Voxel-based Global Illumination

Sinje Thiedemann* Niklas Henrich* Thorsten Grosch† Stefan Müller*

*University of Koblenz-Landau, Germany †University of Magdeburg, Germany

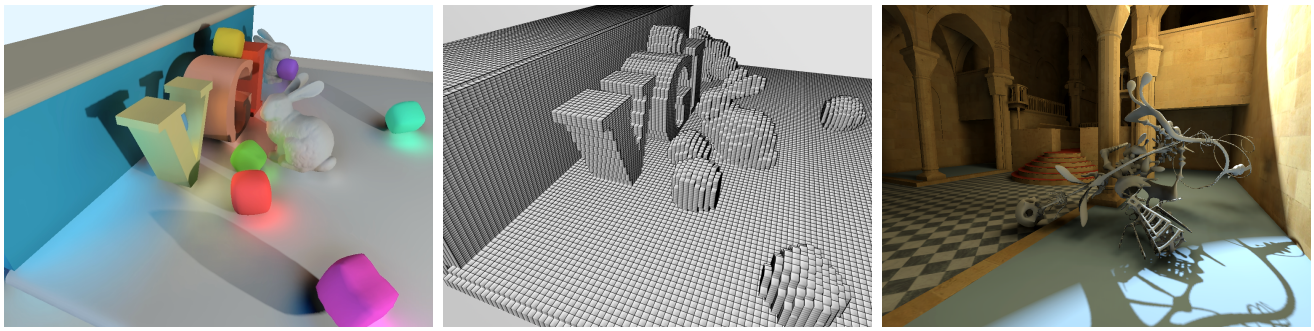

Figure 1: *Using voxel-based visibility (center), we are able to display real-time near-field illumination with directional occlusion (left, 25 fps) and interactive global illumination (right, 4.9 fps). The indirect light is exaggerated for visualization.*

Abstract

Computing a global illumination solution in real-time is still an open problem. We introduce *Voxel-based Global Illumination* (VGI), a scalable technique that ranges from real-time near-field illumination to interactive global illumination solutions. To obtain a voxelized scene representation, we introduce a new atlas-based boundary voxelization algorithm and an extension to a fast ray-voxel intersection test. Similar to screen-space illumination methods, VGI is independent of the scene complexity. Using voxels for indirect visibility enables real-time near-field illumination without the screen-space artifacts of alternative methods. Furthermore, VGI can be extended to interactive, multi-bounce global illumination solutions like path tracing and instant radiosity.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism; I.3.3 [Computer Graphics]: Color, Shading, Shadowing and Texture

Keywords: scene voxelization, global illumination, constant time

1 Introduction

Given a large and dynamic scene, computing global illumination in real-time is still a big problem. Using the original polygonal scene description to compute the light transport is often too time-consuming. As a simplification, many fast screen-space illumination algorithms were developed. These methods have limitations since they can only simulate what is visible in the camera image: If blockers or senders of indirect light become invisible in the camera image, the corresponding illumination effect vanishes. We therefore propose to use a *voxel-based* model to compute the indirect light. This proved to be a good compromise between the accurate, but slow polygons and the fast, but imprecise screen pixels. Using a voxel-based model has several advantages: This is a geometry-independent scene description and many fast voxelization methods were developed recently that allow fully dynamic scenes. Based on a new voxelization method and an improved ray-voxel intersection test, we show that voxel models can be used to simulate near-field illumination with occlusion in real-time. Global illumination simulations can be accelerated, since the low-frequency indirect light and visibility can be well approximated with a coarse voxel model.

We introduce the following contributions:

- A new atlas-based voxelization method and an improved ray-voxel intersection test
- Real-time near-field illumination with voxel visibility
- Interactive global illumination with voxel visibility

The paper is structured as follows: First, we review the related work in Sec. 2. We then describe our atlas-based voxelization method in Sec. 3 and the ray-voxel intersection test in Sec. 4. The illumination methods with voxel-based visibility are described in Sec. 5. In Sec. 6 we show our results before we conclude in Sec. 7.

2 Related Work

Voxelization Scene voxelization describes the process of turning a scene representation consisting of discrete geometric entities (e.g. triangles) into a three-dimensional regular spaced grid. Each cell of the grid encodes specific information about the scene. Depending on the type of voxelization, this information can be different. In the case of a binary voxelization, a cell stores whether geometry is present in this cell or not. The cells can be represented by single bits in a bitmask. In a multi-valued voxelization, the cell can also store information like material or normals. Furthermore, voxelization can be divided into boundary or solid voxelization. Boundary voxelization encodes the object surfaces only, whereas solid voxelization captures the interior of a model as well.

A voxelization method based on slicing was first presented by Fang et al. [2000]. The algorithm made use of multiple settings of the near and far plane of an orthogonal camera to capture differ-

I3D '11, 18-FEB-2011, San Francisco, CA, USA

ent slices of the scene geometry. Another method based on slicing was presented by Crane et al. [2007]. They used the geometry shader to intersect all triangles of the scene with each plane of the three dimensional grid to successively fill each layer. Using custom rasterizers, allowing a more flexible voxelization and thus arbitrary storage methods, was proposed by [Schwarz and Seidel 2010]. A fast binary scene voxelization method was presented by Eisemann et al. [2006]. It uses the RGBA-channels of a texture as a binary mask to encode the boundary of the scene geometry. The depth of a fragment is used as an indicator as to which bit in the mask has to be set. Dong et al. [2004] made the algorithm more robust for geometry which lies parallel to the viewing direction of the voxelization camera. Furthermore, Zhang et al. [2007] proposed to use a conservative rasterization approach to capture more details of the scene geometry. The original authors also proposed an extension for solid voxelization [Eisemann and Décoret 2008]. Voxelization methods based on depth-peeling were proposed by Passalis et al. [2007] and Li et al. [2005]. These methods have the advantage over slicing methods that they depend on the depth complexity of the scene, which in most cases is significantly lower than the number of layers in a volume. Our voxelization method replaces the peeling process by rendering texture atlases for the scene objects. Therefore it is independent of the scene's depth complexity and does not exhibit problems with polygons parallel to the voxelization direction.

Near-field illumination One way to achieve real-time global illumination is to restrict the light transport to the near-field around the receiver point. A simple method is Ambient Occlusion [Landis 2002], here an average visibility value is precomputed and stored on the surface. To extend this for large and dynamic scenes, several screen-space methods (SSAO) were developed [Mittring 2007; Shanmugam and Arikan 2007; Dimitrov et al. 2008]. This can be extended to directional occlusion (SSDO) [Ritschel et al. 2009], that computes directional shadows and bounces of indirect light in image space. Since only information from the deep framebuffer is used for occlusion calculation, all these methods show artifacts if the camera is moving and occluders or sources of indirect light become invisible in screen-space. Reinbothe et al. [2009] presented a voxel-based ambient occlusion method that overcomes these problems. Our approach further improves on this idea: We demonstrate how a voxel-based model can be used to display real-time, near field illumination, including directional occlusion and indirect light. Our method is independent of the camera position, so blockers or senders of indirect light need not be visible in screen-space. In a perceptual experiment, Yu et al. [2009] showed that visibility for indirect light can be approximated without a visible difference. This motivates our approach to use voxels for indirect visibility.

Real-time Global Illumination A real-time global illumination simulation is often only achieved by approximations, here the concept of the Irradiance Volume [Greger et al. 1998] attracted more research interest recently. Being known for their applicability to real-time rendering [Oat 2006] of static indirect lighting, Kaplanyan demonstrated their usage for dynamic indirect lighting [Kaplanyan 2009]. A view-dependent extension to this idea was presented in [Kaplanyan and Dachsbacher 2010]. Although the light propagation is very fast, it is done on a very coarse level which might miss important visual details. In contrast to this, our method is view-independent and works on a much finer resolution.

A whole new class of rendering algorithms based on virtual point lights (VPLs) were inspired by the ideas presented in the context of an Instant Radiosity simulation [Keller 1997]. Reflective Shadow Maps (RSMs) are able to capture the one-bounce indirect illumination of VPLs [Dachsbacher and Stamminger 2005; Dachsbacher and Stamminger 2006]. To be able to evaluate the visibility of VPLs at interactive rates, Ritschel et al. introduced the Imperfect Shadow Maps (ISMs) [2008]. Although the visibility test is very fast, ISMs have problems evaluating the near-field illumination correctly due to the large holes in the shadow map. Using voxels delivers the same accuracy at every distance.

To further reduce the computational effort, homogeneous regions in image-space are identified and the (indirect) illumination as well as the visibility computations are carried out only for representatives of these regions [Nichols and Wyman 2009; Nichols and Wyman 2010]. Recently, Nichols et al. [2010] also used a voxel-based visibility for direct illumination from large area lights.

Path Tracing, introduced by Kajiya [1986], gives an unbiased solution to the rendering equation, introduced in the same paper. Ray Tracing forms the basis of most Path Tracers. With the advent of SIMD instruction sets and multi-core / multi-thread CPUs, real-time Ray Tracing becomes feasible [Wald et al. 2007]. As the programming model of the GPU becomes more flexible, real-time Ray Tracing systems target the GPU as well [Parker et al. 2010]. Since the light that results from multiple diffuse indirect bounces has often a low-frequency nature, using a simplified, voxel-based model of the scene can be an alternative.

3 Binary and Multi-valued Boundary Voxelization

In this section, we present our new boundary voxelization method. It can be used as a *binary* voxelization, where the bits of the RGBA-channels of a 2-dimensional texture are used to encode the voxels, as done by [Eisemann and Décoret 2006]. It can also be used as a *multi-valued* voxelization, where the information is stored in a 3-dimensional texture (one texel per voxel). With the help of a multi-valued voxelization, any type of data (e.g. radiance for diffuse objects, normals or BRDF) can be stored in a voxel grid. The proposed voxelization method extracts the boundaries of the object. There are no restrictions to the objects (e.g. watertight). The method is applicable for dynamic rigid bodies and moderately deforming models as well. The only prerequisite is, that the objects can be stored in a texture atlas and that an appropriate mapping is already generated for the models.

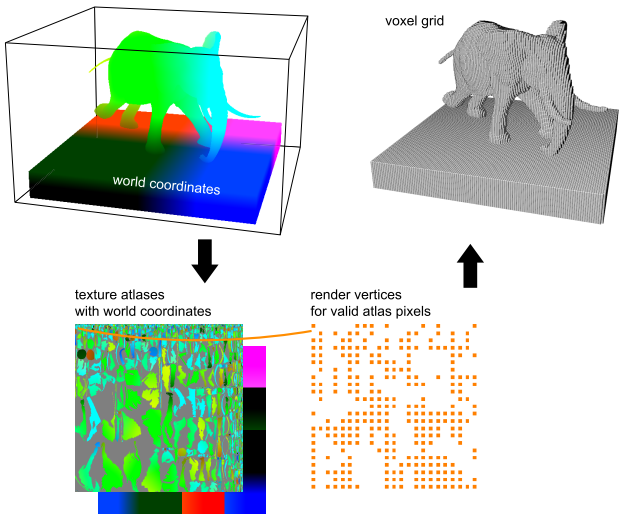

Figure 2: *Atlas-based voxelization: The scene is rendered to texture atlases capturing the world-position and additional data of the scene elements. For each non-empty texel of the texture atlases, a point is rendered and the data is inserted into the voxel grid.*

3.1 Algorithm

Our voxelization algorithm basically follows the same idea as voxelization via depth peeling. Peeling an object and saving its layers to textures before voxelization is replaced by rendering the complete object to a single atlas texture image in one rendering pass (see Figure 2). World-space positions have to be written to the atlas and during that process arbitrary other data can be stored as well. As most mappings from objects to a texture atlas leave unused space in the atlas texture, these unused texels have to be flagged as invalid. Afterwards, all valid entries in the texture atlas are inserted into the grid storing the voxel data using point rendering. As the grid has its own coordinate system, a voxelization camera is defined for that purpose. Please note that the position and orientation of this camera are completely arbitrary. The voxelization camera's frustum defines the region that is voxelized. A vertex is generated for each valid texel of the atlas texture. During a vertex texture fetch, the atlas data is read. The world-space position is used to transform the vertex into the coordinate system of the voxel grid. If a binary voxelization is used, a fragment shader creates a bitmask according to the depth of the point and the corresponding bit is set in the voxel grid (please see [Eisemann and Décoret 2006] for more details). If a multi-valued voxelization is used, the data must be inserted into a 3D-texture. A geometry shader is used to select the appropriate layer of the 3D-texture and the fragment shader outputs the desired data. As modern OpenGL implementations allow to have multiple 3D-textures attached as a render target, a variety of data can be inserted into multiple voxel-grids simultaneously.

3.2 Performance Considerations

Since we draw at least one vertex for each non-empty voxel, the performance is directly related to the number of rendered vertices. This in turn depends on the density of the voxelized model (ratio of full voxels to all voxels). The atlas resolution should therefore be chosen carefully: If we assume that one atlas texel always covers roughly the same surface area in world space, the ideal resolution is achieved if we get a one-to-one mapping from an atlas texel to a voxel position. As shown in Figure 3, using a lower atlas resolution can lead to holes between the voxels. If the atlas resolution is too high, we may end up with overdraw and the same voxel is filled multiple times. In our experiments we manually chose sufficient atlas resolutions.

Figure 3: *Atlas-voxelization of the elephant model (84K triangles) with different atlas resolutions. The resolution of the voxel-grid was set to* 128^3*. The two images on the left show the voxel-grid generated from an atlas with a resolution too low:* 96^2 *(4.4K points, 0.48 ms for atlas rendering and voxelization) and* 144^2 *(10K points, 0.5 ms). For the third result a sufficient atlas resolution of* 368^2 *(65K points, 0.69 ms) was used. In the rightmost image the resolution was set to an unnecessary high atlas resolution of* 1024^2 *(506K points, 2.38 ms).*

Statistics about our method are given in Table1 and 2. Our test system is specified in Sec. 6. As can be seen, our proposed voxelization approach is very fast and thus suitable for real-time applications.

Since many scenes consist of static and dynamic objects, we use a pre-voxelization of all the static parts. At run-time, we only voxelize dynamic objects in each frame and insert them into the static voxelization.

Voxel-grid resolution	Time (ms)	Vertices	Atlas resolution
$64^2 \times 128$	0.52	15k	176×176
$128^2 \times 128$	0.69	65k	368×368
$256^2 \times 128$	1.48	285k	768×768
$512^2 \times 128$	3.37	791k	1280×1280

Table 1: *Timings for a binary voxelization of the elephant model shown in Figure 3. As a binary voxelization encodes the voxels as bits in the RGBA-channels of a 2D-texture, the depth resolution is fixed to 128. The columns show from left to right: voxel grid resolution, time in ms to create the voxel-grid (including atlas rendering), number of vertices which were needed to insert the valid texels of the atlas into the grid and the resolution of the texture atlas.*

Voxel-grid resolution	Time (ms)	Vertices	Atlas resolution
$64^2 \times 128$	1.23	15k	176×176
$128^2 \times 128$	2.01	65k	368×368
$256^2 \times 128$	4.29	285k	768×768

Table 2: *Timings for a multi-valued voxelization of the elephant model. One voxel grid (3D texture) was filled with the direct illumination. The depth resolution is always set to 128 to match the binary voxel grid. Please note that the timings also involve computing the direct illumination for the texture atlas in a deferred shading step. The time in ms to create the voxel-grid (including atlas rendering) is shown in the second column. The third column states the number of vertices which were needed to insert the valid texels of the atlas into the grid. The resolution of the texture atlas is shown in the fourth column.*

3.3 Comparison

In comparison with other boundary voxelization algorithms, the atlas-based method avoids problems with polygons which are viewed from a grazing angle (cf. Figure 4). Alternative methods fill the resulting holes in multiple passes by voxelizing the scene from different viewing directions which are then combined into a single voxel grid. This is more time-consuming and assumes a cube as bounding volume of the scene, wasting lots of empty space. Depth-peeling voxelization approaches also need multiple render passes and depend on the depth complexity of the scene. In contrast, the atlas-based method generates a voxel model that tightly fits the bounding volume of the scene within two render passes: atlas rendering and voxelization (per object).

Solid voxelization (for example by Eisemann et al. [2008]) would be an alternative single-pass voxelization method. In general, solid voxelization requires watertight models and we did not want to limit ourselves to these.

A restriction of our method is that it does not allows strong deformations of the object. Strong deformations can corrupt the mapping from the object to the texture atlas and lead to holes in the voxel-grid. If the deformations are known in advance, one can use different atlas mappings for each stage of the deformation to prevent this. We did not observe this problem for the animated objects we used.

4 Hierarchical ray/voxel intersection test

For visibility computations, we use a ray-voxel intersection test that is based on the recently presented method by [Forest et al. 2009].

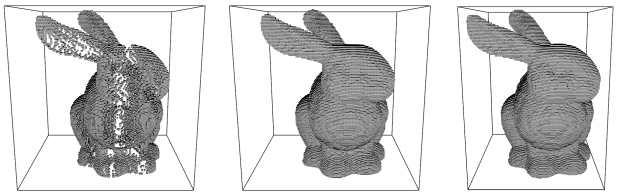

Figure 4: *Boundary voxelization comparison (voxel resolution 128^3) for the Stanford Bunny (70k triangles). Left: single-pass boundary voxelization by Eisemann et al. (0.43 ms), middle: 3-way boundary voxelization (2.53 ms), right: voxelization result with our method (0.51 ms atlas rendering + 0.36 ms voxelization = 0.87 ms, 97K points, 432^2 atlas). Please note that the combination of multiple voxelizations from different directions assumes a cube as bounding volume of the scene, wasting lots of empty space (middle image). Our method works with an arbitrary and thus tighter bounding box.*

Here, a binary voxelization is used which is stored in a 2D texture. Each bit of a texel encodes the presence of geometry at a certain depth. The texture is used to generate an octree-like structure, where each new node is generated with a logical OR operation of the child nodes' bitmasks. This way, the resolution of the depth stays the same at each level. A node effectively represents a *column* of voxels along the direction of 'depth'. The structure is traversed with a parametric traversal algorithm proposed in [Revelles et al. 2000]. The higher depth-resolution at each node allows it to decide more precisely, if the children of a node have to be traversed or not. This is done by intersecting the ray with the extents of the column of the active node. A bitmask, representing the voxels which are intersected by the ray, is generated. This bitmask is compared with the bitmask stored at the node to decide whether an intersection occurs and the node's children have to be traversed.

We propose to use a different traversal method, which does not need the overhead of an octree. In addition to determining the visibility between two points, we present a method to find the first intersection point along a ray as well.

4.1 Visibility between two points in space

Similar to [Forest et al. 2009] we use a binary voxelized scene representation and build a mip-map hierarchy on top of it, whereas the 'depth-direction' keeps its resolution on every mip-map level (cf. Figure 5). To traverse the hierarchy, we start at the coarsest

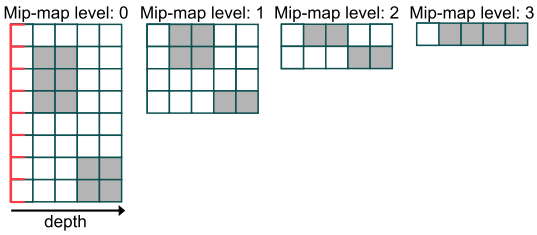

Figure 5: *Construction of the mip-map hierarchy in two dimensions. The depth resolution is identical for all mip-map levels. Each mip-map level is created with a logical OR-operation of the bitmask of the previous level.*

mip-map level and transform the ray into normalized device coordinates NDCs ($[0..1]^3$). The starting point of the ray is projected onto the mip-map texture and used to select the appropriate texel at the current mip-map level. The bounding box in NDCs corresponding to this texel is generated and the intersection with the ray is computed (please note that the intersection with the bounding box must be computed in [Forest et al. 2009] as well). Because one texel stores the full depth resolution, an adaptive bitmask, representing the voxels the ray intersects at the whole column, can be generated. This bitmask and the bitmask stored in the texel of the mip-map hierarchy are used to determine if an intersection occurs. If no intersection occurs, the starting point of the ray is advanced to the last intersection point with the bounding box of the column and the mip-map level is increased. If an intersection occurs, the mip-map level has to be decreased to check whether the intersection still occurs on a finer resolution of the voxelization until the finest resolution of the hierarchy (mip-map level 0) is reached. The algorithm stops if a hit is detected or the length of the remaining ray segment becomes zero. See Figure 6 for an example.

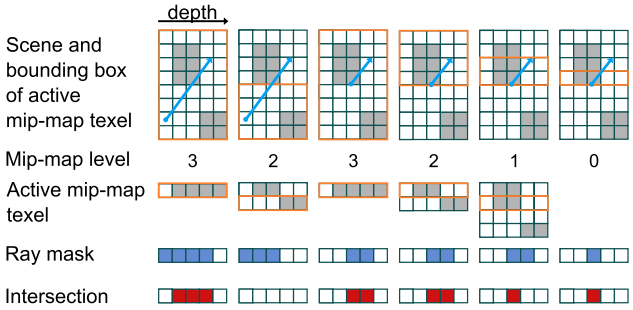

Figure 6: *Hierarchy traversal in 2 dimensions. The rows show from top to bottom: the current extent of the ray (blue) and the bounding box of the current mip-map level (orange), the current mip-map level, the bitmasks of the current mip-map level and the active texel (determined by projecting the start position of the ray onto the image plane), the bitmask of the intersection of the ray with the bounding box at the current mip-map level and finally the intersecting bits of the bitmask of the active texel and the adaptive ray mask.*

4.2 First intersection point along a ray

Up until now, the intersection test can only tell if there is an occluder anywhere in space between two points. We extend the algorithm to determine the position of the first intersection along the ray as well. This can be done by finding the first bit which is set in both the bitmask of the ray and the bitmask of the texel of the mip-map hierarchy. The position of the bit inside the mask can be used to determine its depth in NDCs. The depth can then be used to calculate the position of the intersection along the ray in world coordinates. To find the foremost bit, a logarithm to the base 2 can be used. If the ray direction is opposite to the direction of the voxelization, the position of the last bit has to be found. This can be done by extracting the last bit first (for example with the operation `x = x xor (x-1)`) prior to determining its position.

5 Illumination Methods

The combination of our voxelized scene representation with the optimized intersection test allows us to compute the indirect illumination in several different ways. The goal is to compute the outgoing radiance L_o in direction ω_o at a point $\mathbf{x}$

$$L_o(\mathbf{x}, \omega_o) = \int_\Omega f(\mathbf{x}, \omega_o, \omega_i)\, L_i(\mathbf{x}, \omega_i)\, \cos\theta\, d\omega_i \qquad (1)$$

where $f(\mathbf{x}, \omega_o, \omega_i)$ is the BRDF at $\mathbf{x}$ for the incoming direction ω_i and outgoing direction ω_o. The angle between the receiver normal and ω_i is denoted θ. Using Monte-Carlo Integration, L_o can be approximated as:

$$L_o(\mathbf{x}, \omega_o) \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f_r(\mathbf{x}, \omega_i, \omega_o) \tilde{L}_i(\mathbf{x}, \omega_i) \cos\theta}{p(\omega_i)} \qquad (2)$$

where $p(\omega_i)$ is an arbitrary probability density function and ω_i are N sample directions, generated from p. $\tilde{L}_i$ is the incoming radiance that we approximate based on a voxel model.

As the proposed intersection test effectively traverses the voxelized scene representation along the ray, the time to find the foremost intersection point largely depends on the length of the ray. Keeping the ray-length short allows us to compute the near-field single bounce indirect illumination in real-time while still evaluating the visibility of the indirect light. If we do not limit the extent of the ray, we are also able to evaluate the visibility of distant virtual points lights (VPLs), capturing the indirect illumination of the whole scene. Furthermore, with our multi-valued voxelization approach we realized a path tracing system which is capable of computing multiple bounces of indirect lighting accurately and very fast. Our method works for diffuse as well as glossy BRDFs.

All approaches handle the voxelization of the scene in the same way. At startup, the static scene elements are voxelized. The scene bounding box defines the region to be voxelized. For each frame, only the dynamic scene elements need to be voxelized by inserting their voxels into a copy of the grid storing the static elements. This improves performance for dynamic scenes with large amounts of static geometry.

5.1 Real-Time Near-Field Single Bounce Indirect Light

For a fast near-field illumination, we generate a reflective shadow map (RSM) [Dachsbacher and Stamminger 2005] that contains *direct light*, *position* and *normal* for each pixel. We render shadow maps for spot lights and cube maps for point lights. To obtain the indirect light for a pixel in the camera image, we use a *gathering* approach to compute one-bounce near-field illumination. To solve Eq. 2, we compute N rays starting from the receiver position $\mathbf{x}$ with a maximum distance r, as shown in Figure 7. We noticed that splitting up the computation into multiple passes with one pass per ray increases rendering speed. For each ray, we use our proposed intersection test to determine the first intersection point. If we hit a voxel along the ray we need to determine the direct radiance $\tilde{L}_i$ at this point. The naive solution would be multi-valued voxels where $\tilde{L}_i$ is written to atlases and stored at each voxel. However, this can be implemented more efficiently – especially less memory consuming – by a *back-projection* of the hit point into the RSM which allows us to read the direct radiance stored in the RSM pixel. The direct radiance is valid if the distance between the 3D position of the hit point and the position stored in the RSM pixel is smaller than a threshold ϵ. Otherwise the direct radiance is set to zero because the hit point is in shadow of the light source. The threshold ϵ has to be adjusted to the voxelization discretization v, the RSM pixel size s, the perspective projection, and the normal orientation α. As shown in Figure 7, this leads to $\epsilon = max(v, \frac{s}{cos\alpha} \cdot \frac{z}{z_{near}})$.

This technique allows the computation of one-bounce indirect light within a radius of r – keeping this value small leads to real-time frame rates. In contrast to image-space approaches, the voxel-space method does not depend on the camera position: Senders and blockers which are invisible in the camera image are always detected. As we store the information about the receivers of the direct illumination in the RSM, we do not rely on a multi-valued voxelization and can use our fast binary voxelization instead. Similar to many previous methods, we always compute the indirect light on a lower resolution, with interleaved sampling and a geometry-aware blur. We found that the voxel discretization and particularly the blur prevent glossy materials, so we only use diffuse BRDFs and generate the rays from a cosine density function. Although the accuracy of the reprojection decreases in distant regions of the light source (due to the perspective projection of the RSM), we found the resulting error acceptable because of the quadratic fall-off of the light source.

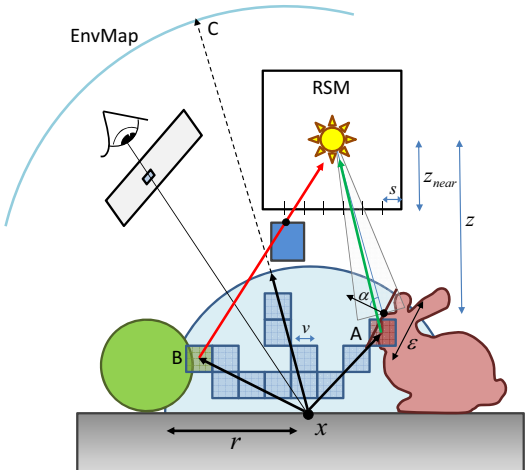

Figure 7: *Near-field indirect light: To compute the indirect light at a point* $\mathbf{x}$*, several rays are shot in the upper hemisphere. The voxels along each ray are traversed until a hit point is found. The hit point is then back-projected into the RSM to obtain the direct radiance at this point. Indirect light is gathered from hit point A, because it is visible from the light source: The distance from hit point A to the RSM pixel position is smaller than the threshold* ϵ*. Hit point B is in shadow of the direct light, because its position is different from the position stored in the RSM pixel. If no intersection point is found within the search radius r, the radiance can optionally be read from an environment map, to simulate directional occlusion (point C).*

5.2 Voxel Path Tracing

Extending the approach described in Sec. 5.1, a fast Path Tracer based on a voxelized scene representation can be built. This way, multiple diffuse and glossy bounces are possible. Typically, Eq. 1 is split into the sum of the direct $L_d(\mathbf{x}, \omega_o)$ and indirect light $L_{id}(\mathbf{x}, \omega_o)$ emitted from $\mathbf{x}$ into direction ω_o. Both integrals are solved with the help of a Monte Carlo approach (Eq. 2). The direct light is computed with deferred shading. The indirect illumination is computed recursively: a single direction $\omega_s \in \Omega = 2\pi$ is chosen using importance sampling of the BRDF $f(\mathbf{x}, \omega_o, \omega_i)$ of the respective surface element $\mathbf{x}$. A ray is cast into that direction and at the hit point $\mathbf{y}$, Eq. 1 is evaluated again to determine the radiance $L_o(y, -\omega_s)$. This is typically repeated until a given length of the path is reached. This recursive approach can not be translated to a GPU Path Tracer directly. Instead, an iterative solution must be used. Our Voxel Path Tracer traces paths up to a certain length l by creating textures $t_1, .., t_l$ for each possible hit point along the path. In the case of diffuse surfaces, the direct radiance L_d at the hit point as well as the BRDF are stored. If all textures are filled, the light is transferred from texture t_l to t_{l-1}, then from t_{l-1} to t_{l-2}, and so on until the final radiance value arrives at t_1.

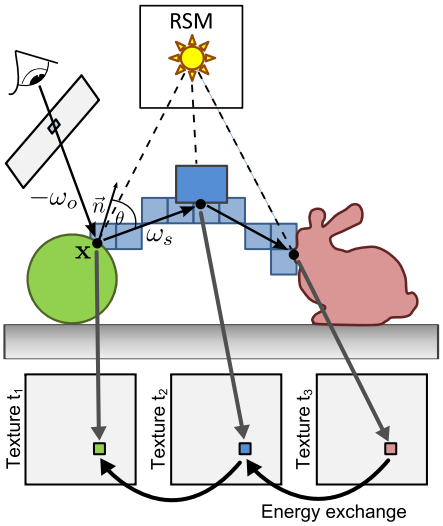

Figure 8: *Iterative Path Tracing. A ray is traced through the scene and at each hit position, the direct light, as well as normal and BRDF are stored in a texture. After the ray tracing has stopped, the energy is propagated backwards along the ray to the image plane.*

In the following, we will describe the approach in more detail. The scene is rendered into a deep framebuffer with position, normal and BRDF. For each pixel of the deep framebuffer, importance sampling of the BRDF is performed and a ray is shot into the hemisphere. In contrast to a one-bounce approach, the binary voxelization and the RSMs are not sufficient to trace the ray further, as it may hit geometry which is not covered in the RSMs. We need a multi-valued voxelization, storing the BRDFs (e. g. specular coefficient and the Phong exponent for glossy materials) and normals for generating random directions in the hemisphere above a hit position. The directly illuminated scene elements are captured in RSMs, storing their respective BRDF (including Phong exponents for glossy material), normal, position and radiant intensity for diffuse materials. If a hit point is found, the direct lighting is estimated and written to a texture. This is done by reprojecting the hit position into the RSMs (see Sec. 5.1). For diffuse materials, the direct radiance is read directly from the RSMs; for glossy surfaces it is evaluated on the fly. The voxel-grids are used to fetch the normal and BRDF. The latter and the hit position are also stored in textures. The hit position is used as the new starting point for the next ray and the process is repeated again. After the whole path is traced through the scene, the information stored in the textures can be used to propagate the energy backwards along the ray, until the final radiance value at the image plane is computed. A progressive rendering scheme is used to successively trace more rays per pixel.

Results show that images produced by a voxel path tracing approach may visually compete with such created by a conventional path tracer.

5.3 VPL-based Methods for Indirect Light

The proposed intersection test can also be used to evaluate the visibility of virtual point lights (Instant Radiosity). For a single bounce, only a binary voxelization of the scene is needed. In a first step, the VPLs are scattered in the scene using the approach of [Dachsbacher and Stamminger 2005]. To compute the radiance for a pixel in the camera image, we gather from the VPLs and use our voxel-based visibility.

This method can be extended to compute multiple bounces of VPLs as well. After the first-bounce VPLs are identified, the position and normal of each VPL is used to trace a ray into the scene (see Sec. 4.2) using an importance sampling of the BRDF. As the hit point might be anywhere in the scene (not visible in the RSM), a multi-valued voxelization of the scene, storing normals and BRDFs is needed. At the hit position, a VPL is generated and stored in an additional texture. During rendering the multiple-bounce VPLs are evaluated as well to determine the radiance arriving at a pixel in the camera image.

6 Results

The voxel-based visibility allows interactive to real-time illumination from the near-field around a receiver point. Figure 9 shows examples for near-field illumination in dynamic scenes. The relevant components of our test system are an NVIDIA GeForce GTX 295 and an Intel Core2Duo 3.16 GHz with 4 GB RAM. All important steps of our methods are running on the GPU. Our implementation uses OpenGL and the GL shading language. The radius r of the near-field illumination is a time-quality parameter: Figure 10 displays the different appearances of a scene when changing the radius from small to large. In comparison to screen-space approaches, VGI does not depend on the content of the deep framebuffer of the viewer camera. As shown in Figure 11, VGI displays indirect light and directional occlusion without the typical screen-space artifacts. Due to the voxel discretization, care must be taken to avoid wrong self-occlusions, as can be seen in Figure 12. Global Illumination examples are shown in Figure 13, here a path tracing solution is shown which is based on voxel visibility. This scene contains diffuse as well as glossy materials. The obtained results are similar to a conventional path tracer, as shown in Figure 14. The overall radiance distribution is maintained, only thin structures lead to over-darkening in indirect shadows due to voxelization. The image in Figure 1 (right) is an example of a two-bounce VPL-based illumination solution with voxel visibility. Figure 15 offers a detailed timing breakdown of the various steps of our near-field method.

7 Conclusion and Future Work

We presented voxel-based global illumination (VGI). Using a voxel-based visibility for indirect light allows real-time near-field illumination and interactive global illumination computation. Furthermore, we introduced a new atlas-based voxelization method and an improved ray-voxel intersection test. As future work we would like to extend our method to large voxel models (e.g. Giga Voxels [Crassin et al. 2009]) and adjust the voxelization to the camera position. Currently, glossy materials are still an open problem since the discretized geometry can become visible in the glossy reflection. For large and animated scenes, the inclusion of a temporal filter should be further investigated to reduce noise and to improve the slight flickering which is sometimes visible if the voxel model changes due to moving geometry. Another interesting direction is volumetric illumination [Ropinski et al. 2008] based on a voxel model.

References

CRANE, K., LLAMAS, I., AND TARIQ, S. 2007. *GPU Gems 3: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison-Wesley Professional, ch. 30 Real-Time Simulation and Rendering of 3D Fluids, 633–675.

CRASSIN, C., NEYRET, F., LEFEBVRE, S., AND EISEMANN, E. 2009. Gigavoxels : Ray-guided streaming for efficient and de-

Figure 9: *Examples of near-field illumination with voxel visibility: The top-left image shows indirect light reflected to animated hands (20 fps). Note how the green indirect light is correctly occluded on the thumb of the right hand. The top-right image shows (exaggerated) directional occlusion from an environment map and indirect light (25 fps). In the bottom row, a larger scene is used with dynamic objects (18 fps). Here, the radius for the indirect light is bigger, so colored light is reflected both to the ground and to the animals. Image resolution is* 1024×768*. Please see the accompanying video for animations.*

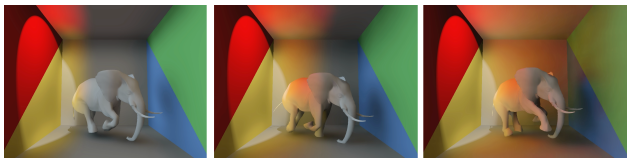

Figure 10: *Voxel-based single bounce illumination with different radii r. Frame rates from left to right: 30 fps, 27.7 fps, 25 fps.*

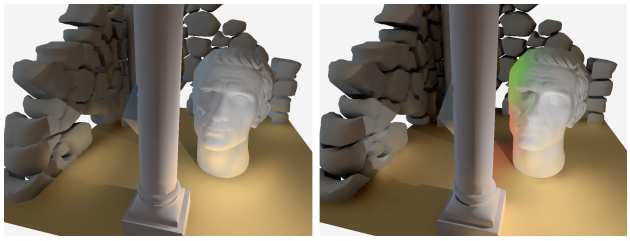

Figure 11: *Left: Screen Space Directional Occlusion and one indirect bounce. Right: Voxel-based Global Illumination. Indirect light is reflected from an object hidden behind the column. Note that SSDO does not display the indirect light from senders not visible in the viewing range.*

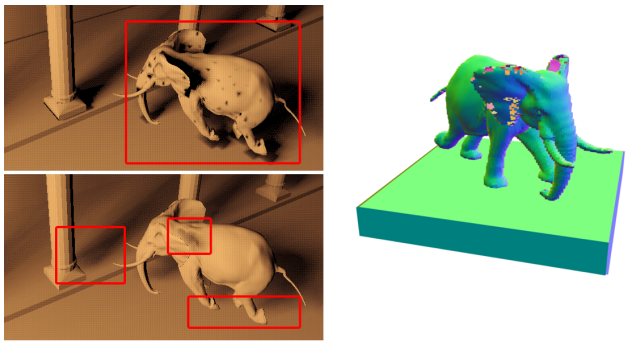

Figure 12: *The start position on the traversed ray has to be advanced by an offset to avoid self intersections (left top). Choosing the offset too large leads to missed objects and floating objects (left bottom). The right image shows a voxelization of surface normals: If two planes fall into the same voxel-cell, the multi-valued voxelization process is unable to capture the information of both. So normals from the backfaces are inserted in some cases.*

tailed voxel rendering. In *I3D '09: Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, ACM.

DACHSBACHER, C., AND STAMMINGER, M. 2005. Reflective shadow maps. In *I3D '05: Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, ACM.

DACHSBACHER, C., AND STAMMINGER, M. 2006. Splatting indirect illumination. In *I3D '06: Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*, ACM.

DIMITROV, R., BAVOIL, L., AND SAINZ, M. 2008. Horizon-split ambient occlusion. In *I3D '08: Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games*, ACM.

DONG, Z., CHEN, W., BAO, H., ZHANG, H., AND PENG, Q. 2004. Real-time voxelization for complex polygonal models. In *Pacific Conference on Computer Graphics and Applications*, IEEE Computer Society, Washington, DC, USA, 43–50.

EISEMANN, E., AND DÉCORET, X. 2006. Fast scene voxelization and applications. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM, 71–78.

EISEMANN, E., AND DÉCORET, X. 2008. Single-pass gpu solid voxelization and applications. In *GI '08: Proceedings of Graphics Interface 2008*, vol. 322, 73–80.

FANG, S., AND CHEN, H. 2000. Hardware accelerated voxelization. *Computers and Graphics 24*.

FOREST, V., BARTHE, L., AND PAULIN, M. 2009. Real-time hierarchical binary-scene voxelization. *Journal of Graphics, GPU, & Game Tools 14*, 21–34.

GREGER, G., SHIRLEY, P., HUBBARD, P. M., AND GREENBERG, D. P. 1998. The irradiance volume. *IEEE Computer Graphics and Applications 18*, 32–43.

KAJIYA, J. T. 1986. The rendering equation. *SIGGRAPH Computer Graphics 20*, 4, 143–150.

KAPLANYAN, A., AND DACHSBACHER, C. 2010. Cascaded light propagation volumes for real-time indirect illumination. In *I3D '10: Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM.

KAPLANYAN, A., 2009. Light propagation volumes in cryengine 3. ACM SIGGRAPH 2009 Courses – Advances in Real-Time Rendering in 3D Graphics and Games Course.

KELLER, A. 1997. Instant radiosity. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press.

LANDIS, H., 2002. Production-ready global illumination. ACM SIGGRAPH Course Notes, Course 16 (RenderMan in Production), July.

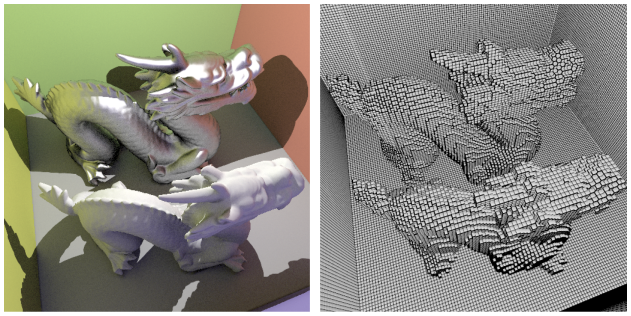

Figure 13: *Path Tracing with voxel-based visibility (32 directions per pixel, 1 bounce, diffuse and glossy materials, 3.5 fps). The voxel-grid has a resolution of* 128^3 *and is shown on the right.*

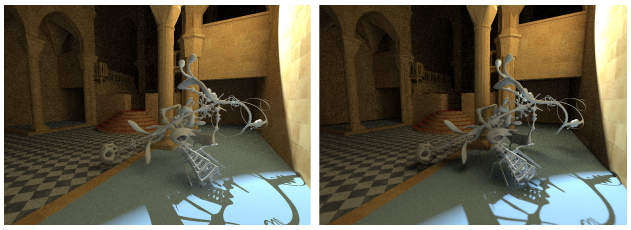

Figure 14: *Path Tracing with 64 samples per pixel, two interreflections. Left: Conventional path tracing (28 minutes). Right: Path Tracing with voxel-based visibility (2.2 seconds). The overall indirect light distribution and the indirect shadows of large objects remain correct. For thin structures like the object in the foreground, the coarse voxelization (*$256^2 \times 128$*) results in over-darkening of the indirect shadow.*

Figure 15: *Sibenik cathedral with animated horse, 28 fps. Timing breakdown of our method (2 spot lights/RSMs,* 128^3 *voxel-grid resolution, 20 sampled directions per fragment, indirect light rendered at* $1/4 \times 1/4$ *image resolution and upscaled to full resolution of* 1024×768*): atlas-voxelization: 0.27 ms, fill g-buffer (+deferred lighting): 5 ms, RSM rendering: 6.9 ms, indirect illumination: 13.6 ms, spatial upsampling filter: 7.7 ms. Rendering the indirect light at* $1/2 \times 1/2$ *image resolution increases the indirect illumination computation time to 36 ms; a full resolution computation takes 123 ms. The total GPU memory consumption of this scene is 350 MB.*

LI, W., FAN, Z., WEI, X., AND KAUFMAN, A. 2005. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison-Wesley Professional, ch. 47 Flow Simulation with Complex Boundaries.

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

NICHOLS, G., AND WYMAN, C. 2009. Multiresolution splatting for indirect illumination. In *I3D '09: Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, ACM.

NICHOLS, G., AND WYMAN, C. 2010. Interactive indirect illumination using adaptive multiresolution splatting. *IEEE Transactions on Visualization and Computer Graphics 16*, 729–741.

NICHOLS, G., PENMATSA, R., AND WYMAN, C. 2010. Interactive, multiresolution image-space rendering for dynamic area lighting. *Eurographics Symposium on Rendering 29*, 4.

OAT, C. 2006. Irradiance volumes for real-time rendering. In *ShaderX5: Advanced Rendering Techniques*, W. Engel, Ed. Charles River Media.

PARKER, S. G., BIGLER, J., DIETRICH, A., FRIEDRICH, H., HOBEROCK, J., LUEBKE, D., MCALLISTER, D., MCGUIRE, M., MORLEY, K., ROBISON, A., AND STICH, M. 2010. Optix: a general purpose ray tracing engine. *ACM Trans. Graph. 29*, 4, 1–13.

PASSALIS, G., THEOHARIS, T., TODERICI, G., AND KAKADIARIS, I. A. 2007. General voxelization algorithm with scalable gpu implementation. *Journal of Graphics, GPU, & Game Tools 12*, 61–71.

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

REVELLES, J., UREÑA, C., AND LASTRA, M. 2000. An efficient parametric algorithm for octree traversal. In *Journal of WSCG*, 212–219.

RITSCHEL, T., GROSCH, T., KIM, M. H., SEIDEL, H.-P., DACHSBACHER, C., AND KAUTZ, J. 2008. Imperfect shadow maps for efficient computation of indirect illumination. *ACM Trans. Graph. (Proc. of SIGGRAPH ASIA 2008) 27*.

RITSCHEL, T., GROSCH, T., AND SEIDEL, H.-P. 2009. Approximating dynamic global illumination in image space. In *I3D '09: Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, ACM, 75–82.

ROPINSKI, T., MEYER-SPRADOW, J., DIEPENBROCK, S., MENSMANN, J., AND HINRICHS, K. H. 2008. Interactive volume rendering with dynamic ambient occlusion and color bleeding. *Eurographics 2008 27*, 2, 567–576.

SCHWARZ, M., AND SEIDEL, H.-P. 2010. Fast parallel surface and solid voxelization on gpus. *ACM Trans. Graph. 29*, 179:1–179:10.

SHANMUGAM, P., AND ARIKAN, O. 2007. Hardware accelerated ambient occlusion techniques on gpus. In *I3D '07: Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, ACM, 73–80.

WALD, I., MARK, W. R., GÜNTHER, J., BOULOS, S., IZE, T., HUNT, W., PARKER, S. G., AND SHIRLEY, P. 2007. State of the Art in Ray Tracing Animated Scenes. In *Eurographics 2007 State of the Art Reports*.

YU, I., COX, A., KIM, M. H., RITSCHEL, T., GROSCH, T., DACHSBACHER, C., AND KAUTZ, J. 2009. Perceptual influence of approximate visibility in indirect illumination. *TAP 6*, 4.

ZHANG, L., CHEN, W., EBERT, D. S., AND PENG, Q. 2007. Conservative voxelization. *The Visual Computer 23*, 783–792.