Interactive Global Illumination Using Implicit Visibility

Zhao Dong
MPI Informatik

Jan Kautz
University College London

Christian Theobalt
Stanford University

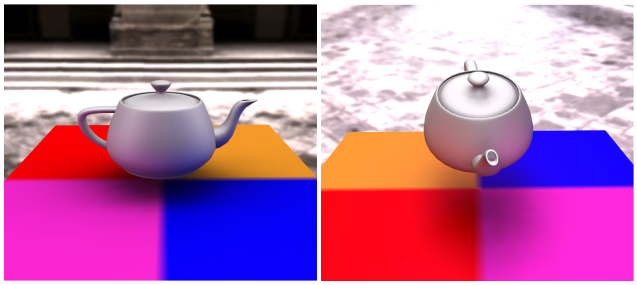
Hans-Peter Seidel
MPI Informatik

Abstract

Rendering global illumination effects for dynamic scenes at interactive frame rates is a computationally challenging task. Much of the computation time needed is spent during visibility queries between individual scene elements, and it is almost illusive to update this information at real-time even for moderately complex scenes. In this paper, we propose a global illumination approach for dynamic scenes that runs at near-real-time frame rates on a single PC. Our method is inspired by the principles of hierarchical radiosity and tackles the visibility problem by implicitly evaluating mutual visibility while constructing a hierarchical link structure between scene elements. By means of the same efficient and easy-to-implement framework, we are able to reproduce a large variety of complex lighting effects for moderately sized scenes, such as interreflections, environment map lighting as well as area light sources.

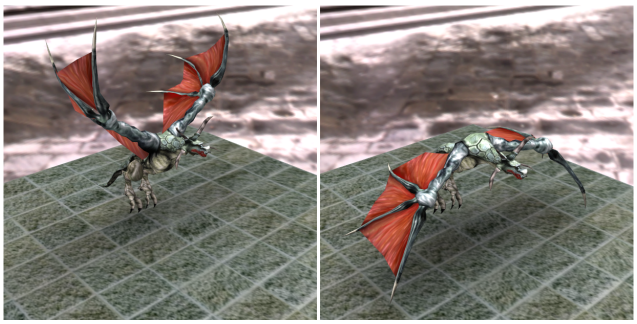
1 Introduction

In order to render a scene photo-realistically many local and global illumination effects have to be faithfully reproduced. Today, real-time rendering of local illumination effects is state-of-the-art and used in many computer games and interactive environments. Unfortunately, scenes rendered in this way often have an artificial look as they lack more sophisticated appearance details such as interreflections. Global illumination computation adds this additional bit of realism by taking into account not only light that comes directly from the light source but also indirectly through reflection from other surfaces. However, the simulation of global illumination effects is very complex and up to now it has been illusive to render full global illumination solutions in real-time on a single PC. The problem's complexity originates from the fact that during lighting simulation every scene element interacts with many others. Furthermore, visibility between scene elements has to be precomputed, as light can only travel between mutually visible points in the scene. This expensive-to-compute information is used in all traditional rendering algorithms [4, 22, 17, 35].



(a) One Bounce, 6x12x12, 6.43FPS (b) One Bounce, 6x12x12, 6.43FPS

Figure 1. Teapot with indirect lighting (3878 vertices).



(a) One Bounce, 6x12x12, 7.53FPS (b) One Bounce, 6x12x12, 7.53FPS

Figure 2. Flying dragon (deformable model, 2670 vertices).

In this paper, we propose a novel algorithm to render global-illumination effects at interactive frame rates on a single PC. The core of our method is a hierarchical radiosity-like link structure describing the light transport between individual scene elements. To overcome the computational bottleneck of having to compute visibility information explicitly at each frame, we propose the concept of *implicit visibility*. By this means, we are able to quickly derive visibility between scene elements implicitly from the hierarchical link structure while it is being built. We propose methods to efficiently construct this link structure and show that the final global illumination solution can

be quickly computed on the GPU (Figure 1). Our method can reproduce interreflections under environment map lighting as well as area light sources at interactive frame rates — even for dynamic scenes with deformable objects. Interactivity is achieved by sparsely sampling visibility, which makes our method most suited for diffuse or low-glossy scenes under large area lighting.

The paper proceeds with a review of relevant related work in Sect. 2. Thereafter, Sect. 3 describes how we reformulate and solve the rendering equation to accommodate the concept of implicit visibility. Sect. 4 describes in detail the construction and administration of our hierarchical link structure, and explains how to efficiently map these concepts onto the GPU. We demonstrate the high visual quality of our results in Sect. 5 and conclude in Sect. 6 with an outlook to future work.

2 Related Work

In the following we review existing techniques and contrast them with our new method.

Radiosity Radiosity [4] is a finite element method to compute a global illumination solution, where links between mutually visible finite elements are created and radiosity is propagated along those links until a steady state is reached. This computation is generally not real-time, even with improvements such as hierarchical radiosity [12]. There exist approaches to handle and make use of temporal coherence in dynamic scenes; a global illumination solution is incrementally updated by shooting negative light to compensate for changes in lighting or geometry [3, 11, 28]. However, updating the link structure is difficult, since visibility needs to be taken into account.

Ray-Tracing The most common method to compute a global illumination solution is the use of ray tracing. Rays of light are followed through the scene until all their energy is deposited in the scene (or a certain recursion depth is reached) [39, 31]. Variants of the original ray-tracing approach include path tracing and photon mapping [22, 17]. All algorithms have in common that rays or photons need to be intersected with the geometry to find the closest hit points. This is a rather costly operation and has long prevented ray-tracing approaches from being interactive Recently, algorithms have been proposed for interactive global illumination using ray tracing [38, 36]. However, a cluster of 24 PCs was required to achieve interactivity.

GPU-Based Approximate Global Illumination The first approaches to use graphics hardware for global illumination were limited to direct illumination with environment maps [14, 20]. I.e., incident illumination could be more complex than simple point sources, however, no indirect illumination was supported. The first approaches to perform near-real-time, indirect illumination on GPUs was limited to static micro-geometry [13].

A coarse approximation of full global illumination can be achieved with ambient occlusion [41], which simply dims incident lighting according to the average visibility at a point. A technique to support ambient occlusion for dynamic models re-used the idea of negative light [2].

The photon tracing technique by Keller [21] regards photons stored at hit points in the scene as secondary light sources. Interactive frame rates can be achieved with this technique but banding artifacts are likely to appear. GPU implementations for radiosity, photon mapping and radiance caching exist as well [5, 29, 23, 10]. Unfortunately, expensive visibility computation prohibits real-time performance of these GPU-based global illumination methods.

One-bounce indirect illumination can be rendered at real-time rates, if no visibility is taken into account [6, 7]. However, this allows light to bleed through surfaces, creating unrealistic results. Several bounces of indirect illumination [27] can be taken into account by iteratively collecting incident lighting. However, real-time rates can only be achieved with very coarse lighting approximations.

Dachsbacher et al. [8] concurrently developed a real-time global illumination method that handles visibility by transferring *anti-radiance*. While this bears many similarities to our algorithm, our implicit visibility handling is slightly more flexible, as we impose no restrictions on the dynamics of the scene, but is also slightly more expensive.

Precomputed Radiance Transfer PRT permits real-time rendering of limited global illumination effects on static objects, such as shadows and diffuse/glossy interreflections [32, 26, 25]. The global illumination solution is simply parameterized by the incident lighting, which is assumed to be represented by means of basis functions, such as spherical harmonics [32] or wavelets [26]. PRT exploits the limitation to static objects by precomputing all the visibility queries and baking them into the parameterized solution.

Dynamic or deformable objects are inherently difficult for PRT techniques, since the visibility cannot be precomputed anymore. Direct illumination on deformable, low-polygonal models can be rendered with a PRT-like technique [19, 30], however interreflections cannot be reproduced in this setting. Similarly, limited dynamic scenes with moving rigid objects can be handled [40, 34], but also without taking indirect illumination into account. Recent work [24, 16] extends these ideas to render interreflections of dynamic rigid objects. Handling deformable models remains a challenge.

In this paper, we present a full global illumination algorithm that circumvents expensive visibility queries by evaluating visibility implicitly during construction of a hierarchical link structure. This rapid visibility approximation

enables rendering of moderately complex, dynamic scenes with deformable models at near real-time frame rates on a single PC. In contrast to previous methods, our approach can handle environment map lighting as well as area light sources.

3 Global Illumination using Implicit Visibility

In the following, we derive the theoretical fundamentals of fast interactive global illumination based on implicit visibility. We start with the rendering equation [18] and rewrite it in such a way that a global illumination solution can be computed in a way similar to early non-diffuse radiosity methods [15]. In contrast to radiosity methods, however, we compute visibility implicitly while building the link structure. The rendering equation can be written as follows:

$$L(x \rightarrow \omega_o) \;=\; L_e(x \rightarrow \omega_o) + \int\limits_{\Omega_x^+} f_r(x, \omega_i' \leftrightarrow \omega_o') \cdot L(x \leftarrow \omega_i) \cdot (\mathbf{n}_x \cdot \omega_i) d\omega_i, \tag{1}$$

where x is a point in the scene, L_e is the emitted light, f_r is the BRDF, $\mathbf{n}_x$ is the normal at x, ω_i and ω_o are the global light and viewing directions, and ω_i' and ω_o' are light and view in local coordinates.

Similar to [15] we discretize the sphere into N_{bin} small spherical bins, each of which has a solid angle Ω_{bin_i}. This allows us to rewrite the rendering equation as:

$$L(x \rightarrow \omega_o) = L_e(x \rightarrow \omega_o) + \sum_{i=1}^{N_{\text{bin}}} K_i(x, \omega_o), \tag{2}$$

with

$$K_i(x, \omega_o) = \int\limits_{\Omega_{\text{bin}_i}} f_r(x, \omega_i \leftrightarrow \omega_o) \cdot L(x \leftarrow \omega_i.) \cdot (\mathbf{n}_x \cdot \omega_i) d\omega_i$$

We now rewrite the K_i as an integral over all surface elements y inside Ω_{bin_i} instead of solid angles:

$$K_i(x, \omega_o) = \int\limits_{y \in \Omega_{\text{bin}_i}} f_r(x, \omega_i \leftrightarrow \omega_o) \cdot L(x \leftarrow \omega_i) \cdot V(x, y)(\mathbf{n}_x \cdot \omega_i) \cdot \tfrac{(\mathbf{n}_y \cdot -\omega_i)}{r^2} dA_y, \tag{3}$$

where V is the binary visibility between two points.

It is now possible to make several simplifying assumptions to speed up the computation. First, we assume that for each element inside a bin the outgoing radiance is constant across its extent. Furthermore, we assume that the size of each element is very small, such that the cosine between the integration direction and the normal is essentially constant. Finally, we assume that surface elements are either completely visible or completely occluded. This allows us to rewrite Equation (3) as:

$$K_i(x, \omega_o) \approx \sum_{j=1}^{\#y \in \Omega_{bin_i}} f_r(x, \omega_{\text{i},j} \leftrightarrow \omega_o) \cdot L(x \leftarrow \omega_{\text{i},j}) \cdot V(x, y_j)(\mathbf{n}_x \cdot \omega_{\text{i},j}) \tfrac{(\mathbf{n}_{y_j} \cdot -\omega_{\text{i},j})}{r^2} A_{y_j}, \tag{4}$$

where $\omega_{\text{i},j}$ is the direction to the surface y_j. Note that, we only evaluate the binary visibility once (between x and the surface element's center) and turn the original integral into a sum over surface elements.

We make the final assumption that a surface element always covers the extent of a spherical bin Ω_{bin_i} completely. This means that only the closest element needs to be considered and Equation (4) becomes:

$$K_i(\omega_o, x) \approx f_r(x, \omega_{\text{i},s} \leftrightarrow \omega_o) \cdot L(x \leftarrow \omega_{\text{i},s}) \cdot (\mathbf{n}_x \cdot \omega_{\text{i},s}) \cdot \tfrac{(\mathbf{n}_{y_s} \cdot -\omega_{\text{i},s})}{r^2} \cdot A_{y_s}, \tag{5}$$

where y_s is the closest surface element.

We use this formulation to render a global illumination solution by means of a radiosity-like algorithm. The discretization into bins allows us to borrow the idea of shadow mapping. We create a (hierarchical) link structure between scene elements, where we store links in the discretized bins at each element, as opposed to a simple list of links used for standard radiosity algorithms. When creating the link structure, each bin will only store the shortest link, i.e., the link connecting to the *closest* surface element. Using this scheme, the visibility information will be implicitly retrieved from the link structure. This can be seen as a variant of omni-directional shadow mapping [1]; for each point x we discretize visibility for its upper hemisphere (Figure 3e).

3.1 Conceptual Overview

Conceptually, our algorithm is very similar to standard radiosity. We create links between scene elements and light sources, and transfer energy between them until the solution is converged (or a certain number of iterations has been reached).

In contrast to standard radiosity, we do not store a simple list of links at each scene element, but structure the links by storing them in bins. A non-hierarchical version of our algorithm would simply try to connect all scene elements with each other. Whenever a link is about to be created, its respective bin is queried and checked if there already exists a link and if that link is shorter or longer than the new link. In case the new link is shorter, it replaces the old one; if not, the old one remains. After all links have been created, normal shooting or gathering iterations can be run to transfer energy. Similar to Immel et al. [15], this allows for diffuse as well as glossy direct and indirect illumination.

Of course, this basic algorithm is inefficient as a non-hierarchical link structure grows quadratically in the number of scene elements. In the following, we therefore develop a hierarchical version of this algorithm, which enables us to obtain near-real-time frame rates for dynamic scenes on a single PC.

4 Hierarchical Implicit Visibility

Our method is visualized in Figure 3 and pseudo-code can be found in Algorithm 1. In a preprocessing step, we create a *geometric hierarchy* for each object. This geometric hierarchy is only computed once and is then re-used at run-time to construct the *hierarchical link structure*, which is the data structure for computing the actual global illumination solution. At run-time, we first update the data associated with the surface elements (positions, etc.), as they might have changed from the last frame. We then construct the hierarchical link structure using implicit visibility as indicated before. After construction, the hierarchical link structure needs to be refined in a second pass to propagate the implicit visibility information to all levels. The propagation of energy is very similar to standard radiosity. We will detail our method in the following.

Algorithm 1 – Main Algorithm

1: *CreateSurfels()*: Create surface elements based on the input geometry information (vertex, face, etc).
2: *CreateGeometricHierarchy()*: Create hierarchical geometric structure for each object.
3: **for** each frame **do**
4: *UpdateElements()*: Update the geometry information for initial geometric hierarchy.
5: *CreateHierarchicalLinks()*: Create hierarchical links between elements.
6: *RefineHierarchicalLinks()*: Refine links (top-down, remove unnecessary links).
7: *PushdownLinks()*: Push all links to leaf node.
8: **for** each light bounce **do**
9: *ComputeIlluminateLeafNodes()*: Gather incident energy from links and compute illumination results in leaf nodes.
10: *PullupEnergy()*: Pull up the indirect lighting energy from leaf nodes.
11: **end for**
12: **end for**

4.1 Geometric Hierarchy Preprocessing

In order to facilitate illumination computations, we represent our objects using a hierarchy of *surface elements*. A surface element is an oriented disk with a position, normal and area. The surface elements at the finest scale are based on the vertices of the input model(s) (Figure 3b). We chose discs centered around vertices as they can be easily computed for any type of mesh. The position and normal information of each surface element is known from the input model. Similar to [2], its area is computed as one-third of the total area of all triangles sharing this vertex.

To speed up the run-time process, we precompute a geometric hierarchy of the surface elements, which is then re-used at run-time. Similar to other radiosity methods, we want to cluster the surface elements in a way such they are adjacent and oriented similarly. Different methods exist to achieve this goal [9, 33]. However, our models are allowed to deform at run-time preventing an optimal precomputed solution. We adopt the simple technique by Bunnel [2] and use UV texture space segments (typically provided by the artist to enable texturing) as the coarsest cluster unit (see Figure 3a for an example).

For each UV segment, we create one *hierarchical quadtree* representing a spatial disc hierarchy for all vertices in the segment. The root node of the tree is a surface element approximating the whole UV segment, the leaf nodes are the discs corresponding to single vertices. Each surface element in the tree can have up to four smaller child surface elements on the next lower level (Figure 3b and c). For each surface-element in the hierarchy, we store its position (average position of all its child surface elements), the overall surface area, as well as the average normal direction. The hierarchical structure is only computed once. However, the average position as well as normal is re-computed every frame in order to support dynamic models. The area of most elements varies very little during animation, therefore, the area does not have to be recalculated for each frame. Please note, that the terms node, surface element and disc are used interchangeably.

4.2 Creating the Hierarchical Link Structure

For each frame, we recompute a hierarchical link structure, which is used to perform light propagation but also implicitly determines visibility. This subsection details how this structure is created and refined (corresponds to steps 4–7 in Algorithm 1).

4.2.1 Update Elements of Geometric Hierarchy

Our method allows objects to move around and even deform. Therefore, the stored geometric information needs to be updated accordingly while preserving the hierarchy. At each frame, we therefore update position and normal of each surface element. The data of each parent node is updated based on its children. Note that the hierarchy itself remains untouched. This process is similar to updating a bounding volume hierarchy in ray-tracing [37].

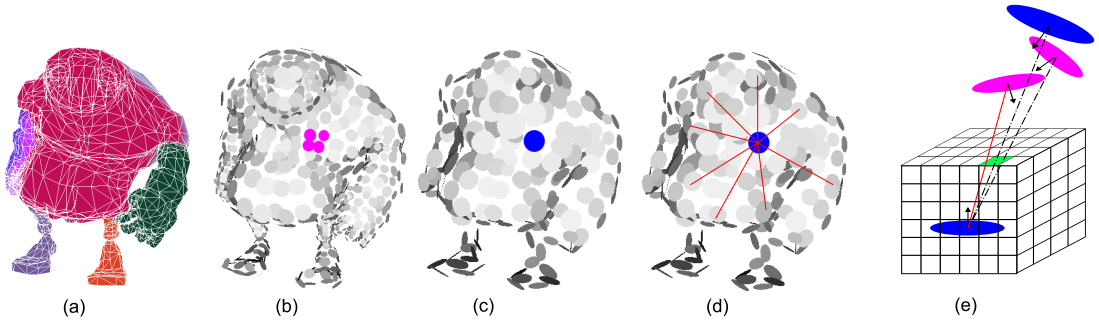


Figure 3. (a) Color-coded surface segments representing the coarsest approximation level of the geometric hierarchy. (b) Four surface elements of the finest level of the hierarchy and the corresponding element on the next coarser level (c). (d) Each surface element features (visibility/radiance) links to many other surface elements. (e) The sphere of directions for each element is discretized into cube-map bins, each one of them storing the shortest link to another disc.

4.2.2 Initial Hierarchical Link Structure

After the geometric information has been updated, we can proceed and build the hierarchical link structure. As stated before, we base the link creation on the precomputed geometric hierarchies. We start by linking *all* top level nodes of the geometric hierarchies. Whenever a link between two nodes (called A and B in the following) is about to be created, we perform the following checks:

- If the solid angle of B as seen from A's position is bigger than the solid angle of the link's respective bin, then the B-node should be subdivided, i.e., we try to link A to B's children (going down the geometric hierarchy). The same check is performed for A as seen from B.
- If there is already a link that has a shorter distance, which is determined by checking the link stored in the respective bin, no link will be created.

In all other cases, we create a link between the two surface elements A and B and store it in the respective bins of A and B.

As can be seen, there are two main metrics to determine whether two nodes can be connected. First, the solid angle determines whether the two surface elements are too big to be connected and should be subdivided. If the surface elements are bigger than a bin, it might happen that bins don't get filled with links, even though there is an element in that direction. This would prevent the implicit visibility to be evaluated correctly. Therefore, we go further down in the hierarchy. Second, the length of the link is used to determine if the other surface element is visible at all.

Discretization We chose the cube-map parameterization to discretize the sphere of directions. In other words, a bin corresponds to a texel in the cube-map. The main advantage of a cube-map lies in the efficient mapping of a direction to a bin.

4.2.3 Refining the Hierarchical Link Structure

During the creation of the hierarchical link structure, it is possible that a surface element and its child nodes contain different links in the same bin/direction because the order in which links are created is arbitrary. This is illustrated in Figure 4. The link creation process first happens to connect surface elements A and B. Then, in the next step, A and C are connected but since they are closer together, the links are created further down in the hierarchy (A and C are

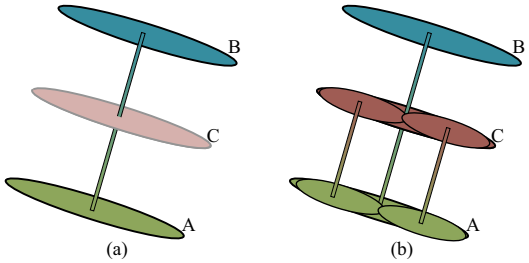


Figure 4. Linking problem: In (a), the elements A and B are connected with a single link. In the next step, the algorithm tries to connect A and C. Since they are close by, both A and C get subdivided and links are created further down in the hierarchy. Now there are inconsistent links at different levels in the hierarchy: A is still linked to B, even though there are shorter links further down in the hierarchy between A and C.

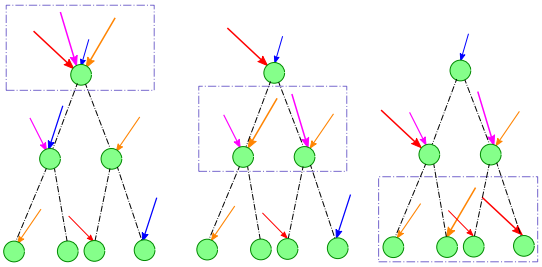


Figure 5. Refining hierarchical links: Different colors refer to different bins, and the length of each arrow represents the link's length. We traverse the tree breadth-first, and compare the links of parent and child nodes. If there is a shorter link in a node further down in the hierarchy, we remove the parent-node's link and push it to the siblings of the node. If there is a longer link, it is removed.

subdivided). The original link between A and B remains however, as the new shorter link is created further down in the hierarchy and does not remove the original link.

The purpose of refining the hierarchical links is to delete those incorrect (and redundant) links. To this end, we traverse the tree(s) in a breadth-first manner. During traversal, we compare the links in the bins of all parent-nodes and the links in the bins of the currently visited node. If, for a given bin, the current node contains a shorter link than a parent node, the parent-node's link is removed and pushed to the siblings of the current node (if they don't contain shorter links). If the parent node contains a shorter link, the child-node's link is simply removed. This refinement removes any incorrect links. Note that the refinement of links can be done in a single traversal of the tree by keeping track of which bins have links further up in the hierarchy.

4.2.4 Push-Down of Links

Our goal is to implement the illumination computation on the GPU. Unfortunately, GPUs only support very limited scatter operations, i.e., data cannot be written to arbitrary positions but usually only to the current raster position. The push part of the push-pull used by hierarchical radiosity algorithms [4] requires a scatter operation, as data is written to all the child nodes of a parent node.

We avoid this scatter operation and enable an efficient GPU implementation by pushing down links from all the interior nodes of our hierarchy to the leaf nodes. More specifically, the previously bidirectional links between two nodes are split into two unidirectional links through which energy is received at each node. All the receiving ends of the links are then pushed down the hierarchy to the leaf nodes. This step can be combined with the link refinement from the previous subsection.

4.3 Illumination Computation

Global illumination is computed in a similar manner to hierarchical radiosity. Energy is transferred between nodes along links. We chose a gathering approach, i.e., at each node, we gather all the energy from all incident links. The incident light is then convolved with the BRDF and converted to outgoing radiance. In case of diffuse BRDFs, the outgoing radiance is constant for all outgoing directions and we just store a single RGB triple. In case of glossy reflections, we augment our bin structure and store the outgoing radiance per direction in it.

As we have pushed down all receiver links to leaf nodes, outgoing illumination is only computed at leaf nodes (no other nodes can receive energy). Nonetheless, just like in hierarchical radiosity, we need to pull up the outgoing energy to the parent nodes, which is achieved by traversing the tree bottom-up and accumulating energies.

These two steps need to be iterated to account for indirect illumination. This procedure can be easily implemented on the CPU, but it unlocks its full potential only when implemented on the GPU.

4.3.1 GPU implementation

We store our surface elements, i.e., positions and normals, in two floating point textures. The hierarchical tree is stored in a texture in a pointer-less manner based on node indices. E.g., a full quadtree with 21 nodes and three hierarchy levels has indices 1–16 for the leaf nodes, indices 17–20 for the second level, and index 21 as the root node. Hence, the index of a node is sufficient to compute the indices of child and parent nodes. A third texture is used to store all the links. In order to allow for fast construction of this texture, we simply flatten the $6 \times N \times N$ bin structure of each node and store its content in the 2D domain (we actually store this data split over several textures). A fourth texture contains the outgoing (and unshot) energy for each node.

Computing the illumination is rather straightforward given these textures. For each leaf node, we loop over all its links and gather and sum the unshot energy from them. It is then converted into outgoing radiance by multiplying with the albedo of the node. After the energy has been gathered at all leaf nodes, we need to perform the traditional push-up operation. The pointer-less tree representation allows us to do this very efficiently by traversing bottom-up through all nodes of the tree. For each node, we accumulate the outgoing radiance weighted by the area ratio. These two steps can be repeated to account for several bounces of indirect illumination.

For final display, we convert the texture containing outgoing radiance into a vertex texture, which is used to set the color at the vertices of the model.

Speedup The direct lighting computation can be sped up, as there are generally few links to the light sources. Instead of going through all bins, we create a special texture that contains for each node: the number of light links and the actual links (indices to nodes). Now, we only need to go through those links to gather energy.

4.4 Light Sources

We support area light sources as well as environmental lighting. Area light sources are geometry like any other object, with the notable difference that their initial outgoing radiance is set to be non-zero.

Environmental lighting could be handle the same way, but allows for a simple optimization. Instead of creating geometry for the environment, we initially omit it completely and create our hierarchy with objects only. We now use the observation that any empty bin (in the leaf nodes) can see the lighting environment and therefore receives light from it. Our optimized direct lighting step (see above) checks for this, and gathers light from the environment for any empty bin.

5 Results

Our method enables interactive rendering of fully dynamic scenes with direct and indirect illumination. Figure 1 shows an example where a teapot reflects the colored pattern of a ground plane. Also note the soft shadow cast by the environmental lighting. This runs at interactive speeds (around 7 FPS) on an NVIDIA 8800. Deformable objects, as shown in Figure 2, can also be handled easily.

Figure 6. Shadow and indirect lighting effects between objects (with (a) 5165 vertices and (b) 4782 vertices).

Figure 6 demonstrates that our method can handle shadows and indirect lighting effects between objects. Note in (b) how there is a green sheen on the grey chess piece.

Discretization	$6\times8\times8$	$6\times12\times12$	$6\times16\times16$
Update Elements	12ms (6%)	12ms (8%)	12ms (11%)
Create Hierarchy	23ms (21%)	43ms (29%)	78ms (38%)
Refine & Push	30ms (29%)	35ms (24%)	60ms (30%)
Illumination	35ms (35%)	40ms (28%)	48ms (24%)
Total	112ms	148ms	218ms

Table 1. Timings for the monster (3378 vertices, 1-bounce illumination, see Fig. 8).

Figure 9 compares a reference image (a) computed with path tracing to a result of our method (b). Despite all the approximations we make, as detailed in Section 3, the differences are minor. Our method produces slightly softer shadows, which is less noticeable for a directional discretization of $6\times16\times16$. The differences become more prominent for coarser discretizations and artifacts appear.

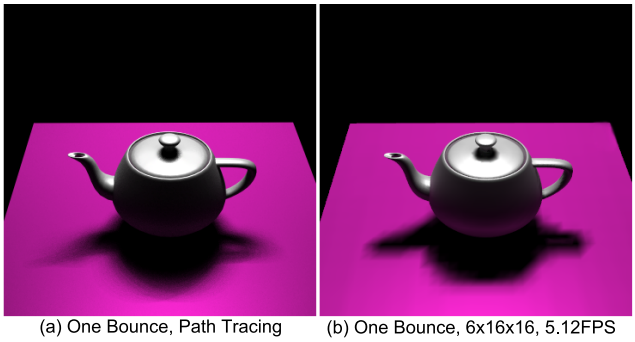
Figure 7 shows a teapot illuminated with an area source. Overall, the shading compares well to the reference image. However, discretization artifacts become obvious in the shadow area. These artifacts can be reduced by either increasing the number of bins or by using larger area lights.

Figure 8 shows a monster on a ground plane with direct, one-bounce indirect, and two-bounce indirect lighting. Three levels of directional discretization are compared. One-bounce lighting is sufficient for a pleasing result. A coarse directional discretization produces slight artifacts, but the speed gains are considerable. Table 1 details the time spent on the different steps of our algorithm for this particular scene. The initial creation of the geometric hierarchy takes about 72ms but is only done once in a preprocess.

Figure 10 compares a reference image (a) computed with path tracing to our method (b). The differences are minor. However, our method renders with several frames per second. We also demonstrate that there is virtually no difference between our hierarchical (b) and a brute-force non-hierarchical version (c). Coarsely tessellated objects can cause light leakage, see (d) where the teapot only has 792 vertices (992 triangles).

Our GPU implementation also supports glossy direct illumination, which we demonstrate in Figure 11. In order to maintain interactive frame rates, we limit indirect illumination to diffuse interreflections in our GPU implementation (even though the proposed method itself can handle glossy interreflections). Our results compare favorably to the reference solution (Figure 11a).

We have found that our algorithm has roughly a complexity of $O(N \log N)$, with N being the number of vertices, which is similar to other hierarchical radiosity methods.



(a) One Bounce, Path Tracing (b) One Bounce, 6x16x16, 5.12FPS

Figure 7. Discretization artifacts may become visible under area lighting. However, increasing the number of bins reduces artifacts.

5.1 Discussion

Despite the high-frame rate and the faithful reproduction of global illumination effects as documented by the ground truth comparisons, the proposed approximations and discretization may lead to visual artifacts. If only a coarse cube-map discretization of the directional hemisphere is used, block artifacts may be visible in the light simulation (e.g., Figure 7 and 8). However, using $6 \times 12 \times 12$ bins, we achieve a good compromise between speed and visual quality.

Additional inaccuracies may occur due to the uneven distribution of solid angles across bins. Furthermore, although the fixed world-space alignment of the bin cube-maps across all geometric hierarchy levels enables fast computation, differences in directional sampling for different surface element orientations may lead to inaccuracies and temporal aliasing when objects undergo deformations (see accompanying video).

Moreover, rendering quality depends on the initial triangulation of the models as we base our lighting simulation on the models' vertices. Starkly uneven triangulation may therefore require re-meshing to prevent artifacts. If a model is not tessellated finely enough, light leakage might occur, as not every bin can be filled with a link for accurate occlusions. However, for $6 \times 16 \times 16$ or fewer bins and models of about 5000 vertices, we have rarely encountered it.

Currently, we also trade rendering performance for accuracy and precompute the geometric hierarchy once, ignoring the fact that an adaptation of the hierarchy according to the deformation may be beneficial.

Despite these trade-offs and approximations, which are necessary to obtain high performance, the good visual quality of our results shows that interactive full global illumination on a single PC is feasible.

6 Conclusions

We presented a new global illumination method that builds on and extends the traditional hierarchical radiosity approach by implicitly computing visibility. This new concept circumvents time-consuming explicit visibility queries, the main performance bottleneck in traditional approaches. Our method allows for rendering of full global illumination solutions for moderately complex and arbitrarily deforming dynamic scenes at near-real-time frame rates on a single PC. It faithfully reproduces a variety of complex lighting effects including diffuse and glossy interreflections, and handles scenes featuring environment map and area light sources.

As part of future work, we plan to investigate explicit temporal coherence strategies to further improve animation quality. Decoupling the tessellation of the mesh from shading computation is another interesting line of research.

References

[1] S. Brabec, T. Annen, and H.-P. Seidel. Shadow Mapping for Hemispherical and Omnidirectional Light Sources. In *Proc. of CGI*, pages 397–408, 2002.

[2] M. Bunnell. Dynamic Ambient Occlusion and Indirect Lighting. In M. Pharr, editor, *GPU Gems 2*, chapter 2, pages 223–233. Addison Wesley, Mar. 2005.

[3] S. Chen. Incremental Radiosity: An Extension of Progressive Radiosity to an Interactive Image Synthesis System. *Computer Graphics (Proc. of SIGGRAPH 90)*, 24(4):135–144, Aug. 1990.

[4] M. Cohen and J. Wallace. *Radiosity and Realistic Image Synthesis*. Academic Press Professional, Cambridge, MA, 1993.

[5] G. Coombe, M. Harris, and A. Lastra. Radiosity on Graphics Hardware. In *Proc. of Graphics Interface*, pages 161–168, 2004.

[6] C. Dachsbacher and M. Stamminger. Reflective Shadow Maps. In *Proc. of I3D*, pages 203–213, 2005.

[7] C. Dachsbacher and M. Stamminger. Splatting Indirect Illumination. In *Proc. of I3D*, pages 93–100, 2006.

[8] C. Dachsbacher, M. Stamminger, G. Drettakis, and F. Durand. Implicit Visibility and Antiradiance for Interactive Global Illumination. *ACM TOG*, 26(3), August 2007.

[9] M. Garland, A. Willmott, , and P. Heckbert. Hierarchical Face Clustering on Polygonal Surfaces. In *Proc. of I3D*, pages 49–58, 2001.

[10] P. Gautron, J. Krivánek, K. Bouatouch, and S. Pattanaik. Radiance Cache Splatting: A GPU-Friendly Global Illumination Algorithm. In *Proc. of EGSR*, pages 55–64, 2005.

[11] D. George, F. Sillion, and D. Greenberg. Radiosity Redistribution for Dynamic Environments. *IEEE CGAA*, 10(4):26–34, 1990.

[12] P. Hanrahan, D. Salzman, and L. Aupperle. A Rapid Hierarchical Radiosity Algorithm. *Proc. of SIGGRAPH*, 25(4):197–206, July 1991.

Figure 8. A monster rendered under environmental lighting with direct illumination, one-bounce, and two-bounce indirect illumination and varying discretizations (3378 vertices).

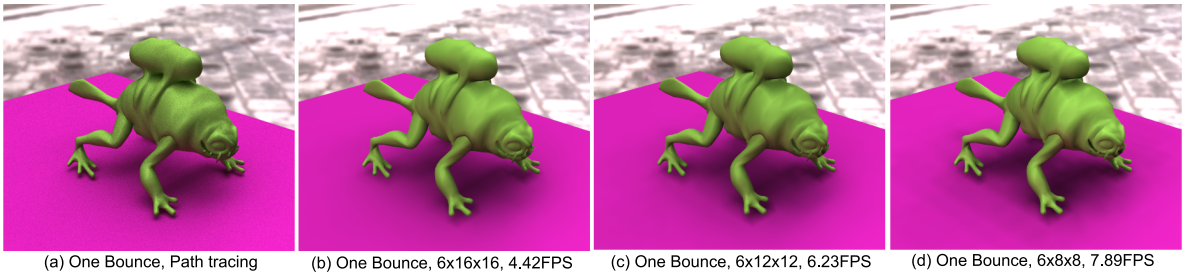


Figure 9. Frog with one-bounce indirect lighting and varying bin discretization (3495 vertices).

[13] W. Heidrich, K. Daubert, J. Kautz, and H.-P. Seidel. Illuminating Micro Geometry Based on Precomputed Visibility. In *Proc. of ACM SIGGRAPH*, pages 455–464, 2000.

[14] W. Heidrich and H. Seidel. Realistic, Hardware-accelerated Shading and Lighting. In *Proc. of ACM SIGGRAPH*, pages 171–178, Aug. 1999.

[15] D. Immel, M. Cohen, and D. Greenberg. A Radiosity Method for Non-Diffuse Environments. In *Proc. of ACM SIGGRAPH*, pages 133–142, 1986.

[16] K. Iwasaki, Y. Dobashi, F. Yoshimoto, and T. Nishita. Precomputed Radiance Transfer for Dynamic Scenes Taking into Account Light Interreflection. In *Proc. of EGSR*, pages 35–44, June 2007.

[17] H. W. Jensen. Global Illumination using Photon Maps. In *Proc. EG Rendering Workshop*, pages 21–30, June 1996.

[18] J. Kajiya. The Rendering Equation. In *Proc. of ACM SIGGRAPH*, pages 143–150, August 1986.

[19] J. Kautz, J. Lehtinen, and T. Aila. Hemispherical Rasterization for Self-Shadowing of Dynamic Objects. In *Proc. of*

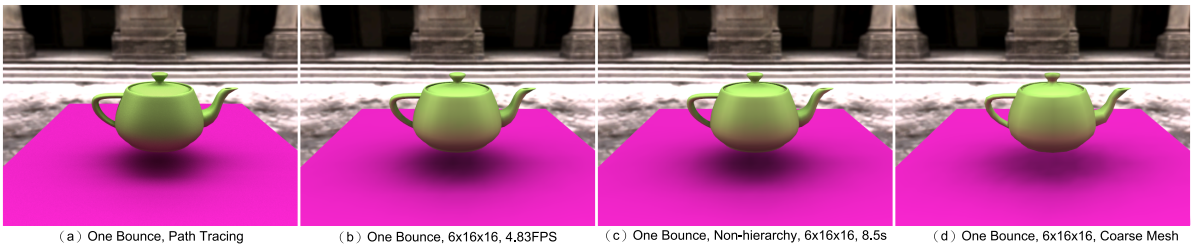


Figure 10. Environment lighting: (a) Ground truth using path tracing. (b) Non-hierarchical CPU implementation of our method. (c) GPU version of our algorithm. (d) Light leaking if mesh tessellation is too coarse (teapot: (a)-(c) 2582 vertices, (d) 792 vertices).

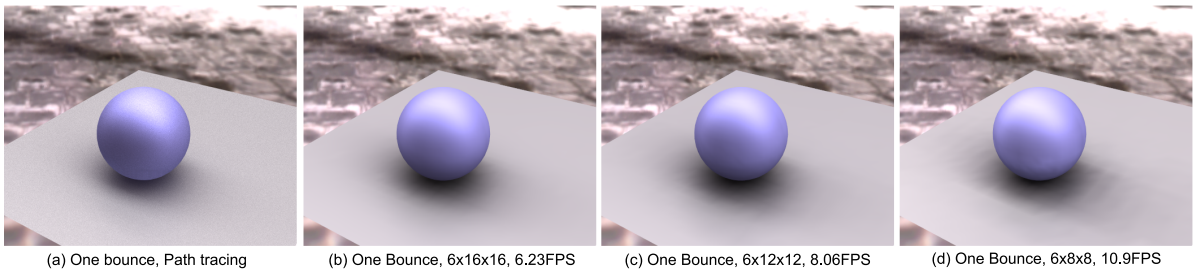


Figure 11. Glossy sphere (2278 vertices).

EGSR, pages 179–184, June 2004.

[20] J. Kautz, P.-P. Vázquez, W. Heidrich, and H.-P. Seidel. A Unified Approach to Prefiltered Environment Maps. In *Proc. of EG Rendering Workshop*, pages 185–196, 2000.

[21] A. Keller. Instant Radiosity. In *Proc. of ACM SIGGRAPH*, pages 49–56, August 1997.

[22] E. Lafortune and Y. Willems. Bidirectional Path Tracing. In *Proc. of Compugraphics*, pages 95–104, 1993.

[23] B. Larsen and N. Christensen. Simulating Photon Mapping for Real-time Applications. In *Proc. of EGSR*, 2004.

[24] X. Liu, M.-H. Pan, R. wang, and H.-J. Bao. Efficient Rendering of Interreflections for Dynamic Scenes. In *Proc. of Eurographics 2007*, 2007.

[25] X. Liu, P.-P. Sloan, H.-Y. Shum, and J. Snyder. All-Frequency Precomputed Radiance Transfer for Glossy Objects. In *Proc. of EGSR*, pages 337–344, 2004.

[26] R. Ng, R. Ramamoorthi, and P. Hanrahan. All-Frequency Shadows Using Non-linear Wavelet Lighting Approximation. *ACM TOG*, 22(3):376–381, July 2003.

[27] M. Nijasure, S. Pattanaik, and V. Goel. Real-Time Global Illumination on GPUs. *Journal of Graphics Tools*, 10(2):55–71, 2005.

[28] C. Puech, F. Sillion, and C. Vedel. Improving Interaction with Radiosity-based Lighting Simulation Programs. In *Proc. of I3D*, pages 51–57, Mar. 1990.

[29] T. Purcell, C. Donner, M. Cammarano, H. Jensen, and P. Hanrahan. Photon Mapping on Programmable Graphics Hardware. In *Graphics Hardware*, pages 41–50, 2003.

[30] Z. Ren, R. Wang, J. Snyder, K. Zhou, X. Liu, B. Sun, P.-P. Sloan, H. Bao, Q. Peng, and B. Guo. Real-Time Soft Shadows in Dynamic Scenes using Spherical Harmonic Exponentiation. *ACM TOG*, 25(3):977–986, 2006.

[31] P. Shirley and R. Morley. *Realistic Ray Tracing*. AK Peters, 2003.

[32] P.-P. Sloan, J. Kautz, and J. Snyder. Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments. In *Proc. of ACM SIGGRAPH*, pages 527–536, July 2002.

[33] B. Smits, J. Arvo, and D. Greenberg. A Clustering Algorithm for Radiosity in Complex Environments. In *Proc. of SIGGRAPH*, pages 435–442, 1994.

[34] W. Sun and A. Mukherjee. Generalized Wavelet Product Integral for Rendering Dynamic Glossy Objects. *ACM TOG*, 25(3):955–966, 2006.

[35] E. Veach and L. Guibas. Metropolis Light Transport. In *Proc. of ACM SIGGRAPH*, pages 65–76, Aug. 1997.

[36] I. Wald, C. Benthin, and P. Slusallek. Interactive Global Illumination in Complex and Highly Occluded Environments. In *Proc. of EGSR*, pages 74–81, 2003.

[37] I. Wald, S. Boulos, and P. Shirley. Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies. *ACM TOG*, 26(1), 2007.

[38] I. Wald, T. Kollig, C. Benthin, A. Keller, and P. Slusallek. Interactive global illumination. In *Proc. of EG Workshop on Rendering*, pages 9–20, 2002.

[39] T. Whitted. An improved illumination model for shaded display. *Communications of ACM*, 23(6):343–349, 1980.

[40] K. Zhou, Y. Hu, S. Lin, B. Guo, and H.-Y. Shum. Precomputed Shadow Fields for Dynamic Scenes. *ACM TOG*, 24(3):1196–1201, 2005.

[41] S. Zhukov, A. Iones, and G. Kronin. An Ambient Light Illumination Model. In *Proc. of EG Rendering Workshop*, pages 45–56, 1998.