Illuminating Micro Geometry Based on Precomputed Visibility

Wolfgang Heidrich[*] Katja Daubert Jan Kautz Hans-Peter Seidel

Max-Planck-Institute for Computer Science

Abstract

Many researchers have been arguing that geometry, bump maps, and BRDFs present a hierarchy of detail that should be exploited for efficient rendering purposes. In practice however, this is often not possible due to inconsistencies in the illumination for these different levels of detail. For example, while bump map rendering often only considers direct illumination and no shadows, geometry-based rendering and BRDFs will mostly also respect shadowing effects, and in many cases even indirect illumination caused by scattered light.

In this paper, we present an approach for overcoming these inconsistencies. We introduce an inexpensive method for consistently illuminating height fields and bump maps, as well as simulating BRDFs based on precomputed visibility information. With this information we can achieve a consistent illumination across the levels of detail.

The method we propose offers significant performance benefits over existing algorithms for computing the light scattering in height fields and for computing a sampled BRDF representation using a virtual gonioreflectometer. The performance can be further improved by utilizing graphics hardware, which then also allows for interactive display.

Finally, our method also approximates the changes in illumination when the height field, bump map, or BRDF is applied to a surface with a different curvature.

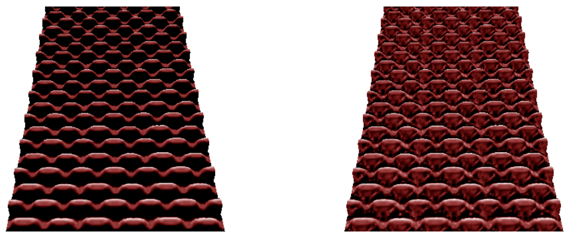
CR Categories: I.3.1 [Computer Graphics]: Hardware Architecture—Graphics processors; I.3.3 [Computer Graphics]: Picture/Image Generation—Bitmap and frame buffer operations; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, Shading, Shadowing and Texture

Keywords: Illumination Effects, Monte Carlo Techniques, Graphics Hardware, Frame Buffer Tricks, Reflectance & Shading Models, Texture Mapping

1 Introduction

Geometry, bump maps and bidirectional reflectance distribution functions (BRDFs) are often considered different levels of detail for the same surface structure. Although there has been a considerable amount of work on generating smooth transitions between these levels [19, 1], the fundamental problem of inconsistent illumination algorithms for the representations still remains.

For example, both simulated and measured BRDFs typically respect not only direct illumination, but also shadowing and masking effects of the micro geometry, as well as indirect illumination resulting from light that scatters between the micro surfaces. Geometry based representations usually consider direct illumination and shadowing/masking, but the indirect illumination is often neglected for performance reasons. Similarly, techniques for shadowing [19] and masking [1] in bump maps have been developed, but most applications do not use them; techniques for light scattering in bump maps have not been available so far. The importance of this indirect, scattered light to the overall appearance is illustrated in Figure 1.



(a) without scattering (b) with scattering

Figure 1: *Indirect light in height fields can have a strong impact on the overall appearance.*

With the advent of new, inexpensive computer graphics hardware that supports both bump mapping (see [23], [22], and [10] for some possible implementations) and rendering with arbitrary BRDFs [15, 10], some newly gained interest in making the transitions consistent has evolved. In this paper we introduce a single, efficient, but high-quality method for illuminating height field geometry and bump maps, as well as for precomputing BRDFs and even higher dimensional data structures. The method is suitable for both ray-tracing and hardware-accelerated rendering.

Our algorithm is based on precomputation and reuse of visibility information in height fields, simulates both shadowing and indirect illumination, and is able to approximate the illumination as the underlying base geometry changes. Thus it is capable of consistently illuminating height field geometry, bump maps, and BRDFs. The main contributions of this paper are:

- An algorithm for efficient computation of indirect light in height fields, based on precomputed visibility information.
- An efficient representation for shadowing and masking in bump maps.
- Hardware-accelerated versions of the above algorithms, using a generalization of Monte Carlo algorithms known as the “Method of Dependent Tests”.
- Techniques to approximate the changes in illumination when the height field is applied to a curved surface.

[*]MPI for Computer Science, Im Stadtwald, 66123 Saarbrücken, Germany, {heidrich,daubert,jnkautz,hpseidel}@mpi-sb.mpg.de

- Application of all methods to simulate BRDFs and render other high dimensional data structures such as light fields and bidirectional texture functions (BTFs).

Throughout this paper we will only deal with height field geometry, which also means that the simulated BRDFs can only originate from height fields. Furthermore, we assume that the bump heights are relatively small compared to their distance from light sources and other objects, so that the only occlusions occurring within a hight field are caused by parts of the same height field, but not by any other geometry in the scene.

The remainder of this paper is organized as follows: In Section 2 we briefly discuss related work. Then, we introduce our data structures for precomputed visibility and their applications to computing indirect illumination in Section 3. We also describe how to make use of graphics hardware for further performance improvements and interactive viewing (Section 3.3), and introduce new data structures for shadowing and masking in Section 4. We then discuss how to adapt both the scattering and the shadow data structures to varying base geometry (Section 5), and finally conclude with some results and a discussion (Section 6).

2 Related Work

Articles that discuss enhancing the original bump map algorithm [2] represent some of the work most closely related to ours. For example, Max [19] shows how to compute the self-shadowing of bump maps with a so-called “horizon map”. This horizon map describes the horizon for a small number of directions (8 in the original paper) at each point in the height field. During rendering, the shadow test then simply determines whether the light direction is above or below the (interpolated) horizon. Stewart [26, 27] introduced a hierarchical approach to determine the visibility in terrains both for occlusion culling and shading. More recently, Stewart used a similar idea to simulate the shadowing in cloth [28].

For the shadowing part of our paper, we use a concept similar to the original horizon map, but in a different representation of the horizon that allows for a highly efficient shadow test, which can also be performed with graphics hardware. In addition, we discuss how the data structure can be adapted to different curvatures of the underlying base geometry.

Masking is in a sense a dual problem to shadowing: where shadowing means that a light ray does not hit a specific surface point because it is occluded by some portion of the height field, masking means that the viewing ray does not hit the point for the same reason. In order to incorporate masking in bump maps, Becker and Max [1] introduced *redistribution bump mapping*, which adjusts the distribution of normals in the bump map with the viewing angle. Another possibility is to blend in coarse approximations of the height field as a displacement map. In our implementation, we take this latter approach, although using the results from Becker and Max should also be possible. Furthermore, we can reuse the horizon data structures, which we also apply for shadowing, to compute BRDFs.

So far, there has been little work on computing the indirect illumination in height fields and bump maps. Although Mostefaoui et al. [20] do integrate the indirect illumination at micro geometry scale into a global illumination simulation, their method relies on both precomputed data (BTFs) and a full geometric description of the features. It does not address the problem of precomputing these BTFs, or of computing the illumination in bump maps.

Several approaches for simulating BRDFs have been proposed in the past. The method by Cabral et al. [3] is based on horizon maps, while Becker and Max [1] use the normal distribution in a bump map. Our method is most closely related to the one from Westin et al. [31], which is based on ray-tracing. At the same time, we borrow the idea of using precomputed visibility information in bump maps from Cabral et al. [3], although our data structures are more comprehensive than simple horizon maps in order to account for indirect illumination.

For the hardware-accelerated variants of our algorithms we require some way of rendering bump mapped surfaces. Any of the recently published algorithms (e.g. [10, 22, 23]) will be sufficient. Mathematically, the hardware implementation of indirect illumination uses a generalization of Monte Carlo integration known as the *Method of Dependent Tests* [7]. This method, which is described in more detail in Section 3.1, uses the same random sampling pattern for estimating an integral at all different points (the indirect illumination in all different points of a height field, in our case). Several other hardware-based algorithms have implicitly used the Method of Dependent Tests in the past, for example most algorithms using the accumulation buffer [9] or the Instant Radiosity algorithm by Keller [16]. Another example is the transillumination method [29], an algorithm for global illumination computations, which is based on propagating light from all surfaces in one direction. Our method improves on this by using precomputed visibility that can be reused for many light paths, and allows for the use of graphics hardware.

While our algorithms yield a significant performance improvement for the generation of single images, they are even more attractive for the computation of higher dimensional data structures such as BRDFs, light fields [18, 8], volume representations [21], and spatially variant BRDFs or bidirectional texture functions (BTFs, [5]), because this allows for a reuse of the precomputed visibility information. Thus, the costly precomputation of visibility can be amortized over a larger amount of reuses. For the same reason our methods are even more attractive if applied to periodic height fields.

3 Light Scattering in Height Fields

To compute the indirect illumination in a height field, we have to solve an integral equation called the Rendering Equation [13]. This requires integrating over the incident illumination in each point of the height field, which can, for example, be achieved with Monte Carlo ray-tracing. The most expensive part of this integration is typically the visibility computation, which determines the surface visible from a given surface point in a certain direction. This is the part that depends on the complexity of the scene, while the computation of the local interaction of the light with the surface has a constant time complexity.

In the case of small-scale height fields that describe the irregularities of a surface, we can make two simplifying assumptions. Most importantly, we only deal with cases where the visibility inside the height field is completely determined by the height field itself, and not by any external geometry. This is equivalent to requesting that no external geometry penetrates the convex hull of the height field, which is a reasonable assumption for the kind of small surface structures that we are targeting. It allows us to precompute the visibility information, and afterwards combine it to a variety of different paths, so that the cost of its computation can be amortized over a larger number of light paths.

Secondly, in the case where we want to use our method to compute a BRDF, we request that the height field geometry is small compared to the remainder of the scene, and therefore any incoming direct light can be assumed parallel. This is necessary simply because the BRDF by definition is a function of *exactly one* incoming direction and *exactly one* outgoing direction. This assumption is not necessary for the other levels of detail, i.e. bump maps and displacement maps.

The visibility restriction requests that no other geometry in the scene will act as an occluder between two points in the height field. If we now assume the height field is attached to a specific, fixed base geometry, we can, for a given point $\mathbf{p}$ on the height field, and a given direction $\vec{d}$, precompute whether the ray originating at $\mathbf{p}$ in direction $\vec{d}$ hits some other portion of the height field, or not. Furthermore, if it does intersect with the height field, we can precompute the intersection point and store it in a data base. Since this intersection point is some point in the same height field, it is unambiguously characterized by a 2D texture coordinate.

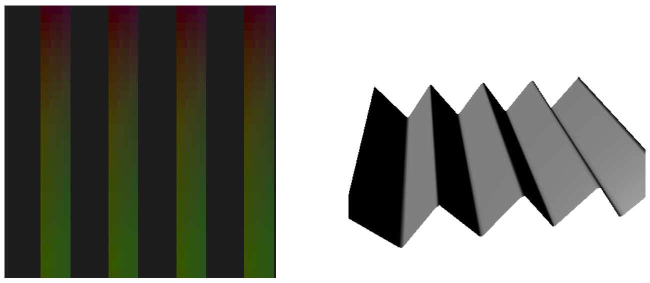


Figure 2: *One of the scattering textures S_i for the triangular height field on the right.*

Now imagine having a set D of N uniformly distributed directions, and having precomputed the visible surface point for every direction $\vec{d} \in D$ and for every grid point in the height field texture. If the height field is periodic, this has to be taken into account for determining this visibility information. Also, for the moment we assume that the height field is applied to a specific base geometry, typically a flat surface. Section 5 deals with an adaptation to varying base geometries.

We store this precomputed data as a set of N texture maps S_i; for each direction $\vec{d_i} \in D$ there is one 2D texture with two components representing the 2D coordinates of the visible point. Each of these textures is parameterized the same way the height field is, i.e. the 2D texture coordinates directly correspond to height field positions $\mathbf{p}$. The texture value also corresponds to a point in the height field and represents the surface point $\mathbf{q}$ that is visible from $\mathbf{p}$ in direction $\vec{d_i}$. Note again, that these visibility textures are only valid for a given, predefined base geometry to which the height field is attached. Section 5 will describe how this information can be used to illuminate the height field when it is attached to other geometries.

By chaining together this visibility information, we can now generate a multitude of different light paths for computing the indirect illumination in the height field. This way, it is possible to implement variants of many existing Monte Carlo algorithms, using the precomputed data structures instead of on-the-fly visibility computations. Below, for example, we outline a simple path tracing algorithm that computes the illumination at a given surface point, but ignores indirect light from geometry other than the height field:

```
radiance( $\mathbf{p}$, $\vec{v}$ ) {
  L:= direct illumination( $\mathbf{p}$ );
  i:= random number in $[1 \ldots N]$;
  if($\mathbf{q} := S_i[\mathbf{p}]$ is valid height field coord.){
    L:= L + $f_r(\mathbf{p}, \vec{v}, \vec{d_i}) \cdot \cos(\angle(\vec{d_i}, \vec{n}_p)) \cdot$
            radiance( $\mathbf{q}$, $-\vec{d_i}$ );
  }
  return L;
}
```

In this algorithm, $\vec{n}_p$ is the bump map normal in point $\mathbf{p}$, and $f_r(\mathbf{p}, \vec{v}, \vec{d_i})$ is the BRDF of the height field in that point. The direct illumination in each point is computed using a bump mapping technique.

Of course the visibility information for direction S_i is only known at discrete height field grid positions. At other points, we can only exactly reconstruct the direct illumination, while the indirect light has to be interpolated. For example, we can simply use the visibility information of the closest grid point as $S_i[\mathbf{p}]$. This nearest-neighbor reconstruction of the visibility information corresponds to a quantization of texture coordinates, so that these always point to grid points of the height field. For higher quality, we can also choose a bilinear interpolation of the indirect illumination from surrounding grid points. In our implementation, we use the nearest-neighbor approach for all secondary intersections by simply quantizing the texture coordinates encoded in the visibility textures S_i. On the other hand, we use the interpolation method for all primary intersections to avoid blocking artifacts. Figure 1b shows a result of this method. For more complex examples, see Section 6.

The simple algorithm above ignores shadowing, but with the technique described in Section 4, which is similar to the one introduced by Max [19], shadows can also be included.

Using similar methods, other Monte Carlo algorithms like distribution ray-tracing [4] can also be built on top of this visibility information. The advantage of using precomputed visibility for the light scattering in height fields, as described in this section, is that the visibility information is reused for different paths. Therefore, the cost of computing it can be amortized over several uses.

3.1 The Method of Dependent Tests

As mentioned above, we have to solve the Rendering Equation [13] in order to determine the indirect illumination in a height field. Based on the precomputed visibility information, we solve the Rendering Equation by Monte Carlo integration of the incident illumination at any given surface point, and obtain the reflected radiance for that point and a given viewing direction.

In general, however, we do not only want to compute the reflected light for a single point on the height field, but typically for a large number of points. With standard Monte Carlo integration, we would use different, statistically independent sample patterns for each of the surface points we are interested in.

The Method of Dependent Tests [7] is a generalization of Monte Carlo techniques that uses the same sampling pattern for all surface points. More specifically, we choose the same set of directions for sampling the incident light at all surface points. For example, as depicted in Figure 3, for all points $\mathbf{p}$ in the height field, we collect illumination from the same direction $\vec{d_i}$.

As pointed out by Keller [17], there are several instances in the computer graphics literature, where the Method of Dependent Tests has been applied implicitly [9, 16]. For example, one of the standard algorithms for the accumulation buffer [9] is a depth-of-field effect, which uses identical sampling patterns of the lens aperture for all pixels. It has been shown by Sobol [25] that the Method of Dependent Tests is an unbiased variant of Monte Carlo integration. Recently, hierarchical versions of the Method of Dependent Tests have been proposed [11, 17], but we do not currently make use of these results.

3.2 Dependent Test Implementation of Light Scattering in Height Fields

Based on the Method of Dependent Tests, we can rewrite Monte Carlo algorithms as a sequence of SIMD operations that operate on the grid cells of the height field. Consider the light path in Figure 3. Light hits the height field from direction $\vec{l}$, scatters at each point in direction $-\vec{d_i} \in D$, and leaves the surface in the direction of the viewer $\vec{v}$.

Since all these vectors are constant across the height field, the only varying parameters are the surface normals. More specifically, for the radiance leaving a grid point $\mathbf{p}$ in direction $\vec{v}$, the important

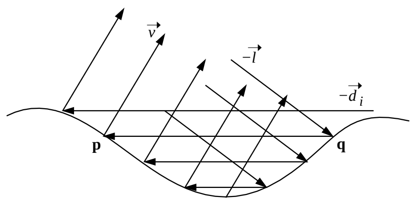


Figure 3: *With the Method of Dependent Tests, the different paths for the illumination in all surface points are composed of pieces with identical directions.*

varying parameters are the normal $\vec{n}_p$, the point $\mathbf{q} := S_i[\mathbf{p}]$ visible from $\mathbf{p}$ in direction $\vec{d}_i$, and the normal $\vec{n}_q$ in that point.

In particular, the radiance in direction $\vec{v}$ caused by light arriving from direction $\vec{l}$ and scattered once in direction $-\vec{d}_i$ is given by the following formula.

$$\begin{aligned} L_o(\mathbf{p},\vec{v}) =& f_r(\vec{n}_p,\vec{d}_i,\vec{v}) < \vec{n}_p,\vec{d}_i > \cdot \\ & \Big(f_r(\vec{n}_q,\vec{l},-\vec{d}_i) < \vec{n}_q,\vec{l} > \cdot L_i(\mathbf{q},\vec{l}) \Big) . \end{aligned} \tag{1}$$

Usually, the BRDF is written as a 4D function of the incoming and the outgoing direction, both given relative to a local coordinate frame where the local surface normal coincides with the z-axis. In a height field setting, however, the viewing and light directions are given in some global coordinate system that is not aligned with the local coordinate frame, so that it is first necessary to perform a transformation between the two frames. To emphasize this fact, we have denoted the BRDF as a function of the incoming and outgoing direction as well as the surface normal. If we plan to use an anisotropic BRDF on the micro geometry level, we would also have to include a reference tangent vector.

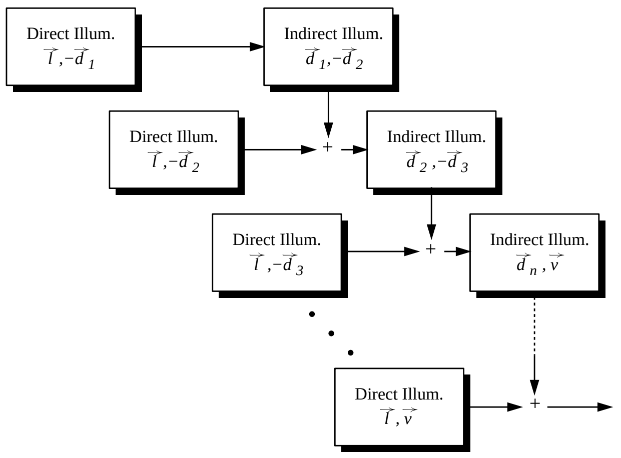
Note that the term in parenthesis is simply the direct illumination of a height field with viewing direction $-\vec{d}_i$, with light arriving from $\vec{l}$. If we precompute this term for all grid points in the height field, we obtain a texture L_d containing the direct illumination for each surface point. This texture can be generated using a bump mapping step where an orthographic camera points down onto the height field, but $-\vec{d}_i$ is used as the viewing direction for shading purposes.

Once we have L_d, the second reflection is just another bump mapping step with $\vec{v}$ as the viewing direction and $\vec{d}_i$ as the light direction. This time, the incoming radiance is not determined by the intensity of the light source, but rather by the content of the L_d texture. For each surface point $\mathbf{p}$ we look up the corresponding visible point $\mathbf{q} = S_i[\mathbf{p}]$. The outgoing radiance at $\mathbf{q}$, which is stored in the texture as $L_d[\mathbf{q}]$, is at the same time the incoming radiance at $\mathbf{p}$.

Thus, we have reduced computing the once-scattered light in each point of the height field to two successive bump mapping operations, where the second one requires an additional indirection to look up the illumination. We can easily extend this technique to longer paths, and also add in the direct term at each scattering point. This is illustrated in the Figure 4.

For the total illumination in a height field, we sum up the contributions for several such paths (some 40-100 in most of our scenes). This way, we compute the illumination in the complete height field at once, using two SIMD-style operations on the whole height field texture: bump mapping for direct illumination, using two given directions for incoming and outgoing light, as well as a lookup of the indirect illumination in a texture map using the precomputed visibility data in form of the textures S_i.

This is in itself a performance improvement over the regular Monte Carlo algorithms presented before, because the illumination



Direct Illum. $\vec{l}, -\vec{d}_1$
Indirect Illum. $\vec{d}_1, -\vec{d}_2$
Direct Illum. $\vec{l}, -\vec{d}_2$
+
Indirect Illum. $\vec{d}_2, -\vec{d}_3$
Direct Illum. $\vec{l}, -\vec{d}_3$
+
Indirect Illum. $\vec{d}_n, \vec{v}$
Direct Illum. $\vec{l}, \vec{v}$
+

Figure 4: *Extending the dependent test scattering algorithm to multiple scattering. Each box indicates a texture that is generated with regular bump mapping.*

in one grid cell will contribute to many other points on the surface in the final image via light scattering. In contrast to standard Monte Carlo, our dependent test approach avoids recomputing this contribution for each individual pixel.

What remains to be done is an efficient test of whether a given point lies in shadow with respect to the light direction $\vec{l}$. While it is possible to interpolate this information directly from the visibility database S_i, we can also find a more efficient, although approximate representation, that will be described in Section 4.

3.3 Use of Graphics Hardware

In addition to the above-mentioned performance improvements we get from the implementation of the Method of Dependent Tests in software, we can also utilize graphics hardware for an additional performance gain. In recent graphics hardware, both on the workstation and on the consumer level, several new features have been introduced that we can make use of. In particular, we assume a standard OpenGL-like graphics pipeline [24] with some extensions as described in the following.

Firstly, we assume the hardware has some way of rendering bump maps. This can either be supported through specific extensions (e.g. [22]), or through the OpenGL imaging subset [24], as described by Heidrich and Seidel [10]. Any kind of bump mapping scheme will be sufficient for our purposes, but the kind of reflection model available in this bump mapping step will determine what reflection model we can use to illuminate our hight field.

Secondly, we will need a way of interpreting the components stored in one texture or image as texture coordinates pointing into another texture. One way of supporting this is the so-called *pixel texture* extension [12, 10], which performs this operation during transfer of images into the frame buffer, and is currently only available on some high-end SGI machines. Alternatively, we can use *dependent texture lookups*, a variant of multi-texturing, that has recently been announced by several vendors for the next generation of consumer level hardware (the Matrox G400 offers a restricted version of this today). With dependent texturing, we can map two or more textures simultaneously onto an object, where the texture coordinates of the second texture are obtained from the components of the first texture. This is exactly the feature we are looking for. In case we have hardware that supports neither of the two, it is quite simple, although not very fast, to implement the pixel texture extension in software: the framebuffer is read out to main memory, and each pixel is replaced by a value looked up from a texture, using the previous contents of the pixel as texture coordinates.

Using these two features, dependent texturing and bump mapping, the implementation of the dependent test method as described above is simple. As mentioned in Section 3.2 and depicted in Figure 3, the scattering of light via two points $\mathbf{p}$ and $\mathbf{q}$ in the height field first requires us to compute the direct illumination in $\mathbf{q}$. If we do this for all grid points we obtain a texture L_d containing the reflected light caused by the direct illumination in each point. This texture L_d is generated using the bump mapping mechanism the hardware provides. Typically, the hardware will support only diffuse and Phong reflections, but if it supports more general models, then these can also be used for our scattering implementation.

The second reflection in $\mathbf{p}$ is also a bump mapping step (although with different viewing- and light directions), but this time the direct illumination from the light source has to be replaced by a per-pixel radiance value corresponding to the reflected radiance of the point $\mathbf{q}$ visible from $\mathbf{p}$ in the scattering direction. We achieve this by bump mapping the surface with a light intensity of 1, and by afterwards applying a pixel-wise multiplication of the value looked up from L_d with the help of dependent texturing. Figure 5 shows how to conceptually set up a multi-texturing system with dependent textures to achieve this result.

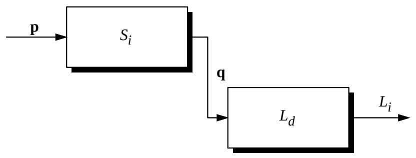


Figure 5: *For computing the indirect light with the help of graphics hardware, we conceptually require a multi-texturing system with dependent texture lookups. This figure illustrates how this system has to be set up. Boxes indicate one of the two textures, while incoming arrows signal texture coordinates and outgoing ones mean the resulting color values.*

The first texture is the S_i that corresponds to the scattering direction d_i. For each point $\mathbf{p}$ it yields $\mathbf{q}$, the point visible from $\mathbf{p}$ in direction d_i. The second texture L_d contains the reflected direct light in each point, which acts as an incoming radiance at $\mathbf{p}$.

By using this hardware approach, we treat the graphics board as a SIMD-like machine which performs the desired operations, and computes one light path for each of the grid points at once. As shown in Section 6, this use of hardware dramatically increases the performance over the software version to an almost interactive rate.

4 Approximate Bump Map Shadows

As mentioned in Section 3.1, we can simply use scattering information stored in S_i for determining the shadows cast in a height field. For example, to determine if a given grid point $\mathbf{p}$ lies in shadow for some light direction, we could simply find the closest direction $\vec{d_i} \in D$, and use texture S_i to determine whether $\mathbf{p}$ sees another point of the height field in direction $\vec{d_i}$.

For a higher quality test, we can precompute a triangulation of all points points on the unit sphere corresponding to the unit vectors $\vec{d_i}$ (since the set of directions is the same for all surface points, this is just one triangle mesh for all points on the height field). The same triangulation will later be used in Section 5 for other purposes. Based on this mesh, we can easily determine the three directions $\vec{d_i}$ that are closest to any given light direction, and then interpolate those directions' visibility values. This yields a visibility factor between 0 and 1 defining a smooth transition between light and shadow.

Although this approach works, we have also implemented a more approximate method that is better suited for hardware implementation and much faster.

We start by projecting all the unit vectors for the sampling directions $\vec{d_i} \in D$ of the upper hemisphere over the shading normal into the tangent plane, i.e. we drop the z coordinate of $\vec{d_i}$ in the local coordinate frame. Then we fit an ellipse containing as many of those 2D points that correspond to unshadowed directions as possible, without containing too many shadowed directions. This ellipse is uniquely determined by its (2D) center point $\mathbf{c}$, a direction $(a_x, a_y)^T$ describing the direction of the major axis (the minor axis is then simply $(-a_y, a_x)^T$), and two radii r_1 and r_2, one for the extent along each axis.

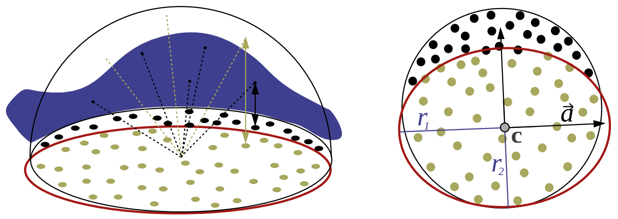


Figure 6: *For the shadow test we precompute 2D ellipses at each point of the height field, by fitting them to the projections of the scattering directions into the tangent plane.*

For the fitting process, we begin with the ellipse represented by the eigenvectors of the covariance matrix of all points corresponding to unshadowed directions. We then optimize the radii with a local optimization method. As an optimization criterion we try to maximize the number of light directions inside the ellipse while at the same time minimizing the number of shadowed directions inside it.

Once we have computed this ellipse for each grid point in the height field, the shadow test is simple. The light direction $\vec{l}$ is also projected into the tangent plane, and it is checked whether the resulting 2D point is inside the ellipse (corresponding to a lit point) or not (corresponding to a shadowed point). This approach is similar to the one described by Max [19] using horizon maps. Only here the horizon map is replaced by a map of ellipses, each uniquely determined by 6 parameters.

Both the projection and the in-ellipse test can mathematically be expressed very easily. First, the 2D coordinates l_x and l_y have to be transformed into the coordinate system defined by the axes of the ellipse:

$$l'_x := < \begin{pmatrix} a_x \\ a_y \end{pmatrix}, \begin{pmatrix} l_x - c_x \\ l_y - c_y \end{pmatrix} >, \quad (2)$$

$$l'_y := < \begin{pmatrix} -a_y \\ a_x \end{pmatrix}, \begin{pmatrix} l_x - c_x \\ l_y - c_y \end{pmatrix} > \quad (3)$$

Afterwards, the test

$$1 - \frac{(l'_x)^2}{r_1^2} - \frac{(l'_y)^2}{r_2^2} \geq 0 \quad (4)$$

has to be performed.

To map these computations to graphics hardware, we represent the six degrees of freedom for the ellipses as 2 RGB textures. Then the required operations to implement Equations 2 through 4 are simple dot products as well as additions and multiplications. Both Westermann et al. [30] and Heidrich and Seidel [10] have shown how such operations can be implemented on graphics hardware. This is possible using the OpenGL imaging subset [24], available on most contemporary workstations, but also using some vendor specific extensions, such as the *register combiner* extension from NVIDIA [22]. Depending on the exact graphics hardware available, the implementation details will have to vary slightly. Thus, they are omitted from this paper, and we refer the interested reader to our technical report [14].

5 Varying the Base Geometry

So far we have only considered the case where the height field is attached to a base geometry of a fixed, previously known curvature, typically a planar object. However, if we plan to use the same height field for different geometric objects, the valleys in a height field widen up or narrow down depending on the local curvature of the object, and the height field can be locally stretched in a non-uniform fashion. This affects both the casting of shadows and the scattering of indirect light. For the shadows, it is obvious that narrower valleys will cause more regions to be shadowed, while in wider valleys more regions are lit.

For the scattering part, the opposite is true. For a point on the bottom of a narrow valley, a large proportion of the solid angle is covered by other portions of the height field, and therefore the impact of indirect light is strong. On the other hand, in a wide valley, most of the light will be reflected back into the environment rather than remaining inside the height field.

In this section we discuss adaptations of the previously described algorithms and data structures to the case where the base geometry changes. To this end, we will assume that the curvature of this base geometry is small compared to the features in the height field. It is then a reasonable assumption that the visibility does not change as the surface is bent. This means that two points in the height field that are mutually visible for a planar base geometry, are also mutually visible in the curved case. Obviously, this assumption breaks down for extreme curvatures, but it generally holds for small ones.

First let us consider the data structures and algorithms for computing scattered, indirect light. Since we have assumed that no extreme changes in visibility occur, the precomputed visibility data i.e. the textures S_i are still valid as the underlying geometry changes. However, as depicted in Figure 7, some parameters of the illumination change. Firstly, there is no longer a fixed global direction $\vec{d}_i$ corresponding to each texture S_i. Rather, the direction changes as a parameter of the curvature and of the distance between two mutually visible points, and becomes different for every point on the surface. Secondly, the normal (and therefore the angles between the normal and other vectors) changes as a function of the same parameters.

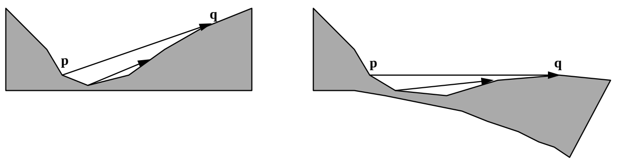


Figure 7: *The directions $\vec{d}_i$ change on a per-pixel basis if the height field is applied to a curved base geometry. The rate of change depends on the distance of two points from each other.*

These changes remove the coherence that we used to map the algorithm to graphics hardware, since now all directions need to be computed for each individual height field point.This requires operations that are currently not possible with graphics hardware. On the other hand, the abovementioned changes are quite easy to account for in a software renderer.

However, there is a third change due to the curvature, which affects all our Monte Carlo algorithms. The set of directions D used to be a uniform sampling of the directional sphere for the case of a given, fixed base geometry. Now, when the height field is applied to a geometry with slightly changed curvature or a non-uniformly scaled one, the directions change as mentioned above. The rate of change depends on the distance of the two mutually visible points. Therefore, the directions do not change uniformly, and, as a consequence, the sampling of directions is no longer uniform. In Monte Carlo terms, this means that the importance of the individual directions has changed, and that this importance has to be taken into account for the Monte Carlo integration. Different light paths can no longer be summed up with equal weight, but have to be weighted by the importance of the respective path. This importance has to be computed for every individual point in the height field.

This requires us to develop an estimate for the importance of a given sample direction, which is explained in the following. We start by interpreting the unit directions $d_i \in D$ for the original geometry as points on the unit sphere, and generate a triangulation of these. Since the sampling of directions is uniform in this planar case, the areas of the triangle fans surrounding any direction d_i will be approximately the same for all d_i, see Figure 8.

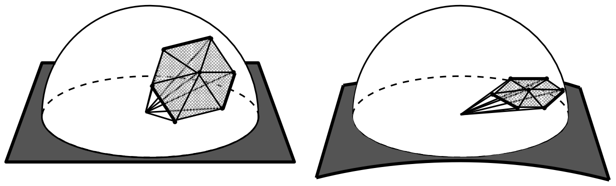


Figure 8: *When a height field is applied to a different base geometry, the importance of the individual directions changes, which is indicated by a change of area of the triangulated unit directions on the sphere.*

Now, if we gradually bend the underlying surface, the points corresponding to the directions will slowly move either towards the horizon or towards the zenith, depending on the sign of the curvature we apply. Note that a change in visibility means that during this movement the triangle mesh folds over at a given point. As mentioned above, we are going to ignore this situation, and restrict ourselves to small curvatures which do not cause such visibility changes.

In this case, the sole effect of the moving points on the unit sphere is that the areas of the triangle fans surrounding each direction change (see Figure 8). This change of area is an estimate for the change in sampling rate, and therefore an estimate for the importance of a particular direction in the curved case. Thus, if we apply a height field to a curved surface, we weight all light paths by the relative area of the triangle fan surrounding the chosen direction.

Now that we have dealt with the adaptation of the scattering data structures, we also have to take care of the shadowing. If we compute the shadows directly from the S_i, as described at the beginning of Section 4, then no changes are required. However, if we are using the 2D ellipses introduced at the end of Section 4, then these ellipses have to be adapted to the local surface curvature.

Starting from the updated scattering directions d_i, we can fit a different ellipse for each point and each surface curvature. However, precomputing and storing this information for a lot of different curvatures is both memory and time consuming. We therefore only precompute a total of five different ellipses: the original one for zero curvature, one each for a slight positive and a slight negative curvature in each of the parametric directions. From this data we can then generate a linear approximation of the changes of ellipse parameters under any given curvature. Again, this only works reasonably as long as the radii of curvature are large compared to the height field features (i.e. as long as the curvatures are small), but for large curvatures we will run into visibility changes anyway.

6 Results

We have implemented the approaches described in this paper both in software, and for two different kinds of graphics hardware. Firstly, we use the SGI Octane, which provides support for pixel textures, but does not have advanced features like multi-texturing, which would help us to reduce the number of rendering passes. On this platform we have implemented the Phong reflection model using the normal map approach described by Heidrich and Seidel [10]. Heidrich and Seidel also describe ways of incorporating other reflection models as well as environment maps through the use of pixel textures. We have not made use of these results.

Secondly, we have used an NVIDIA GeForce 256 with DDR RAM. This graphics board supports multi-texturing with very flexible ways of combining the resulting colors for each fragment (via NVIDIA's register combiner extension [22]). This allows us to perform bump mapping with local illumination and the Phong model in one pass, and helps us to efficiently implement the shadow test. However, the GeForce does not support dependent texture lookups, so that the scattering had to be implemented essentially using a software version of the pixel texture extension.

The first tests we have performed are designed to show whether we can use precomputed visibility to consistently illuminate geometry and bump maps, and also to simulate BRDFs. Figure 9 shows some curved geometry to which the triangular height field from Figure 2 has been applied. In this height field, the faces pointing in one direction are red, and the faces pointing in the other are white. The top row of Figure 9 shows the results of applying the geometry as a displacement map. On the left side, which does not include scattering but shadowing and masking, the separation of the colors becomes apparent, since the top of the geometry is more reddish, while the bottom is white. Due to color bleeding, the image including the scattering term on the right is more homogeneous. The bottom row of the figure shows the geometry with a BRDF that has been computed from the same height field using graphics hardware. Both the version with and the one without scattering show the same kind of behavior as the geometry-based rendering, which illustrates that our technique can be used for smooth transitions between levels of detail.

Both for the rendering of the geometry-based image and for the generation of the BRDF, we first had to generate the visibility data, namely the textures S_i and the ellipse data structures for the shadows. The two leftmost columns of Table 1 show the timings for this precomputation phase and a number of different height fields. The memory requirements for the data structures are quite low: for the scattering in a 32×32 height field with 100 sample directions we generate 100 two-component textures with a size of $32 \times 32 \times 2$ Bytes, which amounts to less than 2 MB of data for the whole scattering information. The shadowing data structure simply consists of two three-component textures, yielding $32 \times 32 \times 6 = 6144$ Bytes.

After the data structures are precomputed, we can efficiently compute images with scattering (100 samples) and shadowing/masking from them using either a software or a hardware renderer. The times for computing the scattering terms in the height field are listed in the third and fourth column of Table 1.

Note that the timings for hardware rendering of small (32×32) height fields including a one-time scattering are well below one second. Thus, we can generate images of scattered height fields at interactive frame rates, although not quite fast enough for applications like games.

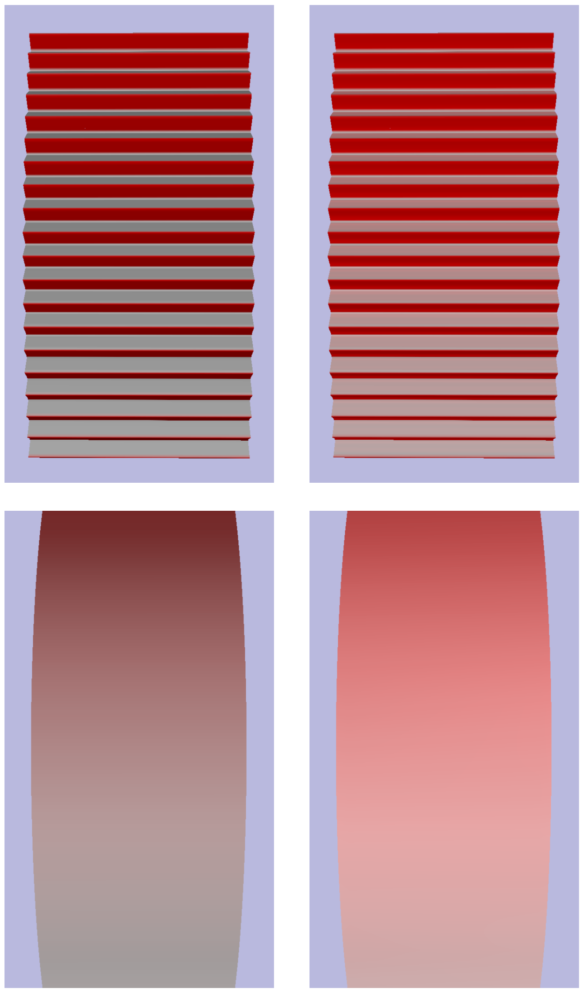


Figure 9: *A comparison of geometry (top) and BRDF (bottom). Left side: without indirect light, right side: with indirect light.*

Height Field	S_i	Shadows	SW	HW
Triangles (32×32)	27	32	10	0.48
Bricks (128×128)	1029	194	12	2.10
Bumps (32×32)	109	65	8	0.48

Table 1: *Timings for precomputation and rendering of different height fields in seconds.*

However, we can use the hardware algorithm to compute higher-dimensional data structures, such as light fields [8, 18] and both space variant and space invariant BRDFs. For example, we can generate a light field consisting of 32×32 images of a height field including scattering terms in just about 6-8 minutes.

As we move to BRDFs, a single BRDF sample is the average radiance from a whole image of the height field. Thus, if we would like to compute a dense, regular mesh of samples for a BRDF, we have to compute a 4-dimensional array of images, and then average the radiance of each image. The BTF [5], on the other hand, is a 6-dimensional data structure obtained by omitting the averaging step, and storing the images directly. These operations can become prohibitively expensive: even for relatively small BRDF resolutions such as 16^4, this would take about 7-8 hours. However, as other researchers have pointed out before [3, 31], it is not necessary to compute this large number of independent samples. Since BRDFs are typically smooth functions, it is sufficient to compute several hundred random samples, and project those into a hierarchical basis such as spherical harmonics.

Using our approach, this small number of samples can be generated within several minutes. To further improve the performance slightly, we can completely get rid of geometry for the computation of BRDF samples, and work in texture space. As described in Sections 3.2 and 3.3, the Method of Dependent Tests already operates in texture space. Only in the last step, when we want to display the result, we normally have to apply this texture to geometry. For the BRDF computation, however, we are only interested in the average of the radiances for the visible surface points. Therefore, if we manage to solve the masking problem by some other means, we do not have to use geometry at all. The masking problem can be solved by using the same data structures as used for the shadow test, only with the viewing direction instead of the light direction. This technique was first proposed by Cabral et al. [3] for their method of shadowing bump maps.

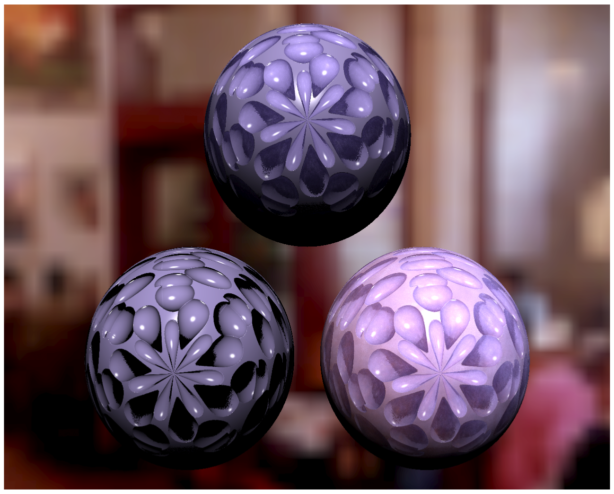


Figure 10: *Three bump-mapped spheres. Bottom left: with shadows only. Top: with shadows and indirect light bouncing off other parts of the bump map. Bottom right: with additional indirect light looked up from an environment map.*

Figure 10 shows some more examples for our technique. The bottom left sphere is rendered with a bump map using only direct light and our shadow test. The top sphere uses the same bump map, but also includes indirect light reflected from other portions of the bump map up to a path length of 4. Finally, in the bottom right sphere, we also include indirect illumination from other parts of the scene, which, in this case, is represented as an environment map, similarly to the method described by Debevec [6]. This is implemented by querying the environment map every time the visibility textures S_i indicate that no intersection occurs with the height field for the given direction.

Figure 11 demonstrates the effect of different curvatures of the underlying geometry, that other researchers have neglected so far. Note that the red faces receive only indirect illumination through scattering from the white faces. We can clearly see the reduced scattering in the case where the curved base geometry causes the valleys to widen up, and at the same time we can see that more regions are shadowed for this case.

Finally, Figures 12 and 13 show some more complicated examples. Figure 12 depicts a backyard scene in which every object except for the floor and the bin has been bump-mapped. This image took 16.2 minutes to render in a resolution of 640×400 pixels.

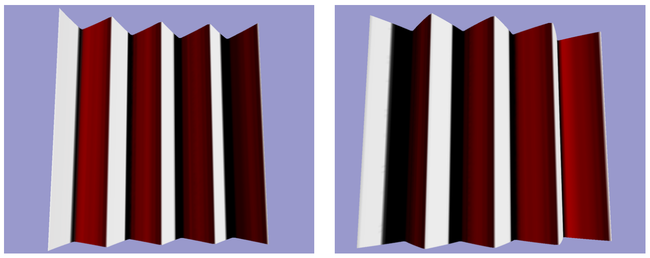


Figure 11: *Changes of indirect light and shadows as the curvature of the base geometry changes. Note that the red faces are exclusively illuminated indirectly via the light scattered from the white faces.*

The scene has been rendered in software with our methods for shadowing and scattering. Figure 13 shows an image of a terrain model rendered with hardware acceleration and bump shadows in real time (≈ 20 fps. on a GeForce 256).

7 Conclusion

In this paper, we have described an efficient method for illuminating height fields and bump maps based on precomputed visibility information. The algorithm simulates both self shadowing of the height field, as well as indirect illumination bouncing off height field facets. This allows us to use geometry, bump maps, and BRDFs as different levels of detail for a surface structure, and to consistently illuminate these three representations.

Using the Method of Dependent Tests, which is a generalization of Monte Carlo techniques, it is possible to map these methods onto graphics hardware. These techniques exploit the new bump mapping features of recent graphics boards, as well as dependent texture mapping, which is currently available only on some high-end systems, but will be a standard feature of the commodity hardware shipping at the end of the year.

Both the software and the hardware implementation of our algorithms can be used to efficiently precompute BRDFs and higher dimensional data structures such as BTFs or shift-variant BRDFs. Finally, we are also able to approximate the effects of different curvatures of the underlying base geometry, which to some extent change shadowing and light scattering in a height field, and therefore also affects representations like the BRDF. This is the first time in the literature that these effects are simulated.

There is a potential for extending the techniques described in this paper in several ways. First of all, we would like to be able to deal with other geometry than height fields, since materials such as cloth and porous materials cannot be represented in this form. In principle it should be easy to extend our algorithms to arbitrary geometry, however. In order to utilize graphics hardware, we have to find an appropriate 2D parameterization for the object so that all surface points can be represented in a texture. The next step could then be to extend the method to participating media, for example to simulate sub-surface scattering. It would also be interesting to explore whether it is useful to apply the developed methods to compute global illumination in macroscopic scenes along the lines of the transillumination method [29].

Finally, a hierarchical version of the Method of Dependent Tests has recently been introduced by Heinrich [11] and Keller [17]. Heinrich proved that this method is optimal for a certain class of functions fulfilling some smoothness criteria, and Keller extended this work to other classes of functions. It would be interesting to see if these results can be utilized to further improve the performance of our algorithm.

8 Acknowledgments

We would like to thank Alexander Keller for pointing out the relationship between our algorithms and the Method of Dependent Tests. Furthermore, we would like to thank him, Michael McCool, and the anonymous reviewers for their valuable comments.

References

[1] B. Becker and N. Max. Smooth transitions between bump rendering algorithms. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, pages 183–190, August 1993.

[2] J. Blinn. Simulation of wrinkled surfaces. In *Computer Graphics (SIGGRAPH '78 Proceedings)*, pages 286–292, August 1978.

[3] B. Cabral, N. Max, and R. Springmeyer. Bidirectional reflection functions from surface bump maps. In *Computer Graphics (SIGGRAPH '87 Proceedings)*, pages 273–281, July 1987.

[4] R. Cook, T. Porter, and L. Carpenter. Distributed ray tracing. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, pages 137–45, July 1984.

[5] K. Dana, B. van Ginneken, S. Nayar, and J. Koenderink. Reflectance and texture of real world surfaces. *ACM Transactions on Graphics*, 18(1):1–34, January 1999.

[6] P. Debevec. Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *Computer Graphics (SIGGRAPH '98 Proceedings)*, pages 189–198, July 1998.

[7] A. Frolov and N. Chentsov. On the calculation of certain integrals dependent on a parameter by the Monte Carlo method. *Zh. Vychisl. Mat. Fiz.*, 2(4):714 – 717, 1962. (in Russian).

[8] S. Gortler, R. Grzeszczuk, R. Szelinski, and M. Cohen. The Lumigraph. In *Computer Graphics (SIGGRAPH '96 Proceedings)*, pages 43–54, August 1996.

[9] P. Haeberli and K. Akeley. The accumulation buffer: Hardware support for high-quality rendering. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, pages 309–318, August 1990.

[10] W. Heidrich and H.-P. Seidel. Realistic, hardware-accelerated shading and lighting. In *Computer Graphics (SIGGRAPH '99 Proceedings)*, August 1999.

[11] S. Heinrich. Monte Carlo Complexity of Global Solution of Integral Equations. *Journal of Complexity*, 14:151–175, 1998.

[12] Silicon Graphics Inc. *Pixel Texture Extension*, December 1996. Specification document, available from http://www.opengl.org.

[13] J. Kajiya. The rendering equation. In *Computer Graphics (SIGGRAPH '86 Proceedings)*, pages 143–150, August 1986.

[14] J. Kautz, W. Heidrich, and K. Daubert. Bump map shadows for OpenGL rendering. Technical Report MPI-I-2000-4-001, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 2000.

[15] J. Kautz and M. McCool. Interactive rendering with arbitrary BRDFs using separable approximations. In *Rendering Techniques '99 (Proc. of Eurographics Workshop on Rendering)*, pages 247 – 260, June 1999.

[16] A. Keller. Instant radiosity. In *Computer Graphics (SIGGRAPH '97 Proceedings)*, pages 49–56, August 1997.

[17] A. Keller. Hierarchical monte carlo image synthesis. *Mathematics and Computers in Simulation*, 2000. preprint available from http://www.uni-kl.de/AG-Heinrich/Alex.html.

[18] M. Levoy and P. Hanrahan. Light field rendering. In *Computer Graphics (SIGGRAPH '96 Proceedings)*, pages 31–42, August 1996.

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

[20] L. Mostefaoui, J.-M. Dischler, and D. Ghazanfarpou. Rendering inhomogeneous surfaces with radiosity. In *Rendering Techniques '99 (Proc. of Eurographics Workshop on Rendering)*, pages 283–292, June 1999.

[21] F. Neyret. Modeling, animating, and rendering complex scenes using volumetric textures. *IEEE Transactions on Visualization and Computer Graphics*, 4(1), January – March 1998.

[22] NVIDIA Corporation. *NVIDIA OpenGL Extension Specifications*, October 1999. Available from http://www.nvidia.com.

[23] M. Peercy, J. Airey, and B. Cabral. Efficient bump mapping hardware. In *Computer Graphics (SIGGRAPH '97 Proceedings)*, pages 303–306, August 1997.

[24] M. Segal and K. Akeley. *The OpenGL Graphics System: A Specification (Version 1.2)*, 1998.

[25] I. Sobol. The use of ω^2-distribution for error estimation in the calculation of integrals by the monte carlo method. In *U.S.S.R. Computational Mathematics and Mathematical Physics*, pages 717–723, 1962.

[26] J. Stewart. Hierarchical visibility in terrains. In *Rendering Techniques '97 (Proc. of Eurographics Workshop on Rendering)*, pages 217–228, June 1997.

[27] J. 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, March 1998.

[28] J. Stewart. Computing visibility from folded surfaces. *Computers and Graphics*, 1999. preprint obtained from http://www.dgp.toronto.edu/people/JamesStewart/.

[29] L. Szirmay-Kalos, T. Fóris, L. Neumann, and B. Csébfalvi. An Analysis of Quasi-Monte Carlo Integration Applied to the Transillumination Radiosity Method. *Computer Graphics Forum (Proc. of Eurographics '97)*, 16(3):271–282, August 1997.

[30] R. Westermann and T. Ertl. Efficiently using graphics hardware in volume rendering applications. In *"Computer Graphics (SIGGRAPH '98 Proceedings)"*, pages 169–178, July 1998.

[31] S. Westin, J. Arvo, and K. Torrance. Predicting reflectance functions from complex surfaces. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, pages 255–264, July 1992.

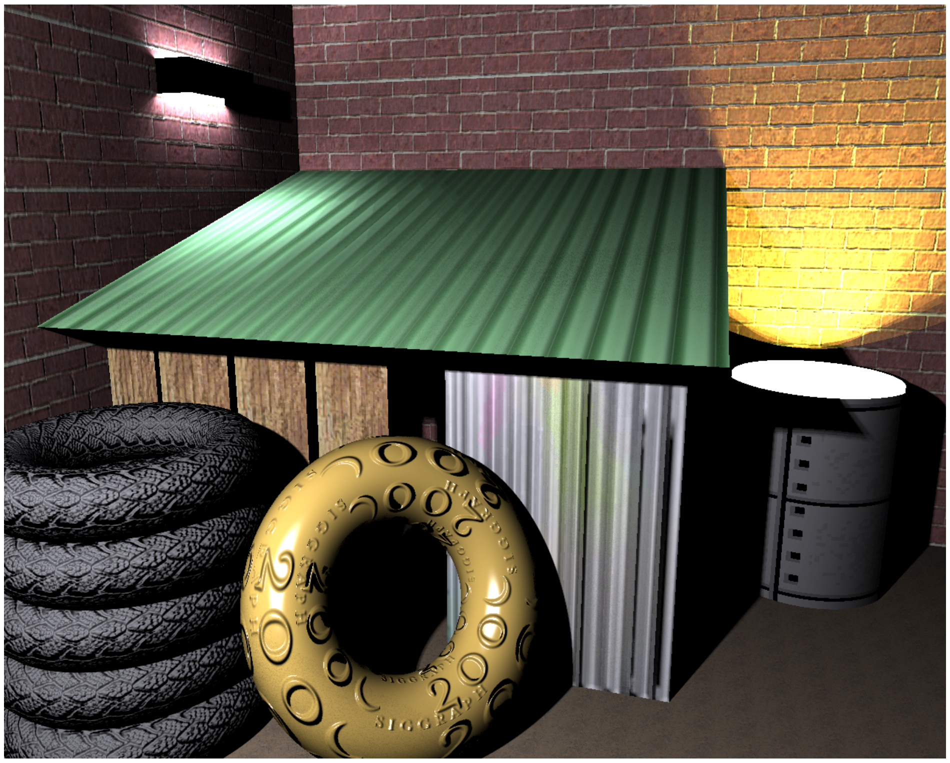


Figure 12: *A more complex scene where all surfaces are bump mapped, including shadowing and indirect light.*

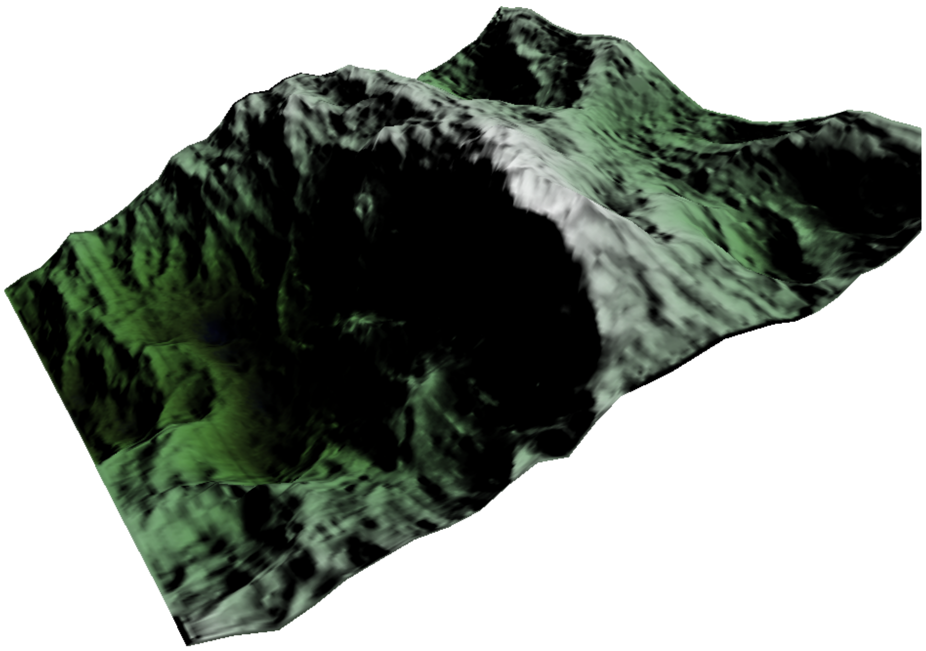


Figure 13: *A terrain model with bump map shadowing rendered in realtime with graphics hardware.*