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

Volume 29 (2010), Number 2

Scalable Height Field Self-Shadowing

Ville Timonen [1,2] and Jan Westerholm [1]

[1] Åbo Akademi University
[2] Turku Center for Computer Science

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 $1 \dots \sqrt{2}N$ 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.

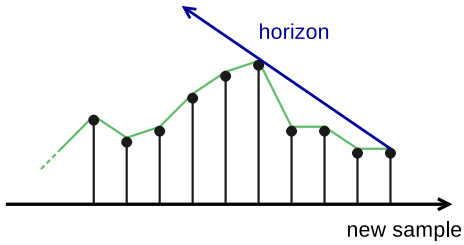

Figure 1: *Each thread supplies a horizon value from its internal representation of the height function for each height sample in its path.*

As it is critical for the performance of current graph-

ics 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.

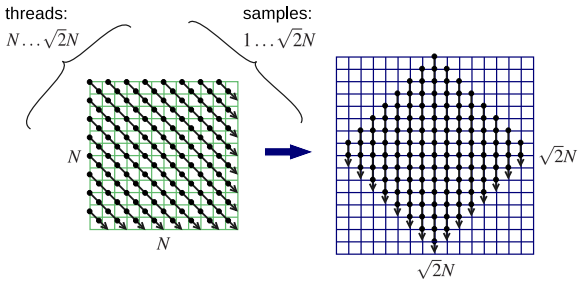

Figure 2: *For each azimuthal direction, $N \ldots \sqrt{2}N$ threads step through $1 \ldots \sqrt{2}N$ samples to cover a height field of resolution N^2 and write the results to an aligned output buffer.*

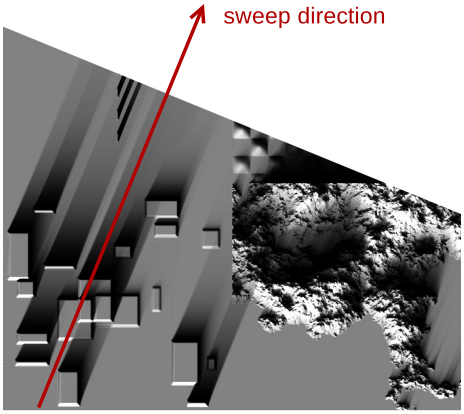

Figure 3: *Parallel threads have extracted the horizon angles for roughly half of their path during one azimuthal sweep. The full result is in Figure 5.*

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.

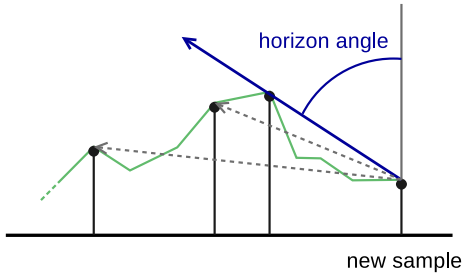

Figure 4: *The horizon angle for a sample is determined by the previous occluder in the convex hull set marked by ●.*

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*

```
v1 ← VECTOR(peek1 → new)
while size > 1 do
  v2 ← VECTOR(peek2 → new)
  if h(v2)d(v1) ≥ h(v1)d(v2) then
    v1 ← v2
    pop
  else
    break
  end if
end while
push(new)
return π/2 − tan^−1 h(v1)/d(v1)
```

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 $0 \dots \pi$.

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

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

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.

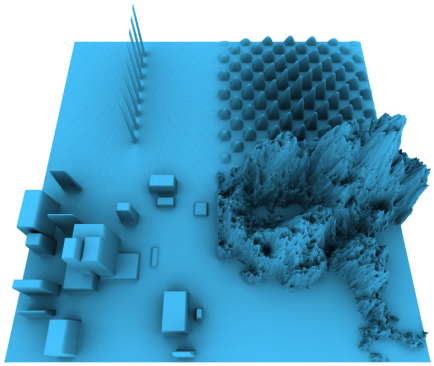

Figure 5: *A 1024^2 height field under uniform ambient lighting ($\phi_N = 64$, 24 fps)*

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.

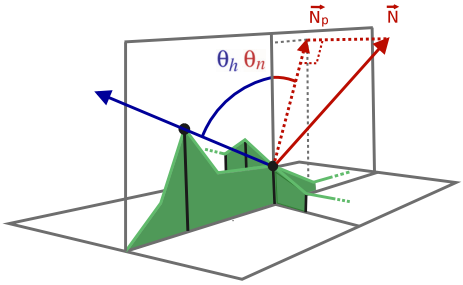

Figure 6: *When the normal $\vec{N}$ is projected onto the azimuthal plane, lighting can be tabulated as a function of the horizon angle θ_h and the angle θ_n for the projected normal.*

The lighting evaluation becomes

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

Precalculation of L_p for direction k becomes

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

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 $(0,0,1)$.

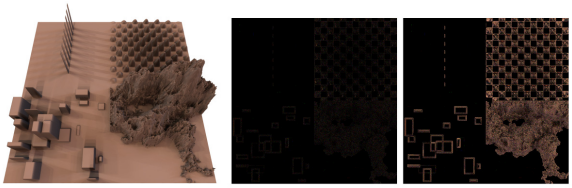

Figure 7: *Error introduced by lowering the normal angle resolution from a reference 256 to 16 (middle) and 8 (right) on a height field shown to the left, multiplied by 50. The average and maximum errors for resolution 8 are 0.16% and 1.66%, and for resolution 16 0.03% and 0.42% respectively.*

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

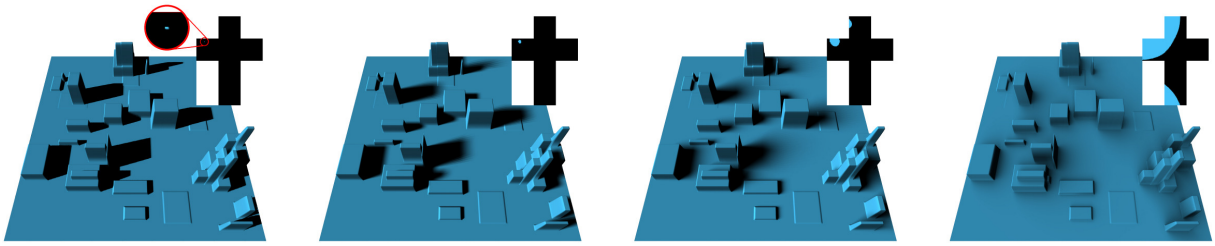

Figure 8: *Differently sized circular light sources with corresponding lighting cube maps on the top.* ($\theta_N = 256, \phi_N = 128$)

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 $\phi_N = 128$.

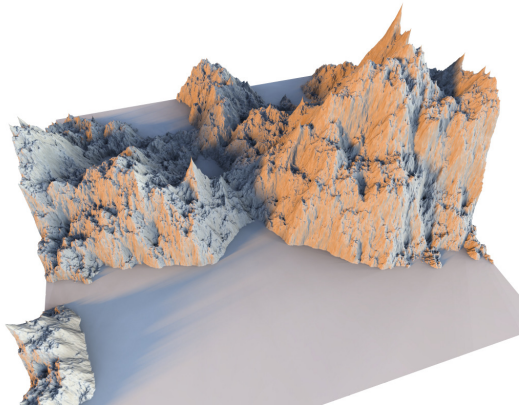

Figure 10: *A fractal terrain of size* 2048^2 *(8M triangles) lit by a single* 256×16 *environment at 20 Hz.*

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.

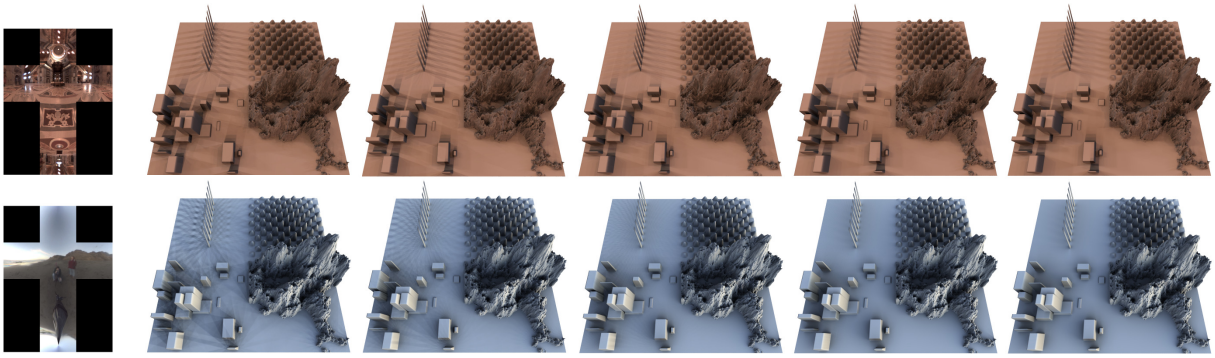

Figure 9: *Increasing azimuthal directions yields smoother results, but its effect depends on the geometric content. On the left are two light environments [Deb98] upon the* 1024^2 *height field,* $\theta_N = 256$*, and the number of azimuthal directions with the corresponding frame rates from left to 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.