proceedings Eurographics Symposium on Rendering*, pages 223–230, June 2005.

[8] Chas Boyd. The directx 11 compute shader. In *ACM SIGGRAPH*, 2008.

[9] Rob Bredow and Sony Pictures Imageworks. Renderman on film. *SIGGRAPH 2002 Course Notes. Course*, 16(6):7, 2002.

[10] M Bunnell. *Dynamic ambient occlusion and indirect lighting*, pages 223–233. Addison-Weseley Professional, 2005.

[11] Per H Christensen. Note# 35: Ambient occlusion, image-based illumination, and global illumination. *PhotoRealistic RenderMan Application Notes*, 2002.

[12] Per H Christensen. Global illumination and all that. *SIGGRAPH 2003 course notes*, 9:31–72, 2003.

[13] Michael F Cohen and Donald P Greenberg. The hemi-cube: A radiosity solution for complex environments. In *ACM SIGGRAPH Computer Graphics*, volume 19, pages 31–40. ACM, 1985.

[14] Daniel Cohen-Or, Yiorgos L Chrysanthou, Claudio T. Silva, and Frédo Durand. A survey of visibility for walkthrough applications. *Visualization and Computer Graphics, IEEE Transactions on*, 9(3):412–431, 2003.

[15] Daniel Cohen-or and Amit Shaked. Visibility and dead-zones in digital terrain maps. *Computer Graphics Forum*, 14:171–180, 1995.

[16] Richard Cole and Micha Sharir. Visibility problems for polyhedral terrains. *J. Symb. Comput.*, 7(1):11–30, 1989.

[17] Carsten Dachsbacher and Marc Stamminger. Reflective shadow maps. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 203–231. ACM, 2005.

[18] Carsten Dachsbacher, Marc Stamminger, George Drettakis, and Frédo Durand. Implicit visibility and antiradiance for interactive global illumination. *ACM Trans. Graph.*, 26(3):61, 2007.

[19] Leila De Floriani and Paola Magillo. Computing point visibility on a terrain based on a nested horizon structure. In *SAC '94: Proceedings of the 1994 ACM symposium on Applied computing*, pages 318–322, New York, NY, USA, 1994. ACM.

[20] Paul Debevec. Rendering synthetic objects into real scenes: bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *Proceedings SIGGRAPH '98*, pages 189–198, New York, NY, USA, 1998. ACM.

[21] Michael Deering, Stephanie Winner, Bic Schediwy, Chris Duffy, and Neil Hunt. The triangle processor and normal vector shader: a vlsi system for high performance graphics. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '88, pages 21–30, New York, NY, USA, 1988. ACM.

[22] Rouslan Dimitrov, Louis Bavoil, and Miguel Sainz. Horizon-split ambient occlusion. In *I3D '08: Proceedings of the 2008 symposium on Interactive 3D graphics and games*, New York, NY, USA, 2008. ACM.

[23] Philip Dutre, Kavita Bala, Philippe Bekaert, and Peter Shirley. *Advanced Global Illumination.* AK Peters Ltd, 2006.

[24] Dominic Filion and Rob McNaughton. Effects & techniques. In *ACM SIGGRAPH 2008 Games*, SIGGRAPH '08, pages 133–164, New York, NY, USA, 2008. ACM.

[25] Jeremy Fishman, Herman Haverkort, and Laura Toma. Improved visibility computation on massive grid terrains. In *GIS '09: Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 121–130, New York, NY, USA, 2009. ACM.

[26] Leila De Floriani and Paola Magillo. Visibility algorithms on triangulated digital terrain models. *International Journal of Geographical Information Systems*, 8(1):13–41, 1994.

[27] Stichting Blender Foundation. Blender 2.63. `http://blender.org/`, 2013.

[28] Wm Randolph Franklin and Clark Ray. Higher isnt necessarily better: Visibility algorithms and experiments. In *Advances in GIS research: sixth international symposium on spatial data handling*, volume 2, pages 751–770. Edinburgh, 1994.

[29] Wm Randolph Franklin, Clark K Ray, and Shashank Mehta. Geometric algorithms for siting of air defense missile batteries. *A], Research Project for Battle*, (2756), 1994.

[30] Cindy M Goral, Kenneth E Torrance, Donald P Greenberg, and Bennett Battaile. Modeling the interaction of light between diffuse surfaces. In *ACM SIGGRAPH Computer Graphics*, volume 18, pages 213–222. ACM, 1984.

[31] Kris Gray. *Microsoft DirectX 9 programmable graphics pipeline.* Microsoft Pr, 2003.

[32] Khronos OpenCL Working Group et al. The opencl specification. *A. Munshi, Ed*, 2008.

[33] Tom R. Halfhill. Parallel Processing with CUDA. *Microprocessor Report*, January 2008.

[34] Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch, and François Sillion. A survey of real-time soft shadows algorithms, dec 2003.

[35] Wolfgang Heidrich, Katja Daubert, Jan Kautz, and Hans-Peter Seidel. Illuminating micro geometry based on precomputed visibility. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 455–464. ACM Press/Addison-Wesley Publishing Co., 2000.

[36] Thai-Duong Hoang and Kok-Lim Low. Efficient screen-space approach to high-quality multiscale ambient occlusion. *The Visual Computer*, 28(3):289–304, 2012.

[37] Jared Hoberock and Yuntao Jia. High-quality ambient occlusion. *GPU gems*, 3:257–274, 2007.

[38] Jing Huang, Tamy Boubekeur, Tobias Ritschel, Matthias Hollnder, and Elmar Eisemann. Separable approximation of ambient occlusion. In *Eurographics 2011 - Short papers*, 2011.

[39] Henrik Wann Jensen. *Realistic image synthesis using photon mapping.* AK Peters, Ltd., 2001.

[40] James T. Kajiya. The rendering equation. In *Proceedings SIGGRAPH '86*, pages 143–150, New York, NY, USA, 1986. ACM.

[41] Anton Kaplanyan. CryENGINE 3: Reaching the speed of light. In Tatarchuk Natalya, editor, *ACM SIGGRAPH 2010 Advances in Real-time Rendering Course*, 2010.

[42] Branko Kaučič and Borut Zalik. Comparison of viewshed algorithms on regular spaced points. In *SCCG '02: Proceedings of the 18th spring conference on Computer graphics*, pages 177–183, New York, NY, USA, 2002. ACM.

[43] Alexander Keller. Instant radiosity. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 49–56. ACM Press/Addison-Wesley Publishing Co., 1997.

[44] Daniel Kersten, Pascal Mamassian, and David C Knill. Moving cast shadows induce apparent motion in depth. *PERCEPTION-LONDON-*, 26:171–192, 1997.

[45] Janne Kontkanen and Samuli Laine. Ambient occlusion fields. In *Proceedings of ACM SIGGRAPH 2005 Symposium on Interactive 3D Graphics and Games*, pages 41–48. ACM Press, 2005.

[46] Eric P Lafortune and Yves D Willems. Bi-directional path tracing. In *Proceedings of CompuGraphics*, volume 93, pages 145–153, 1993.

[47] Samuli Laine and Tero Karras. Two methods for fast ray-cast ambient occlusion. *CGF: Proceedings of EGSR 2010*, 29(4), 2010.

[48] Hayden Landis. Production-ready global illumination. *Siggraph course notes*, 16(2002):11, 2002.

[49] MS Langer and SW Zucker. Shape-from-shading on a cloudy day. *JOSA A*, 11(2):467–478, 1994.

[50] Bradford James Loos and Peter-Pike Sloan. Volumetric obscurance. In *Proceedings of I3D 2010*. ACM, 2010.

[51] Pascal Mamassian, David C Knill, and Daniel Kersten. The perception of cast shadows. *Trends in cognitive sciences*, 2(8):288–295, 1998.

[52] Nelson Max. Horizon mapping: shadows for bump-mapped surfaces. *The Visual Computer*, 4(2):109–117, March 1988.

[53] Morgan McGuire. Ambient occlusion volumes. In *Proceedings of High Performance Graphics 2010*, June 2010.

[54] Morgan McGuire, Michael Mara, and David Luebke. Scalable ambient obscurance. In *High-Performance Graphics 2012*, June 2012.

[55] Morgan McGuire, Brian Osman, Michael Bukowski, and Padraic Hennessy. The alchemy screen-space ambient obscurance algorithm. In *Proc. HPG*, HPG '11, pages 25–32. ACM, 2011.

[56] Àlex Méndez-Feliu and Mateu Sbert. From obscurances to ambient occlusion: A survey. *The Visual Computer*, 25(2):181–196, 2009.

[57] Gavin Miller. Efficient algorithms for local and global accessibility shading. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 319–326. ACM, 1994.

[58] Martin Mittring. Finding next gen: Cryengine 2. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, pages 97–121. ACM, 2007.

[59] George Nagy. Terrain visibility. *Computers & Graphics*, 18(6):763–773, 1994.

[60] Greg Nichols, Jeremy Shopf, and Chris Wyman. Hierarchical image-space radiosity for interactive global illumination. In *Computer Graphics Forum*, volume 28, pages 1141–1149. Wiley Online Library, 2009.

[61] Derek Nowrouzezahrai and John Snyder. Fast global illumination on dynamic height fields. *Computer Graphics Forum: Eurographics Symposium on Rendering*, jun 2009.

[62] Chris Oat and Pedro Sander. Ambient aperture lighting. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses*, pages 143–152, New York, NY, USA, 2006. ACM.

[63] K. Onoue, N. Max, and T. Nishita. Real-time rendering of bumpmap shadows taking account of surface curvature. In *Cyberworlds, 2004 International Conference on*, pages 312–318, 2004.

[64] Georgios Papaioannou, Maria Lida Menexi, and Charilaos Papadopoulos. Real-time volume-based ambient occlusion. *Visualization and Computer Graphics, IEEE Transactions on*, 16(5):752–762, 2010.

[65] Fabio Policarpo and Manuel M Oliveira. Relief mapping of non-height-field surface details. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 55–62. ACM, 2006.

[66] Christoph Reinbothe, Tamy Boubekeur, and Marc Alexa. Hybrid ambient occlusion. *EUROGRAPHICS 2009 Areas Papers*, 2009.

[67] Tobias Ritschel, Carsten Dachsbacher, Thorsten Grosch, and Jan Kautz. The state of the art in interactive global illumination. *Computer Graphics Forum*, 31, February 2012.

[68] Tobias Ritschel, Thorsten Grosch, and Hans-Peter Seidel. Approximating dynamic global illumination in image space. In *I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 75–82, New York, NY, USA, 2009. ACM.

[69] Marc Ruiz, Lázló Szirmay-Kalos, Tamás Umenhoffer, Imma Boada, Miquel Feixas, and Mateu Sbert. Volumetric ambient occlusion for volumetric models. *The Visual Computer*, 26(6-8):687–695, 2010.

[70] Perumaal Shanmugam and Okan Arikan. Hardware accelerated ambient occlusion techniques on gpus. In *Proc. I3D '07*. ACM, 2007.

[71] CN Shen and George Nagy. Autonomous navigation to provide long-distance surface traverses for mars rover sample return mission. In *Intelligent Control, 1989. Proceedings., IEEE International Symposium on*, pages 362–367. IEEE, 1989.

[72] Ying Shen, Li Lin, Mei Yang, and Gao Yurong. Viewshed computation based on los scanning. volume 2, pages 984 –987, dec. 2008.

[73] Dave Shreiner et al. *OpenGL programming guide: the official guide to learning OpenGL, versions 3.0 and 3.1.* Pearson Education, 2009.

[74] Dave Shreiner et al. *OpenGL programming guide: the official guide to learning OpenGL, version 4.3.* Pearson Education, 2013.

[75] Michael Sipser. *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston, 2006.

[76] Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 527–536. ACM, 2002.

[77] Peter-Pike J. Sloan and Michael F. Cohen. Interactive horizon mapping. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, pages 281–286, London, UK, 2000. Springer-Verlag.

[78] John Snyder and Derek Nowrouzezahrai. Fast soft self-shadowing on dynamic height fields. *Computer Graphics Forum: Eurographics Symposium on Rendering*, June 2008.

[79] Cyril Soler, Olivier Hoel, and Frank Rochet. A deferred shading pipeline for real-time indirect illumination. In *ACM SIGGRAPH 2010 Talks*, SIGGRAPH '10, pages 18:1–18:1, New York, NY, USA, 2010. ACM.

[80] A. James Stewart. Fast horizon computation at all points of a terrain with visibility and shading applications. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):82–93, 1998.

[81] L Szirmay-Kalos and T Umenhoffer. Displacement mapping on the gpu — state of the art. volume 27, pages 1567–1592, 2008.

[82] László Szirmay-Kalos, Tamás Umenhoffer, Balázs Tóth, László Szécsi, and Mateu Sbert. Volumetric ambient occlusion for real-time rendering and games. *Computer Graphics and Applications, IEEE*, 30(1):70–79, 2010.

[83] Natalya Tatarchuk. Practical parallax occlusion mapping with approximate soft shadows for detailed surface rendering. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses*, pages 81–112, New York, NY, USA, 2006. ACM.

[84] V. Timonen. Real-time visual simulation of volumetric surfaces. Master's thesis, University of Kuopio, November 2006.

[85] Kostas Vardis, Georgios Papaioannou, and Athanasios Gaitatzes. Multi-view ambient occlusion with importance sampling. In *Proc. i3D*, I3D '13, pages 111–118, 2013.

[86] Eric Veach and Leonidas J Guibas. Metropolis light transport. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 65–76. ACM Press/Addison-Wesley Publishing Co., 1997.

[87] Ingo Wald, William R Mark, Johannes Günther, Solomon Boulos, Thiago Ize, Warren Hunt, Steven G Parker, and Peter Shirley. State of the art in ray tracing animated scenes. In *Computer Graphics Forum*, volume 28, pages 1691–1722. Wiley Online Library, 2009.

[88] Leonard R Wanger, James Ferwerda, and Donald P Greenberg. Perceiving spatial relationships in computer-generated images. *IEEE Computer Graphics and Applications*, 12(3):44–58, 1992.

[89] Lance Williams. Casting curved shadows on curved surfaces. *SIGGRAPH Comput. Graph.*, 12(3):270–274, 1978.

[90] Lance Williams. Pyramidal parametrics. *SIGGRAPH Comput. Graph.*, 17(3):1–11, July 1983.

[91] Craig M Wittenbrink, Emmett Kilgariff, and Arjun Prabhu. Fermi gf100 gpu architecture. *Micro, IEEE*, 31(2):50–59, 2011.

[92] Sergej Zhukov, Andrej Inoes, and Grigorij Kronin. An Ambient Light Illumination Model. In George Drettakis and Nelson Max, editors, *Rendering Techniques '98*, Eurographics, pages 45–56. Springer-Verlag Wien New York, 1998.

Part II

Original publications

Publication P1

Ville Timonen and Jan Westerholm. Scalable Height Field Self-Shadowing. *Computer Graphics Forum*, 29(2), pages 723–731, 2010. *Eurographics Conference 2010.* 3rd Best Paper.

DOI: 10.1111/j.1467-8659.2009.01642.x

EUROGRAPHICS 2010 / T. Akenine-Möller and M. Zwicker
(Guest Editors)

Volume 29 (*2010*), *Number 2*

Scalable Height Field Self-Shadowing

Ville Timonen and Jan Westerholm

Åbo Akademi University

Abstract

We present a new method suitable for general purpose graphics processing units to render self-shadows on dynamic height fields under dynamic light environments in real-time. Visibility for each point in the height field is determined as the exact horizon for a set of azimuthal directions in time linear in height field size and the number of directions. The surface is shaded using the horizon information and a high-resolution light environment extracted on-line from a high dynamic range cube map, allowing for detailed extended shadows. The desired accuracy for any geometric content and lighting complexity can be matched by choosing a suitable number of azimuthal directions. Our method is able to represent arbitrary features of both high- and low-frequency, unifying hard and soft shadowing. We achieve 23 fps on 1024×1024 height fields with 64 azimuthal directions under a 256×64 environment lighting on an Nvidia GTX 280 GPU.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Computer Graphics—Color, shading, shadowing, and texture

1. Introduction

Shadowing is a major contributor to photorealism in computer graphics giving important cues about objects and their environment. Without shadows it is often hard to perceive the shapes of objects or their relative position and magnitude. Hard shadows communicate the exact shape while soft shadows convey distance; a shadow softens gradually as the distance from the caster increases. Without shadows also light sources — their distribution, size, and intensity — become ambiguous. For example, in the cathedral environment of the first row in Figure 9, it can be seen from the shadows that there are multiple small openings from which light enters, while in Figure 10, the elevation and the size of the orange light source can be inferred from the shadows.

In this article we provide a method for general self-shadowing on height field geometry under complex lighting environments. The method takes a height map and an environment light map as input, and produces a color map describing output radiance of the height field. The surface is assumed to exhibit Lambertian reflectance under direct lighting.

Height fields describe geometry by defining the elevation of a plane as a function of $N \times N$ surface coordinates. This allows the geometry to be stored into a scalar 2D texture, and accessed efficiently using graphics hardware. The geometric content of an N^2 size height field can be represented by a polygon mesh of $2(N-1)^2$ triangles. Our method represents (infinitely distant) input lighting as a function of elevation and azimuthal angles. This function can be computed online from cube maps.

GPGPU (general-purpose computation on graphics processing unit) technology allows a more flexible usage of graphics hardware by providing direct access to computing resources. The suitability of GPGPU for raster graphics is discussed in [Lef07]. We have decided to use CUDA [Hal08] and OpenGL for the implementation of our method.

As our primary contribution we present a new algorithm which by utilizing coherency between adjacent samples in the height field along an azimuthal direction is able to calculate the horizon angles for all the points in any given direction in an operation that has linear time complexity in the height field size. The horizon angles are obtained losslessly, i.e., every height field value is taken into consideration in the given direction. Therefore the algorithm can accurately shadow height fields of arbitrary geometric content: sharp edges, thin and tall features, and high and low frequency details with a stable level of performance, implying scalability.

We also present a robust analytically defined model for direct lighting that utilizes the previously generated horizon information. It requires a precalculation phase which is fast and can be executed for every frame. The lighting evaluation is performed for each height field point in each azimuthal direction, thus taking a time linear in the height field size and in the number of azimuthal directions. The lighting model is capable of capturing small point lights, area lights, and arbitrary light environments with high precision. Input lighting can be specified as HDR (high dynamic range) cube maps, which can be rendered or animated on-the-fly. There is also no need for separate representations for low and high frequency light sources, nor has lighting environment complexity any effect on performance. Figure 10 demonstrates a high-resolution height field with shadowing accuracy not previously achieved in real-time.

2. Related work

Height field self-visibility can be determined by the horizon silhouette at each point in the field for a set of azimuthal directions. *Horizon mapping* [Max88] utilizes this observation to shadow bump-mapped surfaces, and it has also been used to shadow static height fields in real-time [SC00]. Methods which consider all points in the height field as occluders for one receiver point — not only the ones lying in discrete azimuthal directions — also exist [Ste98], but they are unsuitable for real-time applications.

Recently, real-time methods for dynamic height fields have been proposed [SN08] [NS09] as well. These methods generate the horizon information on-line by sampling the height field in azimuthal directions for each point separately to find the dominant occluder. For a height field of size N^2, this is an $O(N^3)$ operation for one azimuthal direction if all height field points along the direction are considered for each receiver point. To diminish the sampling load, multiple resolutions of the height field (a multi-resolution height pyramid) are used to satisfy sampling at different distances from the receiver. This approximation is suitable for producing soft shadows, but can not produce high-resolution shadows that extend far from the caster. Our method extracts the exact horizon in lesser time complexity, $O(N^2)$, for any given azimuthal direction, and therefore is able to cast extended high-resolution shadows while retaining high performance. [SN08] and [NS09] use low-order (4th) spherical harmonics — that can be represented by only 16 coefficients — to model input lighting and occlusion, which is suitable for soft-shadowing but incapable of capturing sharp shadows or complex light environments. We model input lighting as a high-resolution environment, which may contain both high- and low-frequency features.

Methods based on calculating *ambient occlusion* are widely used to render soft-shadowing effects on objects [Bun05] [KL05]. These methods usually approximate the geometry surrounding the receiver to calculate the proportion of the visible environment. A family of ambient occlusion methods approximate occlusion in image or screen space [BSD08] [DBS08] [Mit07] [SA07] [BS09] by treating the depth buffer as a height field. As the geometry visible in the depth buffer is a subset of the total affecting geometry, and samples far away from the receiver are unlikely to represent continuous geometry, these methods can only calculate a local approximation of the occlusion by sampling near the receiver. Screen space ambient occlusion has also been extended to render more complex lighting effects, such as indirect illumination and directional lighting [RGS09].

Shadow mapping [Wil78] produces hard shadows from objects by comparing receiver distance from the light and viewer point of view. Although originally used to render shadows from point lights, methods [HLHS03] exist for softening the shadows. These methods however require one pass for each light, and become unsuitable for real-time applications under complex light environments. To render shadows under light environments, [ADM*08] decomposes a cube environment map into multiple light sources, generates shadow maps for each of these using a fast algorithm, and fuses the results. While this is suitable for arbitrary polygon meshes, our method achieves faster performance per amount of geometry and higher resolution shadows for complex light environments also extracted from cube maps, but is specialized to height field geometry.

Height field geometry is usually rendered using two types of methods. *Displacement mapping* methods render the height field as a grid of polygons whose z-components are displaced according to their height value. *Relief mapping* methods render usually only one quad for a surface, and use iterative algorithms in fragment programs to find the first intersection between the viewing ray and the height field. Displacement mapping methods are sensitive to the amount of geometry, whereas relief mapping methods are sensitive to the output image size. A review of these methods is presented in [SKU08]. Relief mapping methods can also render hard shadows by determining if a light source is occluded by the height field by searching for an intersection between the height field and the light source. In [Tat06] this is extended to produce approximate soft shadows for area lights. Shadowing light environments using these methods would require an occlusion search for each light, and without optimizations such as using a height pyramid this would have worse performance than in [SN08].

3. Summary of core ideas

The problem setting of height field self-shadowing is as follows. We would like to know the horizon for each point in the height field as a function of azimuthal direction, in order to determine the amount of light coming from the environment. One way to approximate this is to determine the horizon for a set of discrete azimuthal (with respect to the height field plane) directions. Therefore, for each point in the height field, and for each azimuthal direction from that point, we need to find the point that occludes the horizon the most. Intuitively, the problem can be thought of as standing at each point in the height field, turning around 360 degrees, and finding the edge of the sky. What makes this computationally challenging is the fact that the highest horizon can be cast by any point in the azimuthal plane (the plane perpendicular to the height field's ground plane oriented towards the azimuthal direction).

We present a solution to this problem by describing, in Section 4, a method for traversing the height field in a way that facilitates very efficient occluder extraction. The occluder extraction is described in detail in Section 5. What is new in our approach is the linear-time algorithm that is able to find the horizon both more accurately and an order of magnitude faster than before. We demonstrate its efficiency on present day GPGPUs.

We use CUDA terminology [NVI09b] for the different types of memory and concepts such as *threads* and *kernels*. The key idea in our method is to use threads to calculate horizons for whole lines of height field points instead of calculating them independently for each one. Each thread keeps a robust representation of the height samples it has processed so far along the line in a *convex hull subset*. This representation can be incrementally updated and used to extract horizon angles in such a way that the total time complexity for processing a line of n samples is $O(n)$. There are no approximations involved in determining the horizon angle except those coming from the inherent precision of the data types used. We are not aware of this method having been introduced in a field outside computer graphics before.

A practical use for this horizon information is to calculate the amount of incident light a point on the surface receives from its environment. As the second part of our contribution, in Section 6, we describe a model for direct environment lighting that features unbounded accuracy and can therefore capture the full detail of the generated self-shadowing information. The lighting model is derived from an analytical definition, first for uniform ambient light environments. We then extend the concept to precalculate arbitrary environment lighting into a 2D table that can be indexed by normal and horizon angles during evaluation. The table represents accumulated incident light weighed by the angle of the surface normal's projection to the azimuthal plane, and limited by the horizon angle. The final lighting can then be accurately evaluated by multiplying a sample from this table by the length of the projected normal. The precalculation phase is fast and accepts cube maps as input.

We finish the paper by presenting results of performance, scalability, and shadowing accuracy. We are able to extract occluders for dynamic high-resolution height fields, and utilize this information to light the surface from dynamic high-resolution environment light maps in real-time.

4. Computation framework

Graphics APIs offer programmability of the hardware by allowing the application to supply its own programs for specific stages of the rendering pipeline. Because rasterisation is not currently exposed to the application, a fragment program can output only one value to each output buffer into a predefined position. Our method relies on each calculation outputting a series of values into different positions in the output buffer, and is therefore an unsuitable candidate for a fragment program. This is the main reason we decided to implement our algorithm using GPGPU technology. In this chapter we describe the rasterisation used and its thread topology.

A height field describes a 2-dimensional surface in 3 dimensions, $(x, y, h(x, y))$, where h is given at discrete coordinates, i.e. as a height map. By describing the surface this way, the geometry along a straight line in the (x, y) plane can be traversed by sampling the height function at corresponding points. To sample the height function, we use bi-linear filtering [NVI09b] provided by graphics hardware.

We determine the occlusion as the horizon angle from zenith at each height field point in discrete azimuthal directions. The horizon is determined by another point in the azimuthal direction whose height-to-distance ratio (slope) is the highest as measured from the receiver point. Instead of extracting horizon angles independently for each height field point the height field is marched along the azimuthal directions in parallel lines (Figure 2). An unfinished occlusion sweep for one azimuthal direction is shown in Figure 3.

For a height field of size $N \times N$ and for one azimuthal direction, between N and $\sqrt{2}N$ lines are processed, one thread for each line. A thread steps through $1 \ldots \sqrt{2}N$ height samples along its path, so that a total of N^2 evenly spaced samples are processed for the direction, covering the height field. Each thread is responsible for keeping a representation of the geometry along the path up to the latest sample. From this data, the thread decides for each new height sample a horizon angle backwards along its path, as illustrated in Figure 1.

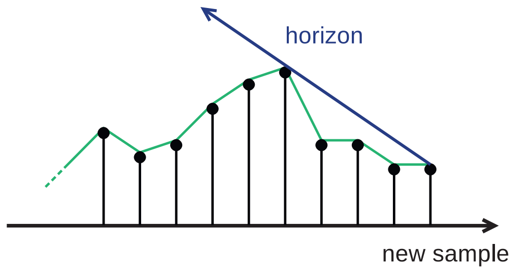


Figure 1: *Each thread supplies a horizon value from its internal representation of the height function for each height sample in its path.*

As it is critical for the performance of current graphics hardware to coalesce writes into larger transactions [NVI09b], the output values are written to a buffer in such a way that threads are axis-aligned and write into consecutive memory locations, as demonstrated in Figure 2. For an $N \times N$ source height field, output buffers are $\sqrt{2}N \times \sqrt{2}N$ in size, and they have to be rotated back when fusing the results.

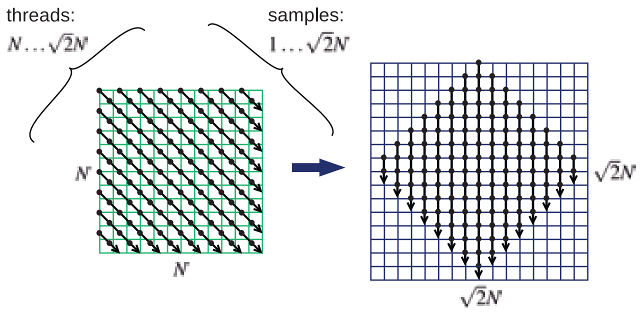


Figure 2: *For each azimuthal direction, $N \ldots \sqrt{2}N$ threads step through $1 \ldots \sqrt{2}N$ samples to cover a height field of resolution N^2 and write the results to an aligned output buffer.*

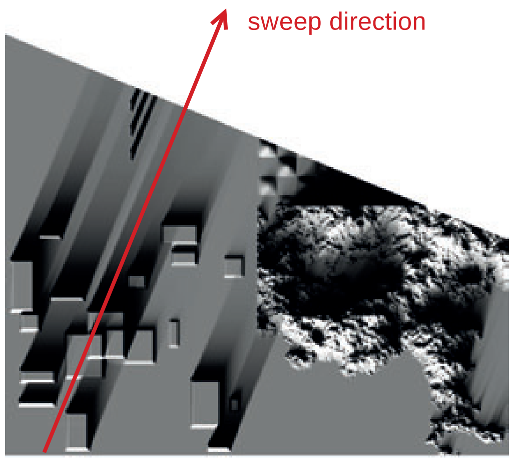


Figure 3: *Parallel threads have extracted the horizon angles for roughly half of their path during one azimuthal sweep. The full result is in Figure 5.*

5. Occlusion extraction

The purpose of the occlusion extraction stage is to extract horizon angles for height samples efficiently and correctly. This process is done in threads that map to lines in the height field. A thread steps along the line one height sample at a time, calculates the horizon angle for each consecutive sample, and writes the results to consecutive lines in the output buffer. The threads keep a robust representation of the height function along the line in memory from which the dominant occluder is deduced. Each new sample is a potential occluder for future samples and is therefore always initially included in this representation.

The dominant occluder for an arbitrary new sample is one from the *convex hull subset* of previous samples. Moreover, elements of the convex hull can have a direct line of sight only to neighboring elements. Therefore, when the new sample is made part of the convex hull set, its horizon angle can be deduced directly from the previous element, as illustrated in Figure 4.

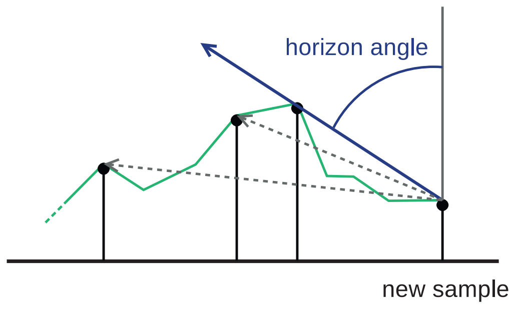


Figure 4: *The horizon angle for a sample is determined by the previous occluder in the convex hull set marked by ●.*

We can also state this formally. When sorted by distance, the convex hull set can be defined as

$$\frac{h_n - h_{n+i}}{d_n - d_{n+i}} \geq \frac{h_m - h_{m+j}}{d_m - d_{m+j}}, \quad n \leq m \quad n+i \leq m+j \tag{1}$$

where for occluder of index k, h_k is its height and d_k its distance along the line. From this definition we obtain the largest occlusion for an element of index n as

$$\max_{i \in [1,n]} \left(\frac{h_{n-i} - h_n}{d_{n-i} - d_n} \right) = \frac{h_{n-1} - h_n}{d_{n-1} - d_n} \tag{2}$$

In computational geometry, algorithms for the efficient construction of convex hulls have been studied in [Gra72] and [Mel87]. We can exploit the incremental nature of our method and the structure of the height field geometry to construct a simple linear-time algorithm to process a line of height samples.

For simplicity we implement the convex hull set here as a stack. To retain a valid convex hull when incrementally adding occluders, the stack has to be popped until Equation 1 holds for the last and the new element. Therefore adding a new element while retaining convexity, and finding the occluder casting the smallest horizon angle on the new element are achieved with the same operation. A pseudo code algorithm of this operation is shown in Algorithm 1.

Algorithm 1 Processing a new height map sample *new*

```
$v_1 \leftarrow \text{VECTOR}(peek_1 \rightarrow new)$
**while** $size > 1$ **do**
    $v_2 \leftarrow \text{VECTOR}(peek_2 \rightarrow new)$
    **if** $h(v_2)d(v_1) \geq h(v_1)d(v_2)$ **then**
        $v_1 \leftarrow v_2$
        $pop$
    **else**
        $break$
    **end if**
**end while**
$push(new)$
**return** $\frac{\pi}{2} - tan^{-1}\frac{h(v_1)}{d(v_1)}$
```

Pop and *push* are standard stack operations, $peek_1$ returns the last element without modifying the stack, and $peek_2$ returns the second to last. The inverse tangent can be efficiently calculated using [Has53] as introduced in [SN08], which requires only one floating point division and two computationally light branches.

For a thread processing N elements, there will be exactly N pushes on the stack and less than N pops. One iteration performs $n+1$ comparison operations for n pops, and therefore the total number of comparisons for an entire thread is at most $2N$, yielding a total time complexity of $O(N)$. This property of the algorithm gives it its desired performance charasteristics. Another desired feature of the algorithm is that it does not skip, or use an approximation for, even distant occluders, and can return horizon angles in the full range of $0\ldots\pi$.

6. Lighting

As it is possible to extract high-resolution horizon maps using the algorithms described in Sections 4 and 5, it is useful to have a scalable lighting model capable of representing the full resolution. As our second contribution we first present incident lighting in an analytical form, and then show how it can be efficiently calculated in real-time for uniform ambient lighting and for arbitrary light environments.

We first recapitulate the rendering equation [Kaj86] in this context. We assume the surfaces to exhibit Lambertian reflectance and to emit no radiance. While our method is not restricted to Lambertian surfaces, its suitability for other BRDFs would warrant a separate investigation and is beyond the scope of this paper. Lighting calculations are performed in the coordinate system of the height field plane, where the equation for output radiance becomes

$$L_o(L_i, o, \vec{N}, \mathbf{x}) = \frac{1}{\pi}\int_{\Omega} L_i(\vec{e})o(\mathbf{x},\vec{e})(\vec{N}\cdot\vec{e})\,d\vec{e}, \qquad \vec{N}\cdot\vec{e} \geq 0 \tag{3}$$

The integral extends over the hemisphere around the point $\mathbf{x}$ with the normal $\vec{N}$. L_i is the input radiance as a function of direction $\vec{e}$, and o is a binary visibility term as a function of the point $\mathbf{x}$ and direction $\vec{e}$.

If the integral is discretized into n equally sized azimuthal swaths, it can be expressed as

$$L_o(L_i, o', \vec{N}, \mathbf{x}) = \frac{1}{\pi}\sum_{k=0}^{n-1}\int_{\frac{\pi}{n}(2k-1)}^{\frac{\pi}{n}(2k+1)}\int_0^{\theta_k} L_i(\vec{e})(\vec{N}\cdot\vec{e})sin\theta\,d\theta\,d\phi,$$
$$\theta_k = \min\left(o'(\mathbf{x},k), tan^{-1}\left(\frac{\vec{N}_x cos(\frac{\pi}{n}2k) + \vec{N}_y sin(\frac{\pi}{n}2k)}{\vec{N}_z}\right)\right) \tag{4}$$

The angle θ_k is the horizon angle $o'(\mathbf{x},k)$ from zenith for the azimuthal direction k at $\mathbf{x}$ clamped to satisfy $\vec{N}\cdot\vec{e} \geq 0$. The clamping is evaluated at $\phi = \frac{\pi}{n}2k$.

6.1. Uniform ambient lighting

Solving the integrals in Equation 4 with $\vec{e}$ expressed in spherical coordinates and assuming constant input lighting ($L_i = c$) gives

$$L_o(o', \vec{N}, \mathbf{x}) = c\frac{\vec{N}_z}{n}\sum_{k=0}^{n-1} sin^2\theta_k + c\frac{sin(\frac{\pi}{n})}{\pi}\sum_{k=0}^{n-1} \tag{5}$$
$$\left(\left(\theta_k - \frac{1}{2}sin(2\theta_k)\right)\left(\vec{N}_x cos\left(\frac{\pi}{n}2k\right) + \vec{N}_y sin\left(\frac{\pi}{n}2k\right)\right)\right)$$

As $cos\left(\frac{\pi}{n}2k\right)$ and $sin\left(\frac{\pi}{n}2k\right)$ remain constant for one azimuthal direction throughout the height field, only $sin^2\theta_k$ and $sin(2\theta_k)$ will have to be calculated for each height field point. Also, if the inverse tangent in Algorithm 1 is calculated after clamping the slope by the normal, Equations 4 and 5 can be evaluated with three floating point divisions and less than ten multiplications and additions. Figure 5 features uniform lighting.

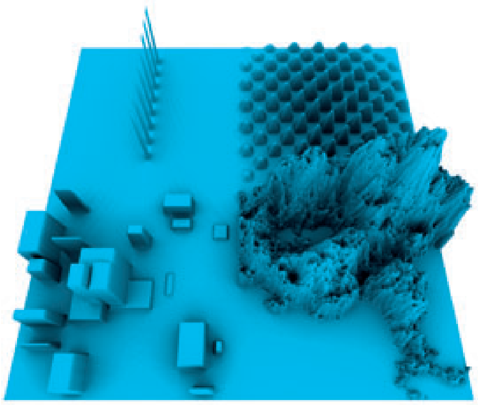


Figure 5: *A* 1024^2 *height field under uniform ambient lighting (*$\phi_N = 64$*, 24 fps)*

6.2. Arbitrary light environments

To represent arbitrary, infinitely distant, light environments we can precalculate lighting for each azimuthal direction for a specific L_i as a function of $\vec{N}$ and $o'(\mathbf{x},k)$.

If $\vec{N}$ is decomposed into two orthogonal components, one perpendicular to $\vec{e}$, the remaining component $\vec{N}_p$ is the only contributor to the dot product $\vec{N}\cdot\vec{e}$. Furthermore, the length of the component can be dissociated from the precalculated light function L_p, allowing an efficient definition of $L_p^k(\theta_n,\theta_h)$ as a function of the normal angle θ_n and the horizon angle θ_h towards the azimuthal direction k, as illustrated in Figure 6.

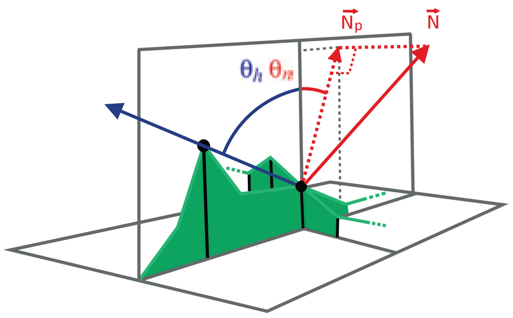


Figure 6: *When the normal $\vec{N}$ is projected onto the azimuthal plane, lighting can be tabulated as a function of the horizon angle θ_h and the angle θ_n for the projected normal.*

The lighting evaluation becomes

$$L_o(L_p,o',\vec{N},\mathbf{x}) = \sum_{k=0}^{n-1}\left(|\vec{N}_p|L_p^k\left(\theta_n^k,o'(\mathbf{x},k)\right)\right) \tag{6}$$

Precalculation of L_p for direction k becomes

$$L_p^k(\theta_n,\theta_h) = \frac{1}{\pi}\int_{\frac{\pi}{n}(2k-1)}^{\frac{\pi}{n}(2k+1)}\int_0^{\min\left(\theta_n+\frac{\pi}{2},\theta_h\right)} L_i(\theta,\phi)sin\theta cos(\theta-\theta_n)\,d\theta\, d\phi \tag{7}$$

If the same environment map sample for a specific θ is used throughout one k, i.e. $L_i(\theta,\phi)=L_i(\theta,k)$, Equation 5 can be incrementally used to compute L_p in a separate pass.

As for the resolutions of θ_n and θ_h in L_p, the horizon angle θ_h directly defines the resolution of the light environment in conjunction with the azimuthal direction count ϕ_N. The resolution of θ_n on the other hand should be selected to be as low as possible while retaining an acceptable level of error in the result. The resolution only affects the weighing of the environment samples and the cutoff horizon angle induced by $\vec{N}\cdot\vec{e}\geq 0$. Also, the results from consecutive normal angles can be linearly interpolated during the sampling of L_p.

Figure 7 shows different normal angle resolutions and the corresponding error. A resolution as low as 8 usually produces results indistinguishable to the naked eye from higher resolutions, and a resolution of 16 is a safe choice without losing much of the benefit offered by texture caching. We actually use uneven resolutions (7, 15, 31...) in order to produce exact results for normals that point directly upwards $(0,0,1)$.

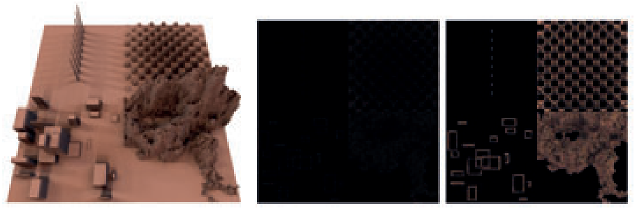


Figure 7: *Error introduced by lowering the normal angle resolution from a reference 256 to 16 (middle) and 8 (right) on a height field shown to the left, multiplied by 50. The average and maximum errors for resolution 8 are 0.16% and 1.66%, and for resolution 16 0.03% and 0.42% respectively.*

Figure 8 demonstrates the ability of this light model to represent both point and area light sources.

7. Implementation

Our implementation runs entirely on the GPU and is based on OpenGL 3 and CUDA 2. Various implementational alternatives were tested and the ones described here produced the best results in our environment. The APIs and the performance characteristics of their implementations are subject to change.

The algorithm input consists of one or two OpenGL side PBOs (pixel buffer objects): a floating point height map and (if environment lighting is used) a floating point RGB cube map. The output is either a monochromatic or an RGB (for environment lighting) floating point OpenGL texture. The entire pipeline uses HDR (high dynamic range) values.

The algorithm can be broken down into the following stages that are executed for each frame.

Preprocessing

The source height field and the environment cube light map are passed as 32 bit floating point OpenGL PBOs to CUDA, and further copied to CUDA arrays for bi-linearly filtered sampling. Surface normals are created from the height data by summing unnormalized normals from each of the four quads connected to the height sample. Not normalizing the quadrants produces less artifacts on very sharp edges. After the summing, the normals are normalized and stored as per-component 8 bit fixed points, and bi-linearly filtered during sampling. L_p is generated as described in Section 6 using 32 bit floating point color components. The resulting light table is bi-linearly filtered during sampling.

Occlusion extraction

The height field is swept through for each azimuthal direction as described in Section 4. Occluders are stored in local memory (off-chip) in preallocated arrays that are large

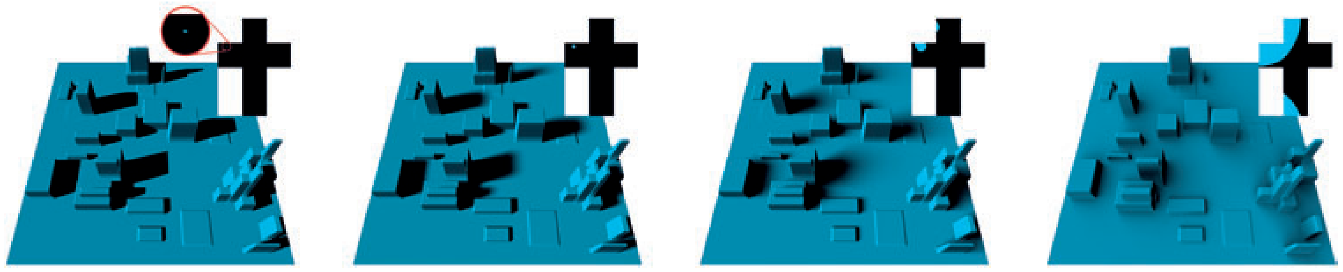


Figure 8: *Differently sized circular light sources with corresponding lighting cube maps on the top.* ($\theta_N = 256, \phi_N = 128$)

enough for no overflow to occur using half precision (16 bit) floats for their height and unsigned fixed point (16 bit) for their distance, fitting an occluder into 32 bits. Threads write 32 bits of data to the output buffer every fourth iteration consisting of four 8 bit unsigned fixed point horizon angle values. When uniform ambient lighting is used, light values are computed directly instead.

Packing and lighting

Before passing the data for final blending to OpenGL, four (ϕ, $\phi + \frac{\pi}{2}$, $\phi + \pi$, and $\phi + \frac{3\pi}{2}$) azimuthal directions are linearly combined for reduced overhead during PBO passing between OpenGL and CUDA. As the horizon data is 8 bit and threads process 32 bit elements for optimal efficiency, shared memory communication between threads is required. Communication is also required for efficient texture transposing. If environment lighting is used the lighting calculation is also carried out in this stage for reduced computation. The reduction is made possible by sharing one normal for the four azimuthal directions, producing sampling coordinates for L_p with less operations. The resulting lighting is stored using OpenGL compatible 10+11+11 bit floating point BGR components.

Fusing of intermediate results

The previously generated intermediate results are independent and can be fetched in multiple passes if video memory reservation is an issue. The intermediate results are copied into textures which are rotated and blended in a 32+32+32 bit floating point RGB frame buffer to produce the final result in OpenGL.

When constructing the thread blocks for occlusion extraction, at least two aspects should be considered. Firstly, the heads of the threads should be aligned to form groups that write consecutive memory addresses (share a common first row in the output buffer) to allow write coalescing. Secondly, the threads within the whole thread block should have similar spans for maximum utilization of the SIMT (single-instruction multiple-thread) hardware. Processing multiple azimuthal directions in one kernel invocation increases the amount of available threads with similar spans. However, packing threads of *equal* span will interfere with write coalescing, since threads with exactly the same span are bound to be either at the other end of the texture, or belong to a different direction. As the amount of threads necessary for efficient memory coalescing is lower than the optimal size of a thread block, these two goals are not mutually exclusive.

Although having similar spans, adjacent threads may still not execute each iteration of Algorithm 1 synchronously due to different occluder stack contents. Furthermore, when the occluder stacks are of different sizes during runtime, memory coalescing cannot occur. Fortunately, the last two occluders in the stack can be stored as separate variables (in registers) saving two memory accesses that would otherwise be required in each iteration (when $size > 1$), diminishing the memory coalescing problem and improving the overall performance.

In order to estimate the memory consumption of the occluder stacks we note that the maximum size that a convex hull may have in a $N \times N$ height field is $\sqrt{2}N$ (e.g. a half sphere extending from corner to corner of the height field). Preallocating arrays according to this observation would yield an occluder storage equal to the output buffer ($\sqrt{2}N \times \sqrt{2}N$ elements) in size. In practice, however, CUDA runtime only has to allocate space for threads that are scheduled to run, which is typically less than the total number of threads. Also, most height fields require convex hulls of only fraction of the maximum size. For instance, the actual convex hull sizes in Figure 9 peaked at 3% of the theoretical maximum (43 elements). According to our benchmarks, the size of the occluder arrays has a negligible effect on performance, excluding that coming indirectly from memory consumption (e.g. by affecting the number of passes required).

8. Results

The performance of our algorithm is directly related to the height field size and to the number of azimuthal directions, but is rather insensitive to geometric content. As the exact horizon is extracted for each height field point for all azimuthal directions, there are no other tunable parameters involved which trade the accuracy of visibility calculations for speed.

Figure 9 illustrates the effect of azimuthal direction count

Table 1: *Performance measurements*

HF res.	FPS for uniform, environment lighting				
	$\phi_N = 16$	32	64	128	256
	Nvidia GTX 280 (1 GB)				
512^2	160, 114	118, 87	74, 61	44, 38	24, 22
1024^2	76, 62	45, 39	24, 23	12, 12	6.4, 6.3
2048^2	24, 24	13, 13	6.6, 6.7	3.4, 3.4	1.7, 1.7
4096^2	5.4, 6.8	2.8, 3.5	1.4, 1.8	0.7, 0.9	0.4, 0.5
	Nvidia 8800 GTS (512 MB)				
512^2	110, 108	62, 68	33, 38	17, 20	8.6, 10
1024^2	33, 38	17, 21	8.8, 11	4.4, 5.5	2.2, 2.8
2048^2	8.8, 10	4.6, 5.4	2.3, 2.8	1.2, 1.4	0.6, 0.7

Table 2: *Execution time distribution*

Stage	Proportion	
	$\phi_N = 16$	128
	Env. lighting	
Normals and light precalculation	15%	3%
Occluder extraction	38%	46%
Packing and lighting	20%	32%
OpenGL (incl. CUDA interop.)	27%	19%
	Uniform	
Normals	9%	1%
Occluder extraction and lighting	65%	76%
Packing	8%	11%
OpenGL (incl. CUDA interop.)	18%	12%

on the quality of rendering. Banding artefacts due to azimuthal undersampling might appear when a low number of directions is used. Uneven complex geometry helps to hide banding, but a height field with sharp edges and planarity might require up to 128 azimuthal directions before very good results are obtained. Currently, the suitable number of azimuthal directions has to be selected manually.

Table 1 lists frame rates for combinations of height field resolutions and azimuthal directions. Height field content is shown in Figure 9 and the environment lighting resolutions are 256 (θ_N) times the number of azimuthal directions (ϕ_N). All stages listed in Section 7 were included for each frame. Table 2 shows typical execution time distributions between the different stages. A 1024^2 height field (Figure 9) was used for the tests, and the data was gathered using CUDA Profiler [NVI09a]. The execution stages do not overlap in time.

The precalculation of normal vectors and the light function for environment lighting execute in approximately constant time for any number of azimuthal directions and any height field size, and therefore consume a larger portion of the execution time when the number of azimuthal directions is low. It is also worth noting that these precalculation kernels take significantly longer CPU time than GPU time, indicating relatively high overhead in data copying and kernel invocation. Also, binding OpenGL PBO resources in CUDA has some overhead — included in the OpenGL phase — that grows proportionally larger with a lower number of azimuthal directions.

As occluder extraction is a problem that has many uses — and even in this context can be used with other lighting methods — observing its performance independently can be useful. Extracting only the horizon angles for one azimuthal direction on each point of a 1024^2 height field is accomplished in 0.30 ms (>3300 Hz), when measured using $\phi_N = 128$.

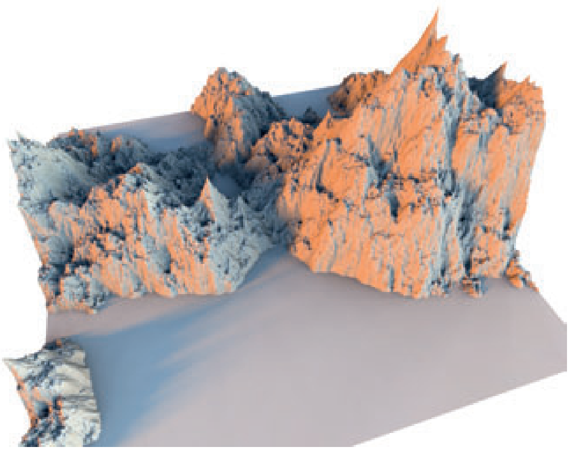


Figure 10: *A fractal terrain of size* 2048^2 *(8M triangles) lit by a single* 256×16 *environment at 20 Hz.*

9. Conclusions

We have presented a new real-time method to render self-shadows on dynamic height fields under dynamic light environments. Its computation is parallel and suitable for current GPGPUs. Our method determines visibility as the exact horizon in a set of azimuthal directions in time linear in height field size. This allows scaling to large height fields with arbitrary geometric content. We also presented a lighting model capable of representing complex high-resolution light environments extracted on-line from HDR cube maps, allowing for accurate real-time direct lighting of height fields. Our method is faster and more general than previous methods.

Our method could also be used to calculate ambient occlusion or direct lighting in offline rendered graphics which require greater accuracy and scaling. For example, a 4096×4096 height field (34M triangles) can be lit from 1024 azimuthal directions in 8 seconds, its computation fitting into the video memory of commodity graphics cards, and the result being comparable to exhaustive ray-tracing.

Acknowledgements
We would like to thank Dr Jukka Arvo for his fruitful comments.

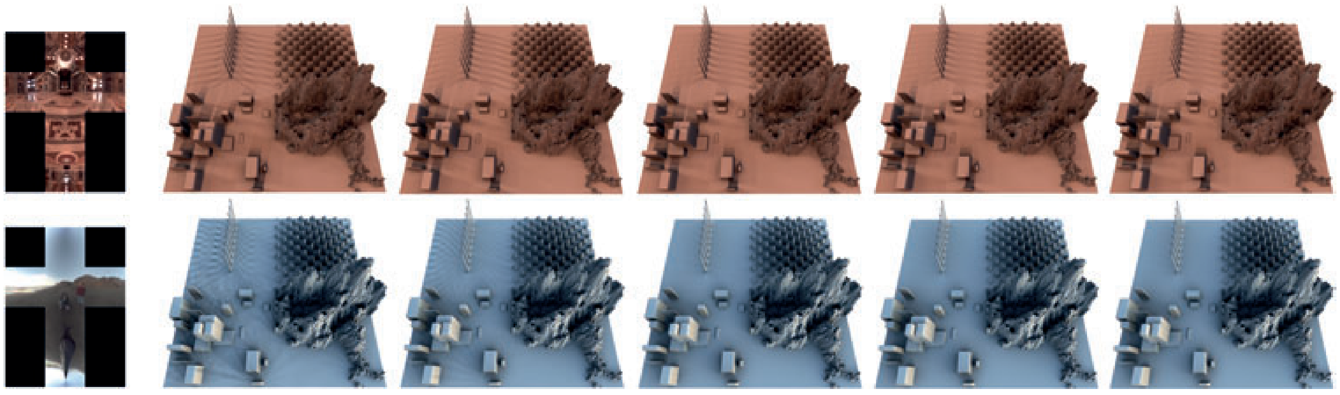


Figure 9: *Increasing azimuthal directions yields smoother results, but its effect depends on the geometric content. On the left are two light environments [Deb98] upon the* 1024^2 *height field,* $\theta_N = 256$, *and the number of azimuthal directions with the corresponding frame rates from left to right are: 16 (62 Hz), 32 (39 Hz), 64 (23 Hz), 128 (12 Hz), and 256 (6.3 Hz).*

References

[ADM*08] ANNEN T., DONG Z., MERTENS T., BEKAERT P., SEIDEL H.-P., KAUTZ J.: Real-time, all-frequency shadows in dynamic scenes. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers* (New York, NY, USA, 2008), ACM, pp. 1–8.

[BS09] BAVOIL L., SAINZ M.: Multi-layer dual-resolution screen-space ambient occlusion. In *SIGGRAPH '09: SIGGRAPH 2009: Talks* (New York, NY, USA, 2009), ACM, pp. 1–1.

[BSD08] BAVOIL L., SAINZ M., DIMITROV R.: Image-space horizon-based ambient occlusion. In *SIGGRAPH '08: ACM SIGGRAPH 2008 talks* (New York, NY, USA, 2008), ACM, pp. 1–1.

[Bun05] BUNNELL M.: *Dynamic ambient occlusion and indirect lighting*. Addison-Weseley Professional, 2005, pp. 223–233.

[DBS08] DIMITROV R., BAVOIL L., SAINZ M.: Horizon-split ambient occlusion. In *I3D '08: Proceedings of the 2008 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2008), ACM, pp. 1–1.

[Deb98] DEBEVEC P.: Rendering synthetic objects into real scenes: bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *Proceedings SIGGRAPH '98* (New York, NY, USA, 1998), ACM, pp. 189–198.

[Gra72] GRAHAM R.: An efficient algorithm for determining the convex hull of a finite planar set. *Inf. Process. Lett. 1* (1972), 132–133.

[Hal08] HALFHILL T. R.: Parallel Processing with CUDA. *Microprocessor Report* (January 2008).

[Has53] HASTINGS C.: Approximation theory, note 143. *Math. Tables Aids Comput 68*, 6 (1953).

[HLHS03] HASENFRATZ J.-M., LAPIERRE M., HOLZSCHUCH N., SILLION F.: A survey of real-time soft shadows algorithms, dec 2003.

[Kaj86] KAJIYA J. T.: The rendering equation. In *Proceedings SIGGRAPH '86* (New York, NY, USA, 1986), ACM, pp. 143–150.

[KL05] KONTKANEN J., LAINE S.: Ambient occlusion fields. In *Proceedings of ACM SIGGRAPH 2005 Symposium on Interactive 3D Graphics and Games* (2005), ACM Press, pp. 41–48.

[Lef07] LEFOHN A.: Gpgpu for raster graphics. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses* (New York, NY, USA, 2007), ACM, p. 11.

[Max88] MAX N.: Horizon mapping: shadows for bump-mapped surfaces. *The Visual Computer 4*, 2 (Mar. 1988), 109–117.

[Mel87] MELKMAN A.: On-line construction of the convex hull of a simple polygon. *Inf. Process. Lett. 25* (1987), 11–12.

[Mit07] MITTRING M.: Finding next gen: Cryengine 2. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses* (New York, NY, USA, 2007), ACM, pp. 97–121.

[NS09] NOWROUZEZAHRAI D., SNYDER J.: Fast global illumination on dynamic height fields. *Computer Graphics Forum: Eurographics Symposium on Rendering* (June 2009).

[NVI09a] NVIDIA CORPORATION: *CUDA Profiler*. 2009.

[NVI09b] NVIDIA CORPORATION: *NVIDIA CUDA Programming Guide 2.3*. 2009.

[RGS09] RITSCHEL T., GROSCH T., SEIDEL H.-P.: Approximating dynamic global illumination in image space. In *I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2009), ACM, pp. 75–82.

[SA07] SHANMUGAM P., ARIKAN O.: Hardware accelerated ambient occlusion techniques on gpus. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2007), ACM, pp. 73–80.

[SC00] SLOAN P.-P. J., COHEN M. F.: Interactive horizon mapping. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000* (London, UK, 2000), Springer-Verlag, pp. 281–286.

[SKU08] SZIRMAY-KALOS L., UMENHOFFER T.: Displacement mapping on the gpu — state of the art. vol. 27, pp. 1567–1592.

[SN08] SNYDER J., NOWROUZEZAHRAI D.: Fast soft self-shadowing on dynamic height fields. *Computer Graphics Forum: Eurographics Symposium on Rendering* (June 2008).

[Ste98] STEWART A. J.: Fast horizon computation at all points of a terrain with visibility and shading applications. *IEEE Transactions on Visualization and Computer Graphics 4*, 1 (1998), 82–93.

[Tat06] TATARCHUK N.: Practical parallax occlusion mapping with approximate soft shadows for detailed surface rendering. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses* (New York, NY, USA, 2006), ACM, pp. 81–112.

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. *SIGGRAPH Comput. Graph. 12*, 3 (1978), 270–274.

Publication P2

Ville Timonen. Low-Complexity Intervisibility in Height Fields. *Computer Graphics Forum*, 31(8), pages 2348–2362, 2012. Invited to *Eurographics Conference 2013*.

DOI: 10.1111/j.1467-8659.2012.03155.x

COMPUTER GRAPHICS *forum*
Volume 31 (2012), number 8 pp. 2348–2362

Low-Complexity Intervisibility in Height Fields

Ville Timonen

Åbo Akademi University
vtimonen@abo.fi

Abstract

Global illumination systems require intervisibility information between pairs of points in a scene. This visibility problem is computationally complex, and current interactive implementations for dynamic scenes are limited to crude approximations or small amounts of geometry. We present a novel algorithm to determine intervisibility from all points of dynamic height fields as visibility horizons in discrete azimuthal directions. The algorithm determines accurate visibility along each azimuthal direction in time linear in the number of output visibility horizons. This is achieved by using a novel visibility structure we call the convex hull tree. The key feature of our algorithm is its ability to incrementally update the convex hull tree such that at each receiver point only the visible parts of the height field are traversed. This results in low time complexity; compared to previous work, we achieve two orders of magnitude reduction in the number of algorithm iterations and a speedup of 2.4 to 41 on 1024^2 height fields, depending on geometric content.

Keywords: height field, intervisibility, global illumination

ACM CCS: I.3.7 [Computer Graphics]: Computer Graphics—Color, shading, shadowing, and texture

1. Introduction

The two major applications for visibility algorithms in computer graphics rendering are geometry culling and lighting. Firstly, culling is used to accelerate rendering by determining which geometry should not be sent down the rendering pipeline, i.e. which geometry is invisible from the viewer. Secondly, lighting algorithms need to know the visibility of light sources from each illuminated receiver point. As generally all scene geometry acts as light sources, global illumination algorithms need to determine the visibility of the whole scene from each receiver point. This problem is similar to that in geometry culling, except orders of magnitude more complex, as visibility from millions of receiver points needs to be determined. The problem is largely solved for static geometry as most of the computation can be performed as a pre-pass, but dynamic geometry remains an open problem.

In this paper we present a method to determine intervisibility of height field geometry. Height maps describe geometry by defining the elevation of a plane as a function of $N \times N$ surface coordinates. They can be used as standalone objects or to describe meso- and micro-structure on the surfaces of a larger-scale object. Another recently popular application for height field algorithms is producing lighting effects in screen space, where the depth buffer of a rendered scene is treated as a height field with effects applied in post-processing. For non-graphics related applications of height field visibility algorithms, see the survey [Nag94].

Current interactive height field intervisibility methods determine approximate or local visibility in order to produce effects such as soft global illumination [NS09] or color bleeding [RGS09]. Current methods are limited to local, approximate, or noisy effects mainly due to poor scaling of visibility calculations. The approach that current methods use involves sampling, for each receiver independently, the surrounding height field where in order to test n sender points for one receiver point, n iterations are performed. Intervisibility searches based on this approach are variations of what we in this paper will call the *naïve method*: for each receiver point K azimuthal directions are chosen and, for each direction, the height field is traversed outwards from the receiver one unit length step at a time.

Table 1: *The average number of visible points per direction for the 1024^2 height fields in Figure 7.*

Height field	Visible points	Visible/total
Fractal terrain	27.1	5.6 %
Brick surface	8.94	1.8 %
Sine grid	22.8	4.7 %
Blocks	5.56	1.1 %

The main problem in the naïve method is that it scales linearly with respect to the search distance, while visibility in all practical height fields should scale *sub-linearly*. In Table 1 we counted the average number of visible points for a single azimuthal direction for different types of height fields ($N = 1024$) and compared it to the number of evenly spaced points that were tested for visibility (roughly $N/2$). In Section 6, we will measure the scaling to be between $O(N^{0.01})$ and $O(N^{0.65})$.

In this paper we present a new way of calculating visibility in height fields. The key feature of our method is its ability to traverse only the visible geometry by effectively culling the non-visible geometry. Another advantage of our method is that it produces a more compact description of the visibility than a simple enumeration of the visible points: for each receiver point we determine a list of local visibility horizons where two consecutive horizons always enclose all adjacent visible height field points. Our algorithm runs in time linear in the number of output visibility horizons and is dependent on the height field content. Compared to previous algorithms, we achieve two orders of magnitude reduction in the number of iterations required to extract accurate intervisibility on 1024^2 height maps bringing the complexity to manageable levels, and a speed up of 2.4 to 41 compared to the naïve method, depending on the height field content.

It can be argued that in rendering there is a trend towards performing an increasing portion of shading and lighting as a screen-space pass. It will be interesting to see what the full potential of such methods are. Current screen-space methods are not able to effortlessly scale to full illumination solutions, but rather settle for producing a limited set of effects that are computationally feasible. While there are many obstacles to overcome before global illumination with arbitrary materials and integrated light sources is feasible in screen-space, our contribution is to overcome the one with the highest computational complexity: intervisibility.

2. Previous Work

Determining height field intervisibility is essentially a problem of computational geometry. Finding the visible areas of a terrain from a single viewpoint [KZ02] [FHT09], a line [CS89], or a region [BWW05] is a problem extensively researched and largely solved. However, algorithms that find terrain intervisibility at all surface points are significantly less studied, and applying single-viewpoint methods to each height field point is intractable for interactive applications.

Global illumination methods for height fields require intervisibility determination, but unlike our method they so far exclusively use a scheme where the same visibility search procedure is performed independently for each height field point. The naïve approach [CoS95] [SLYY08] is to solve the visibility in a set of azimuthal directions where each direction is traversed from the receiver point outwards in unit length steps, and each time the previous slope maximum is exceeded the new point is known to be visible from the receiver point. Our method produces results identical to the naïve method.

The naïve approach can be accelerated by applying level of detail (LOD) methods: [NS09] generate multiple levels of detail of the height field and use the lower resolution levels and sparser sampling when traversing farther from the receiver, as first introduced in [SN08]. While faster, the approximated visibility favors using the method only for soft effects.

Ambient occlusion methods with falloff terms need to know the distance of occluding geometry and therefore have to solve the intervisibility problem as well. Screen-space ambient occlusion methods [Mit07] [DBS08] approximate *local* scene visibility in image-space by treating the depth buffer as a height field. The same approach has been used to produce global illumination effects such as color bleeding [RGS09], which is extended with LOD in [SHR10]. Intervisibility in these methods is determined from sender and receiver normals only and any occluding geometry in-between is ignored. While fast and sufficient for approximate near-field effects, scaling to far-field is problematic: an occlusion search between sender and receiver is required as suggested by the authors of [RGS09], making the intervisibility search the same as used by the naïve method.

Implementations of the naïve method usually trade banding for noise by randomizing sampling patterns per receiver. In future work, in Section 10, we discuss ways to apply LOD and to trade banding for noise with our method as well.

More exotic ways to sample the height field have also been introduced, such as performing a very sparse randomized occlusion search per pixel and gathering the final occlusion values from a small neighborhood around a receiver [HBR*11]. Alternatively, instead of taking simple height samples, line and area samples can be taken to approximate the overall occlusion of the near-field geometry [LS10]. While these methods produce fast results, they don't scale well to occlusion effects of arbitrary length and are non-trivial to extend to indirect illumination.

Global illumination methods for *generic* geometries are diverse [DBBS06] and also need to address the intervisibility problem. Excluding various approximations, these methods traverse all scene geometry for each receiver primitive. The problem then becomes making sure that only the effect of the frontmost layer around the receiver is accounted for. This has been tackled in [Bun05] and [DSDD07] by running several iterations of the algorithm where each iteration removes extraneous contribution of the overlapping layers. Another solution to overcome this problem is to use shadow maps (see the survey [HLHS03]) to determine the receivers from the point of view of point light sources. Fast approximate shadow maps [RGK*08] [ADM*08] can be used to make this approach feasible when many light sources need to be considered. None of these approaches make use of the characteristics of height field geometry, and their visibility solutions become prohibitively expensive for anything but very small height fields when accurate results are preferred.

Another common approach to solving visibility of scene geometry is ray tracing, where visibility is queried by shooting independent rays from a receiver and determining the closest geometry the rays intersect. Let K be the number of azimuthal directions in which rays from each point are cast, and let p_a be the average number of visible height field points from one receiver point in one azimuthal direction. Then on an $N \times N$ height field at least N^2Kp_a optimally chosen rays have to be traced to determine intervisibility at the same accuracy as our method. Not including the complexity of tracing one ray, this already is higher than the time complexity of our method, $O(N^2Km_a)$, where $m_a \leq p_a$ denotes the average number of visibility horizons.

We use a compact way to unambiguously describe intervisibility on a line by local horizons as introduced in [DFM94]. In this model, a local horizon from a viewpoint is defined at each transition from visibility to invisibility. From local horizons it is possible to produce the *casting set* as used in [NS09], which is the set of points visible from the receiver point.

Incidentally, it was shown in [TW10] that a visibility horizon is defined by the points of a convex hull, and that a line sweep algorithm can incrementally determine global visibility horizons for n points in $O(n)$ time by using a convex hull stack. In order to determine intervisibility, we extend the ideas of [TW10]: we track *a set of convex hulls* instead of only one and introduce a novel tree structure to hold them. Through efficient tree update operations, we maintain the same linear complexity: n local horizons are extracted in $O(n)$ time.

3. Height Field Processing

In this section we describe the highest level framework of our method with a focus of the process on the scale of the whole height field. In Section 4 we describe the process of solving visibility on the scale of a single line. Section 5 defines in detail the algorithm that is executed for each point

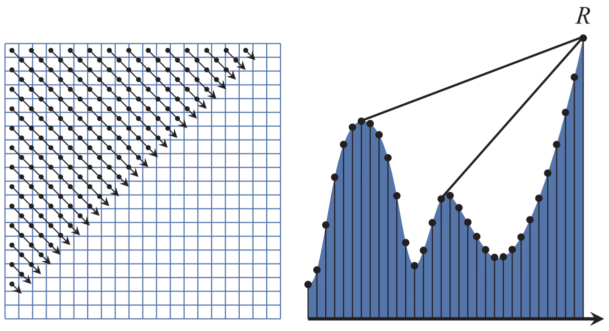


Figure 1: *For one azimuthal direction, the height field is processed in parallel lines (left). For each line, the visibility horizons are extracted backwards along the line for the most recent step, receptor R (right).*

on the line. Section 6 establishes the complexity of the algorithm and analyzes its scaling with respect to the height field size and content. An implementation on the GPU and optimizations on the code are covered in Section 7 and the implementation efficiency with respect to available hardware resources is discussed in Section 8. Section 9 showcases the actual performance and Section 10 discusses the accuracy of our method and ways to utilize the visibility information.

The input to our algorithm is a height map consisting of a regular grid of $N \times N$ height values. For each height field point our algorithm determines visibility in K discrete azimuthal directions by performing K sweeps through the height field. For each direction $\phi_k = \frac{k}{K}2\pi, 0 \leq k < K$, the height field is swept through in parallel lines that are unit length apart thus calculating the given azimuthal direction for all height field points in one sweep.

These lines are stepped through one unit length step at a time, and visibility backwards along the line is determined for each step in turn, as demonstrated in Figure 1. At each step lighting contribution is gathered from the visible segments of the line and the result is written into the sweep's output buffer. The output buffers are axis-aligned such that the processed lines map to vertical lines in each output buffer, as demonstrated in Figure 2. As the maximum number of lines as well as the maximum length of a line in an arbitrary direction can be at most $\sqrt{2}N$, the output buffers are of size $\sqrt{2}N \times \sqrt{2}N$.

After the sweeps have been performed, each of the K output buffers contain N^2 result values. Finally, results across the output buffers are accumulated into a single result buffer the size of the input height field, $N \times N$, shown in Figure 2. At this point visibility of an average of roughly $K\frac{N}{2}$ height field samples have been considered for each of the N^2 sampled receptor points. Unlike previous methods, we require substantially fewer than $K\frac{N^3}{2}$ iterations to achieve this, as shown later in this paper.

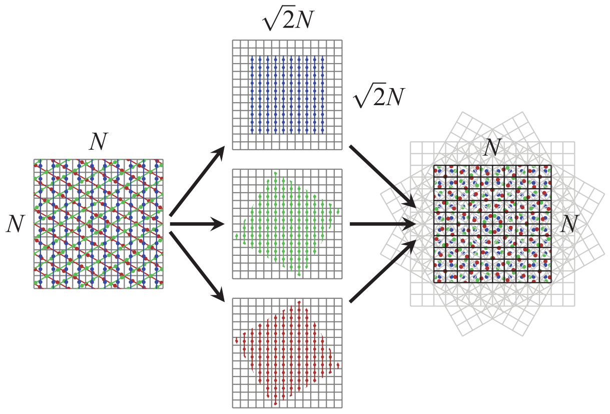


Figure 2: *Three sweeps ($K = 3$), denoted by different colours, are performed over the input height field (left), their results are written to axis-aligned output buffers (center), and finally accumulated in the result buffer (right).*

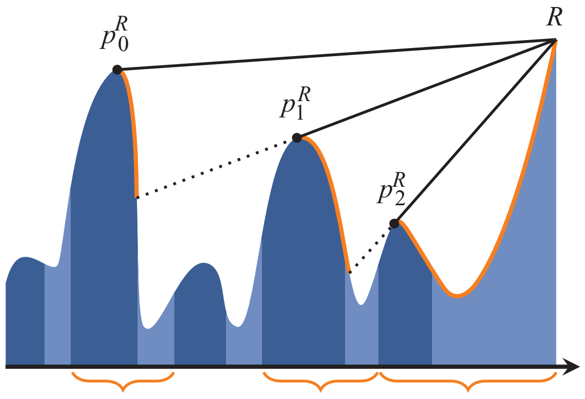


Figure 3: *The local horizons formed between points p_i^R and R unambiguously describe visible segments (denoted at the bottom) and the extent of their visibility (the surface in orange) at the receptor R. Convex sections are dark and concave light.*

4. Line Processing

Visible points of the height field along a line are naturally grouped into continuous parts. Such a part can only start at a point in the *convex* section of the line: when viewed from the receptor R, the point is a local maximum with neighbors that are below it. Convex sections are separated from each other by *concave* sections, and therefore the line can be split into strictly alternating convex-concave sections. We call a pair of convex-concave sections a *segment*, and use these segments to determine visibility along a line.

From now on we refer to a height field sample at step i as point p_i, denoting its height by h_i and its distance from the start of the line by d_i. In this notation p_0 is the first point on the line and p_n is the receptor R. When traversing a line, each point p_i is determined to belong to either a convex or a concave section of the line by the following function:

$$C(p_i) = \begin{cases} \textit{convex} & \text{if } i = 0 \quad \text{or} \quad 2h_i > h_{i-1} + h_{i+1}, \\ \textit{concave} & \text{if } i \neq 0 \quad \text{and} \quad 2h_i < h_{i-1} + h_{i+1}, \\ C(p_{i-1}) & \text{otherwise} \end{cases} \tag{1}$$

Note that in case the three points p_{i-1}, p_i and p_{i+1} forming a straight line the convexity status is inherited from the previous point.

The *visible* segments can be found using local horizons as demonstrated in Figure 3. The local horizons are lines-of-sight formed between R and points p_i^R on the surface such that each line-of-sight is locally tangent to the surface at p_i^R and does not intersect the surface between R and p_i^R. The line-of-sight between p_0 and R is also a horizon if it does not intersect the surface.

There are as many local horizons as there are visible segments along the line, and each horizon lies on the convex part of a visible segment. The beginning of the visibility is available as the endpoint of the horizon, p_i^R. The end of the segment's visibility generally lies between two surface points and its coordinate is not directly available from the horizons. Instead, the next horizon (through p_{i+1}^R and R) intersects the visible segment exactly at the end of the visibility. The last visible segment includes R and does not have a following horizon, in which case the visibility reaches all the way to p_{n-1}. The least amount of information required to describe the complete line visibility from R is an array of the unsigned integer distance values d_i of p_i^R.

In order to iteratively derive the local horizons for each R (p_n) along the line without having to go over points $p_0 \dots p_{n-1}$ each time, we track a convex hull for each segment, starting from the beginning of the segment (p_j) and ending in p_n as demonstrated in Figure 4. The convex hulls therefore are the groups of points ordered by their distance:

$$\left\{ p_{S_j}, \{j, \{i_c\}, n\} \in S_j \text{ and } j < i_c < n \text{ and} \right.$$
$$\left. \frac{h_{S_j[i]} - h_{S_j[i-1]}}{d_{S_j[i]} - d_{S_j[i-1]}} > \frac{h_{S_j[i+1]} - h_{S_j[i]}}{d_{S_j[i+1]} - d_{S_j[i]}} \right\}, \tag{2}$$
$$j = 0 \text{ or } (C(p_j) = \textit{convex} \text{ and}$$
$$C(p_{j-1}) = \textit{concave} \text{ and } 0 < j < n)$$

In other words, we defined the set of upper convex hulls that start from the first point in each convex section and end in R. The first convex section always starts from p_0.

Convex hulls are efficient in determining occlusion between a receiver that is included in the hull and geometry behind the hull: points on the hull can have a direct

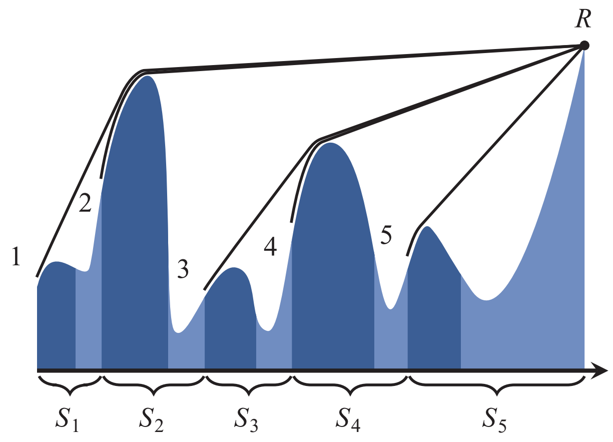


Figure 4: *A set of convex hulls (1 . . . 5) are formed from the convex sections of segments $S_1 \dots S_5$ to the receptor R (p_n) at the far right.*

line-of-sight only to their neighbors and thus the highest occlusion is cast by the point previous to the receiver in the hull. The last point of each convex hull defined in Equation 2 is the receptor R, and the points second to last are p_i^R. Therefore the edges between the last two points of the hulls are the local horizons for R. Duplicate horizons, however, emerge from this definition: the convex hull of each *invisible* segment produces the same horizon as the convex hull of one visible segment. This is due to the possibility of convex hulls sharing points close to R. In fact, the vast majority of the segments are usually invisible.

To overcome this problem we maintain a *convex hull tree* instead of maintaining a separate convex hull for each segment. Points included in the convex hull tree are the union of points in the separate convex hulls as defined in Equation 2. Each separate convex hull is still existent as part of the convex hull tree and we call such a part a *convex hull path*. A path will start from p_j (a *leaf node* of the tree) and end up in R (the *root node*). The branches are ordered such that the first child of a parent is the one farthest away from the parent or—equivalently—having the highest height of the children. There are no redundant points in the tree and the direct children of the root node are exactly the points p_i^R. Therefore, when the convex hull tree is up-to-date, extracting the local visibility horizons involves nothing more than going over the children of the root node. Figure 5 demonstrates a convex hull tree in a fractal terrain height field along one line.

When a new step along the line is taken, a new R is introduced and becomes the new root of the tree. The core algorithm for determining visibility using the convex hull tree therefore adjusts the tree after the introduction of a new root such that each convex hull path is valid (convex). This algorithm is recursive in nature and described next.

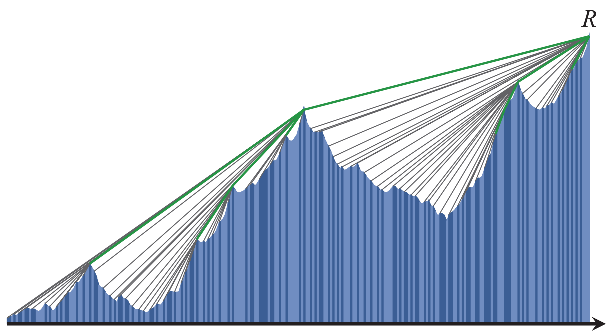


Figure 5: *A convex hull tree along one line during a sweep on a fractal terrain. Green links are shared by multiple convex hull paths.*

5. Point Processing in the Convex Hull Tree

In this section we describe how a new point is added to the present convex hull tree. The input of this phase is the next height field point p_n along the line being traversed. The point is first added as the new root of the convex hull tree, making the previous point p_{n-1} (the old root) its only child. The tree is then processed using a recursive algorithm until all paths from the root to the leaves are convex. As the algorithm is applied incrementally it can be assumed that the paths were valid before the addition of the new root. Therefore, it is enough to process paths only to the point where convexity once again holds.

The algorithm is first invoked using the triplet (root's first child's first child → root's first child → root) as its parameter. We are naming the elements of such a triplet ($child_T \rightarrow parent_T \rightarrow root$). The last element of the triplet will always be the root element of the tree, and $child_T$ the first child of $parent_T$. First, the algorithm checks whether the vertices of the triplet are convex. If they are, no action is needed and the call returns. If the convexity check fails, then $parent_T$ needs to be removed from this path, causing $child_T$ to be connected directly to $root$. The edge from $root$ to $child_T$ will be above the edge from $root$ to $parent_T$, and therefore the correct position for $child_T$, as a $root$'s child, is before $parent_T$. If $child_T$ was also the last child of $parent_T$, $parent_T$ gets orphaned and is removed. Otherwise the second child of $parent_T$ takes the place of the first child. After these changes it is necessary to proceed both one step deeper and one step wider from $child_T$ in the tree. Figure 6 illustrates the described process.

When proceeding deeper, the previous $child_T$ becomes the new $parent_T$ and the first child of $child_T$ becomes the new $child_T$. When stepping wider, $parent_T$ stays the same and its (newly assigned) first child becomes the new $child_T$. The process continues recursively until the convexity checks for each branch pass and the algorithm stops, at which point all convex hull paths are valid. Algorithm 1 lists the pseudocode

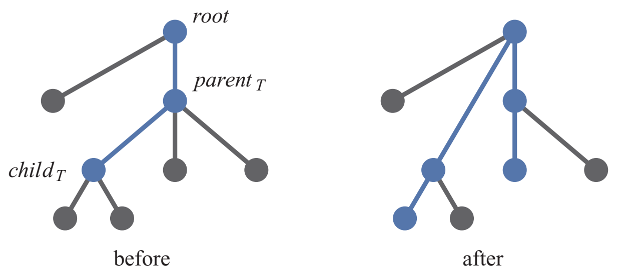


Figure 6: *Triplet ($child_T \rightarrow parent_T \rightarrow root$) fails the convexity check (left), causing $child_T$ to disconnect from $parent_T$ and connect to $root_T$ (right). Two new triplets are then processed, shown in blue.*

for the recursive function. After the convex hull tree is valid again, the visibility information is available as *root*'s direct children (as a linked list) as described in Section 4.

Finally, after the convexity has been established, we determine whether the previous step started a new segment. When the beginning of a segment is detected, the segment's first element is duplicated in the tree and set to be the last child of the root. The duplicated elements form the leaf nodes of the convex hull tree and are permanent throughout the line.

6. Complexity

In this section we observe the complexity of our convex hull tree processing algorithm on a line of n steps. When sweeping through an $N \times N$ height field, $1 \leq n \leq \sqrt{2}N$. Let m_i denote the number of visible horizons and t_i the number of iterations of Algorithm 1 at step i, $0 \leq i < n$ on the line. Then the total number of horizons on a line is given by $m = \sum_{j=0}^{n-1} m_j$ and the total number of iterations by $t = \sum_{j=0}^{n-1} t_j$. We first prove that our algorithm's complexity is linear in the total number of produced output horizons ($t = O(m)$), and then analyze the complexity of the horizons.

Algorithm 1: *RecConvexity($child_T$, $parent_T$, root)*

```
if !convex($child_T \rightarrow parent_T \rightarrow$ root)
    connect $child_T$ to root before $parent_T$
    if $child_T$ has a next sibling
        first child of $parent_T \leftarrow$ next sibling of $child_T$
        // Step wider
        RecConvexity(next sibling of $child_T$, $parent_T$, root)
    else
        delete $parent_T$

    if $child_T$ has a first child
        // Step deeper
        RecConvexity(first child of $child_T$, $child_T$, root)
```

Table 2: *The average number of horizons m_i and iterations t_i for one azimuthal direction at a given point (denoted by m_a and t_a, respectively).*

Height field	m_a	t_a	t_a/naïve
Fractal terrain	9.25	11.7	2.4%
Brick surface	3.06	4.80	0.99%
Sine grid	1.68	3.02	0.62%
Blocks	2.55	3.76	0.78%

The total number of horizons m on all points of a line of length n is at least n. We distinguish between three types of iterations of the algorithm and show that these are either $O(n)$ or $O(m)$:

(1) An iteration that fails the convexity check with a child that does not have a next sibling results in the deletion of the parent node. As there are at most n elements in the tree, there can be at most n iterations of this type.

(2) An iteration that passes the convexity check will return without further invocations of the algorithm. Each iteration of this type corresponds to exactly one visible horizon formed by the parent node (directly connected to the root). Also, no other iteration can produce the same horizon because such an iteration would need to have the same parent, and all such iterations would need to emanate from this iteration. Therefore the number of iterations of this type will be exactly m.

(3) The last iteration type is the one for which the convexity check fails, but results in the child being detached from its previous parent and connected to the root without the parent being deleted. As the parent and its previous sibling were directly connected to the root and formed consecutive horizons before this iteration, the iteration will introduce a new horizon (formed by the child) between the two. As this inevitably increases the number of horizons, there can be at most m iterations of this type.

The total number of iterations t is therefore at most $2m + n$, or, of complexity $O(m)$.

Table 2 presents empirical results for the number of visible horizons and iterations of Algorithm 1 for height fields shown in Figure 7. The figures are measured from sweeps in 256 directions ($K = 256$) and averaged for one height field point and for one azimuthal direction. The height fields are of size 1024^2 ($N = 1024$) and the naïve method performs 484 iterations per point on average. Compared to this figure, the number of iterations required to produce the visibility information is reduced by two orders of magnitude.

As a visibility description, horizons are in all cases at least as compact as a point-to-point description: if the visibility

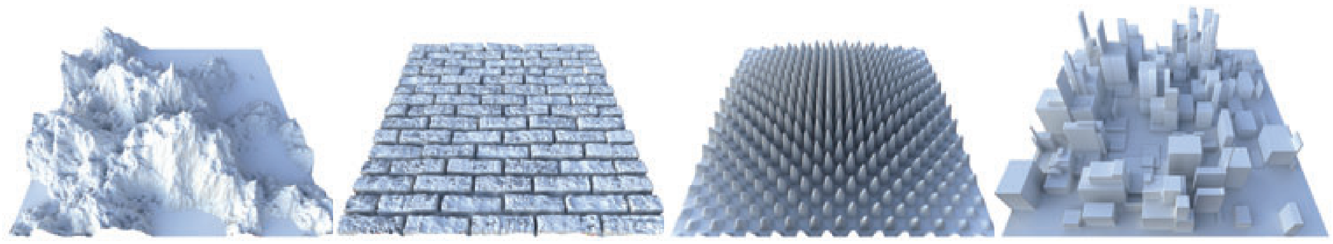


Figure 7: *The four 1024^2 height fields used as test data.*

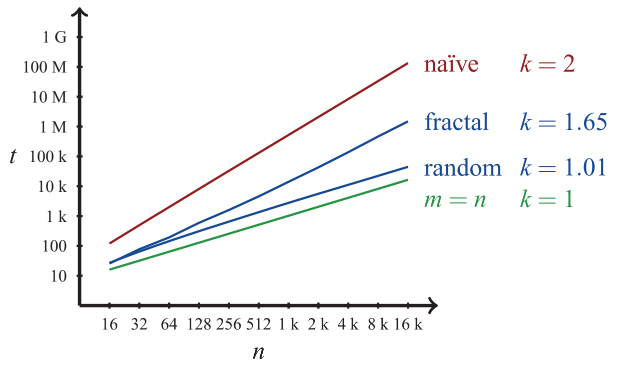


Figure 8: *The number of required iterations t for a line of n steps for fractal terrains and random data. The naïve method (in red) and linear scaling (in green) as references.*

consisted of individual scattered points there would be one horizon (that can be expressed by a single surface coordinate) for each point. In practice it is common that the visibility of each segment spans multiple consecutive points for which only one horizon is required. This can be seen by comparing Tables 1 and 2.

Determining visibility for a line of n samples using our method has the worst-case complexity equal to that of the naïve method defined in Section 1, $O(n^2)$. This happens, for instance, in a bowl-shaped height field where every other point is depressed. On the other hand, if the height field is dome-shaped, the complexity is $O(n)$. We measure practical complexity by presenting iteration counts for various height field examples and by measuring scaling in n.

In order to measure scaling, we differentiate between two types of content scaling which most practical height fields exhibit a combination of. First, we measure increasing *detail* using fractal terrains: every time n is doubled, more resolution is added between the existing points using a uniform distribution with half the range. Second, we measure increasing *extent* with a constant level of detail using random data: when n increases, more height data is randomized from a fixed finite height range. Figure 8 shows the average number of iterations t for a line of length n as a function of n. We measure scaling by k in $t = O(n^k)$. For the naïve method $k = 2$, and for linear scaling $k = 1$. As the axes are logarithmic, we fit first-order polynomials to the graph (for $n \geq 512$) and attain k from their slopes. Visibility of the fractal terrains seem to exhibit a scaling of roughly $O(n^{1.65})$, whereas the random data quickly settles to near-linear scaling of $O(n^{1.01})$. The data suggest that the scaling in most practical cases is well below the quadratic scaling of the naïve method.

We noticed that the average segment length in these two height field examples is the same for all n, indicating that the scaling is due to changes in the amount of visible horizons, not due to changes in the average coverage of one horizon. This means that the number of visible points as listed in Table 1 would scale similarly, and the observed complexity applies generally to visibility and not just to our visibility description. An increase in detail could, however, also cause an increase in segment lengths. This would be the case if the sine grid shown in Figure 7 were to scale up without the grid size changing (the number of sine 'domes' staying the same): every point would continue to have the same number of visible segments but the number of points belonging to them would increase. This type of increase in detail would yield linear scaling in our algorithm, and demonstrates the power of the compactness of our visibility description.

7. Implementation

As current hardware accelerated graphics libraries have a fixed rasterization stage that does not allow writing lines into a framebuffer, we have chosen to use GPGPU. We use OpenGL 4 and CUDA 3 in our implementation, and in this section use CUDA terminology. Notes on performance apply to the NVidia Fermi architecture [Nvi09].

The high-level framework begins with the passing of the source height field as a texture from OpenGL to CUDA. Visibility calculations are then performed in one kernel, one thread mapping to one line in the height field. As many azimuthal directions are processed simultaneously as allowed by the amount of available graphics memory. One output buffer for each K is produced, of which $\pi/2$ rotations are linearly accumulated in CUDA reducing the number of buffers to a quarter. Once the resulting $K/4$ buffers have been passed back to OpenGL as textures, they are sampled and additively blended in a floating point frame buffer.

In the actual visibility algorithm we implement a node of a convex hull tree by allocating the following:

(i) one half-precision (16 bit) float for the height h_i
(ii) one 15 + 1 bit unsigned integer for the distance d_i (also index of self, i)
(iii) one 16 bit unsigned integer for the index of the first child (null if a leaf)
(iv) one 16 bit unsigned integer for the index of the next sibling

A node allocated this way fits into 8 bytes. Elements are stored in global memory using indices that correspond to element distances, with the exception of leaf nodes. Leaves are stored using indices one smaller than their distance which are known to be empty when the leaf is forked, and the 1 bit flag is set to denote the offset.

The distance-based allocation yields sparse arrays, but according to our benchmarks produced better overall performance than having separate distance and index data and packing the elements densely. The optimal data layout depends on the content of the height field and the target GPU architecture due to memory coalescing and caching efficiency. We found good overall performance from a quasi-parallel allocation in which the same indices of two to eight consecutive threads are sequentially laid out in memory. Using the maximum amount of 48 kB of L1 cache on each multiprocessor for global memory accesses also produced the best performance. A cacheless architecture would not benefit from the temporal coherence of memory accesses and we expect a fully parallel allocation to yield best results on such GPUs.

To implement the recursion in Algorithm 1, we use a stack stored in global memory. The quasi-parallel allocation scheme used for the convex hull trees produced the best performance for the stacks as well.

In Algorithm 1, when the convexity check fails and $child_T$ is connected to $root$, it is necessary to adjust both $parent_T$ and its previous sibling. However, as we use a single linkage scheme in which each node has links to its next sibling and to its first child only, we need to carry both the parent and its previous sibling (called $parent_{prev}$ from now on) by pushing them onto the stack. The use of single linkage also makes breadth-first traversal preferable for the following reason: current $child_T$ will be the $parent_{prev}$ of the next breadth iteration, however if depth is processed first and all children of $child_T$ are connected to the root, then $child_T$ is removed making the previous $parent_{prev}$ obsolete. Tracking the changing of $parent_{prev}$ would require another stack.

Algorithm 2 OptimizedConvexity(root, inSafeZone)

Functions self(n), child(n), and next(n) return memory locations of node n, node n's first child, and node n's next sibling, respectively. Function read(p) returns the node from memory location p and write(n) stores node n to memory location self(n). Operators are C style, and calls requiring memory accesses are underlined.

```
preSet = true
regression = null

// { $child_T$, $parent_T$, $parent_{prev}$, history }
s = { child(child(root)), child(root), null, 1 }

while stack not empty || preSet
    if !preSet
        s = stack.pop()
    preSet = false

    if self(s.$parent_T$) == self(regression)
        s.$parent_T$ = regression

    if self(s.$child_T$) &&
            !convex(s.$child_T$ $\rightarrow$ s.$parent_T$ $\rightarrow$ root)
        if !self(s.$parent_{prev}$)
            child(root) = self(s.$child_T$)

        if !inSafeZone && (child(s.$child_T$) ||
                child(s.$parent_T$) == self(s.$child_T$))
            stack.push(child(s.$child_T$) ?
                read(child(s.$child_T$))
                    : null, s.$child_T$, s.$parent_{prev}$, s.history >> 1)

        if next(s.$child_T$) || next(s.$parent_T$)
            s.$parent_{prev}$ = s.$child_T$
            if next(s.$parent_{prev}$)
                s.$child_T$ = read(next(s.$parent_{prev}$))
            else
                s.$child_T$ = null
                s.$parent_T$ = next(s.$parent_T$) ?
                        read(next(s.$parent_T$)) : null
            s.history = 2
            preSet = true
    else
        // & is bitwise and, ^ is bitwise xor
        if self(s.$child_T$) && s.history & 2
            child(s.$parent_T$) = self($child_T$)
            write(s.$parent_T$)
        if self(s.$parent_{prev}$) && s.history ^ 1
            next(s.$parent_{prev}$) = self(s.$parent_T$)
            write(s.$parent_{prev}$)
        regression = s.$parent_{prev}$
```

An optimized implementation of Algorithm 1 is listed as Algorithm 2. As established, each time the convexity check fails $child_T$ is connected to the root causing changes to linkage. With some state tracking (variables *regression* and $history$) these changes can be postponed and consolidated to the fail block (lines 32–38) by pushing extra entries onto the stack that have a null child. This also allows more efficient hardware scheduling as threads do the expensive global writes in one place instead of blocking the execution in several places. As a result, reads were halved and writes were cut down to about a tenth as compared to an algorithm that uses single linkage and flushes changes to memory immediately. Also, as pushes onto the stack are large and stress the memory system, the latest stack element (breadth traversal) can be passed on in a register variable, cutting down stack accesses to a quarter on average.

There are two cases when running the convexity algorithm is unnecessary. The first is when the new sample forms the new root and inherits the old root as its child without further changing the tree. The second and more important case is when the new sample replaces the old root without affecting the rest of the tree. Without specially treating this case all children of the old root are traversed, and, as their convexity checks fail, their first children are also traversed, whose convexity checks all pass. This is an expensive operation, and depending on height field content, may occur frequently. The first condition for the old root being straightforwardly replaced is when all its children get connected to the new root. This can be easily tested for by checking if the convexity check for (the last child of the old root → the old root → the new sample) fails, in which case all previous siblings of $child_T$ also fail. The second condition is that there will be no other changes to the children of the old root, which is a little harder to test. Essentially it is necessary to know whether the convexity checks on the first grandchildren of the old root will pass with the new root as well.

We address these situations by maintaining a *safe zone* between two distance boundaries that enclose the current and some future steps. Edges from the first grandchildren of the root to their parents are then projected against the distance boundaries. The line between the lowest intersection points at the boundaries forms a conservative upper bound for the safe zone, demonstrated in Figure 9. When the old root is determined to be replaced by a new sample landing within the safe zone, height and distance of the root can be safely renewed by those of the new sample.

The expense of being able to handle this special case efficiently is the need to keep the safe zone up-to-date by iterating over the first grandchildren of the root whenever the convex hull tree changes or the second boundary is stepped over. Algorithm 3 describes the process of updating the safe zone at boundaries b_1 and b_2. Although the performance impact of this optimization depends on the height field content, it was always beneficial in our benchmarks and improved

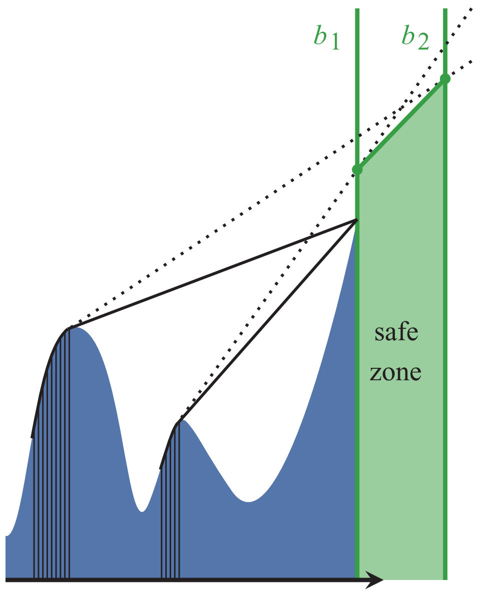


Figure 9: *The lowest projected heights of the second level horizons on the boundaries b_1 and b_2 form a zone where it is safe to replace the root.*

the performance on average by 50%. As a minor optimization, the safe zone information is used to limit the convexity algorithm (line 19 of Algorithm 2) in cases where the root is not being completely replaced and the convexity algorithm has to be invoked, but no grandchildren are affected.

Algorithm 3: *SafeZoneUpdate(root, b1, b2)*

```
b1_h, b2_h ← ∞
c ← first child of root
do // Loop over root's children
    if exists gc ← first child of c
        // Project grandchild-child line on boundaries
        b1_h ← min(b1_h, line(gc → c) at distance b1_d)
        b2_h ← min(b2_h, line(gc → c) at distance b2_d)
while exists c ← next sibling of c
```

8. Efficiency

In this section we discuss the efficiency of our implementation on an NVidia GTX 480 graphics card in order to give a frame of reference to the previous and the following sections. The two aspects we focus on are the utilization of computational resources and the efficiency of memory accesses. The naïve method is highly efficient from both aspects: neighboring height field points undergo almost exactly the same processing, allowing high utilization, while requiring sampling on neighboring height field coordinates at all times, allowing for optimal texture cache efficiency.

Concurrent threads in our algorithm, however, may perform significantly different amounts of computation per step.

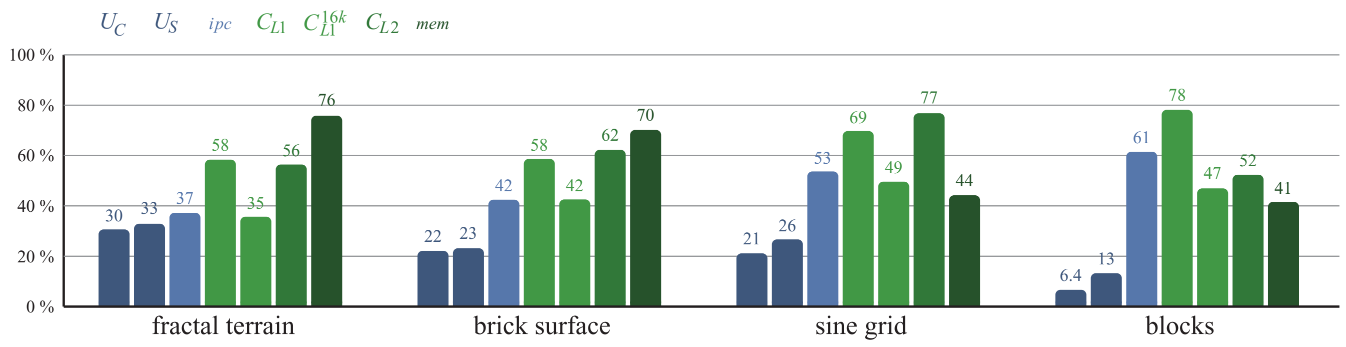


Figure 10: *Efficiency measurements for four test height fields. From the left, the metrics are: utilization in convexity algorithm (U_C), utilization during safe zone update (U_S), instruction rate (ipc), L1 cache hit ratio (C_{L1}), 16 kB L1 cache hit ratio (C_{L1}^{16k}), L2 cache hit ratio (C_{L2}), and DRAM bandwidth (mem).*

In current GPU architectures this yields low SIMT utilization as threads within a warp will have to wait until the longest running thread is finished. Memory accessing in our algorithm is not quite optimal either: elements of convex hull trees are accessed in a pattern where concurrent threads rarely access consecutive memory locations. Also, one thread rarely accesses consecutive indices of its own convex hull tree in a temporally local manner, but rather switches from a branch to the next in subsequent iterations of the algorithm. Therefore there is little *immediate* coherency to be exploited for either efficient memory coalescing or caching. Fortunately, however, the data set for one thread in an N^2 height field is $\sqrt{2}N \times 8$ bytes at most, significantly less on average, and only a small portion of the tree is traversed at each iteration, making transparent caches useful afterall.

To quantify the aspects of computational utilization and memory access, we use seven separate metrics:

Utilization in convexity algorithm (U_C) measures the ratio of active threads in a warp (that consists of 32 consecutive threads) during the while loop in Algorithm 2. This measure reaches 100% when all threads in a warp execute the same number of iterations.

Utilization during safe zone update (U_S) measures the ratio of active threads in a warp during the safe zone update which requires looping over the children of the root. This measure reaches 100% when all threads simultaneously decide to update the safe zone and do so in the same number of iterations, e.g. have the same number of visible horizons.

Instruction rate (ipc) measures the average number of instructions issued per clock cycle. A multiprocessor in the GF100 architecture is able to issue at most 2 warp-wide instructions per cycle.

L1 cache hit ratio (C_{L1}) measures the ratio of memory requests that were satisfied by the L1 cache. This includes accesses to both the convex hull trees and the stacks.

16 kB L1 cache hit ratio (C_{L1}^{16k}) measures the ratio of memory requests that were satisfied by the L1 cache when configured to 16 kB.

L2 cache hit ratio (C_{L2}) measures the ratio of memory requests that could not be satisfied by the L1 cache but were satisfied by the L2 cache.

DRAM bandwidth (mem) measures the amount of bytes per second read or written by the memory controller relative to the maximum throughput as measured by the *bandwidthTest* tool in NVIDIA GPU Computing SDK.

Data for the first two metrics are gathered internally by our algorithm. Only the number of loop iterations is measured; the loss in utilization caused by divergence during if-else branching is not included in the measurement. Of these two metrics U_C is more important as the convexity algorithm dominates the overall execution time. Data for the last five metrics are extracted using NVIDIA Compute Visual Profiler. Unless otherwise stated, the L1 cache is configured to 48 kB.

There are four main configurables in our algorithm. Optimal values for them depend on the height field content, but as the values are currently not automatically adjusted at runtime, we use the same values for all height fields. The configurables and the values used are:

- Thread block size, 64
- The number of neighboring height field lines packed together in a thread block, 64
- The number of consecutive threads for which the same indices in the convex hull tree are laid out sequentially in memory, 4
- The number of consecutive threads for which same indices in the stack are laid out sequentially in memory, 8

Table 3: *The average sweep time for the naïve and our method, and relative speedup.*

Height field	Time	Speedup
naïve	21.2 ms	
fractal terrain	8.85 ms	*2.4x*
brick surface	5.05 ms	*4.2x*
sine grid	1.61 ms	*13x*
blocks	0.52 ms	*41x*
Figure 11	3.14 ms	*6.8x*
Figure 12	0.59 ms	*36x*

In Figure 10 we observe the seven metrics for the four test cases shown in Figure 7. We consider the algorithm efficient when measured by ipc, however the utilization U_C is relatively low and has a high impact on the effective instruction rate *per thread*. A work queueing approach might be able to balance load and improve efficiency considerably, but is beyond the scope of our paper. Our algorithm is also memory intensive, and caching plays an important role in overall performance. An increase in the amount of L1 cache per thread would further reduce the strain on the memory subsystem, and would likely improve performance.

9. Performance

We measure our visibility calculation performance against the naïve method implemented as an OpenGL fragment program. Table 3 lists the average time taken to perform a single sweep on the test height fields. The performance of the naïve method does not depend on the height field content. Despite the shortcomings in the efficient mapping of our algorithm to current GPUs as discussed in Section 8, our method is still significantly faster than the naïve method in all test cases.

Choosing the number of azimuthal directions K is a balancing act between performance and approximation accuracy. In principle, to cover every single height field point for each receiver point, $O(N)$ directions need to be processed. A lot fewer are usually adequate, but the appropriate value of K depends on e.g. the geometric content, BRDF, and frequency and intensity of the surface exit radiance. Different values of K for the indirect lighting component are demonstrated on a diffuse monochromatic surface under outdoor lighting in Figure 14. Solving visibility with $K = 32$ over the varying geometry in Figure 11 is achieved at 10 fps.

10. Discussion

10.1. Lighting

Although lighting methods that utilize the visibility information produced by our visibility algorithm are future work, in this section we outline ways to utilize our visibility description and also demonstrate, through trivially gathering the surface radiance, that also lighting within the lowered complexity is possible.

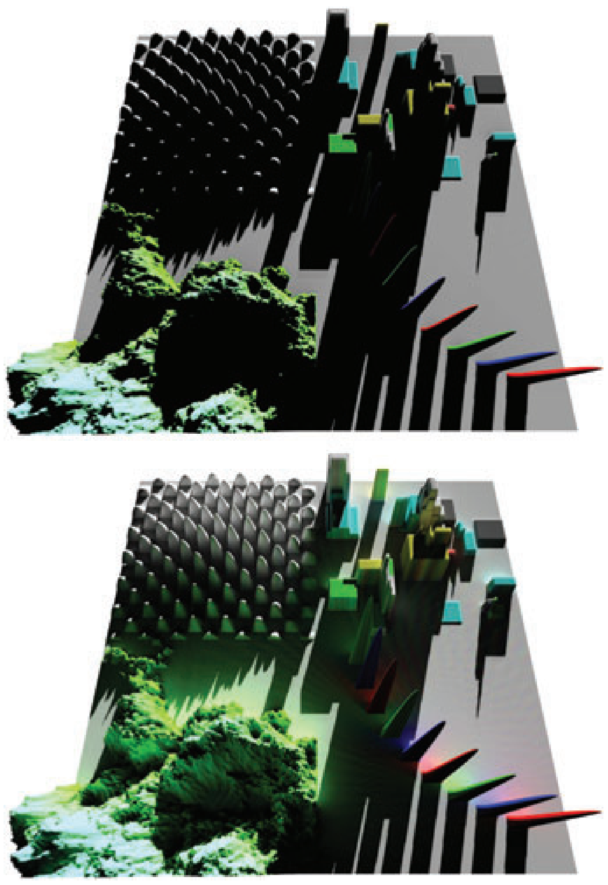


Figure 11: *Top: direct illumination from a point light source on a 1024^2 height field. Bottom: an additional indirect light bounce. Accurate illumination in 64 directions is achieved in 1.21 seconds per frame (19 ms per direction).*

The visibility horizons as produced by our method provide exact angular coordinates for both the beginning and the end of a segment's visibility. The Cartesian coordinate is only available for the beginning. We suggest three ways to utilize this type of visibility information, in an order of increasing computational complexity:

(1) As a line on the height field is traversed, record its accumulated exit radiance in a way that allows sampling the total segment radiance using the angular coordinates.
(2) Record the accumulated exit radiance so that it can be sampled using distance coordinates. To find the exact Cartesian end coordinate of the visibiliy use any of the various existing intersection search algorithms. Two properties might prove useful: (i) there is exactly one intersection point between the horizon line and

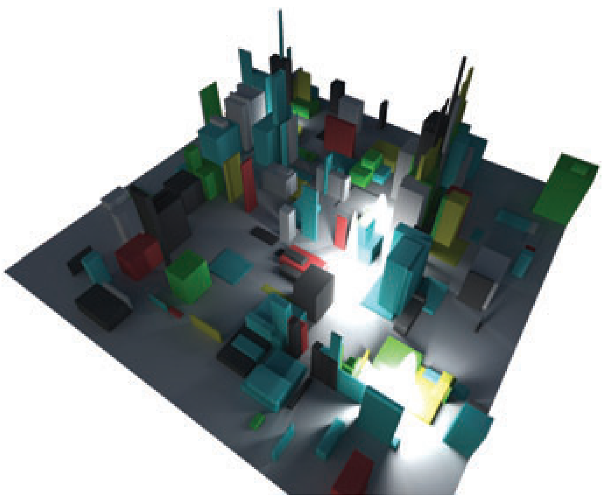


Figure 12: *A 1024^2 height field with self-illuminating parts. Illumination in 256 directions achieved in 1.68 seconds per frame (6.6 ms per direction).*

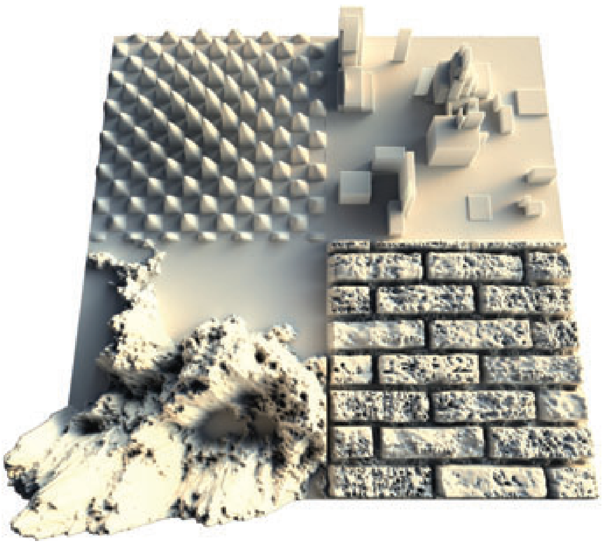


Figure 13: *The height field used for error testing. The surface exhibits diffuse reflection and is lit under outdoor lighting.*

the segment and (ii) the behaviour of the derivative is known due to the convexity/concavity requirements.

(3) Traverse the visible points one at a time by starting from each beginning coordinate and stepping towards the receiver point until below the next horizon (cf. Figure 3). This allows sampling of only the visible points (plus the partially visible at each horizon boundary). The visible segments can be traversed in parallel.

The potentially visible height field points and their exit radiance along the line have already been traversed when it is time to determine incident radiance for a receptor. Therefore we find promise in fast lighting methods that accumulate a description of exit radiance per segment such that it can, ideally, be sampled directly with the information available in our visibility horizons (alternative 1).

We demonstrate a trivial use of the 3. (slowest) alternative by sampling through the exit radiance one point at a time and computing the contribution on the receiver analytically. While this method is slow compared to visibility determination, it performs in the complexity of Table 1 and Figure 8 while producing exact lighting. Figure 11 demonstrates global lighting with one indirect bounce under direct lighting of a single point light. The complete lighting performed this way is faster than only finding the visible points using the naïve method.

Figure 12 demonstrates global lighting (direct lighting in a dim 256×256 lighting environment plus one indirect bounce) with parts of the height field emitting light by themselves. In this case, we achieve pixel-perfect lighting with an unbound extent at a per-direction rate higher than that of the approximative method in [NS09]. Multiple indirect bounces can be simulated by repeating the indirect lighting phase using the output of the previous iteration as the new surface radiance.

10.2. Error analysis

Errors in our method come from (i) sampling along each line, and (ii) the discretization of the visibility search into K azimuthal directions. All image-space methods suffer from the first issue: geometry is visited at sampled points, which generally are not the original height field points. Using heavy supersampling along a line is considered to produce the ground truth. The more identifiable approximation in our method is the discretization into azimuthal directions. Visibility solved using a large value of K represents the ground truth in this aspect.

In this section we analyze the error introduced by these two issues as compared to the respective ground truths. As the test case, we use a diffuse height field under outdoor lighting, shown in Figure 13. The error is visualized in Figure 14 and average error plotted in Figure 15. Generally the error in indirect illumination is amplified by high-frequency details in geometry and glossiness of the surface.

Azimuthal undersampling results in banding that is especially apparent on flat regions of a height field, however less visible on uneven regions. One might argue that our method is imbalanced for determining visibility along each line accurately while crudely undersampling azimuthally, however quantitative analysis implies that accurate visibility determination is necessary even for a small number of azimuthal

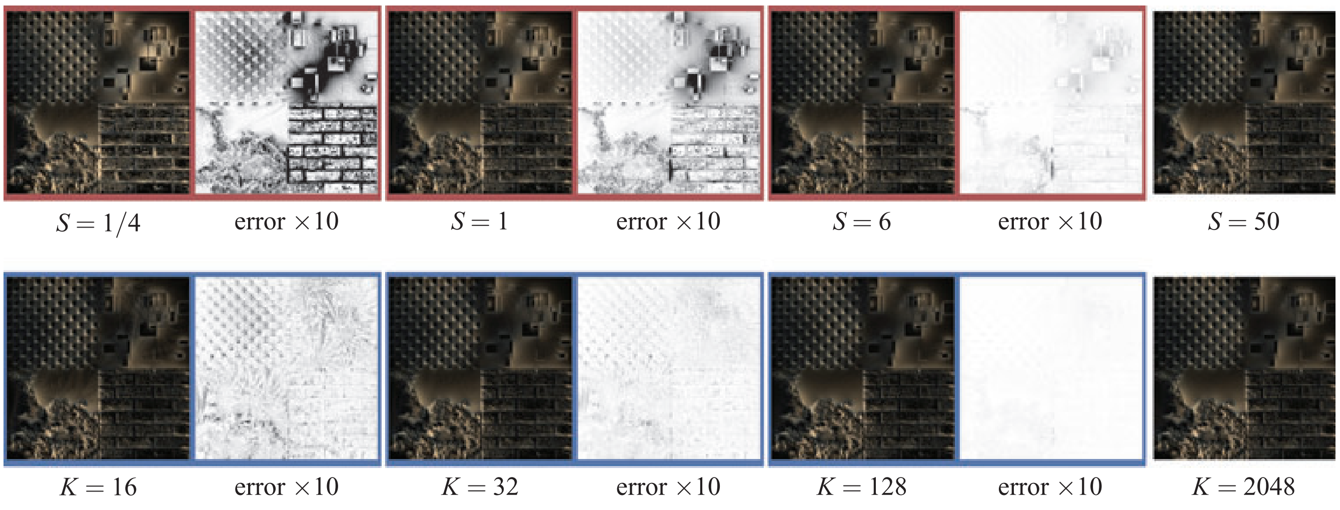


Figure 14: *The indirect illumination component from Figure 13 and the corresponding error (white = 0%, black = 10%) against ground truth (right) for different levels of visibility supersampling (on red) and values of azimuthal directions (on blue).*

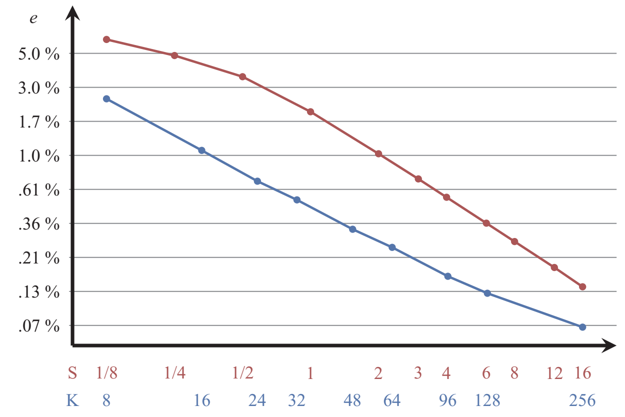


Figure 15: *The average error e per pixel for different values of azimuthal directions K and supersampling S. The axes are logarithmic.*

directions: Figure 15 shows that $K = 16$ introduces an error that is much smaller than that coming from unit length sampling ($S = 1$). Qualitatively though, banding can be the most outstanding visual artefact and in the next section we discuss a way to trade it for noise.

10.3. Future work

As future optimizations, it may be possible to trigger a simplification of the convex hull tree for parts far away from the receptor every few dozen steps. Also because the algorithm has traversed the data of the line already, the simplification process can utilize the exit radiance and geometry information along the line to estimate the impact of the simplification in order to control the error in the resulting approximation. Any such simplification can also be performed while processing height field samples for the first time. For example, it might be a reasonable optimization to ignore fine levels of roughness on otherwise flat surface regions and favor extrusions over depressions when determining exit radiance from the simplified regions. Other, non-sweep based methods, are unaware of the contents of a line before taking the actual samples, and therefore cannot trivially apply said data-aware optimizations.

Previous methods based on the naïve method that perform an independent visibility search for each receiver point can randomize azimuthal directions per receiver, and therefore trade banding for noise. Trading banding for noise is also possible in our method: one can perform a sparse sweep in each direction (process every n-th line) while simultaneously increasing K. Afterwards, when gathering results for a point in the result buffer, one accumulates lighting only from directions that have a line that crosses the receiver point. Furthermore, a line does not have to strictly cross the receiver, but a configurable distance epsilon can be used to provide a way to trade noise for blur. Randomizing azimuthal directions in previous methods incurs a penalty of lowered texture cache hit ratio, the impact of which can be significant. Our method takes very few height field samples per receiver and is not bottlenecked by sampling, making it practically immune to this penalty.

When it comes to applications, height field methods have proven useful in producing lighting effects in screen space. The main approximations of depth buffer geometry are (i) not counting geometry outside the visible frame buffer and (ii) taking only the first (visible) depth layer into account. It is possible to alleviate these limitations [BS09], and we expect our algorithm to prove useful in producing properly occluded *global* screen-space indirect illumination effects not yet seen in interactive graphics. Thus an interesting avenue of further research is to investigate the possibility of producing global scene lighting entirely in screen space, with light sources rendered in HDR as part of the scene geometry.

11. Conclusion

Determining intervisibility on surfaces of objects is a problem that needs to be solved in various applications, including global illumination systems in computer graphics. Unfortunately, the problem is complex and its current solutions computationally expensive. For height field geometry, the problem can be reduced from 2.5D to 1.5D domain by approximating visibility in discrete azimuthal directions. The most compact way currently known to express intervisibility in the 1.5D case are local visibility horizons.

In this paper we have presented a novel algorithm that determines local visibility horizons using *incremental convex hull trees*. Visibility from every height field point is determined to a number of azimuthal directions in time that is linear in the number of output visibility horizons, making the algorithm of optimal time complexity. We have showed that the proportion of visibility to the whole height field is low, giving our algorithm an advantage over the previous methods. In practice, we achieve a reduction by two orders of magnitude in the number of iterations required to produce the accurate visibility information. Our method is also amenable for GPGPU implementations and we have demonstrated that such an implementation can achieve significantly better performance than previous work.

Acknowledgments

I would like to thank Prof. Jan Westerholm for his advice and support and Dr. Jukka Arvo for his helpful comments on the manuscript.

References

[ADM*08] Annen T., Dong Z., Mertens T., Bekaert P., Seidel H.-P., Kautz J.: Real-time, all-frequency shadows in dynamic scenes. *ACM Trans. Graph. 27*, 3 (2008), 1–8.

[BS09] Bavoil L., Sainz M.: Multi-layer dual-resolution screen-space ambient occlusion. In *SIGGRAPH '09: Talks*. (New York, NY, USA, 2009) ACM.

[Bun05] Bunnell M.: *Dynamic ambient occlusion and indirect lighting*. Addison-Weseley Professional, 2005, pp. 223–233.

[BWW05] Bittner J., Wonka P., Wimmer M.: Fast exact from-region visibility in urban scenes. In *Rendering Techniques 2005 (Proceedings Eurographics Symposium on Rendering)*, Bala K., Dutré P. (Eds.), Eurographics, Eurographics Association, 2005, pp. 223–230.

[CoS95] Cohen-or D., Shaked A.: Visibility and dead-zones in digital terrain maps. *Computer Graphics Forum 14* (1995), 171–180.

[CS89] Cole R., Sharir M.: Visibility problems for polyhedral terrains. *J. Symb. Comput. 7*, 1 (1989), 11–30.

[DBBS06] Dutre P., Bala K., Bekaert P., Shirley P.: *Advanced Global Illumination*. AK Peters Ltd, 2006.

[DBS08] Dimitrov R., Bavoil L., Sainz M.: Horizon-split ambient occlusion. In *I3D '08: Proceedings of the 2008 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2008), ACM.

[DFM94] de floriani L., Magillo P.: Computing point visibility on a terrain based on a nested horizon structure. In *SAC '94: Proceedings of the 1994 ACM Symposium on Applied Computing* (New York, NY, USA, 1994), ACM, pp. 318–322.

[DSDD07] Dachsbacher C., Stamminger M., Drettakis G., Durand F.: Implicit visibility and antiradiance for interactive global illumination. *ACM Trans. Graph. 26*, 3 (2007), 61.

[FHT09] Fishman J., Haverkort H., Toma L.: Improved visibility computation on massive grid terrains. In *GIS '09: Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (New York, NY, USA, 2009), ACM, pp. 121–130.

[HBR*11] Huang J., Boubekeur T., Ritschel T., holländer M., Eisemann E.: Separable approximation of ambient occlusion. In *Eurographics 2011 - Short papers* (2011), 29–32.

[HLHS03] Hasenfratz J.-M., Lapierre M., Holzschuch N., Sillion F.: A survey of real-time soft shadows algorithms. *Computer Graphics Forum 22*, 4 (Dec 2003), 753–774.

[KZ02] Kaučič B., Zalik B.: Comparison of viewshed algorithms on regular spaced points. In *SCCG '02: Proceedings of the 18th Spring Conference on Computer Graphics* (New York, NY, USA, 2002), ACM, pp. 177–183.

[LS10] Loos B. J., Sloan P.-P.: Volumetric obscurance. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2010), I3D '10, ACM, pp. 151–156.

[Mit07] Mittring M.: Finding next gen: Cryengine 2. In *SIGGRAPH '07: ACM SIGGRAPH 2007 Courses* (New York, NY, USA, 2007), ACM, pp. 97–121.

[Nag94] NAGY G.: Terrain visibility. *Computers & Graphics 18*, 6 (1994), 763–773.

[NS09] NOWROUZEZAHRAI D., SNYDER J.: Fast global illumination on dynamic height fields. *Computer Graphics Forum: Eurographics Symposium on Rendering 28*, 4 (June 2009), 1131–1139.

[Nvi09] NVIDIA Corporation: *Next Generation CUDA Compute Architecture: Fermi. Whitepaper,* Santa Clara, CA, 2009.

[RGK*08] RITSCHEL T., GROSCH T., KIM M. H., SEIDEL H.-P., DACHSBACHER C., KAUTZ J.: Imperfect shadow Mmaps for efficient computation of indirect illumination. *ACM Trans. Graph. (Proc. of SIGGRAPH ASIA 2008) 27*, 5 (2008).

[RGS09] RITSCHEL T., GROSCH T., SEIDEL H.-P.: Approximating dynamic global illumination in image space. In *I3D '09: Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2009), ACM, pp. 75–82.

[SHR10] SOLER C., HOEL O., ROCHET F.: A deferred shading pipeline for real-time indirect illumination. In *ACM SIGGRAPH 2010 Talks* (New York, NY, USA, 2010), SIGGRAPH '10, ACM, pp. 18:1–18:1.

[SLYY08] SHEN Y., LIN L., YANG M., YURONG G.: Viewshed computation based on los scanning. In *2008 International Conference on Computer Science and Software Engineering* (Dec. 2008), vol. 2, pp. 984–987.

[SN08] SNYDER J., NOWROUZEZAHRAI D.: Fast soft self-shadowing on dynamic height fields. *Computer Graphics Forum: Eurographics Symposium on Rendering 27*, 4 (June 2008), 1275–1283.

[TW10] TIMONEN V., WESTERHOLM J.: Scalable height field self-shadowing. *Computer Graphics Forum (Proceedings of Eurographics 2010) 29*, 2 (May 2010).

Publication P3

Ville Timonen. Line-Sweep Ambient Obscurance. *Computer Graphics Forum*, 32(4), pages 97–105, 2013. *Eurographics Symposium on Rendering 2013*. Best Student Paper.

DOI: 10.1111/cgf.12155
Eurographics Symposium on Rendering 2013
Nicolas Holzschuch and Szymon Rusinkiewicz
(Guest Editors)

Volume 32 (2013), Number 4

Line-Sweep Ambient Obscurance

Ville Timonen[†]

Turku Centre for Computer Science
Åbo Akademi University

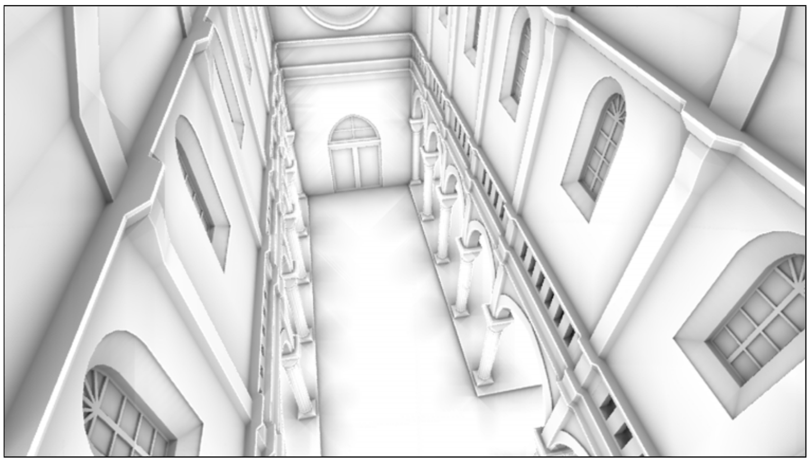


Figure 1: *SSAO rendered by our method at 1920×1080 (+10% guard band) in 1.7 ms on a GeForce GTX 480.*

Abstract

Screen-space ambient occlusion and obscurance have become established methods for rendering global illumination effects in real-time applications. While they have seen a steady line of refinements, their computational complexity has remained largely unchanged and either undersampling artefacts or too high render times limit their scalability. In this paper we show how the fundamentally quadratic per-pixel complexity of previous work can be reduced to a linear complexity. We solve obscurance in discrete azimuthal directions by performing line sweeps across the depth buffer in each direction. Our method builds upon the insight that scene points along each line can be incrementally inserted into a data structure such that querying for the largest occluder among the visited samples along the line can be achieved at an amortized constant cost. The obscurance radius therefore has no impact on the execution time and our method produces accurate results with smooth occlusion gradients in a few milliseconds per frame on commodity hardware.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Computer Graphics—Color, shading, shadowing, and texture

1. Introduction

Ambient occlusion and obscurance (AO) have become de-facto parts of global illumination implementations, and their screen-space evaluation (SSAO) has been widely adopted in real-time applications. Since its inception in 2007, SSAO has seen a steady line of improvements both in render quality and render time. However, its computational complexity has remained fundamentally the same. The distance of the AO effect is defined in eye-space and may thus cover a large range in screen-space, and high quality results still require

[†] e-mail: vtimonen@abo.fi

more samples per pixel than is practically affordable in real-time applications.

One established SSAO method is Horizon-Based Ambient Occlusion (HBAO) [BSD08] [Bav11] which accumulates obscurance from a set of azimuthal directions, finding the largest occluder in each direction by ray marching. While HBAO's physically based treatment of geometry scales well quality-wise, the number of samples that is necessary in each azimuthal direction for high quality results is prohibitive performance-wise. Given K azimuthal directions and N steps along each direction, HBAO's time complexity per pixel is $O(KN)$.

We propose a method to calculate the same results in $O(K)$ time with unlimited range. Instead of calculating the obscurance independently for each receiver pixel, we perform line sweeps over the screen. Each line is traversed incrementally, and the visited geometry along the line is stored in an internal data structure which can be queried in amortized constant time for the largest falloff attenuated occluder at the new pixel. This significant reduction in the time complexity of SSAO allows the fast production of high quality renderings which is not impacted by the range of the effect. Since we are not allowed to choose the azimuthal directions for each pixel freely, but rather use a set of directions shared by multiple screen pixels, our main visual artefact is banding which can be alleviated by increasing K.

2. Previous Work

Evaluating AO from the geometry in the depth buffer was first proposed by [Mit07] and [SA07] who sample a volume around the receiver pixel and determine the occlusion from the number of points that fall below the depth field. As scene geometry not merely blocks incoming light but also reflects it, an empirically selected falloff function [ZIK98] is usually introduced that weighs the sampled scene points according to their distance from the receiver by putting more weight to nearby occluders. Ambient occlusion extended by a falloff function has been termed ambient *obscurance*. Since [Mit07] and [SA07], several works such as [BS09] [LS10] [MOBH11] [MML12] have refined the quality and rendering speed of SSAO methods.

Evaluating each sampled scene point independently from each other ignores the fact that the evaluation of occluders in roughly the same azimuthal direction is not separable: A tall nearby occluder might make occluders behind it invisible such that these do not contribute to occlusion regardless of their elevation. To address this issue [BSD08] takes a more physically based approach along the lines of Horizon Mapping [Max88], whereby the highest horizon within a certain range is searched for a set of azimuthal directions. While this approach scales well with respect to image quality and obscurance radius, it is expensive because many height field samples have to be taken in each azimuthal direction. Our method produces results essentially identical to [BSD08] but we find the largest occluder in $O(1)$ time for one azimuthal direction within an unbounded radius.

We build upon the observation by [TW10] that sweeping through a height field in lines and incrementally building the convex hull of the visited geometry allows the extraction of global horizons in constant time for the purpose of horizon mapping. In ambient obscurance, where the falloff function attenuates occlusion with distance, the global horizon is often very far away and contributes little to occlusion, making the convex hull of little use in determining AO. Also, [TW10] scans the height field densely and accumulates rotated versions of the sweeps. This results in banding unless many azimuthal directions are scanned, which in turn becomes computationally expensive.

Our contribution over [TW10] is three-fold:

- Instead of using a geometrical convex hull, we form a hull based on the falloff weighted obscurance.
- We generalize the scans to arbitrary sampling densities and propose a way to gather results sparsely per-pixel, which allows trading high render times for edge-respecting blur when the number of azimuthal scanning directions is increased.
- We also suggest special line sampling patterns for cases where depth buffer values cannot be interpolated (often the case in SSAO).

3. Overview

The observation behind HBAO is that SSAO is physically separable per pixel in azimuthal directions. We also exploit this observation and calculate obscurance in K discrete azimuthal directions. For physically correct ambient obscurance, in each azimuthal direction the falloff weighted occlusion should be integrated from the tangent plane upwards until the global horizon as shown to the left in Figure 2. This results in unavoidably high complexity as obscurance has to be gathered in many segments.

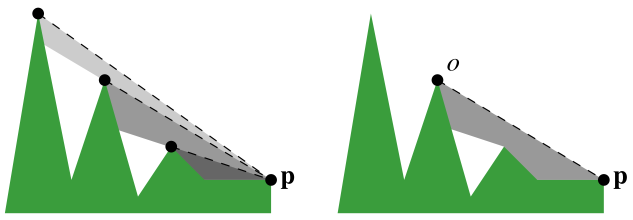


Figure 2: *Left: Obscurance gathered in segments of the elevation angle between the tangent plane and the global horizon onto receiver* **p**. *Each segment is shaded according to the strength of the falloff term. Right: Obscurance from the largest falloff weighted occluder, o, only.*

However, if we are willing to make the sacrifice that we calculate obscurance in each azimuthal direction from only one occluder along that direction—the one that casts the largest obscurance, as shown to the right in Figure 2—an order of magnitude more headroom is made available complexity-wise, which we in this paper show how to exploit. We define *largest occluder* as the occluder that would cast the largest amount of obscurance on the receiver were it the only occluder along the azimuthal direction. Fortunately, it turns out that the visual impact of only considering the largest occluder per direction is modest, as illustrated in Figure 3.

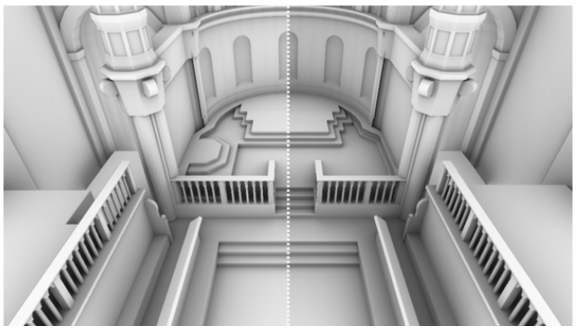


Figure 3: *Right: Full obscurance. Left: Obscurance from the largest falloff weighted occluder only.*

Approximating obscurance from a single occluder typically yields underestimated obscurance because geometry below the largest occluder tends to be closer than the largest occluder (higher falloff term) and also because obscurance coming above the largest occluder is simply ignored. However, given the approximated nature of SSAO the results are entirely plausible and can be further compensated by lifting the falloff function or by adjusting brightness/constrast in post-process.

3.1. Our Method

Instead of calculating AO for each receiver pixel independently (done predominantly in prior work) we traverse through the depth field along lines that cover the framebuffer evenly as shown in Figure 4. In a threaded implementation

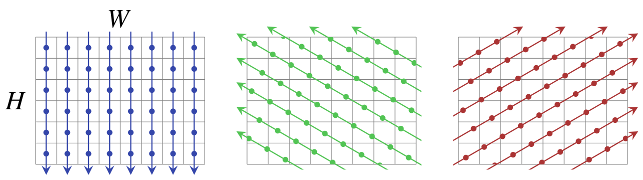


Figure 4: *Sweeps in $K = 3$ directions over a 8×6 framebuffer. A total of $K\cdot W\cdot H$ samples/receivers are considered.*

each thread processes one line, and the line is traversed one constant length step at a time. At each step the depth field is sampled and the sampled point is deprojected into eye-space. This point is then stored onto a stack using an algorithm (described in detail in Section 4) that has an amortized constant cost per step. Processing a line of N steps therefore has the complexity of $O(N)$. At any given point the largest occluder is always found at the top of the stack.

Our method is compatible with obscurance estimators that evaluate obscurance from a set of azimuthal directions and as input take the horizon angle and the distance to the occluder. In this paper we have chosen to use HBAO's obscurance estimator:

$$AO(\mathbf{p},\vec{n}) \approx \frac{1}{K}\sum_{k=0}^{K-1}\Big(sin(t)+(sin(h)-sin(t))\,\rho(||\vec{h}_k||)\Big), \tag{1}$$

$$\vec{D}_k = (sin(\alpha), cos(\alpha)), \alpha = k\cdot 2\pi/K,$$

$$\vec{u}_k = -(p_{xy}\cdot\vec{D}_k, p_z), \vec{t}_k = (-\vec{n}_z, \vec{n}_{xy}\cdot\vec{D}_k), \vec{h}_k = (\vec{o}_{xy}\cdot\vec{D}_k, \vec{o}_z),$$

$$sin(t) = \hat{t}_k\cdot\hat{u}_k, sin(h) = \hat{h}_k\cdot\hat{u}_k$$

for receiver point $\mathbf{p}$ with normal $\vec{n}$. The eye is located at the origin and $\vec{u}_k$ is the zenith vector towards the eye. All vectors denoted by subscript k are projected onto the 2D plane along the azimuthal direction defined by the vector $\vec{D}_k$. Vector $\vec{o}$ points from $\mathbf{p}$ towards the largest occluder along the azimuthal direction. The terms are illustrated in Figure 5. It should be noted that the first $sin(t)$ in the sum in Eqn. 1 gets canceled by opposing directions when both directions use the same tangent plane. As we discuss strategies where this is not the case, in Section 4.1, we include the term in our obscurance estimator to ensure that AO does not evaluate to a value larger than 1.

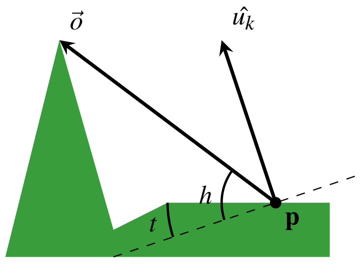


Figure 5: *Illustration of the terms used by our obscurance estimator in Eqn. 1. Vector $\vec{o}$ is the vector from the receiver* $\mathbf{p}$ *to the largest occluder, $\hat{u}_k$ is the unit zenith vector towards the eye, and t and h are the tangent plane and horizon angles, respectively, from the zenith normal.*

Our method can use any monotonically decreasing falloff function ρ and for this paper we have selected an inverse quadratic function similar to the function reported aesthetically agreeable in [FM08]:

$$\rho(d) = \frac{r}{r+d^2} \tag{2}$$

where r is used to control the decay rate.

The obscurance results written along the processed lines are gathered per pixel in a separate phase, described in Section 5. In Section 5 we also cover how lines are positioned in the framebuffer and how sampling coordinates should be chosen. Rendered images and execution times are then presented in Section 6 and compared against the most relevant previous work.

4. Line Sweeping

In this section we describe the process of sweeping through one line in the framebuffer. The output of this process are obscurance values for the points along the line written to an intermediate buffer.

4.1. Obscurance Hull

We first recapitulate the main idea behind *incremental convex hulls* as introduced in [TW10] as our data structure is motivated by it. In [TW10] height field points are iterated along a line and incrementally inserted onto a stack that holds the convex subset of the visited points. In order to keep the stack convex before pushing a new point in, elements are popped from the stack (shown in red in Figure 6) until the last 2 points on the stack and the point to be added form a convex set. The three points form a convex set when the vector from the new point to the last point on the stack has a lower slope than the vector from the new point to the point second to last in the stack. Because the stack can be assumed to have been convex in the previous iteration, due to induction it will remain convex after pushing the new point in. In the convex hull the global horizon for the new point is cast by the point next to it, which is now second to last in the hull (shown in blue in Figure 6).

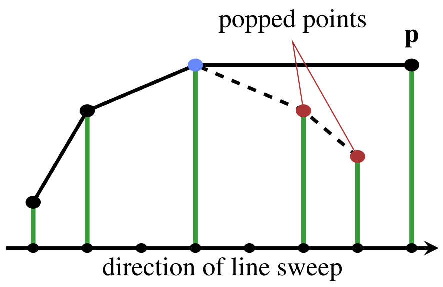


Figure 6: *A convex hull before (dashed line) and after (solid line) adding the new point* **p**. *The global horizon for* **p** *is cast by the point next to it in the hull, marked in blue.*

For the purpose of determining SSAO however, the global horizon might be far away from the receiver, and, according to the falloff function, might cast an insignificant obscurance on the receiver. Since we are trying to find the point between the global horizon and the receiver that casts the largest *falloff weighted* occlusion we have to use an additional criterion to geometrical convexity for the hull. Instead of popping the stack until the point to which the slope from the receiver is the lowest is at the top, we pop the stack until the point which casts the largest obscurance onto the receiver is at the top. The main difference to [TW10] is the boolean function (with the three points as parameters) which determines whether to pop elements from the stack: Instead of testing for convexity, we compare the obscurance (Eqn. 1) from the last 2 points in the stack onto the new point. Points are popped from the stack until the last points casts the largest obscurance, or until the global horizon is reached. We refer to a hull formed according to these criteria as an *obscurance hull*. The obscurance hull will not necessarily be convex and points in the beginning of the stack are progressively losing weight due to the falloff function.

However, it is possible that in some extreme cases the obscurance hull does not return the largest occluder for a receiver. In order to provide some intuition on when this can happen, consider ρ to be a step function that puts full weight to occluders within distance r and zero to others. Next, consider that the colored points in Figure 6 are within r from **p** and a new point, **q**, is encountered after **p** along the line. Now, let **q** be at a height where the popped (red) points are visible to **q** and within r, whereas the blue point falls outside r. In this case one of the popped points casts the largest obscurance on **q** but instead **p** is returned by the obscurance hull. Cases like this are rare but can occur in areas of rapid depth changes. We measured the average error in the final AO value caused by these degenerate cases to be small—between 0.02% and 0.3% in scenes presented in this paper.

Some attention must be paid to the fact that the tangent plane of the receiver appears in Eqn. 1. Obscurance is cut by $sin(t)$ which is not globally fixed. Therefore forming an obscurance hull using one tangent plane might not give the correct obscurance onto a receiver that has a different tangent plane. Eqn. 1 will be used to determine which points are to be culled atop the stack, and also for calculating the obscurance on the receiver once the largest occluder is found. There are three main strategies for choosing the tangent plane for these two operations:

1. Points are culled and obscurance is calculated using the receiver's real tangent. This however will cause the obscurance hull to rapidly unfold when the tangent becomes steep (high $sin(t)$), which may result in future largest occluders being culled and therefore in missing occlusion.
2. Points are culled assuming a globally fixed tangent while the obscurance is calculated using the receiver's real tangent. This makes occlusion discontinued because the obscurance hull gives the largest occluder based on different criteria than the actual per-receiver obscurance is calculated with.
3. The tangent plane is fixed for all calculations ignoring the real tangents of the points.

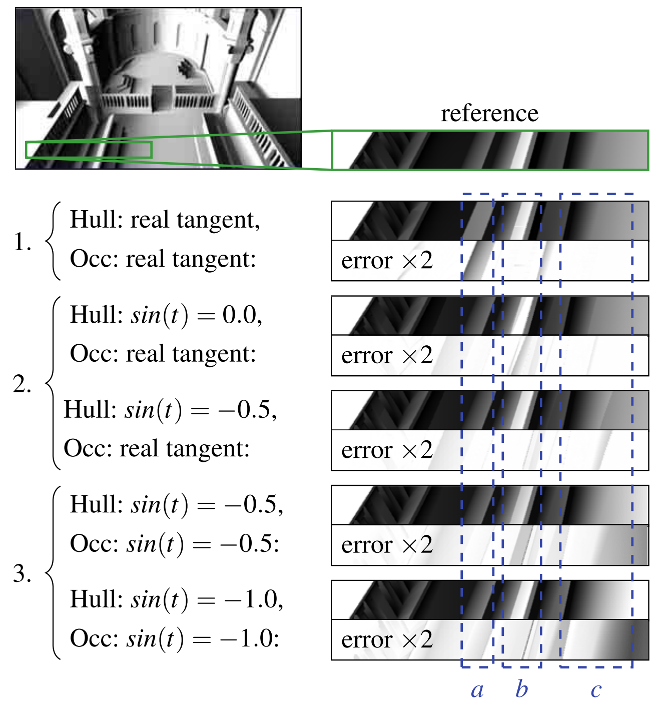


Figure 7: *Various strategies for treating the tangent plane $sin(t)$ when constructing the obscurance hull ("Hull:") and when calculating the obscurance ("Occ:"). The images show the contribution of a single AO sweep directed to the right. A cut from the Sibenik scene is shown for each strategy above the respective error image. Areas of interest a, b, and c are highlighted in blue.*

Figure 7 visualizes ambient obscurance contribution from a single sweep in the Sibenik scene using the three strategies. A cut from the scene is shown from a part where the tangent varies with respect to the sweep direction. The reference image is created by going through all steps along the sweep line for each receiver individually in brute-force and picking the largest occluder.

Strategy 1 may result in occlusion that is flat in regions where, instead, a gradient should appear (area *a* in Figure 7). Along a sweep, strategy 2 may switch from an occluder to the next at a point where occlusions cast by the consecutive occluders do not match. This causes a jump in the occlusion value which is perceptually prominent (area *c* in Figure 7). Strategy 3 both constructs the hull and evaluates occlusion using the same tangent value and therefore produces consistent, however biased, obscurance. The bias comes from the fact that the tangent plane splits the occlusion integral in two parts whose falloff terms differ: The first part is below the tangent plane and has $\rho = 1$ and the second is above the tangent plane and has $\rho \leq 1$ that is dependent on the distance to the occluder. If $sin(t)$ is chosen large, part of the integral that should be evaluated using the occluder's distance (i.e. is above the real tangent) may instead be considered to be below the fixed tangent plane and evaluated using $\rho = 1$ causing overestimated occlusion. This is visible in area *b* when $sin(t) = -0.5$. If $sin(t)$ is chosen small, the opposite may happen: Integral below the real tangent plane may get weighted using the distance to the occluder, causing underestimated occlusion, which is most visible in area *c* when $sin(t) = -1.0$.

While fixing the tangent (strategy 3) does not result in geometrically correct calculations, or even the smallest absolute out of the three options, we consider its error most suitable to the approximated nature of SSAO. Also, the aesthetically chosen falloff term can be used to compensate for underocclusion. In the following sections we have chosen to fix all tangents to $sin(t) = -0.85$ which produces mainly underocclusion.

4.2. Algorithm

Algorithm 1 lists the pseudocode for processing one line in the framebuffer. For a line of M steps there are exactly M pushes to the stack and therefore at most M pops. In addition, there is one iteration per step that terminates the loop at lines 12 – 18 without causing a pop, which results in finding the point that casts the highest obscurance. Therefore the inner loop will perform between M and $2M$ iterations in total and yields the time complexity of $O(M)$ for the algorithm.

5. Gathering Line Sweep Results

In this section we cover how lines are spread out in the framebuffer, how sampling coordinates are selected along the lines, and finally how the obscurance results from processed lines are gathered per final rendered pixel.

5.1. Line Placement

In order to densely evaluate obscurance for each of the K directions lines can be placed 1 pixel width apart and steps along each line can be chosen to be 1 pixel width long. As a result obscurance is evaluated at $W \cdot H$ pixels for each K directions (previously illustrated in Figure 4). Since the same set of azimuthal directions is shared by all pixels, banding may become visible unless a large K is used. Increasing K eventually hides banding, but render times also increase linearly in K. In order to speed up rendering when a large K is used it is often desirable to evaluate obscurance more sparsely.

Instead of placing lines 1 pixel width apart, we can spread them D_L pixel widths apart where D_L is a given fixed value. Also, instead of taking 1 pixel width steps along each line, we can take D_S pixel width steps. Having D_L and D_S larger than 1 effectively causes obscurance in one azimuthal direction to be evaluated sparsely. In order to gather the sparse results for each receiver pixel in the frame buffer, we select

Algorithm 1 SweepLine(float2 pos, float2 dir, int steps)

Functions peek1() and peek2() return the last and the second to last element of the stack, respectively.

```
while (steps−−)
{
    float3 p = deProj(sampleDepth(pos))
    float2 $p_k$ = float2(p.xy · dir, p.z)

    // Unit vector towards the camera
    float2 $u_k$ = $-p_k/||p_k||$

    float2 h1 = hull.peek1() − $p_k$
    float2 h2 = hull.peek2() − $p_k$

    while (occlusion(h1, $u_k$) < occlusion(h2, $u_k$) &&
            h1·$u_k$/||h1|| < h2·$u_k$/||h2||)
    {
        hull.pop()
        h1 = h2
        h2 = hull.peek2() − $p_k$
    }

    writeResult(occlusion(h1, $u_k$))
    hull.push($p_k$)
    pos += dir
}

float occlusion(float2 h, float2 u)
{
    // sin(t) = −0.85
    return sin(t) + max(0, h·u/||h||− sin(t)) · ρ(||h||)
}
```

the nearest point in each K direction at which obscurance was evaluated and average the results. Figure 8 illustrates sparse sweeps using $D_L = D_S = 2.0$. Each sweep therefore subsamples the depth field in a rotated grid pattern.

When the subsampled results from K sweeps are gathered, a different set of points will be selected for each screen pixel but the full azimuth of K directions is included and therefore no noise is produced to the image. Instead blur is produced, because the values that are averaged are sampled from the neighborhood of the receiver pixel and not exactly at it. To limit blurring, the average can be taken only from points that have the normal, the depth (used in this paper), or both within a threshold of the receiver, which is similar to edge-awareness used by blur filters in previous methods. Previous methods usually apply a large blur kernel to hide banding or noise in post-process, whereas we hide banding by increasing K and counter increase in execution time, in exchange for blur, by gathering sparsely evaluated obscurance values. The K nearest obscurance values for each screen pixel are found within the radius of $\sqrt{(D_S/2)^2 + (D_L/2)^2}$ which for

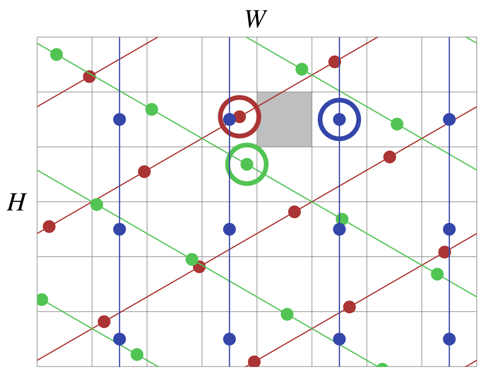


Figure 8: *Sparse ($D_S = D_L = 2$) sampling in 3 azimuthal directions ($K = 3$). The K highlighted points are selected for the screen pixel marked in gray.*

sparsities used in this paper are small compared to a typical post-process blur filter radius.

5.2. Sampling Coordinates

If the depth field is assumed not to be an infinitely thick volume, depth samples cannot be interpolated across depth discontinuities without causing temporal and spatial artefacts. In [BSD08] this is addressed by snapping sampling coordinates to texel centers, and methods relying on mipmaps (such as [MML12]) do not actually average values but instead use the values found in the base level of the depth buffer.

However, sampling coordinates along arbitrary lines in our method do not usually hit texel centers. Always snapping to texel centers, on the other hand, produces artefacts because the pattern of visited samples is not the same for every receiver along the line as shown in Figure 9. This is especially prominent on steep surfaces where a small deviation from the center of the sweep line causes large jumps in the sampled depth values. One possibility is to use linear interpolation when traversing along a line until an edge is detected (depth or normal changes too much from the previous point), in which case the sampling coordinate is snapped to texel center and sampled again. While this eliminates the sampling artefacts, it incurs some overhead and impacts performance.

However, it is possible to choose the K directions and D_S such that the snapped sampling coordinates of previously visited samples at any receiver form the same pattern. The 4 trivial cases are the axis aligned directions, for which any D_S can be used. For larger values of K, directions can be chosen using a grid of $(2n+1) \times (2n+1), n \in \mathbb{Z}^+$ texels as a template as shown in Figure 10. This gives $K = 8n$ aligned directions. For each direction we choose the step length D_S and the direction such that they match the vector from the center of the grid to each of the outer edge texel centers.

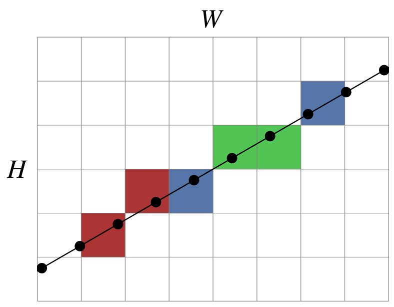


Figure 9: *The colored texels will be sampled when sampling coordinates are snapped to texel centers. For two different receivers along the line (in blue), the sampling patterns (in red and green) relative to the receiver differ (1 left/0 down and 2 left/1 down vs. 1 left/1 down and 2 left/1 down) and show up as noise in the obscurance on slanted surfaces.*

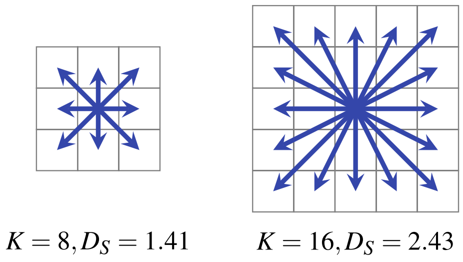


Figure 10: *Aligned sampling patterns and their average step length D_S. The axis aligned directions use D_S that is the average of the rest of the directions.*

While this allows safe snapping of sampling coordinates to texel centers, the average D_S increases along with K. Fortunately this is not a problem, since increasing K is often offset by making the sampling sparser (larger D_S and D_L) such that banding is traded for blur without increasing the execution time. The average number of calculated obscurance values per pixel is $K/(D_S \cdot D_L)$.

Choosing sampling directions according to the box pattern causes directions near the diagonals to be sampled more densely. In order to avoid bias resulting from this, contribution along directions should be weighted according to the azimuthal coverage of each direction, such that smaller weights are given to the diagonal directions. This will result in slightly higher resolution sampling near the diagonals instead of bias.

Figure 11 shows a scene with a slowly decaying falloff (to accentuate banding) and different values of K, D_S, and D_L such that the number of obscurance values per pixel remains constant. Obscurance values, here, are gathered per-pixel by respecting depth edges but ignoring normals. Respecting normals as well can be used to further contain blur.

Our implementation uses an intermediate floating point buffer the size of $K \cdot W \cdot H/(D_S \cdot D_L)$ elements to store the obscurance values of the line sweeps. For $D_S = 4.87, D_L = 2.46, K = 16, W = 1920, H = 1080$ (Figure 13) and $D_S = 2.83, D_L = 2.12, K = 8, W = 1920, H = 1080$ (Figure 1) this is approximately 11 MB. Under tight memory constraints it is also possible to scatter the results to the framebuffer directly using atomic additions during line sweeps, but as this produces highly uncoalesced writes on a GPU we found its performance notably worse than using the intermediate buffers.

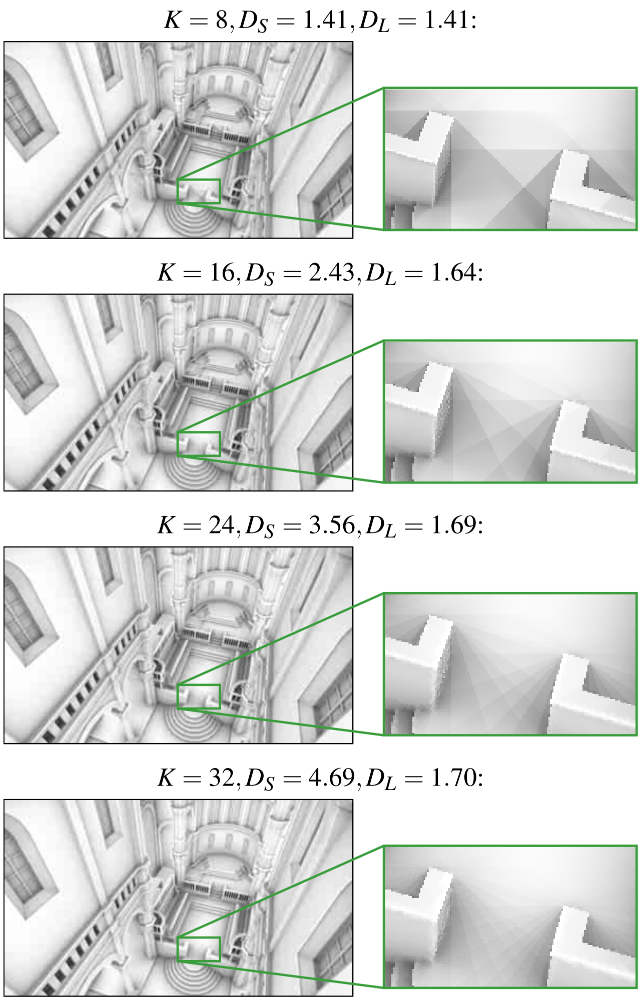

$K = 8, D_S = 1.41, D_L = 1.41$:
$K = 16, D_S = 2.43, D_L = 1.64$:
$K = 24, D_S = 3.56, D_L = 1.69$:
$K = 32, D_S = 4.69, D_L = 1.70$:

Figure 11: *Different number of azimuthal directions K with sampling sparsities such that the density of obscurance values per pixel remains constant ($K/(D_S \cdot D_L) = 4$).*

6. Results

Since traversing lines that vary in their length and thus write a different number of output values does not map well to fragment shaders, our algorithm requires either compute shaders or GPGPU. We have selected to use CUDA and made the sources available under the BSD license at http://wili.cc/research/lsao/. The benchmarks are performed on an NVidia GeForce GTX 480 GPU. The HBAO method used as the reference is implemented as an OpenGL fragment shader. For quality and performance comparison between HBAO and other recent SSAO methods, refer to [VPG13] and [McG10].

HBAO requires a falloff function that decays to 0, and we have chosen the following falloff function for HBAO:

$$\rho_0(d) = max\left(0, \frac{r(1+C)}{r+d^2} - C\right) \tag{3}$$

This function has roughly the same shape as Eqn. 2 for small C. Compared to Eqn. 2 it is sunken by C, clipped to 0, scaled to start from 1, and reaches zero at $d = \sqrt{r/C}$. In this section we use $C = 0.3$.

Figure 12 shows two scenes rendered at 1280(+256)×720(+144) (20% guard band) using two different rates of decay r for the falloff function. Our method uses configurations for $K = 8$ and $K = 16$ shown in Figure 11, whereas the number of HBAO steps N have been hand-picked for each scene and are scaled per-pixel to cover the eye-space falloff radius. The execution time of HBAO is different for the two falloff decay rates because a different number of steps has to be taken to cover the bulk of the falloff function with the same granularity. The execution time of our method depends mainly on the variance in the number of iterations of the inner loop in Algorithm 1 within warps of threads, and does not vary significantly. In all scenes our method performs roughly $2K$ iterations per pixel on average ($\approx K$ during line sweeps and K for gathering the results) whereas HBAO has to perform an order of magnitude more, $K \cdot N$.

Table 1 shows scaling with respect to screen resolution in the Sponza scene at $K = 16$ (bottom left in Figure 12). HBAO has to use larger N at higher resolutions to cover the eye-space falloff at the same screen-space accuracy, whereas our method has a constant per-pixel cost and scales linearly in the resolution. The execution time of HBAO in fact increases slightly faster than cubicly in the number of screen pixels because of increased texture cache misses, whereas the slower than quadratic scaling in the execution time of our method at lower screen resolutions is due to the small number of threads (i.e. lines to sweep) which impacts hardware utilization. Our method takes very few texture samples and is not much impacted by a texture cache miss penalty.

Table 1: *Total render times of our method and HBAO at different resolutions using 20% guard band. The scene is shown in Figure 12 to the bottom left.*

Screen resolution	Our method	HBAO
800×600	1.49 ms	10.5 ms
1280×720	2.56 ms	24.2 ms
1920×1080	5.24 ms	92.5 ms
2560×1600	9.58 ms	249 ms

Our method consists of two stages: The line-sweep stage and the result gathering stage. Table 2 shows execution time breakdown for these two stages when K is increased but $K/(D_S \cdot D_L)$ is kept constant. Even though the amount of intermediate sweep data stays the same, more data is read per pixel which shows up as a steady increase in the execution time of the gather stage. Execution time of the line-sweep stage, on the other hand, decreases slightly because the work is split into a larger number of threads that run shorter, which improves hardware utilization.

Table 2: *Render time breakdown per stage for our method at 1280(+256)×720(+144) for cases shown in Figure 11.*

Configuration	Line-sweep	Gather
$K = 8, D_S = 1.41, D_L = 1.41$	1.91 ms	0.38 ms
$K = 16, D_S = 2.43, D_L = 1.64$	1.80 ms	0.59 ms
$K = 24, D_S = 3.56, D_L = 1.69$	1.67 ms	0.73 ms
$K = 32, D_S = 4.69, D_L = 1.70$	1.60 ms	0.90 ms

Finally, two more scenes rendered at 1080p resolution are shown in Figures 1 and 13. Both scenes are rendered at 1.33 obscurance evaluations per pixel in roughly 2 ms; Figure 1 with $K = 8, D_S = 2.83, D_L = 2.12$ and Figure 13 with $K = 16, D_S = 4.87, D_L = 2.46$. Since our method uses a globally fixed set of K azimuthal directions, a small K can result in severe banding. Our primary way of fighting banding is by increasing K, and our primary way of fighting high render times due to a high K is by controlling the sparsity parameters D_L and D_S. In summary, the configuration of our algorithm is a balancing act of performance, banding, and blur: A small K and high D_L and D_S improve render times, whereas a high K reduces banding and small D_L and D_S reduce blurring.

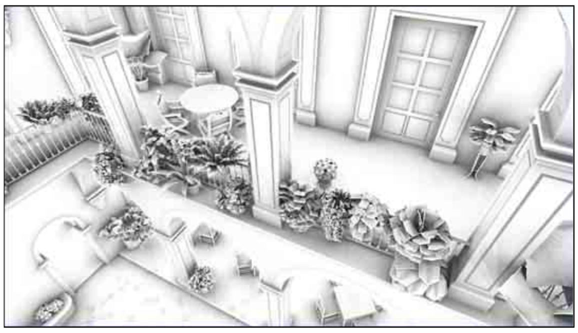


Figure 13: *SSAO rendered by our method at 1920(+192)×1080(+108) in 2.3 ms on a GeForce GTX 480.*

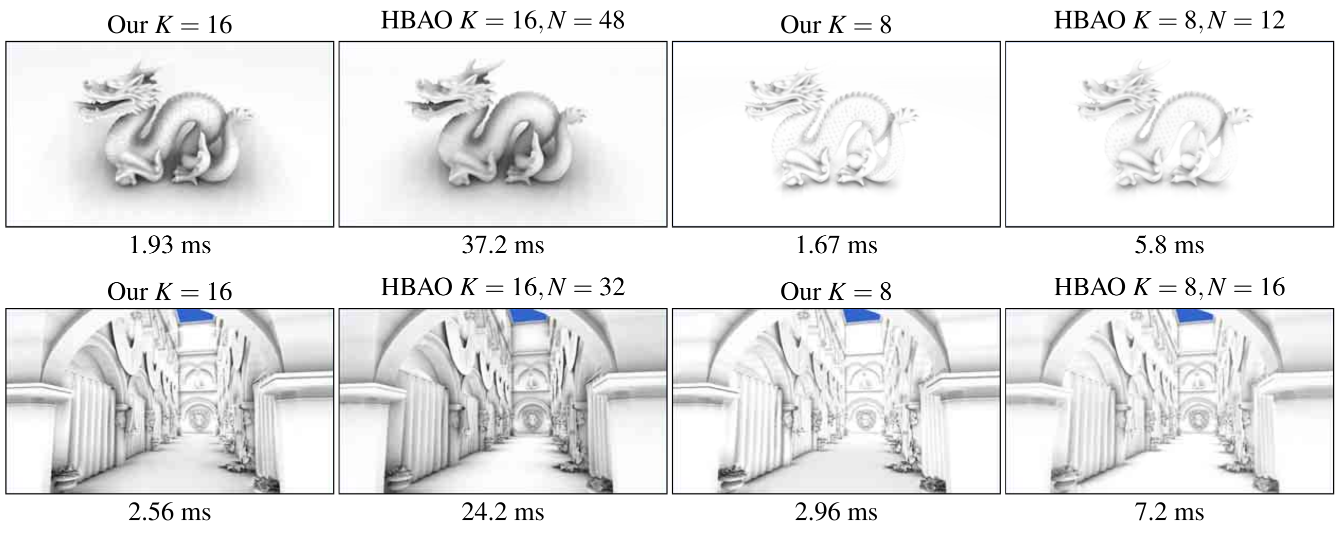


Figure 12: *Scenes rendered at 1280(+256)×720(+144) using different falloff decay rates by our method and HBAO. For HBAO, the number of steps along each of the K azimuthal directions is denoted by N.*

7. Conclusion

Our method is the first attempt to reduce the underlying time complexity of SSAO since its introduction in 2007. Previous methods rely exclusively on strategies that, in order to determine visibility of (and eventually occlusion from) m sampled scene points around n receivers, require $O(mn)$ work. Many of the m points are not visible to the receiver or cast only a small amount of occlusion. In contrast, our method finds the largest falloff weighted occluders along K azimuthal directions for n receivers in $O(Kn)$ time. The falloff radius has no impact on the performance or on the image quality of our method. The largest occluder is found in constant time at per-pixel accuracy regardless of its distance from the receiver, therefore avoiding exhaustive sampling based searches used by previous methods.

Our method uses a globally fixed set of K azimuthal directions and is prone to exhibit banding for small K. In order to avoid having to increase execution time linearly in K to hide banding, it is possible to accept blur instead by evaluating obscurance at a lower than per-pixel density along the depth field and gather, per final screen pixel, the nearest value from each azimuthal direction. Overall our method greatly improves the render times of medium to large range SSAO effects and scales well to high resolutions.

References

[Bav11] BAVOIL L.: Horizon-based ambient occlusion using compute shaders. *NVIDIA Graphics SDK 11 Direct3D* (2011). 2

[BS09] BAVOIL L., SAINZ M.: Multi-layer dual-resolution screen-space ambient occlusion. In *SIGGRAPH '09 Talks* (2009), ACM. 2

[BSD08] BAVOIL L., SAINZ M., DIMITROV R.: Image-space horizon-based ambient occlusion. In *SIGGRAPH '08: ACM SIGGRAPH 2008 talks* (New York, NY, USA, 2008), ACM, pp. 1–1. 2, 6

[FM08] FILION D., MCNAUGHTON R.: Effects & techniques. In *ACM SIGGRAPH 2008 Games* (New York, NY, USA, 2008), SIGGRAPH '08, ACM, pp. 133–164. 3

[LS10] LOOS B. J., SLOAN P.-P.: Volumetric obscurance. In *Proceedings of I3D 2010* (2010), ACM. 2

[Max88] MAX N.: Horizon mapping: shadows for bump-mapped surfaces. *The Visual Computer 4*, 2 (Mar. 1988), 109–117. 2

[McG10] MCGUIRE M.: Ambient occlusion volumes. In *Proceedings of High Performance Graphics 2010* (June 2010). 8

[Mit07] MITTRING M.: Finding next gen: Cryengine 2. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses* (2007), ACM, pp. 97–121. 2

[MML12] MCGUIRE M., MARA M., LUEBKE D.: Scalable ambient obscurance. In *High-Performance Graphics 2012* (June 2012). 2, 6

[MOBH11] MCGUIRE M., OSMAN B., BUKOWSKI M., HENNESSY P.: The alchemy screen-space ambient obscurance algorithm. In *Proc. HPG* (2011), HPG '11, ACM, pp. 25–32. 2

[SA07] SHANMUGAM P., ARIKAN O.: Hardware accelerated ambient occlusion techniques on gpus. In *Proc. I3D '07* (2007), ACM. 2

[TW10] TIMONEN V., WESTERHOLM J.: Scalable Height Field Self-Shadowing. *Computer Graphics Forum (Proceedings of Eurographics 2010) 29*, 2 (May 2010), 723–731. 2, 4

[VPG13] VARDIS K., PAPAIOANNOU G., GAITATZES A.: Multi-view ambient occlusion with importance sampling. In *Proc. i3D* (2013), I3D '13, pp. 111–118. 8

[ZIK98] ZHUKOV S., INOES A., KRONIN G.: An Ambient Light Illumination Model. In *Rendering Techniques '98* (1998), Drettakis G., Max N., (Eds.), Eurographics, Springer-Verlag Wien New York, pp. 45–56. 2

Publication P4

Ville Timonen. Screen-Space Far-Field Ambient Obscurance. In Proceedings of the *High Performance Graphics 2013*, pages 33–43, ACM.

Screen-Space Far-Field Ambient Obscurance

Ville Timonen[*]
Turku Centre for Computer Science
Åbo Akademi University

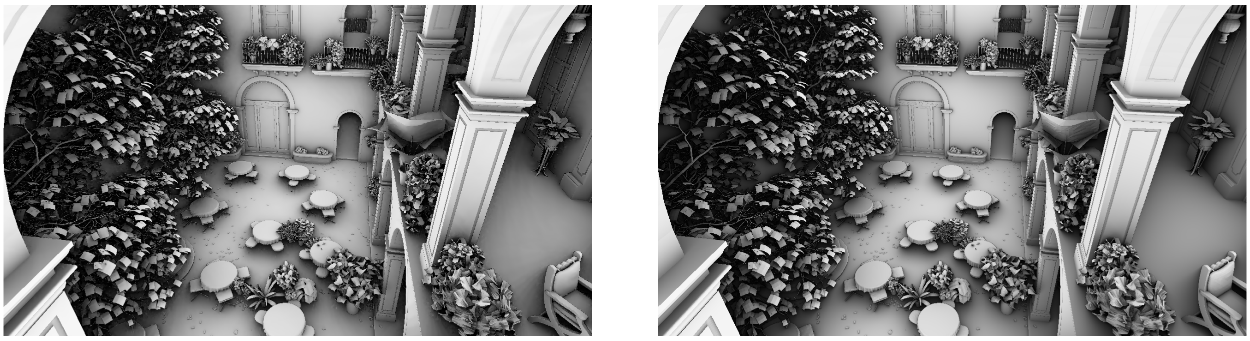


Figure 1: *Left: screen-space far-field ($\geq$ 15 px) occlusion component solved by our method in 4.6 ms on a 1280(+256)$\times$720(+144) depth buffer. Right: ray traced screen-space reference result.*

Abstract

Ambient obscurance (AO) is an effective approximation of global illumination, and its screen-space (SSAO) versions that operate on depth buffers only are widely used in real-time applications. We present an SSAO method that allows the obscurance effect to be determined from the entire depth buffer for each pixel. Our contribution is two-fold: Firstly, we build an obscurance estimator that accurately converges to ray traced reference results on the same screen-space geometry. Secondly, we generate an intermediate representation of the depth field which, when sampled, gives local peaks of the geometry from the point of view of the receiver. Only a small number of such samples are required to capture AO effects without undersampling artefacts that plague previous methods. Our method is unaffected by the radius of the AO effect or by the complexity of the falloff function and produces results within a few percent of a ray traced screen-space reference at constant real-time frame rates.

CR Categories: Computer Graphics [I.3.7]: Computer Graphics—Color, shading, shadowing, and texture

Keywords: ambient occlusion, screen space

1 Introduction

Ambient occlusion approximates global illumination under the assumption that the scene is uniformly lit by blocking part of the incident light due to surrounding occluders. Ambient *obscurance* extends ambient occlusion by introducing a falloff term which attenuates the occlusion effect as a function of occluder distance. The appropriate choice for a falloff term depends on various factors and therefore an ambient obscurance algorithm should be able to support any such distance dependent falloff function.

Screen-space ambient occlusion and obscurance (SSAO) methods have recently become very popular in real-time applications because they only require the depth buffer as input and are therefore easily applied as a post-process or plugged into a deferred renderer, work on fully dynamic scenes, and are insensitive to scene complexity. The falloff term is defined in eye-space distances, which means that the obscurance radius in screen-space depends on the camera's distance to the geometry and may get arbitrarily large. If this was not the case, objects would change appearance as they get closer to the camera. Indeed, an AO effect that conveys the proper global illumination feel often extends a significant radius in screen-space. The best any SSAO method can do given the information available in the depth buffer is represented by a ray tracer: From each receiver pixel rays are traced over the hemisphere to their nearest intersection with the depth field, and then falloff and cosine weighted.

A majority of real-time SSAO methods rely on taking point samples of the depth buffer in the immediate pixel neighborhood of the receiver. This approach is efficient for gathering ambient occlusion from a small neighborhood around the receiver, making screen-space near-field occlusion largely a solved problem. However, as the sampled environment grows in radius, geometry will be missed and noise is produced. Keeping up with the increased radius requires quadratic work and has not been found feasible, even after accepting a blurrable amount of noise. It is possible, however, to use mipmapped depth data such that lower resolution levels are used when sampling farther from the receiver. This approach is used by most state-of-the-art methods and does not miss geometry, but simply averaging the depth field corrupts occluders as seen from the receiver and causes erroneous results.

In this paper our primary contribution is an intermediate representation of the depth field that can be sampled at various distances from the receiver to get virtual scene points that reconstruct features important for AO. The intermediate representation is generated in a

[*]e-mail: vtimonen@abo.fi

pre-pass which scans through the depth field, runs in time that is linear in the depth field size, and generally takes a small fraction of the total time. Our secondary contribution is an obscurance estimator that is fast to evaluate and converges accurately to the AO integral [Zhukov et al. 1998]. When evaluated with scene samples from our intermediate representation, the estimator reaches results within a few percent of the ray traced screen-space reference at real-time frame rates.

Our algorithm executes in three passes: First the depth field is preprocessed by scanning along multiple azimuthal directions, next the output is traversed orthogonally to the scanning directions to pre-integrate dominant occluders, and finally obscurance is evaluated per-pixel from the reconstructed occluders. The key features of our SSAO solution are:

- Constant time, unbounded radius (the effect may span the entire screen)
- Does not miss important occluders (no noise or need to filter)
- Supports arbitrary falloff functions (no render time evaluation)

2 Previous work

In this section we cover only previous work most relevant for our method; for a recent review of ambient occlusion and SSAO methods, consult [Ritschel et al. 2012].

The main branch of present SSAO methods follows from the works of [Mittring 2007] and [Shanmugam and Arikan 2007] where point samples around the receiver are taken to approximate the visibility of the hemisphere. In order to avoid overocclusion when samples farther from the receiver are evaluated, it is important to know whether there is intersecting geometry closer to the receiver which would render the sampled point invisible. To this end, it is possible to connect the samples along one azimuthal direction to get one horizon value instead, as done in [Bavoil et al. 2008]. AO is calculated based on the global horizon angle and rays below the horizon are assumed to be occluded. However in ambient *obscurance*, when a non-constant falloff term is used, occluders' distances below the horizon affect the amount of occlusion and need to be known. Our method tracks the horizon incrementally as geometry is traversed outwards from the receiver, and occlusion coming from geometry visible to the receiver below the global horizon is properly weighted by distance.

Global horizons for a height field are calculated efficiently in [Timonen and Westerholm 2010] for direct lighting of a height field, however for ambient obscurance the same single-horizon problem applies: No information is kept of the geometry below the global horizon, and weighting the occlusion properly according to a falloff function is not possible. Errors can get arbitrarily large because it is not known how far the geometry below the global horizon is from the receiver. [Timonen 2013] fits this method to SSAO by finding the largest falloff attenuated occluder for each direction instead of the global horizon. While this is much more useful for SSAO, geometry above the largest occluder is ignored and the distance to the geometry below the largest occluder is still unknown. While this is suitable for approximate SSAO, results do not converge to a ray traced reference or scale to very high quality like those of our method. Also, both [Timonen and Westerholm 2010] and [Timonen 2013] sample the height field along straight lines whereas we account for the visibility of the geometry over the full azimuth. Considering geometry only along a set of straight lines significantly accentuates banding (cf. Figure 10).

Lower resolution (mipmapped) depth buffers can be used for sampling farther from the receiver as done by [Bavoil and Sainz 2009] [Hoang and Low 2012] [McGuire et al. 2012]. An artefact-free sampling of this multi-resolution representation is not a trivial task, as shown in [Snyder and Nowrouzezahrai 2008]. Regardless of the used low-pass filter, reducing the depth field over an area into a single-valued texel does not capture the view-dependency when the depth field is viewed from an arbitrary receiver. While we also use a resolution hierarchy, we capture information of the enclosed geometry such that it retains the approximated local peaks as viewed from any receiver. Furthermore, levels in our hierarchy are well-aligned (do not overlap or have gaps), making artefact-free sampling straight-forward.

Methods that sample an area around the receiver that is fixed in screen-space may produce fast results [Loos and Sloan 2010], but these methods neither scale to far-field AO nor respect a falloff term. It is also possible to use a forward rendering approach to AO whereby scene geometry prior to rendering is expanded and occlusion is spread onto the area of influence. This approach is pursued in [McGuire 2010], but the method does not scale to far-field effects because of increased overocclusion and high fillrate requirements.

A near-field screen-space search can be coupled with a far-field *world-space* method. A voxelization of the scene is ray traced in [Reinbothe et al. 2009], and [Shanmugam and Arikan 2007] use spherical proxies to approximate scene polygons. World-space methods have significantly different characteristics to screen-space methods: While they have the possibility to include geometry not visible in the depth buffer, they are forced to evaluate the visibility of many geometric primitives per pixel. This is costly and prone to produce overocclusion. Results and performance depend on the scene geometry whereas pure screen-space methods are insensitive to scene complexity.

Finally, purely ray traced AO methods such as [Laine and Karras 2010] produce results similar in quality to ours but do not suffer from the limitations of screen-space information. These methods are, however, at least an order of magnitude slower.

3 Algorithm overview

Our algorithm takes as input the depth and normal buffers, the projection matrix, and a pointer to the falloff function. The depth and normal buffers can change freely between frames, and the depth buffer may include optional guard bands. Geometry within the guard bands are considered as occluders, but obscurance values are not calculated for pixels in the guard band. The output of our algorithm is a floating point map of the ambient light.

We evaluate the 2D ambient obscurance integral in K azimuthal slices. In Section 4 we describe our obscurance estimator that is evaluated for each screen pixel. It takes scene points along each azimuthal direction as input. In order to generate these points, our method first scans the depth buffer in parallel lines in K azimuthal directions and writes out an intermediate representation at regularly spaced intervals along the lines as described in detail in Section 6. This is illustrated for one azimuthal direction to the left in Figure 2. This intermediate data is then optionally (when the depth field can be assumed continuous) traversed perpendicular to the azimuthal scan direction and turned into a prefix sum, as shown to the center in Figure 2. The purpose of the prefix sum is to allow averaging of the intermediate data over each of the K azimuthal sectors which effectively avoids azimuthal undersampling and reduces banding. This phase is described in Section 7. Finally, the prefix sum is sampled per pixel, shown to the right in Figure 2, to construct points (as input to our obscurance estimator) that track local peaks of the depth field. As a reference method we use a mipmapped depth

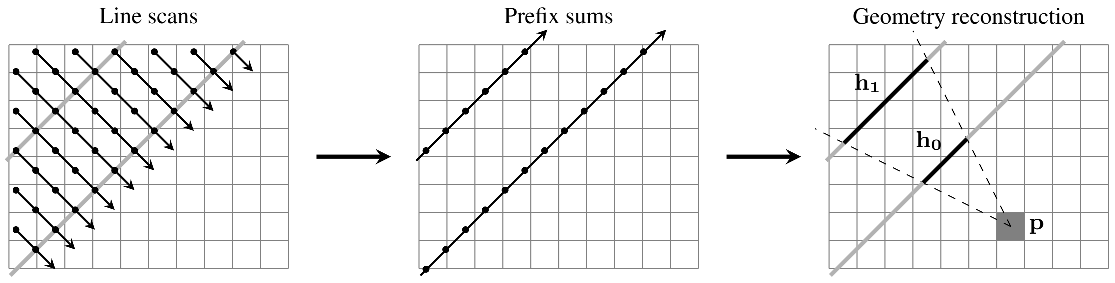


Figure 2: *The three main stages of our method for one azimuthal direction. The first stage scans the depth buffer in parallel lines (arrows to the left) and outputs intermediate geometry at regularly spaced intervals (gray lines). The second stage traverses the output (arrows in the middle) across multiple scan lines and generates prefix sums. The third stage samples the prefix sums per pixel (one pixel highlighted in gray to the right) to construct scene points* $\mathbf{h_i}$ *used as input by our obscurance estimator.*

buffer, described in Section 5, which is used in present state-of-the-art. Results are presented in Section 8.

The remaining perceptually dominant artefact in our method is banding. We propose three mutually complementary strategies to reduce banding:

1. Averaging scene points over sectors in continuous depth fields (Section 7)
2. Sparse evaluation of sectors which trades banding for blur and reduces render times (Section 10)
3. Jittering sampling directions per-pixel which trades banding for noise (Section 11.1)

Our method is most useful for finding far-field occluders and can be coupled with a lighter weight near-field search as discussed in Section 9. The usual limitations of depth buffer geometry apply and are discussed with regards to our method in Section 11.

4 Obscurance estimator

We build our obscurance estimator such that it converges to the original definition of ambient obscurance [Zhukov et al. 1998]:

$$AO(\mathbf{p},\vec{n}) = \frac{1}{\pi}\int_{\Omega} \rho(d(\mathbf{p},\vec{\omega}))\max(0,\vec{n}\cdot\vec{\omega})d\vec{\omega} \tag{1}$$

where $d(\mathbf{p},\vec{\omega})$ is the distance of the nearest occluder from $\mathbf{p}$ in direction $\vec{\omega}$, $\rho \rightarrow [0,1]$ is the falloff term as a function of distance, and Ω denotes the unit sphere in $\mathbb{R}^3$. The falloff function should be smooth and it typically applies that $\rho(0) = 1$ and $\rho(\infty) = 0$, but the exact type and rate of decay is defined by the application. We support any such falloff function and its complexity only affects pre-calculation, not runtime performance.

For geometrically correct SSAO the surrounding geometry for a receiver has to be traversed in azimuthal directions starting from near the receiver and progressing outwards, an approach first taken by [Bavoil et al. 2008]. This way the nearest occluder along a direction from the receiver is guaranteed to be found and contribution from invisible geometry (behind a nearer occluder) can be correctly ignored. At each receiver we consider the vector from the receiver to the camera to represent zenith, and the sphere around the receiver is split into K azimuthal sectors. Therefore the 2D integral of Eqn. 1 is decomposed into K 1D integrals. Each of the 1D integrals is evaluated on a plane that includes the zenith vector and the vector pointing towards the azimuthal angle $k2\pi/K$:

$$AO(\mathbf{p},\vec{n}) \approx \frac{1}{\pi}\sum_{k=0}^{K-1}\left(w_k\int_0^{\pi}\rho(d(\mathbf{p},\vec{\theta}_k))\max(0,\vec{n}_k\cdot\vec{\theta}_k)sin\theta d\theta\right) \tag{2}$$

where $\vec{\theta}_k$ is a unit vector in the k:th azimuthal plane towards the horizon angle θ, and $\vec{n}_k$ is the projection of $\vec{n}$ onto this plane. Geometry will be sampled from the depth buffer along equal size azimuthal sectors in screen-space which map to relative sizes w_k as measured around the zenith for each $\mathbf{p}$. The term $sin\theta$ accounts for the width of the sector as a function of the horizon angle and goes to zero near the zenith.

The integral in Eqn. 2 is evaluated piecewise from depth field points in each sector. The points should be culled to form a series of increasing slopes as measured from the receiver $\mathbf{p}$, i.e. the points should be visible to $\mathbf{p}$. In practice, when the depth field is sampled progressively farther from the receiver, the largest horizon angle from the receiver to the sampled scene point is tracked and new points are known to be visible when their horizon angle exceed the previous maximum horizon angle. As we will only evaluate obscurance from a set of points along a sector that are apart from each other, we do not know the distance of the geometry between the points. For conservative obscurance, we assume that each sampled scene point represents a "slab" of geometry that extends outwards from the camera along the negative zenith direction, as illustrated in Figure 3. The slabs extend the thickness of the depth field; to infinity if the depth field is assumed continuous. While it would

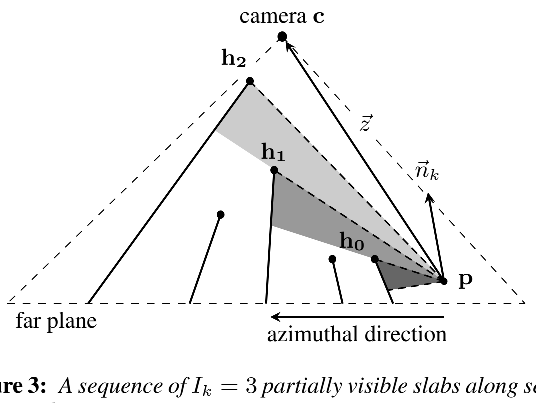


Figure 3: *A sequence of* $I_k = 3$ *partially visible slabs along sector k. Points* $\mathbf{h_i}$ *are the top points of the visible slabs. The strength of the falloff term is noted by shading.*

require a slab for every visible point in the depth field along the azimuthal direction to represent the integral in Eqn. 2 *exactly*, we note that only a few (less than 10) points are required to give values not more than 1% off from the ray traced values if the points are chosen carefully, which we will show in Section 8.

The piecewise evaluation of Eqn. 2 in I_k visible slabs now becomes:

$$\int_0^{\pi} \rho(d(\mathbf{p}, \vec{\theta}_k)) \max(0, \vec{n}_k \cdot \vec{\theta}_k) sin\theta d\theta \approx$$

$$\sum_{i=0}^{I_k - 1} ||\vec{n}_k|| \left(L(a_i, b_i, c_i, d) - L(a_{i-1}, b_i, c_i, d) \right), \qquad (3)$$

$$a_i = \angle(\mathbf{h_i} - \mathbf{p}, \vec{z}),$$

$$b_i = ((\mathbf{p}_y - \mathbf{c}_y)(\mathbf{h_i}_x - \mathbf{c}_x) - (\mathbf{p}_x - \mathbf{c}_x)(\mathbf{h_i}_y - \mathbf{c}_y))/||\mathbf{h_i} - \mathbf{c}||,$$

$$c_i = \angle(\mathbf{h_i} - \mathbf{c}, \vec{z}), d = \angle(\vec{n}_k, \vec{z})$$

where L is a 4D pre-calculated table. The four arguments of L, in order, are

1. The angle a_i of the vector from the receiver to the sampled scene point
2. The closest distance b_i from the receiver to the line formed by the slab, $\mathbf{h_i} + t(\mathbf{h_i} - \mathbf{c})$
3. The angle c_i of the slab
4. The angle d of the projected normal

where all angles are with respect to the zenith vector $\vec{z} = \mathbf{c} - \mathbf{p}$. L is constructed in an offline pre-pass by generating the corresponding slabs and evaluating the falloff weighted integral numerically by ray casting. We have implemented L as a 3D texture where c_i and d share an axis. The sharing is implemented by first splitting the axis into segments, one for each discretized value of d, and then placing consecutive values of c_i consecutively within each segment. Therefore a_i, b_i, and c_i can be linearly interpolated and d is chosen as the nearest discretized value. Although L is of high dimensionality, the involved functions are very smooth and therefore low resolutions are sufficient which helps to keep the texture size moderate (within a few MB).

5 Reference method: mipmap

Our obscurance estimator needs as input the scene points along azimuthal directions in the depth buffer for each receiver. In the simplest case these points can be direct depth buffer samples deprojected into eye-space. While this is the most widely taken approach in current SSAO methods, it does not scale well to far-field: Dense sampling translates into high render times and sparse sampling causes undersampling artefacts because important geometry might be missed.

The approach taken by previous state-of-the-art SSAO methods is to generate a depth pyramid (mipmap) that has a series of lower resolution levels of the depth buffer. Regardless of the filter used to generate the lower resolution levels from the base level, this approach does not retain the view-dependent information of the depth field necessary for accurate AO. We tried several filters including the ones covered in [McGuire et al. 2012] and considered averaging to produce the best results as it does not introduce sudden changes to obscurance like, for example, max-mipmaps do. However, when the depth field cannot be assumed continuous, only points in the original depth buffer can be used. In this case we found max-mipmaps to perform best and we use them in Section 11. Until Section 11 we assume a continuous depth field and compare our geometry representation against averaged mipmaps.

In the mipmap method, from each receiver point, we start traversing the surrounding depth buffer along each of the K azimuthal directions by first sampling the base level texture at a distance of one texel. After each sample, the step size is multiplied by a constant and the sampling distance is accumulated by the step size. This yields an exponentially sparser sampling. We found the constant of 1.5 to produce the best performance-quality tradeoff when used with $K = 16$ and these parameters are used in Section 8.

We use trilinear filtering available in hardware, and choose the mipmap level in such a way that the sample's coverage of the depth buffer matches the sector's width at any given sampling distance. In order to avoid sudden changes in obscurance when the last sampling position goes outside the depth buffer, an extra sample is always taken at the very edge of the depth buffer.

6 Intermediate geometry along scanlines

In this section we describe how our method scans the depth buffer in order to create an intermediate representation of the depth field. We scan the depth buffer in a dense set of parallel lines for each K azimuthal direction and incrementally track the depth field profile along those lines. The parallel lines are spaced one texel width apart along the depth buffer and the lines are traversed one texel width step at a time. In a threaded implementation one thread processes one line. A scan along one azimuthal direction in a depth buffer is shown to the left in Figure 4.

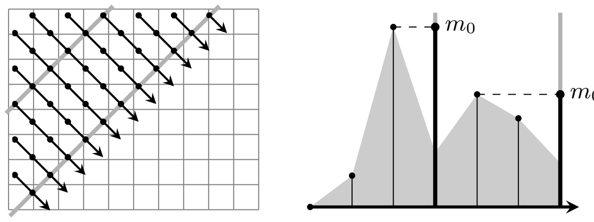


Figure 4: *The depth buffer (grid shown in the background) is scanned in one of the K azimuthal directions in parallel lines (arrows). The maximum height m_0 of each line every $B_0 = 3$ steps (thick gray lines) is written to a buffer holding the intermediate data. Progression along one line is shown to the right.*

We base our method on the intuition that points important for AO are local peaks in the depth field. To track these local peaks, we find the highest (nearest to the camera) depth field points along the lines. In practice, we step through each line incrementally and at each point we sample the depth buffer and deproject the scene point into eye-space. The maximum height value is then remembered along the line until B_0 steps have been taken. After B_0 steps the maximum height is written into an intermediate geometry buffer and reset. This is illustrated to the right in Figure 4. The process is repeated until the end of the depth buffer.

However, which local peak has the highest contribution to AO is dependent on the angle at which the receiver views the peak. The maximum height value is guaranteed to represent the highest horizon value for a receiver that is at the same height, i.e. when the peak is viewed directly horizontally. However, receivers (points along the line) may reside at various heights and therefore view the depth field peaks from different angles. Instead of storing the max height value as viewed directly horizontally, we store 2 max height values: one as viewed horizontally (m_o) and one as viewed at a downwards angle (m_1). We call the angles along which the max height values are viewed *receiver angles*. This is illustrated to the left in Figure 5.

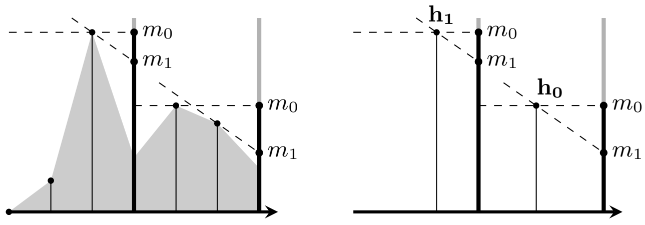


Figure 5: *The maximum heights (denoted by m_0 and m_1) as viewed along 2 receiver angles are written to the intermediate buffer every $B_0 = 3$ steps. The virtual points ($\mathbf{h_i}$) as geometry used by the obscurance estimator are reconstructed at the intersection of the corresponding receiver angles positioned at m_0 and m_1, as shown to the right.*

During the evaluation of the obscurance estimator the intermediate geometry buffer is read and a virtual point is reconstructed at the intersection of the receiver angles as shown to the right in Figure 5. Intuitively this virtual point is a view-dependent (for a likely receiver) approximation of the highest peak within the interval of B_0 steps along the scanning line. After culling the invisible points per receiver, these points can be directly used as h_i by the obscurance estimator in Eqn. 3.

We have chosen to use slopes 0 (horizontal) and -1 (45 degrees downward) for the receiver angles. We found that the choice for the receiver angles does not make a large difference to results, however it is important that there are two different angles such that the reconstructed virtual point will have both a depth and a distance value. Algorithm 1 lists the pseudocode for scanning one line in the depth buffer.

Algorithm 2 lists the pseudocode for evaluating obscurance at one screen pixel along one azimuthal direction. Since SSAO is separable in azimuthal directions, Algorithms 1 and 2 can be calculated sequentially for each K, in which case our method requires $O(W_0 \cdot H_0/B_0)$ space for the intermediate geometry buffer, where W_0 and H_0 are the depth buffer dimensions including guard bands. For a typical case of $W_0 = 1280+256, H_0 = 720+144, B_0 = 10$ this is roughly 1 MB. If the application is not memory constrained it is faster to evaluate Algorithm 1 for all K simultaneously as to maximize the number of concurrent threads and therefore improve utilization of a GPU. For $K = 16$ the respective memory requirement becomes 16.2 MB.

The accuracy of our intermediate geometry becomes progressively better compared to mipmaps when the interval size increases. Due to the falloff function occluders far from the receiver get less weight and also map to smaller swaths of the horizontal angle than nearby occluders. Therefore it is sensible to construct occluders progressively more sparsely when farther from the receiver. Similarly to building multiple resolutions of the depth field in the form of mipmaps, our intermediate geometry can be made into a 1D pyramid. We form levels of the intermediate geometry such that their intervals increase exponentially from the base level's, B_0. Therefore the interval of level n is $B_n = B_0 \cdot 2^n$. The levels can be efficiently generated by taking the max m_0 and m_1 from the two corresponding lower level intervals. Generating the exponential hierarchy roughly doubles the required space but reduces the per-pixel time complexity from $O(n)$ to $O(log(n))$ where n is the pixel distance from the receiver to the edge of the depth buffer.

Algorithm 1 ScanLine(float2 pos, float2 dir, int steps, int lineNo)

Pos *is the coordinate of the first step in the depth buffer and* dir *is a vector for one step along the scanline.*

```
float m0 = $-\infty$
float m1 = $-\infty$

while (steps$--$)
{
    float3 p = deProj(sampleDepth(pos), pos)
    // p is projected onto k:th azimuthal plane
    float2 pk = (p.xy · dir, p.z)

    m0 = max(m0, pk.y)
    // s = slope of the downwards receiver angle ($-1$)
    m1 = max(m1, pk.y + s·pk.x)

    if (steps modulo B0 == 0)
    {
        // iBuf = the intermediate buffer (output of this stage)
        iBuf[lineNo][steps/B0] = (m0, m1)
        m0 = $-\infty$
        m1 = $-\infty$
    }

    pos += dir
}
```

Algorithm 2 EvalObscurance(float2 pixelPos, float2 dir)

```
int lineNo = find the line with direction dir closest to pixelPos
int iVal = find the nearest interval in lineNo that is at least B0
        steps from pixelPos

float3 p = deProj(sampleDepth(pixelPos), pixelPos)
// zScale scales z onto the slanted azimuthal plane
float zScale = $\sqrt{1 + (\text{p.x/p.z} \cdot \text{dir.y} - \text{p.y/p.z} \cdot \text{dir.x})^2}$
float2 pk = (p.xy · dir, p.z·zScale)

float (AO, maxAngle) =
        EvalNearField(pixelPos, dir, distance to iVal)

while (iVal $\geq$ 0) {
    float (m0,m1) = iBuf[lineNo][iVal]
    // s = slope of the downwards receiver angle
    float2 h = ((m0 $-$ m1)/s, m0·zScale)
    float angle = $\angle((\text{h} - \text{p}_\text{k}), -\vec{\text{p}_\text{k}})$ // c is at the origin
    if (angle > maxAngle)
    {
        // Obs(ai, ai-1, hi, p) evaluates i:th segment from Eqn. 3
        AO += Obs(angle, maxAngle, h, pk)
        maxAngle = angle
    }
    iVal$--$
}

return AO
```

7 Averaging sectors

In Section 6 we described how to construct virtual points for the obscurance estimator defined in Section 4 from geometry along a single line in the depth buffer. We propose this approach when it is not possible to average or interpolate depth field values, which is the case in Section 11 where the depth field is assumed to represent a volume of a finite thickness. In this section we assume that the depth field is continuous and averaging is thereby allowed.

Ideally the virtual points should represent the entire sector instead of the thin texel wide line along the center of the azimuthal sector. The sector's width increases linearly in the distance from the receiver as demonstrated in Figure 6. In Figure 6 lines contributing to the obscurance at $\mathbf{p}$ are shown as arrows. The horizontal intervals are equal to the gray lines perpendicular to the scanning direction previously shown to the left in Figure 4. In order to construct the averaged point for the highlighted middlemost interval shown in Figure 6, we simply average m_0 and m_1 over the parallel lines $l_A ... l_B$ fitting into the sector at that specific distance: $(m_0^a, m_1^a) = 1/(l_B - l_A + 1) \cdot \Sigma_{l_i=l_A}^{l_B} \mathrm{iBuf}[l_i][iVal]$. When the virtual point is constructed (line 14 in Algorithm 2) m_0^a and m_1^a are used instead of m_0 and m_1. In order to calculate the average in constant time, we turn the buffer iBuf into a per-interval prefix sum iBuf^P such that $\mathrm{iBuf}^P[l_i][iVal] = \Sigma_{l=0}^{l_i} \mathrm{iBuf}[l][iVal]$. From the prefix sum the average over any line range $l_0 ... l_1$ for interval $iVal$ can then be efficiently calculated as $(\mathrm{iBuf}^P[l_1][iVal] - \mathrm{iBuf}^P[l_0 - 1][iVal])/(l_1 - l_0 + 1)$.

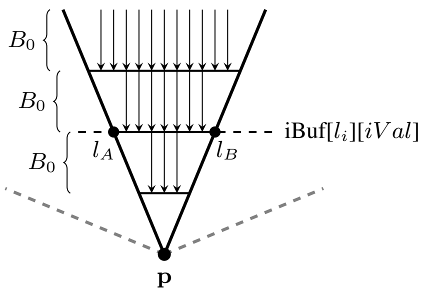


Figure 6: *The downward arrows denote scan lines along one azimuthal scanning direction. Lines indexed $l_i \in [l_A, l_B]$ at the highlighted interval (constant index $iVal$) fit into the sector from receiver $\mathbf{p}$ and their m_0 and m_1 should be averaged.*

We therefore introduce another stage between Algorithm 1 and 2 which traverses the intermediate geometry buffer iBuf *perpendicularly* to the scan direction in Algorithm 1 and accumulates m_0 and m_1 values over lines. This stage produces the prefix summed version iBuf^P which can, in fact, be built in-place over the original iBuf.

Finally, when evaluating obscurance, instead of averaging the intervals across the entire sector width, the obscurance can be evaluated in multiple segments to increase azimuthal resolution. While doing so does not produce results quite as accurate as if the number of sectors K is increased by a corresponding factor, evaluating a sector in multiple segments is computationally lighter than increasing the number of sectors and has the same effect on reducing banding. More importantly, evaluation in multiple segments does not require extra azimuthal scans over the depth buffer. We have chosen to split each sector in half and evaluate obscurance in 2 segments per sector. From now on, we denote this by adding a multiplier to K, e.g. $K = 8 \times 2$ for eight azimuthal directions and two segments.

8 Results

We ran our algorithm on AMD Radeon HD 7970 (OpenCL) and NVIDIA GeForce GTX 580 (CUDA). Sources are available under the BSD license online at `http://wili.cc/research/ffao/`. The mipmap method using our obscurance estimator and Horizon-Based Ambient Occlusion (HBAO) [Bavoil et al. 2008] are implemented as OpenGL fragment shaders. Performance and quality comparison between other recent SSAO methods can be found in [Vardis et al. 2013] and [McGuire 2010].

Our algorithm calculates the far-field SSAO in 3 kernels:

1. The *Scan* kernel scans through the depth buffer in K azimuthal scanning directions in parallel lines and finds local peaks of the depth field at regularly spaced intervals.
2. The *Prefix sum* kernel reads through the values over multiple lines in a direction perpendicular to the scan direction and generates prefix sums.
3. The *Obscurance* kernel reconstructs virtual points from the prefix sums that are averaged over the azimuthal sector width at each screen pixel. The final obscurance value per pixel is calculated by evaluating the obscurance estimator with the virtual points.

We have chosen two scenes as our main test material. The first scene is an architectural scene with simple planar geometry (Figure 7, top), and the second scene shows complex geometry and foliage (Figure 7, bottom). The scenes are rendered using exponentially decreasing falloff functions whereby in the first scene the falloff function decays slower than in the second scene. All renderings use a 10% guard band (extending 10% of the visible framebuffer width or height at each side) which is denoted by postfixing it to the resolution in parentheses.

For our method we use $K = 8 \times 2$ and $K = 16 \times 2$ and for the mipmap method we use $K = 16$. As reference we use ray tracing on the same geometry (a single-layer depth buffer with a 10% guard band). In the ray traced result rays with cosine-weighted directions are cast around the hemisphere for each receiver pixel, and stepped through in small steps until geometry is being intersected. The intersection distance is then weighted by the falloff function and accumulated to the result. This can arguably be considered the best result any method can do with the available screen-space data.

In addition to difference images, we measure the error using two metrics: e_A measures the average per-pixel variation from the ray traced values and $e_{<5\%}$ measures the number of pixels within 5% of the ray traced values. A high value in $e_{<5\%}$ denotes that only few pixels behave abnormally, which also implies temporal stability since the reference values do not wave or flicker. In Figure 7 we have rendered the two scenes using our method and the mipmap method and compared the results against the ray tracing.

Recall that our obscurance estimator conservatively assumes a scene point to represent a slab of geometry which extends along the negative zenith. However, actual depth field geometry between slabs can be closer to the receiver. The error coming from the overestimated distance is relative to the density of the slabs, which we in this section keep constant at roughly 8 slabs per azimuthal direction. The average error introduced by this in the first scene is $e_A \approx 0.6\%$ and in the second scene $e_A \approx 0.8\%$, which sets a lower bound for the error as K is increased. Obscurance as estimated from the geometry produced by our method is very accurate even when a low number of sectors is being used, mainly because the evaluated virtual scene points are tailored to capture the features of the scene geometry that specifically contribute to AO. The

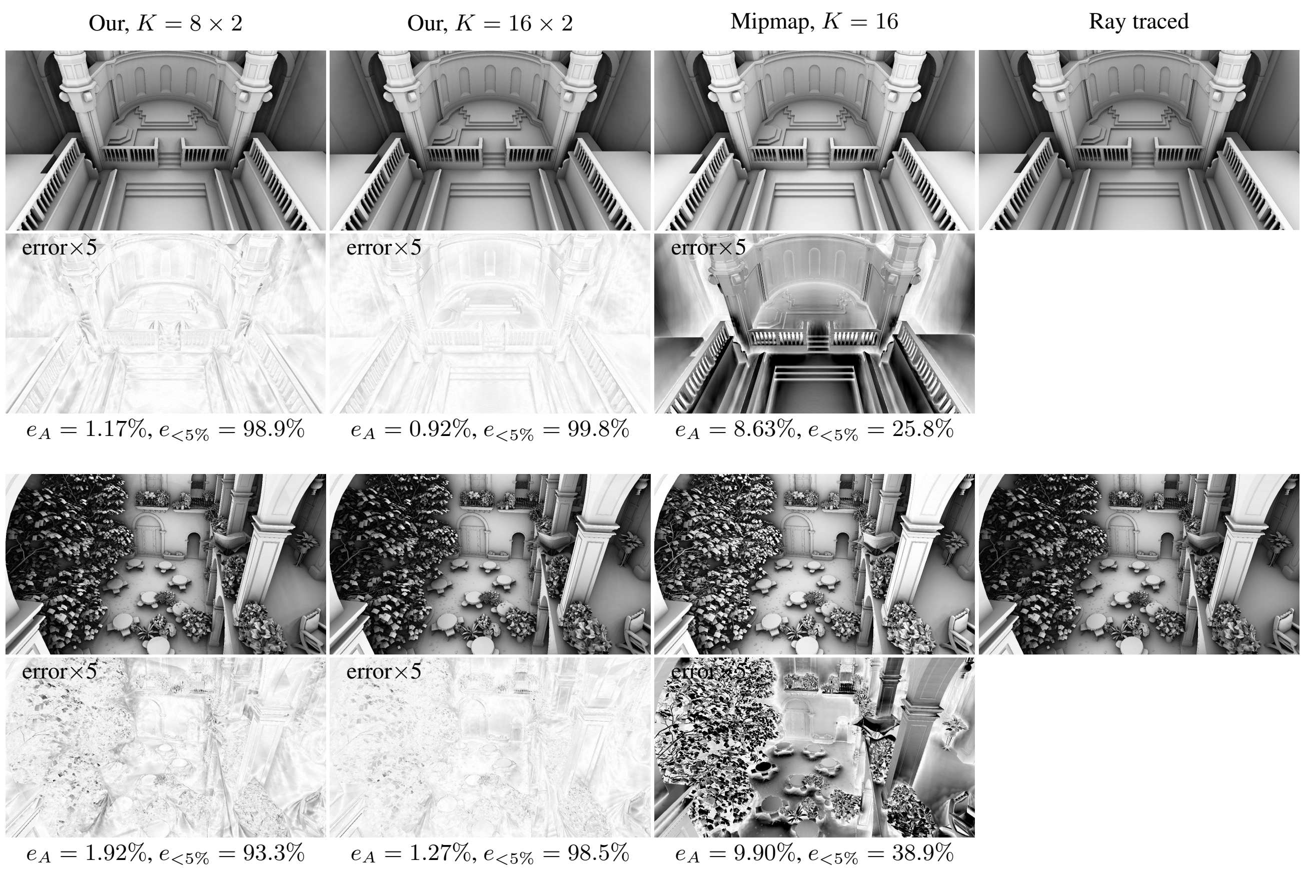

Our, $K = 8 \times 2$
Our, $K = 16 \times 2$
Mipmap, $K = 16$
Ray traced
error×5
error×5
error×5
$e_A = 1.17\%$, $e_{<5\%} = 98.9\%$
$e_A = 0.92\%$, $e_{<5\%} = 99.8\%$
$e_A = 8.63\%$, $e_{<5\%} = 25.8\%$
error×5
error×5
error×5
$e_A = 1.92\%$, $e_{<5\%} = 93.3\%$
$e_A = 1.27\%$, $e_{<5\%} = 98.5\%$
$e_A = 9.90\%$, $e_{<5\%} = 38.9\%$

Figure 7: *Two scenes rendered by our method and the mipmap method and their respective error images (white = 0%, black $\geq$ 20%, brighter is better), average error (e_A, lower is better) and the number of pixels within 5% ($e_{<5\%}$, higher is better) of the ray traced reference.*

mipmap method is inadequate in capturing the "profile" of the geometry within the sample's radius and produces erroneous results. Furthermore, the mipmap method converges to the ray traced values very slowly: It takes over 300 ms to achieve the same level of error as in our method at $K = 8 \times 2$ and several seconds to match $K = 16 \times 2$.

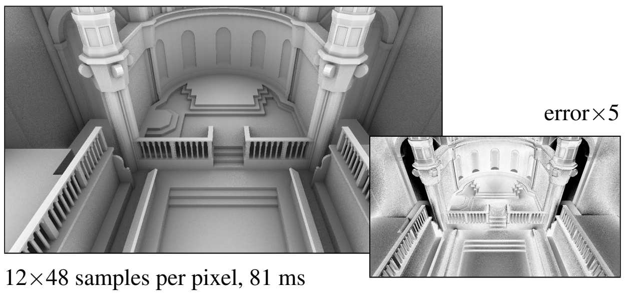
While the quantitative error in our method is small, there can still be banding that is perceptually prominent. The level of banding depends on the geometric content: Bands are cast by sharp tall edges and are visible on planar surfaces. Averaging the virtual points over the widths of each sector reduces banding, especially from occluders far from the receiver where the banding is almost completely removed. Banding is discussed in more detail in Section 10. Temporal coherence can be a major concern in SSAO methods that exhibit undersampling, whereas the dense azimuthal scans employed by our method do not skip geometry. Overall we observe that the results of our method look temporally stable (under motion) which is to be expected given the small variation with respect to the stable ray traced values.

In comparison, Figure 8 shows results as rendered by HBAO using the same falloff function. HBAO does not assume a continuous depht field. Instead, it assumes that geometry between two consecutive visible points along an azimuthal direction is at the same distance from the receiver as the higher of the two visible points. As HBAO's obscurance estimator is not built to converge to Eqn. 1 results look dissimilar to the ray traced reference. HBAO requires very many samples per pixel to cover far-field effects accurately which shows up as impractically high render times.

Let $W_0 \times H_0$ be the resolution of the depth buffer with guard bands, and $W \times H$ without. Then the time complexity of our method for kernel 1 is $O(K \cdot W_0 \cdot H_0)$, for kernel 2 $O(K \cdot W_0 \cdot H_0/B_0)$, and for kernel 3 $O(K \cdot W \cdot H \cdot log(W_0 + H_0))$. We consider the scaling favorable, as the per-pixel cost increases only logarithmically in the resolution while the full depth field is still considered for each pixel.

Our method is insensitive to the geometric content of the depth buffer and the obscurance radius has no effect on the render times; obscurance is gathered from the entire guard banded depth buffer for every pixel. Table 1 lists the total execution time of the second scene in Figure 7 for our method and for the mipmap method using two different GPUs and two common screen resolutions. All timings only include far-field AO, which starts at approximately $1.5B_0$ pixels from the receiver. The same far-field boundary is used for both methods.

Table 2 shows how the execution time is split between the three stages of our method. In the Obscurance kernel, we measure the average number of constructed virtual points to be 7.8 per sector per pixel. The mipmap method takes an average of 8.5 samples per sector per pixel.



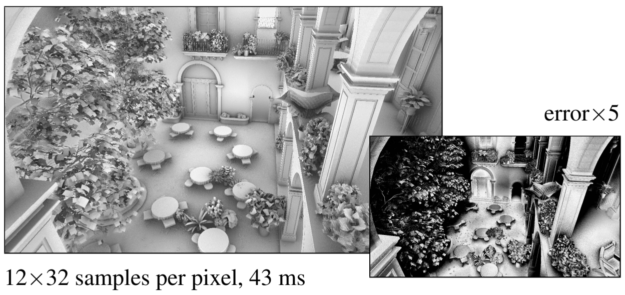


Figure 8: *Scenes from Figure 7 as rendered by HBAO on a GeForce GTX 580 at 1280(+256)×720(+144). Obscurance is calculated in $K = 12$ azimuthal directions that are randomly rotated per pixel.*

Table 1: *Total render times of the far-field AO component.*

Method	Radeon 7970	GTX 580
$1280(+256) \times 720(+144)$, $B_0 = 10$:		
Our, $K = 8 \times 2$	7.26 ms	12.0 ms
Our, $K = 16 \times 2$	13.3 ms	23.6 ms
Mipmap, $K = 16$	19.2 ms	17.7 ms
$1920(+384) \times 1080(+216)$, $B_0 = 10$:		
Our, $K = 8 \times 2$	16.7 ms	29.4 ms
Our, $K = 16 \times 2$	31.6 ms	58.1 ms
Mipmap, $K = 16$	31.5 ms	37.9 ms

9 Integration with near-field

From a time complexity point of view our method is effective in treating near-field obscurance as well, however our method's benefits become significant only when the geometry enclosed by the interval B_i covers a large distance. In order to bring the nearest interval in our method closer to the receiver, B_0 has to be reduced, which increases the execution time and the memory footprint. We suggest that our method be combined with a lightweight near-field search that gathers obscurance from an area around the receiver that is at least a couple of pixels in radius. Where exactly the boundary between the near-field and our method should be depends on the characteristics of the near-field search: Our method should generally take over at a distance where the near-field method is no longer faster. The interval of the base level, B_0, determines the nearest distance at which our method can take over. Halving B_0 causes one extra interval per sector to be evaluated (roughly a constant increase in execution time), and reduces the area of influence of the near-field search to quarter. Table 3 lists our method's execution times for different values of B_0.

When integrating a near-field method with our method, the near-field method should be executed first and provide the maximum horizon angles from the near-field range along the K sectors for each pixel as shown at line 9 in Algorithm 2. After this, our method continues accumulating the obscurance from the horizon angle upwards until the edge of the depth field. We expect $B_0 \in [5, 10]$ to be a suitable choice for a typical state-of-the-art near-field search. Figure 9 shows the contribution of the far-field and the near-field obscurance components on a 720p depth buffer using $B_0 = 10$.

Table 2: *Render time breakdown of our method per kernel.*

Phase	Radeon 7970	GTX 580
$1280(+256) \times 720(+144)$, $K = 8 \times 2$, $B_0 = 10$:		
Scan	0.537 ms	0.489 ms
Prefix sum	0.945 ms	0.617 ms
Obscurance	5.77 ms	10.9 ms

Table 3: *Far-field render times of our method using different values for the base level interval B_0.*

Base level	Radeon 7970	GTX 580
$1280(+256) \times 720(+144)$, $K = 8 \times 2$:		
$B_0 = 20$	6.06 ms	9.77 ms
$B_0 = 10$	7.26 ms	12.0 ms
$B_0 = 5$	8.89 ms	14.7 ms

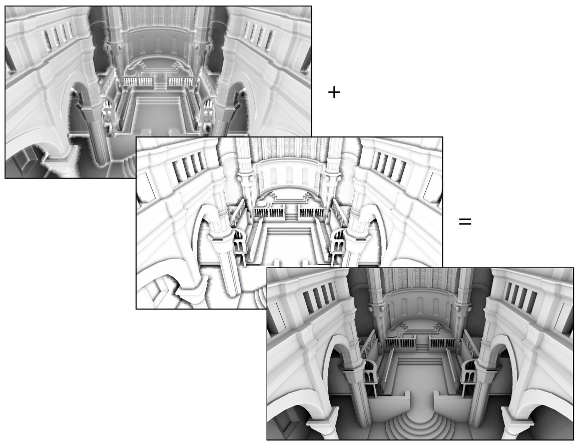


Figure 9: *The far-field component ($\geq$ 15 px) as produced by our method and the near-field component ($<$ 15 px) together form the final ambient obscurance result (bottom).*

10 Banding

While the error in our method is small, as established in Section 8, there still might be some visible banding even though we average occluders across the width of a sector. In order to gain intuition on why banding happens, consider the case where an occluder—say, a wall—enters an otherwise flat sector in a linear motion. If the sector was split into infinitely many subsectors, obscurance would increase linearly as the wall occupied a larger swath of the sector. In our method, the entering wall increases the average *height* of an interval linearly, which might not map to linear change in obscurance. This is especially evident for very tall occluders which cause the obscurance value to increase faster than linearly when only a small portion of the occluder is occupying the sector. So, while the obscurance values at band boundaries in our method do not jump abruptly, they don't follow the physically correct curve either. In Figure 10 we show a split screen of a scene rendered using $K = 8 \times 2$ averaged sectors as described in Section 7 and then using $K = 16$ straight sampling lines that go through the center line of each sector. Averaging eliminates far-field banding almost completely, which is often the hardest to rid, but near-field banding still persists.

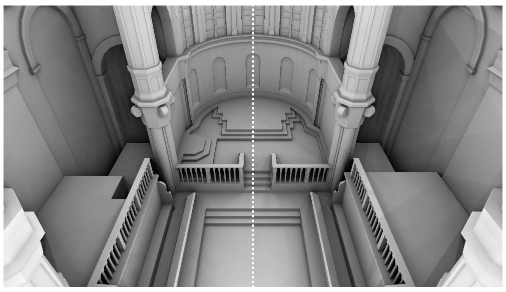


Figure 10: *Left: our method using $K = 8 \times 2$ averaged sectors. Right: our method using $K = 16$ straight sampling lines.*

One solution to the banding problem is to increase the number of scanning directions K which shows up as a roughly linear increase in execution times of the Scan and Prefix sum stages. Instead of evaluating all K sectors for every pixel, it is possible to evaluate the sectors sparsely. We use the Separable Approximation of Ambient Occlusion (SAAO) approach from [Huang et al. 2011], and evaluate $K = 18 \times 2$ sectors in groups of 3×3 pixels. Obscurance is therefore evaluated in an interleaved pattern such that only $K = 2 \times 2$ sectors are evaluated per pixel and the results are gathered using an edge-aware 3×3 box filter as a post-process. *Any* 3×3 pixel neighborhood includes all $K = 18 \times 2$ sectors and therefore no noise is produced to the image. SAAO produces errors primarily at edges and depth discontinuities in the depth buffer. While the far-field AO component can also change by unbounded amounts between adjacent pixels in the screen, the error is mainly in the near-field.

We have incorporated SAAO in our method by combining a 3×3 separated far-field AO with a full near-field AO. The primary artefacts are small integration errors (noise) at the boundary of the far-field and near-field AO components because they are of different sparsity. The error can be hidden to a large extent by a selective blur that uses a small intensity threshold and does not currupt the image. In Figure 11 a result without blurring is shown to the right, which shows minor noise, and the image to the left includes a 5×5 bilateral box blur. Figure 1 is also rendered using the method shown to the left.

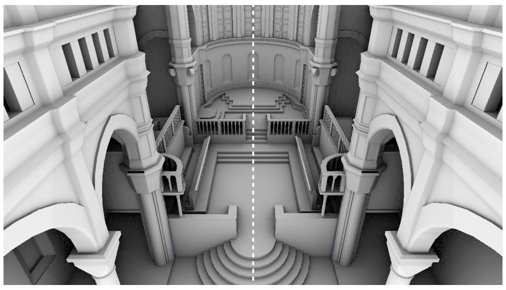


Figure 11: *Our method using $K = 18 \times 2$ with SAAO. Left: selectively blurred result. Right: result without blurring.*

We implement SAAO by introducing two new kernels:

Box average kernel is an edge-aware filter which averages depth and the normal vectors from a 3×3 pixel neighborhood of each framebuffer pixel. If the dot product of the normal of the source pixel and the candidate pixel is higher than 0.5 and the relative difference of pixel depths is below 3%, the pixel is accepted into the average. The accepted pixels are the pixels from which the far-field obscurance is gathered in the Gather kernel, and therefore a bit mask representing the accepted pixels is carried to the Gather kernel. The averaged depth and normal vectors are used in the Obscurance kernel instead of the original pixel's values.

Gather kernel combines the per-pixel near-field obscurance with an average of the far-field obscurance from pixels that were marked as accepted in Box average kernel. Results are optionally blurred using a bilateral box filter of size 5×5 with a threshold of 7% (pixels within a maximum of 7% color difference are included into the average).

The SAAO enabled render times are shown in Table 4. The execution time of the Obscurance kernel decreases linearly in the number of evaluated sectors. In fact, the Scan and Prefix sum kernels now take more time than the Obscurance kernel in some cases, and speeding up these two kernels would be the logical next step in improving the execution times and is briefly discussed in Section 13. Overall, SAAO is an efficient way to trade banding for noise or blur while also improving render times significantly.

Phase	Radeon 7970	GTX 580
$K = 18 \times 2$ (3×3 separation), $B_0 = 10$, $1280(+256) \times 720(+144)$:		
Scan	1.07 ms	1.08 ms
Prefix sum	1.09 ms	1.04 ms
Box average	0.265 ms	0.400 ms
Obscurance, sep.	1.60 ms	3.00 ms
Gather (with blur)	0.255 (0.591) ms	0.290 (0.652) ms
total (with blur)	4.28 (4.62) ms	5.81 (6.17) ms
$1920(+384) \times 1080(+216)$:		
Scan	2.32 ms	3.08 ms
Prefix sum	2.03 ms	2.08 ms
Box average	0.566 ms	0.894 ms
Obscurance, sep.	3.82 ms	7.25 ms
Gather (with blur)	0.557 (1.31) ms	0.651 (1.46) ms
total (with blur)	9.29 (10.0) ms	14.0 (14.8) ms

Table 4: *Render time breakdown of our method per kernel with SAAO enabled.*

11 Limitations of depth buffer geometry

Scene geometry in a depth buffer is incomplete in two ways: (i) geometry outside the view frustum is unknown and (ii) geometry below the first depth layer is unknown. The first limitation is usually addressed by introducing a guard band around the depth buffer. In this paper we have used a guard band of 10% (extending the depth buffer by 10% of its width or height in each direction). As the screen-space radius of the obscurance effect may become arbitrarily large when scene geometry is close to the camera, it cannot be entirely contained within a guard band in any SSAO method. However, the slower the decay of the falloff function the larger the guard band generally needs to be and thus becomes important to our method. Fortunately the Z pre-pass is usually quick and lower resolution rasterization can be used in the guard bands to further minimize its cost. As long as the Z pre-pass does not have a high cost, we recommend even larger than 10% guard bands when calculating far-field AO effects in screen-space. It is also possible to construct a simplified *world-space* representation of occluders around the camera that are outside the depth buffer and accumulate SSAO with occlusion from them using a global AO method, however this approach is outside the scope of this paper

Most SSAO methods use only a single depth layer and make a generic assumption about the geometry below the nearest depth layer. Such an approach can never produce correct results in all scenes and the artefacts vary. Until this section we have assumed the depth field to be continuous, i.e. an infinitely thick volume. This has the benefits, for example, that depth field points can be averaged and interpolated, and objects appearing behind nearer geometry within the view frustum will not cause abrupt changes to obscurance. The downside is that obscurance is often overestimated, and to a large degree if there are thin objects, such as chains hanging in the air, near the camera. Because we in this paper advocate a high quality and physically correct SSAO, we find more promise in approaches that attempt to fill the missing scene geometry with real information of the scene instead of fitting a scene-dependent assumption. In previous work such information has been introduced in the form of multiple depth layers [Bavoil and Sainz 2009] and multiple views [Vardis et al. 2013]. Extending our method into that direction is left as future work.

Instead, we briefly demonstrate our method under the assumption that the depth field has a fixed finite thickness, an approach taken by many prior works such as [Loos and Sloan 2010] [McGuire et al. 2011] [McGuire et al. 2012]. While this will not work for arbitrary views or scenes that have varied objects, it produces plausible results when the depth field thickness is carefully selected and matches that of the viewed objects. Fixing the thickness requires a small change in the obscurance estimator in Eqn. 3: a_{i-1} is replaced with $max(a_{i-1}, \angle(\mathbf{h_i}+t(\mathbf{h_i}-\mathbf{c})/||\mathbf{h_i}-\mathbf{c}||-\mathbf{p}, \vec{z}))$ where t is the thickness of the depth field.

When the depth field thickness is finite the depth field becomes discontinuous. Therefore it is not allowed to generate new points through interpolation or averaging. This limitation does not much impact direct depth buffer samples which can be snapped to texel centers as done in [Bavoil et al. 2008]. In our method this means that averaging across the sector's width as described in Section 7 cannot be used, however we still retain the advantage over direct samples that our method tracks the local peaks along each sampling line. The mipmap method, however, is most impacted: Not only is interpolation spatially and across mip levels forbidden, but lower resolution level textures have to reuse values found from the base level and no averaging is possible. Out of various filters we found max-mipmaps to produce best results for mipmapping.

In Figure 12 we show the Stanford Dragon as rendered by our obscurance estimator with a fixed depth field thickness using our intermediate geometry samples, direct depth buffer samples, and max-mipmaps. All methods evaluate roughly 8 far-field samples per azimuthal direction. SAAO is not used. Direct depth field samples and our method produce banding especially since sector averaging cannot be used for mitigation. Therefore our method does not use the Prefix sum stage and reconstructs scene points as per Algorithm 1 and 2 directly. Our method produces smooth obscurance *along* each azimuthal direction whereas direct depth buffer samples produce artefacts depending on whether the samples hit or miss local peaks in the depth field. Due to the missed geometry, direct sampling also produces systematic underocclusion. Max-mipmaps do not miss geometry but systematically overestimate it by always picking the largest occluder within the sample's radius. Also, as linear interpolation cannot be used the results are blocky.

11.1 Jittering

It is possible to jitter the sampling directions per-pixel to trade banding for noise. In our method this can be achieved by adding or substracting an offset value from the sampled line indices l_A and l_B shown in Figure 6. The offset is randomly selected per pixel, and sampling along every direction is offset by the same amount as not to cause bias. The offset is scaled according to the distance from the receiver to the interval. Here, when averaging is not allowed and only a single line is sampled along each azimuthal direction, banding becomes especially severe if jittering is not used. Overall we find that jittering the sampling direction within the sector boundaries efficiently eliminates banding in return for some noise. Our method is the cache-friendliest of the three methods with respect to jittering because neighboring pixels access the same intervals which are laid out in memory consecutively and accesses are likely to hit the same cache lines. Mipmapping exhibits better cache locality than the sparser direct samples and its render times are not impacted significantly by jittering.

12 Conclusion

We have presented a method to solve ambient obscurance in screen-space from occluders that are beyond the immediate neighborhood of the receiving pixel. We do this by first scanning the depth buffer in a number of azimuthal directions while tracking local height maxima and writing them into an intermediate geometry buffer. After creating prefix sums of the intermediate geometry buffer, it can be sampled per-pixel to obtain approximated local peaks in the environment as seen from the receiver point, at various distances. These reconstructed scene points are then evaluated using an obscurance estimator to approximate the AO integral over the receiver's hemisphere. The obscurance effect in our method is only limited by the falloff term, and our method can incorporate any such term without its evaluation affecting render times. Overall our method is able to produce very high quality AO effects that are close to a ray traced screen-space reference.

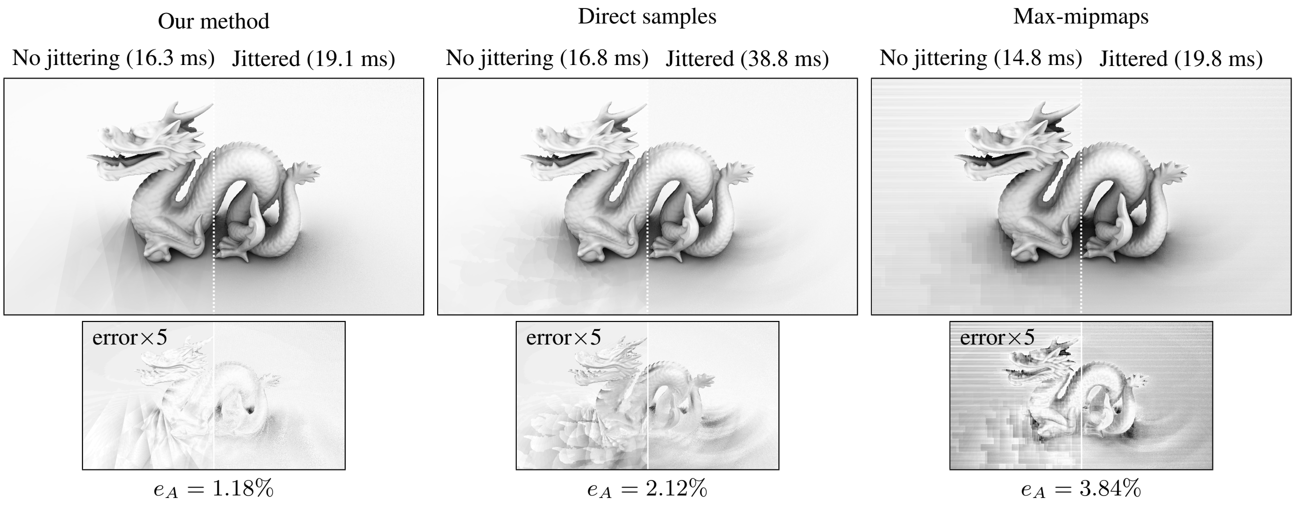
The intended use for our algorithm is to couple it with a lightweight near-field search to build a robust SSAO solution that accurately integrates ambient obscurance from the entire guard banded depth buffer.

13 Future work

Currently we scan the depth buffer densely, which is justifiable since the Obscurance stage takes most of the execution time and is not impacted by scanning density. However, a dense scan becomes costly when SAAO is used as a significant amount of the total time is spent in the Scan and Prefix sum stages that scale linearly in the number of scanned lines. It is possible to leave out some of the parallel lines during azimuthal scans without much impact on the calculated obscurance because the lines are always approximately facing the receiver and therefore have a limited contribution to AO. Ideally, processing every n:th line reduces the execution time of the Scan and Prefix sum stages by the factor $1/n$.

Overall there are four main strategies to reduce the render time of our method:

- Sparse scans as described above
- Separated obscurance evaluation sparser than 3×3 (which is used in Section 10), such as 5×5
- Reducing K and increasing the number of segments in which each sector is evaluated (e.g. $K = 8 \times 4$)



Our method
No jittering (16.3 ms) Jittered (19.1 ms)
Direct samples
No jittering (16.8 ms) Jittered (38.8 ms)
Max-mipmaps
No jittering (14.8 ms) Jittered (19.8 ms)
error×5
error×5
error×5
$e_A = 1.18\%$
$e_A = 2.12\%$
$e_A = 3.84\%$

Figure 12: *The Stanford Dragon rendered in* $K = 16$ *azimuthal directions with a hand-picked thickness* t *using our intermediate geometry (left), direct depth buffer samples (middle), and max-mipmaps (right). The right side of each image uses azimuthal directions that are randomly jittered per-pixel. The resolution is 1280(+256)×720(+144) and the render times are reported for the far-field (*$B_0 = 10$*) AO component on a GeForce GTX 580.*

- Constructing fewer points (larger intervals) per azimuthal direction per pixel.

We are also investigating the possibility of extending our method to handle multiple depth layers [Bavoil and Sainz 2009] or multiple views [Vardis et al. 2013] which—when coupled with sufficiently large guard bands—would alleviate the screen-space problem of missing scene geometry. This could allow our method to produce results comparable to global ray tracing.

References

BAVOIL, L., AND SAINZ, M. 2009. Multi-layer dual-resolution screen-space ambient occlusion. In *SIGGRAPH '09 Talks*, ACM.

BAVOIL, L., SAINZ, M., AND DIMITROV, R. 2008. Image-space horizon-based ambient occlusion. In *SIGGRAPH '08 Talks*.

HOANG, T.-D., AND LOW, K.-L. 2012. Efficient screen-space approach to high-quality multiscale ambient occlusion. *The Visual Computer 28*, 3, 289–304.

HUANG, J., BOUBEKEUR, T., RITSCHEL, T., HOLLÄNDER, M., AND EISEMANN, E. 2011. Separable approximation of ambient occlusion. In *Eurographics 2011 - Short papers*.

LAINE, S., AND KARRAS, T. 2010. Two methods for fast ray-cast ambient occlusion. *CGF: Proceedings of EGSR 2010 29*, 4.

LOOS, B. J., AND SLOAN, P.-P. 2010. Volumetric obscurance. In *Proceedings of I3D 2010*, ACM.

MCGUIRE, M., OSMAN, B., BUKOWSKI, M., AND HENNESSY, P. 2011. The alchemy screen-space ambient obscurance algorithm. In *Proc. HPG*, ACM, HPG '11, 25–32.

MCGUIRE, M., MARA, M., AND LUEBKE, D. 2012. Scalable ambient obscurance. In *High-Performance Graphics 2012*.

MCGUIRE, M. 2010. Ambient occlusion volumes. In *Proceedings of High Performance Graphics 2010*.

MITTRING, M. 2007. Finding next gen: Cryengine 2. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, ACM, 97–121.

REINBOTHE, C., BOUBEKEUR, T., AND ALEXA, M. 2009. Hybrid ambient occlusion. *EUROGRAPHICS 2009 Areas Papers*.

RITSCHEL, T., DACHSBACHER, C., GROSCH, T., AND KAUTZ, J. 2012. The state of the art in interactive global illumination. *Computer Graphics Forum 31* (Feb.).

SHANMUGAM, P., AND ARIKAN, O. 2007. Hardware accelerated ambient occlusion techniques on gpus. In *Proc. I3D '07*, ACM.

SNYDER, J., AND NOWROUZEZAHRAI, D. 2008. Fast soft self-shadowing on dynamic height fields. *Computer Graphics Forum: Eurographics Symposium on Rendering* (June).

TIMONEN, V., AND WESTERHOLM, J. 2010. Scalable Height Field Self-Shadowing. *Computer Graphics Forum (Proceedings of Eurographics 2010) 29*, 2 (May), 723–731.

TIMONEN, V. 2013. Line-Sweep Ambient Obscurance. *Computer Graphics Forum (Proceedings of EGSR 2013) 32*, 4.

VARDIS, K., PAPAIOANNOU, G., AND GAITATZES, A. 2013. Multi-view ambient occlusion with importance sampling. In *Proc. i3D*, I3D '13, 111–118.

ZHUKOV, S., INOES, A., AND KRONIN, G. 1998. An Ambient Light Illumination Model. In *Rendering Techniques '98*, Springer-Verlag Wien New York, G. Drettakis and N. Max, Eds., Eurographics, 45–56.

Appendix A

Algorithm listings

A.1 Horizon map computation

ProcessLineP1 outputs, in *horizonPoints*, the points that cast the highest horizon at each line point

```
// Operators are C-style
float2 getPoint(float2 stepCoord, float2 step)
    // HF(float2) samples the height field at the given coordinate
    return (stepCoord · step, HF(stepCoord))

bool isConvex(float2 p, float2 h0, float2 h1)
    float2 v1 = h0 − p;
    float2 v2 = h1 − p;
    return $v1_y/v1_x < v2_y/v2_x$

ProcessLineP1(float2 stepCoord, float2 step,
        int stepNum, float2 horizonPoints[stepNum])
    int stepIndex = 0
    // Handling the two first steps separately
    stack.push(getPoint(stepCoord, step))
    horizonPoints[stepIndex++] = NULL // Not defined for the first point
    stack.push(getPoint(stepCoord += step, step))
    horizonPoints[stepIndex++] = stack.peek(1)
    while (stepIndex < stepNum)
        float2 p = getPoint(stepCoord += step, step)

        // peek(n) returns the n:th last element in the stack
        while (stack.size() ≥ 2 &&
                !isConvex(p, stack.peek(1), stack.peek(2)))
            stack.pop()

        // The last element in the stack is the end point of the horizon
        horizonPoints[stepIndex++] = stack.peek(1)
        stack.push(p)
```

A.2 Intervisibility computation

ProcessLineP2 outputs, in *horizonPoints*, vectors holding the points that cast the local horizons at each line point

```
correctConvexity(node &child, node &parent, node &root)
    if (!isConvex(root, parent, child))
        root.insert(parent, child) // Connect child to root before parent
        if (child.next() != NULL) // If child has a next sibling
            parent.firstChild() = child.next()
            // Step wider in the tree
            correctConvexity(child.next(), parent, root)
        else
            root.disconnect(parent) // Remove the orphaned node

        if (child.firstChild() != NULL)
            // Step deeper in the tree
            correctConvexity(child.firstChild(), child, root)

ProcessLineP2(float2 stepCoord, float2 step,
      int stepNum, vector<float2> horizonPoints[stepNum])
    int stepIndex = 0
    node first, root
    // Handling the first two line steps separately
    first.vertex() = getPoint(stepCoord, step)
    horizonPoints[stepIndex++] = NULL // No visibility horizons for
          the first point
    root.vertex() = getPoint(stepCoord += step, step)
    horizonPoints[stepIndex++].insert(first) // Insert() adds to the vector

    root.firstChild() = first
    bool onConvexPart = isConvex(getPoint(stepCoord−step, step),
         getPoint(stepCoord, step), getPoint(stepCoord+step, step))
```

```
while (stepIndex < stepNum)
    node oldRoot = root // Copy the node
    root.clearLinkage() // Sets firstChild() and next() to NULL
    root.vertex() = getPoint(stepCoord += step, step)
    root.firstChild() = oldRoot // Old root becomes the only child

    correctConvexity(root.firstChild().firstChild(),
          root.firstChild(), root)

    // Output the local horizon points
    for (node horizon = root.firstChild();
          horizon != NULL; horizon = horizon.next())
        horizonPoints[stepIndex].insert(horizon.vertex())
    stepIndex++

    // Finally checking convexity
    bool thixConvexity = isConvex(getPoint(stepCoord−step, step),
         getPoint(stepCoord, step), getPoint(stepCoord+step, step))
    if (!onConvexPart && thisConvexity)
        // Convex part begun, forking the first point as a leaf
        node leaf
        leaf.vertex() = getPoint(stepCoord−step, step)
        root.insert(NULL, leaf)

    onConvexPart = thisConvexity
```

ISBN 978-952-12-3036-3
ISSN 1239-1883