Eurographics Symposium on Rendering (2005)
Kavita Bala, Philip Dutré (Editors)

Radiance Cache Splatting: A GPU-Friendly Global Illumination Algorithm

Pascal Gautron † Jaroslav Křivánek ‡ Kadi Bouatouch § Sumanta Pattanaik ¶

Abstract

Fast global illumination computation is a challenge in several fields such as lighting simulation and computer-generated visual effects for movies. To this end, the irradiance caching algorithm is commonly used since it provides high-quality rendering in a reasonable time. However this algorithm relies on a spatial data structure in which nearest-neighbors queries and data insertions are performed alternately within a single rendering step. Due to this central and permanently modified data structure, the irradiance caching algorithm cannot be easily implemented on graphics hardware. This paper proposes a novel approach to global illumination using irradiance and radiance cache: the radiance cache splatting. *This method directly meets the processing constraints of graphics hardware since it avoids the need of complex data structure and algorithms. Moreover, the rendering quality remains identical to classical irradiance and radiance caching. Our renderer shows an implementation of our algorithm which provides a significant speedup compared to classical irradiance caching.*

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism - Rendering, Global Illumination

1. Introduction

The aim of global illumination computation is to simulate multiple interreflections of light in a scene.As computers become more and more powerful, high-quality global illumination computation gets employed in a growing number of fields, such as architectural design, cinema and video games. Generally, the computation is performed using ray tracing and Monte Carlo sampling, and is very costly. A number of methods have been proposed to reduce the computational cost of global illumination. Several approaches have been proposed to render globally illuminated scenes in real-time, such as [WBS03, GWS04, TPWG02, WS99]. However, interactive methods based on ray tracing rely on parallel processing using several computers to maintain a reasonable frame rate. An efficient approach to global il-

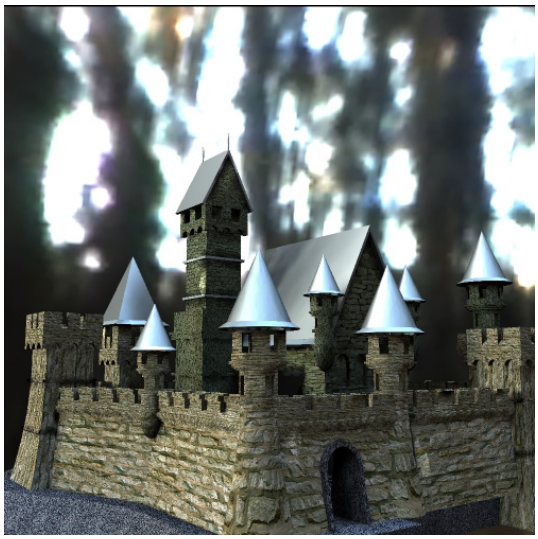


Figure 1: *The Castle scene (58K triangles) illuminated by an environment map. Our renderer computes first-bounce glossy global illumination in 10.1 s at resolution* 1000×1000.

† IRISA (Université de Rennes 1), Rennes, France - University of Central Florida, Orlando, FL, USA
‡ IRISA (Université de Rennes 1), Rennes, France - University of Central Florida, Orlando, FL, USA - Czech Technical University , Prague, Czech Republic
§ IRISA (Université de Rennes 1), Rennes, France
¶ University of Central Florida, Orlando, FL, USA

lumination using ray tracing is the irradiance and radiance caching [WRC88, KGPB05]. The irradiance caching method is being employed for architectural design using the Radiance software [War04, War94].

In this paper, we propose a new method for irradiance and radiance caching which leverages graphics hardware and computes global illumination at a time order of magnitude faster than currently available caching-based methods. As shown in [TL04], first bounce global illumination takes into account most of the light transfer in a scene, and provides realistic results in most cases. Moreover, Tabellion et al. [TL04] show that a coarsely tessellated scene is sufficient to compute an accurate indirect component of the global illumination solution. Therefore, this paper focuses on the computation of first bounce global illumination in moderately complex scenes. Our method provides a fast and simple way of computing indirect lighting in a simplified geometry, while the direct lighting can be computed independently by an offline renderer using a detailed geometry.

We reformulate the irradiance and radiance caching algorithms by defining a fast image-space method based on splatting. This method makes an extensive use of graphics hardware and can be used for fast, high quality rendering or interactive visualization of globally illuminated scenes. Unlike [PBMH02, PDC*03], we avoid the need of representing and traversing complex data structures on the Graphics Processing Unit (GPU).

This paper is organized as follows: Section 2 presents previous techniques used for fast global illumination using graphics hardware. After an overview of the irradiance and radiance caching algorithms in Section 3, Section 4 presents our rendering algorithm. Section 5 details the implementation of our GPU-based renderer, and Section 6 discusses the results obtained with our algorithm in both high-quality rendering and interactive walkthroughs.

2. Related Work

Much research has been done in GPU-accelerated global illumination computation during the past years. This section describes related GPU-based global illumination algorithms.

Radiosity [GTGB84] is a classical method for global illumination computation. This approach is based on the calculation of energy transfer between all surface elements in a scene. Therefore, many visibility tests are required to perform an accurate computation, making this method very costly. Many attempts to hardware acceleration for radiosity have been developed in the last decades. Among them the hemi-cube approach [CG85, SP89] uses graphics hardware to identify the patches visible from a given patch in the scene. More recently, [CHL04] and [CHH03] propose methods for GPU-based radiosity. The former relies on texturing and visibility testing, whereas the latter uses the GPU to process the radiosity matrix.

The method described in [TPWG02] makes use of graphics hardware to display the results of global illumination computation at interactive rates. In this approach, the scene is adaptively tesselated, and the incoming radiance is computed for each vertex using parallel ray tracing. Unlike in our method, the GPU is only used for interpolating the incoming radiance across triangles using Gouraud shading. Unfortunately, high quality rendering requires to tesselate each surface into many triangles, yielding performance drop. Moreover, this method focuses only on interactive visualization: the rendering time for high quality global illumination is not improved by this approach.

In [NPG04, NPG03], Nijasure et al. propose a method for non diffuse global illumination computation using graphics hardware. The incoming radiance function at a number of locations *a priori* selected is sampled and projected into the spherical harmonics basis. Then the incoming radiance at any surface point is estimated by interpolating the incoming radiance at nearby sample locations. Although the authors demonstrate real-time performance, the main drawback of this method is the choice of sample points. In [NPG04, NPG03] these points are placed on a regular grid inside the volume of the scene, therefore not adapting to the lighting complexity.

In the Precomputed Radiance Transfer (PRT) approaches, the radiance transfer between surfaces of an object is precomputed offline and represented using spherical harmonics [SKS02, SHHS03] or wavelets [LSSS04, WTL04]. Using this precomputed information, the global illumination solution can be computed and displayed at interactive rates using the GPU. Although the PRT approaches allow real-time, high quality relighting, they rely on a costly precomputation which prevents from using them easily in complex dynamic scenes.

Wand and Straßer [WS03] describe a GPU-based method for real-time caustics computation. This algorithm relies on the selection of sample points on glossy surfaces. Each sample point is considered as a pinhole camera that projects the incoming light on diffuse receiver surfaces. This method handles several dynamic light sources and objects at interactive rates, but speed drops quickly as quality improves.

The Reflective Shadow Maps method [DS05] aims at computing first-bounce global illumination in realtime. This approach is based on an extension of shadow mapping, in which each element of the shadow map stores the incoming light flux. Although the this method offers realtime performance, it does not consider occlusion for the computation of indirect lighting, then yielding physically incorrect results.

Several attempts have been made to compute global illumination using GPU-based ray tracing. These methods rely on the versatility of programmable graphics hardware and use fragment shaders to perform ray-primitive intersections [CHH02, PBMH02]. The work described in [PBMH02] has been extended to photon map [Jen01] rendering [PDC*03].

In addition, another photon map rendering method is presented in [MM02]. Those approaches suffer from the same drawback: the GPU architecture does not allow to handle complex data structures such as trees, which are commonly used in ray tracing optimization and photon map storage. Therefore, the photon map is stored in a regular grid [PDC*03], or in a costly hash table [MM02]. The related nearest-neighbors queries have been simplified to meet the data structure and GPU constraints, yielding quality or performance drop.

Three other approaches for GPU-accelerated photon map rendering have been proposed. Larsen et al. [LC04] use graphics hardware to perform the costly final gathering: the photon map is built on the Central Processing Unit (CPU) using the classical method defined in [Jen01]. For each surface, an "approximate illumination map" is built using the data contained in the photon map. The GPU is used to perform final gathering and caustics filtering. In this paper, we take advantage of this approach to accelerate global illumination computation. The approaches presented in [SB97] and [LP03] use the GPU for irradiance reconstruction: each photon is rendered as a textured quadrilateral. The corresponding texture represents the kernel function for the photon. Although those methods show encouraging results, they are bounded by the large number of photons required to render a high quality image.

Besides hardware acceleration, many other methods have been proposed to speed up global illumination computation. Among them, the approaches based on the storage and the interpolation of incoming radiance provide fast and accurate results. Such methods include the photon maps [Jen01] and the shading cache [TPWG02]. The irradiance caching [WRC88] provides a fast and accurate way of computing indirect diffuse interreflections. [KGPB05] proposes the radiance caching, an extension of irradiance caching for the computation of indirect glossy lighting. This latter uses hemispherical harmonics [GKPB04] to represent the incoming radiance function and account for view-dependent BRDFs.

The method proposed in this paper reformulates the irradiance and radiance caching algorithms to allow an easy and efficient GPU implementation.

3. Irradiance and Radiance Caching Overview

Due to the similarity between irradiance and radiance caching, we refer to these algorithms as *(ir)radiance* caching in the remainder of this document. The (ir)radiance caching algorithms are based on the following observation: "the indirect illuminance tends to change slowly over a surface" [WRC88]. Therefore, these methods exploit spatial coherence by sparsely sampling and interpolating indirect incoming radiance. For each sample point, an *(ir)radiance record* stores the sampled incoming radiance. The records are stored in the *(ir)radiance cache*. If a point $\mathbf{p}$ in the scene is surrounded with a set of (ir)radiance records S_r, the indirect incoming lighting at point $\mathbf{p}$, $E(\mathbf{p})$, can be estimated by Eq. (1) [WRC88].

$$E(\mathbf{p}) = \frac{\sum_{k \in S_r} w_k(\mathbf{p}) E_k}{\sum_{k \in S_r} w_k(\mathbf{p})} \tag{1}$$

where E_k is the computed incoming lighting at $\mathbf{p}$ and $w_k(\mathbf{p})$ is the weighting function of record k evaluated at $\mathbf{p}$ (see the next section for the definition of w_k). In the case of irradiance cache, $E(\mathbf{p})$ represents the irradiance at point $\mathbf{p}$. For radiance cache, $E(\mathbf{p})$ stands for the incoming radiance function. The record set S_r is computed by querying the (ir)radiance cache. In order to optimize the rendering speed, a spatial data structure such as an octree is used to represent the (ir)radiance cache. More details on (ir)radiance caching and incoming radiance interpolation can be found in [WH92, War94, KGPB05, KGBP05].

4. Our Algorithm

In this paper, our aim is to reformulate the (ir)radiance caching algorithm to take advantage of the GPU computing power for first-bounce global illumination. The GPUs are SIMD (Single Instruction Multiple Data) processors. Such processors cannot handle efficiently complex data structures such as octrees. Therefore, we propose a fast rendering algorithm which avoids the need for nearest-neighbors queries and spatial data structures in the (ir)radiance caching. Moreover this approach aims at reducing the CPU workload by performing most of the computation on the GPU. The core of our approach is the *radiance cache splatting*, which determines the contribution of each record to the indirect lighting of visible objects. The radiance cache splatting and the whole rendering algorithm are described hereafter. For the sake of clarity, our algorithm is first presented in the case of irradiance caching. If necessary, specific details about the extension to radiance caching are given at the end of each subsection.

4.1. Radiance Cache Splatting

As described in Section 3, the irradiance caching algorithm relies on the computation and the interpolation of irradiance records. For a point visible through a pixel, the irradiance caching determines which records contribute to the indirect lighting of this point. The radiance cache splatting uses the opposite approach: for a given record, our algorithm determines which visible points it contributes to by splatting the record on the image plane. The result of radiance cache splatting is stored in the *radiance splat buffer*, which has the same size as the frame buffer. Each pixel $SPLATBUFF(x,y)$ of the radiance splat buffer is a pair (L_o, w), where L_o is the sum of the weighted contribution of each record, and w is the cumulated weight.

As described above, the radiance cache splatting (Algorithm 1) is designed for the computation of the contribution of an irradiance record to the indirect lighting of visible points. Our approach is based on the equation used in the irradiance caching interpolation scheme (Eq. (1)). The weight allocated to record k at point $\mathbf{p}$ with normal $\mathbf{n}$ is defined in [WRC88] as:

$$w_k(\mathbf{p}) = \frac{1}{\frac{\|\mathbf{p}-\mathbf{p}_k\|}{R_k} + \sqrt{1-\mathbf{n}\cdot\mathbf{n}_k}} \tag{2}$$

where $\mathbf{p}_k$, $\mathbf{n}_k$ and R_k are respectively the location of record k, its normal and the harmonic mean distance to the objects visible from $\mathbf{p}_k$. The user-defined value a represents the accuracy of the computation. This value is used to threshold the weighting function: record k contributes to the estimate of the outgoing radiance at point $\mathbf{p}$ if and only if

$$w_k(\mathbf{p}) \geq \frac{1}{a} \tag{3}$$

Substituting Eq. (3) into Eq. (2) and assuming $\mathbf{n} = \mathbf{n}_k$, one can see that record k can contribute to the estimate of the outgoing radiance at point $\mathbf{p}$ only if:

$$\|\mathbf{p} - \mathbf{p}_k\| \leq aR_k \tag{4}$$

Therefore, Eq. (4) guarantees that a record k cannot contribute to the outgoing radiance of a point outside a sphere I_k centered at $\mathbf{p}_k$, with radius $r_k = aR_k$.

Algorithm 1 Radiance cache splatting

```
Let k = {p_k, n_k, E_k, R_k} be the considered record
Determine the bounding box of I_k on the image plane
for all pixel P(x,y) = {p, n, ρ_d} in the bounding box do
  // Evaluate weighting function at p
  w = 1 / (||p - p_k|| / R_k + sqrt(1 - n·n_k))
  if w ≥ 1/a then
    //Compute the contribution of record k at point p
    E'_k = E_k (1 + n_k × n · ∇_r + (p - p_k) · ∇_t)
    // Compute the outgoing radiance
    L_o = ρ_d E'_k
    // Accumulate into the radiance splat buffer
    SPLATBUFF(x,y).L_o += w L_o
    SPLATBUFF(x,y).w += w
  end if
end for
```

Given a camera, the *radiance cache splatting* splats the sphere I_k onto the image plane (Figure 2). The weighting function (Eq. (2)) is evaluated for each point visible through pixels covered by I_k. Then, the weight is tested against the accuracy value (Eq. (3)). For each pixel passing this test, our algorithm computes the contribution of record k to the outgoing radiance estimate.

The outgoing radiance contribution L_o to a point $\mathbf{p}$ as seen through a pixel is obtained by evaluating the following integral:

$$L_o(\mathbf{p}, \omega_o) = \int_H L_i(\mathbf{p}, \omega_i) f(\omega_o, \omega_i) \cos(\theta_i) d\omega_i \tag{5}$$

where ω_i and ω_o are respectively the incoming and outgoing directions. $L_i(\mathbf{p}, \omega_i)$ is the radiance incoming at $\mathbf{p}$ from direction ω_i. $f(\omega_o, \omega_i)$ is the surface BRDF evaluated for the directions ω_i and ω_o. In the case of irradiance caching, we only consider diffuse interreflections. Therefore, Eq. (5) simplifies to:

$$L_o(\mathbf{p}) = \rho_d \int_H L_i(\mathbf{p}, \omega_i) \cos(\theta_i) d\omega_i = \rho_d E(\mathbf{p}) \tag{6}$$

where ρ_d is the diffuse surface reflectance, and $E(\mathbf{p})$ is the irradiance at point $\mathbf{p}$. Therefore, the contribution of record k to the outgoing radiance at point $\mathbf{p}$ is

$$L_o = \rho_d E'_k(\mathbf{p}) \tag{7}$$

where $E'_k(\mathbf{p})$ is the irradiance estimate of record k at point $\mathbf{p}$. This estimate is obtained using irradiance gradients:

$$E'_k = E_k(1 + \mathbf{n}_k \times \mathbf{n} \cdot \nabla_r + (\mathbf{p} - \mathbf{p}_k) \cdot \nabla_t) \tag{8}$$

where ∇_r and ∇_t are respectively the rotational and translational gradients for record k. The computation of irradiance gradients is detailed in [WH92, KGBP05].

Note that our splatting approach can be used with any weighting function containing a distance criterion. In this paper we focus on the weighting function defined in [WRC88], although the function proposed in [TL04] could also be employed.

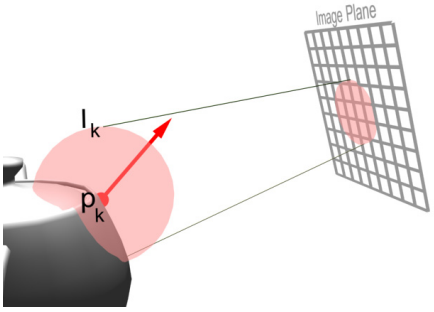


Figure 2: *The sphere I_k around the position $\mathbf{p}_k$ of the record k is splatted on the image plane. For each point within the sphere splat, the contribution of record k is accumulated into the radiance splat buffer.*

Extension to Radiance Caching The BRDFs of glossy surfaces are view-dependent. Therefore, Eqs. (6) and (8) cannot be used in this case. As described in [KGPB05], both the incoming radiance contribution and cosine-weighted BRDF are represented using hemispherical harmonics. Due to the basis functions orthonormality [GKPB04], Eq. (5) reduces

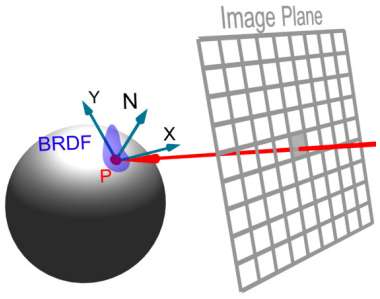


Figure 3: *The radiance cache splatting requires per-pixel information about geometry and materials: the hit point, local coordinate frame, and BRDF.*

to:

$$L_o(\mathbf{p},\omega_o) = \int_H L_i(\mathbf{p},\omega_i) f(\omega_o,\omega_i)\cos(\theta_i)\mathrm{d}\omega_i = \sum_{l=0}^{n-1}\sum_{m=-l}^{l} c_l^m(\omega_{out})\lambda_l^{m\prime}(\mathbf{p}) \quad (9)$$

where l is the order of the projection used for BRDF and incoming radiance representation. c_l^m and $\lambda_l^{m\prime}(\mathbf{p})$ are respectively the projection coefficients of the BRDF and the projection coefficients of the incoming radiance of record k interpolated at point $\mathbf{p}$. As described in [KGPB05, KGBP05] this estimate is obtained by applying translational gradients to the incoming radiance stored in record k. Then, since the local coordinate frames at points $\mathbf{p}_k$ and $\mathbf{p}$ might differ, we use hemispherical harmonics rotation to align the incoming radiance estimate with the local frame at $\mathbf{p}$. See [GKPB04] and [KGPB05, KGBP05] for details about hemispherical harmonics rotation and incoming radiance estimation.

Using the radiance cache splatting, the contribution of a record to the outgoing radiance estimate at points visible from the current camera is computed independently of other records. The outgoing radiance contribution of each cache record is accumulated in the radiance splat buffer. The process of generating the final image uses the contents of the radiance splat buffer to display the global illumination solution.

4.2. Indirect Lighting Rendering

The final indirect lighting image is generated in five main steps (Algorithm 2). Given a camera, the first step consists in obtaining per-pixel information about viewed objects: their position, local coordinate frame and BRDF (Figure 3).

In the second and third steps, the rendering process determines where new (ir)radiance records are necessary to achieve the user-defined accuracy of indirect illumination computation. In Step 2, each existing record (possibly computed for previous frames) is splatted onto the splat buffer using the procedure described in Section 4.1. Step 3 consists in reading back the radiance splat buffer into the CPU

Algorithm 2 Indirect lighting rendering

// Step 1
Generate geometric and reflectance data of objects viewed through pixels (GPU)
Clear the splat buffer
// Step 2
for all cache records **do**
  // The radiance cache is empty for the first image,
  // and non empty for subsequent images
  Algorithm 1: splat the records onto the radiance splat buffer (GPU)
end for
// Step 3
Read back the radiance splat buffer from GPU to CPU
// Step 4
for all pixels (x,y) in the radiance splat buffer **do**
  if $SPLATBUFF(x,y).w < a$ **then**
    Compute a new incoming radiance record at corresponding hit point (GPU/CPU): see Section 5.2 for technical details
    Apply Algorithm 1: splat the new record (CPU)
  end if
end for
// Step 5
for all cache records **do**
  Apply Algorithm 1: splat all the newly generated records (GPU)
end for
//Normalize the radiance splat buffer (GPU)
for all pixels (x,y) in the radiance splat buffer **do**
  $SPLATBUFF(x,y).L_o /= SPLATBUFF(x,y).w$
end for
Combine the radiance splat buffer with direct lighting (GPU)

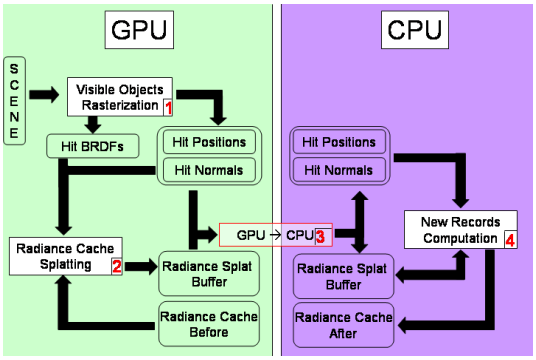


Figure 4: *The irradiance cache filling process. The numbers show the steps defined in Algorithm 2. During this process, the irradiance cache stored on the CPU is updated whereas the copy on the GPU remains untouched.*

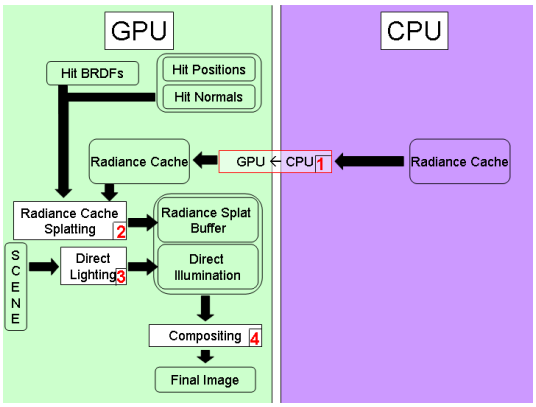


Figure 5: *The final rendering task. The numbers show the processing order described in steps 4 and 5 of Algorithm 2.*

memory. In Step 4, the algorithm traverses the radiance splat buffer to determine where new irradiance records are required to achieve the user-defined accuracy. For each pixel (x,y) in the radiance splat buffer, the cumulated weight is tested against the accuracy value a:

$$SPLATBUFF(x,y).w < a \tag{10}$$

If a pixel (x,y) passes this test, the existing cache records are insufficient to achieve the required accuracy. Therefore, a new record is generated at the location visible from pixel (x,y), and is splatted by the CPU onto the radiance splat buffer (Figure 4).

It should be noted that unlike previous approaches, the radiance cache splatting method do not rely on a specific traversal algorithm to ensure smooth interpolation. Therefore, the traversal of the radiance splat buffer can be performed linearly without introducing interpolation artifacts.

Once $SPLATBUFF(x,y).w \geq a$ for every pixel, the data stored in the cache can be used to display the indirect illumination according to the accuracy constraint. At that time in the algorithm, the irradiance cache stored on the CPU memory differs from the cache stored on the GPU: the copy on the GPU represents the cache before the addition of the records described above, while the copy on the CPU is up-to-date.

The last rendering step is the generation of the final image using the cache contents (Figure 5). The (ir)radiance cache on the GPU is updated, then the radiance cache splatting algorithm is applied on each newly generated cache record. Hence the radiance splat buffer contains the cumulated record weight and outgoing radiance contribution of all the (ir)radiance records. Then, as described in Eq. (1), the cumulated contribution of each pixel cumulated weight. This process yields an image of the indirect lighting in the scene from the current point of view. Combined with direct lighting, this method generates a fast, high quality global illumination solution.

As described above, our algorithm no longer relies on complex data structure for cache record storage. Moreover, the nearest-neighbors queries are replaced by simple sphere splatting and weighting function evaluation. These properties make the radiance cache splatting well-suited for GPU implementation.

5. Implementation

Our algorithm has been implemented using OpenGL and the OpenGL Shading Language (GLSL). This section gives technical information about the implementation of (ir)radiance cache splatting. After detailing the overall rendering algorithm, we discuss the use of GPU for (ir)radiance records computation.

5.1. Rendering Algorithm

As shown in Figures 4 and 5, our rendering algorithm makes intensive use of the GPU computational power. In this implementation, we have chosen to rely on the basic capability provided by graphics hardware: fast geometric primitive rasterization along with vertex and fragment processing. Hence the data required by our algorithm (Figure 3) is generated by rasterizing the scene geometry on the GPU using dedicated shaders. Figure 3 shows that this step includes the storage of per-pixel BRDF. In the case of diffuse BRDFs, the information consists of the diffuse reflectance of the material (i.e. one RGB value). In the case of glossy BRDFs, the per-pixel information is only one identifier. During splatting, this identifier is used to fetch the corresponding BRDF. The BRDFs are stored in a texture using the method described in [KSS02].

The (ir)radiance cache splatting can be performed either using the GPU or the CPU. As described in 4.2, our implementation uses both depending on the current context. The (ir)radiance cache splatting on the GPU is performed by drawing a quadrilateral tightly bounding the splatted sphere on the image plane. The position and size of the quadrilateral are computed using a vertex shader. Then, each fragment is processed so that its value represents the record's weighted contribution. The fragment is accumulated in the radiance splat buffer using the floating-point blending capabilities provided by graphics hardware. The final normalization (i.e. the division by the cumulated weight) is performed using a fragment shader in an additional pass, for the final display. The GPU implementation is either used to splat the whole (ir)radiance cache before adding new records, or to display the final image.

The traversal of the radiance splat buffer is performed on the CPU. Consequently, the GPU-based splatting cannot be used efficiently during the record addition step, since it would require to read back the radiance splat buffer from GPU memory once per added record. Hence our program needs a CPU implementation of (ir)radiance cache splatting. While this implementation is designed the same way as for

the GPU, Figure 3 shows that the (ir)radiance cache splatting requires information about the visible objects. This information is computed on the GPU, then read back to the CPU's memory once per frame. The overhead introduced by the data transfer from GPU to CPU turns out to be very small when using PCI-Express hardware.

Once all necessary records are computed and splatted on the radiance splat buffer, the final picture containing both direct and indirect lighting has to be generated. In our implementation, the direct lighting computation is carried out by the GPU: a fragment shader evaluates the BRDFs per-pixel, while shadow maps [Wil78,BP04] are used to simulate shadowing effects. The normalization of the radiance splat buffer and the combination with direct lighting is finally done in a fragment shader before displaying.

5.2. Record computation

The incoming (ir)radiance associated with a record is generated using both CPU and GPU. As in [SP89, LC04], our renderer uses the rasterization engine to sample the hemisphere above each record position in order to compute the incoming radiance and gradients. To compensate the incomplete hemisphere coverage Larsen et al. [LC04] divide the obtained irradiance by the plane coverage ratio. We propose a more accurate method, in which border pixels are extended so that they fill the remaining solid angle (Figure 6). Hence the incoming radiance is considered constant within the extension of each pixel. In our test scenes, this method yields more accurate results than the approach of Larsen et al.(Table 1). Moreover, a key aspect of this method is that the directional information of the incoming radiance remains plausible. Therefore, indirect glossy lighting can also be rendered correctly.

	Plane sampling	[LC04]	Our method
RMS Error	18.1%	10.4%	5.8%

Table 1: *RMS error of 10000 irradiance values computed in the Sibenik Cathedral scene. Our method yields more accurate results than previous approaches, while preserving the directional information of the incoming radiance.*

As shown in [LC04] the irradiance is defined by a weighted sum of the pixels of the sampling plane. This sum can be calculated either using the GPU and automatic mipmap generation [LC04], or using frame buffer readback and CPU-based summation (Figure 7). Due to the high transfer rates provided by PCI-Express, CPU-based computation turned out to be faster on the computer we used. However, the PCI-Express architecture does not avoid pipeline stalls: the GPU remains idle during the readback and during the computation of the records on the CPU. Future work will consider multithreading and scheduling for enhanced performance.

The same approach is used to compute the incoming radiance function for radiance cache records. Instead of computing the irradiance, we project the pixel values onto the hemispherical harmonics basis using the CPU.

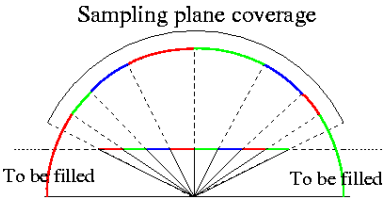


Figure 6: *The hemisphere sampling is replaced by rasterizing the scene geometry on a sampling plane. Since this plane does not cover the whole hemisphere, we use a border compensation method to account for the missing directions. Border pixels are extended to avoid zero lighting coming from grazing angles, yielding more plausible results.*

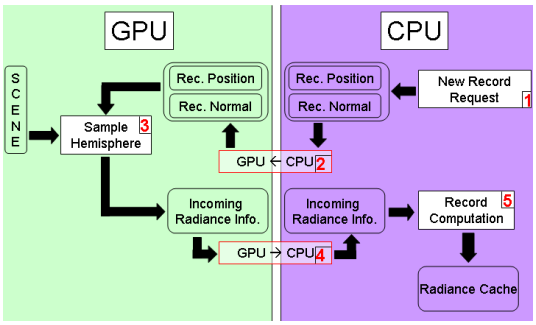


Figure 7: *New record computation process. The numbers show the order in which the tasks are processed.*

6. Results

This section discusses the results obtained using our implementation of radiance cache splatting. The images and timings presented here have been generated using an NVIDIA Quadro FX 3400 PCI-E and a 3.6 GHz Pentium 4 CPU with 1 GB RAM.

6.1. Fast High Quality Rendering

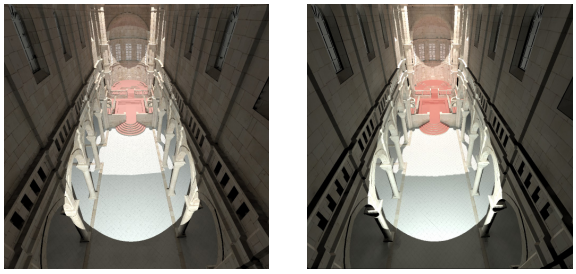
In this section, our aim is non-interactive, high quality rendering of global illumination. First, we compare the results obtained with our GPU-based renderer with the well-known Radiance [War04] software in the context of irradiance caching. Second, we discuss the results of radiance caching in glossy environments using our renderer.

We have compared the rendering speed of the Radiance software and our renderer in diffuse environments: the *Sibenik Cathedral* and the *Sponza Atrium* (Figure 9). The

images are rendered at a resolution of 1000×1000 and use a 64×64 resolution for hemisphere rasterization. The results are discussed hereafter, and summarized in Table 2.

a) Sibenik Cathedral This scene contains 80K triangles, and is lit by two light sources. The image is rendered with an accuracy parameter of 0.15. At the end of the rendering process, the irradiance cache contains 4076 irradiance records. The radiance cache splatting on the GPU is performed in 188 ms. The Radiance software rendered this scene in 7 min 5 s while our renderer took 14.3 s, yielding a speedup of about $30\times$.

b) Sponza Atrium This scene contains 66K triangles and two light sources. Using an accuracy of 0.1, this image is generated in 13.71 s using 4123 irradiance records. These records are splatted on the GPU in 242.5 ms. Using the Radiance software with the same parameters, a comparable image is obtained in 10 min 45 s. In this scene, our renderer proves about $47\times$ faster than the Radiance software.



(a) Radiance (b) Our renderer

Figure 8: *The Sibenik Cathedral scene (80K triangles). The images show first bounce global illumination computed with Radiance (a) and our renderer (b)*

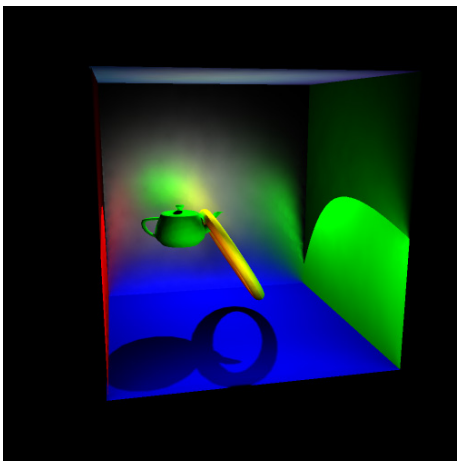
	Sibenik	Sponza
Triangles	80K	66K
Accuracy	0.15	0.1
Radiance time (s)	425	645
Our renderer time (s)	14.3	13.7
Speedup	29.7	47.1

Table 2: *Rendering times obtained using Radiance and our renderer for high quality rendering in diffuse environments. Each image is rendered at resolution* 1000×1000.

Our renderer also contains an implementation of radiance caching, hence supporting the computation of global illumination in glossy environments. If the environment also contains diffuse surfaces, our renderer uses either radiance or irradiance caching depending on the considered surface.

The *Cornell Box* scene presented in Figure 9(b) contains a glossy back wall (Phong BRDF, exponent 20), while the other objects are diffuse. The glossy BRDF and incoming

(a) Sponza Atrium



(b) Cornell Box

Figure 9: *Images obtained with our renderer. The Sponza Atrium (66K triangles) contain only diffuse surfaces. The Cornell Box (1K triangles) contains a glossy back wall.*

radiance function are projected into the hemispherical basis using order 10 representation. The accuracy parameters are 0.25 for both radiance and irradiance caching. Figure 9(b) was rendered in 12.18 s using 3023 irradiance records and 869 radiance records. The GPU-based splatting for the irradiance cache is performed in 65.91 ms. The radiance cache is splatted in 935.5 ms.

Figure 1 shows an example of radiance cache splatting in a more complex glossy environment: the *Castle* scene contains about 57K triangles. In this scene, the glossiness of the roofs is obtained using a Phong BRDF with exponent 15. The BRDF and incoming radiance functions are represented by order 5 projection on the hemispherical harmonics basis. The accuracy parameter we used is 0.25 for irradiance caching and 0.2 for radiance caching. At the end of the rendering process, the irradiance and radiance cache respec-

tively contain 3204 and 1233 records, computed in 10.1 s. The splatting on the GPU is performed in 58.6 ms for the irradiance cache. The radiance cache records are splatted in 493.7 ms.

The results presented above show that our algorithm is able to render fast high-quality glossy global illumination. The simplicity of our algorithm also allows its use for progressive rendering in interactive applications.

6.2. Interactive Visualization of Global Illumination

An important aspect of (ir)radiance caching is that the values of the records do not depend on the viewpoint. Therefore, records computed for a given frame can be reused in subsequent frames. Hence the radiance cache splatting approach can also be used in the context of interactive visualization of global illumination, and progressive rendering. Since the direct lighting is computed independently, the user can walk through the environment while the irradiance and radiance caches are filled on the fly. Figure 10 shows sequential images of *Sam* scene (63K triangles) obtained during an interactive session with an accuracy parameter of 0.5 and resolution 512×512. The global illumination is computed progressively, by adding at most 100 new records per frame. Our renderer provides an interactive frame rate (between 5 and 32 fps) during this session, allowing the user to move even if the global illumination computation is not completed. The accompanying videos presents interactive walkthroughs in diffuse and glossy environments: *Sam*, the *Sibenik Cathedral* and the *Castle*.

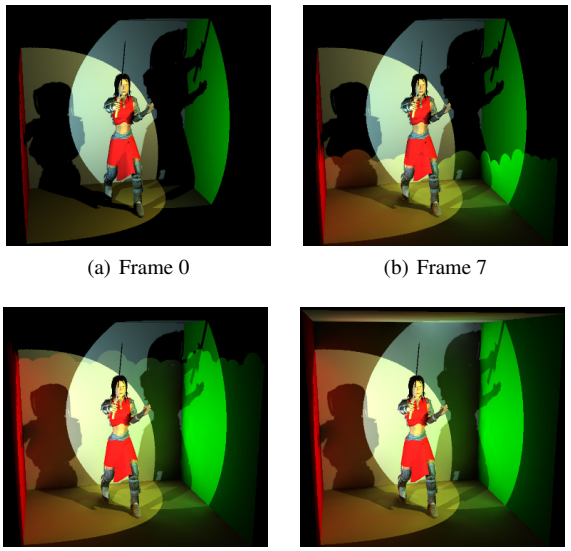
6.3. Discussion: Speedup Analysis

Previous subsections show that our method achieves significant speedup compared to the Radiance software by using the GPU for both record computation and final rendering.

In the (ir)radiance caching algorithm, the first and most expensive rendering step consists in computing the value of the cache records using hemisphere sampling. In this paper we propose to reduce this cost by using a sampling plane along with an accurate, novel compensation method. Therefore, the incoming radiance values are computed accurately using fast GPU rasterization and shadow maps.

Once all the needed (ir)radiance records are computed, the final rendering in Radiance relies on ray tracing and nearest-neighbors queries to calculate the outgoing radiance for each pixel. Although the cost involved is not dominant compared to the records computation, the speedup due to the simplicity of radiance cache splatting allows to display a globally illuminated scene at interactive rates.

Therefore, our global illumination method leverages graphics hardware using both novel and previous approaches, yielding a significant overall speedup while reducing the CPU workload.



(a) Frame 0 (b) Frame 7

(c) Frame 11 (d) Frame 14

Figure 10: *A progressive rendering session for interactive visualization of the* Sam *scene (63K triangles). Our renderer computes at most 100 new records per frame, hence maintaining an interactive frame rate (5 fps) during the global illumination computation. When the (ir)radiance cache is full, the global illumination solution is displayed at 32 fps.*

7. Conclusion and Future Work

In this paper, we proposed a reformulation of the (ir)radiance caching algorithm, by defining the radiance cache splatting. This method takes advantage of the latest graphics hardware to perform both the computation of irradiance and radiance records and the final rendering of global illumination. Our renderer shows a speedup of more than $29\times$ compared to the Radiance software for high quality rendering. Moreover, we show interactive performance for global illumination visualization in moderately complex scenes. To our knowledge, the radiance cache splatting is the first implementation of irradiance and radiance caching using programmable graphics hardware. We believe that our method could be integrated into film production renderers for fast and accurate computation of indirect illumination.

In the future, we plan to extend this method in several directions. Among them, some challenging improvements are the handling of multiple light bounces and higher frequency BRDFs. Moreover, future work will consider the extension of our algorithm to highly complex models using GPU-based optimizations such as in [BWPP04].

Even though our method shows significantly faster results than previous approaches, high quality global illumination is still not computed interactively. Therefore, we would also like to speed up the rendering process to reach real-time performance.

References

[BP04] BUNNELL M., PELLACINI F.: *GPU Gems: Shadow map antialiasing*, 1 ed. Addison Wesley, 2004, pp. 185–192.

[BWPP04] BITTNER J., WIMMER M., PIRINGER H., PURGATHOFER W.: Coherent hierarchical culling: Hardware occlusion queries made useful. In *Proceedings of Eurographics* (2004), pp. 615–624.

[CG85] COHEN M., GREENBERG D. P.: The hemi-cube: A radiosity solution for complex environments. In *Proceedings of SIGGRAPH* (1985), vol. 19, pp. 31–40.

[CHH02] CARR N. A., HALL J. D., HART J. C.: The ray engine. In *Proceedings of SIGGRAPH/Eurographics Workshop on Graphics Hardware* (2002), pp. 37–46.

[CHH03] CARR N. A., HALL J. D., HART J. C.: GPU algorithms for radiosity and subsurface scattering. In *Proceedings of SIGGRAPH/Eurographics Workshop on Graphics hardware* (2003), pp. 51–59.

[CHL04] COOMBE G., HARRIS M. J., LASTRA A.: Radiosity on graphics hardware. In *Proceedings of Graphics Interface* (2004), pp. 161–168.

[DS05] DACHSBACHER C., STAMMINGER M.: Reflective shadow maps. In *Proceedings of the Symposium on Interactive 3D graphics and games* (2005), pp. 203–231.

[GKPB04] GAUTRON P., KŘIVÁNEK J., PATTANAIK S., BOUATOUCH K.: A novel hemispherical basis for accurate and efficient rendering. In *Proceedings of Eurographics Symposium on Rendering* (2004), pp. 321–330.

[GTGB84] GORAL C. M., TORRANCE K. E., GREENBERG D. P., BATTAILE B.: Modelling the interaction of light between diffuse surfaces. In *Proceedings of SIGGRAPH* (1984), vol. 18, pp. 212–222.

[GWS04] GÜNTHER J., WALD I., SLUSALLEK P.: Realtime caustics using distributed photon mapping. In *Proceedings of Eurographics Symposium on Rendering* (2004), pp. 111–121.

[Jen01] JENSEN H. W.: *Realistic Image Synthesis Using Photon Mapping*. AK Peters, July 2001.

[KGBP05] KŘIVÁNEK J., GAUTRON P., BOUATOUCH K., PATTANAIK S.: Improved radiance gradient computation. In *Proceedings of SCCG* (2005), pp. 149–153.

[KGPB05] KŘIVÁNEK J., GAUTRON P., PATTANAIK S., BOUATOUCH K.: Radiance caching for efficient global illumination computation. *To appear in IEEE Transactions on Visualization and Computer Graphics* (2005).

[KSS02] KAUTZ J., SLOAN P.-P., SNYDER J.: Fast, arbitrary brdf shading for low-frequency lighting using spherical harmonics. In *Proceedings of Eurographics workshop on Rendering* (2002), Eurographics Association, pp. 291–296.

[LC04] LARSEN B. D., CHRISTENSEN N.: Simulating photon mapping for real-time applications. In *Proceedings of Eurographics Symposium on Rendering* (2004), pp. 123–131.

[LP03] LAVIGNOTTE F., PAULIN M.: Scalable photon splatting for global illumination. In *Proceedings of GRAPHITE* (2003), pp. 1–11.

[LSSS04] LIU X., SLOAN P.-P., SHUM H.-Y., SNYDER J.: All-frequency precomputed radiance transfer for glossy objects. In *Proceedings of Eurographics Symposium on Rendering* (2004), pp. 337–344.

[MM02] MA V. C. H., MCCOOL M. D.: Low latency photon mapping using block hashing. In *Proceedings of SIGGRAPH/Eurographics Workshop on Graphics Hardware* (2002), pp. 89–98.

[NPG03] NIJASURE M., PATTANAIK S., GOEL V.: Interactive global illumination in dynamic environments using commodity graphics hardware. In *Proceedings of Pacific Graphics* (2003), pp. 450–454.

[NPG04] NIJASURE M., PATTANAIK S., GOEL V.: Real-time global illumination on the GPU. *To appear in Journal of Graphics Tools* (2004).

[PBMH02] PURCELL T. J., BUCK I., MARK W. R., HANRAHAN P.: Ray tracing on programmable graphics hardware. In *Proceedings of SIGGRAPH* (2002), pp. 703–712.

[PDC*03] PURCELL T. J., DONNER C., CAMMARANO M., JENSEN H. W., HANRAHAN P.: Photon mapping on programmable graphics hardware. In *Proceedings of Graphics Hardware* (2003), pp. 41–50.

[SB97] STURZLINGER W., BASTOS R.: Interactive rendering of globally illuminated glossy scenes. In *Proceedings of Eurographics Workshop on Rendering* (1997), pp. 93–102.

[SHHS03] SLOAN P.-P., HALL J., HART J., SNYDER J.: Clustered principal components for precomputed radiance transfer. In *Proceedings of SIGGRAPH* (2003), pp. 382–391.

[SKS02] SLOAN P.-P., KAUTZ J., SNYDER J.: Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *Proceedings of SIGGRAPH* (2002), 527–536.

[SP89] SILLION F., PUECH C.: A general two-pass method integrating specular and diffuse reflection. In *Proceedings of SIGGRAPH* (1989), vol. 23, pp. 335–344.

[TL04] TABELLION E., LAMORLETTE A.: An approximate global illumination system for computer generated films. In *Proceedings of SIGGRAPH* (2004), pp. 469–476.

[TPWG02] TOLE P., PELLACINI F., WALTER B., GREENBERG D. P.: Interactive global illumination in dynamic scenes. In *Proceedings of SIGGRAPH* (2002), pp. 537–546.

[War94] WARD G. J.: The Radiance lighting simulation and rendering system. In *Proceedings of SIGGRAPH* (1994), pp. 459–472.

[War04] WARD G. J.: *Radiance Synthetic Imaging System*. http://radsite.lbl.gov/radiance, 2004.

[WBS03] WALD I., BENTHIN C., SLUSALLEK P.: Interactive global illumination in complex and highly occluded environments. In *Proceedings of Eurographics Symposium on Rendering* (2003), pp. 74–81.

[WH92] WARD G. J., HECKBERT P. S.: Irradiance gradients. In *Proceedings of Eurographics Workshop on Rendering* (1992), pp. 85–98.

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. In *Proceedings of SIGGRAPH* (1978), pp. 270–274.

[WRC88] WARD G. J., RUBINSTEIN F. M., CLEAR R. D.: A ray tracing solution for diffuse interreflection. In *Proceedings of SIGGRAPH* (1988), pp. 85–92.

[WS99] WARD G., SIMMONS M.: The holodeck ray cache: an interactive rendering system for global illumination in nondiffuse environments. *ACM Trans. Graph. 18*, 4 (1999), 361–368.

[WS03] WAND M., STRASSER W.: Real-time caustics. In *Proceedings of Eurographics* (2003), pp. 611–620.

[WTL04] WANG R., TRAN J., LUEBKE D.: All-frequency relighting of non-diffuse objects using separable BRDF approximation. In *Proceedings of Eurographics Symposium on Rendering* (2004), pp. 345–354.

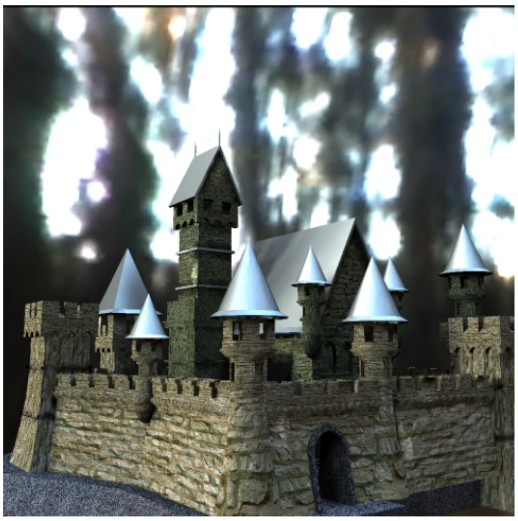
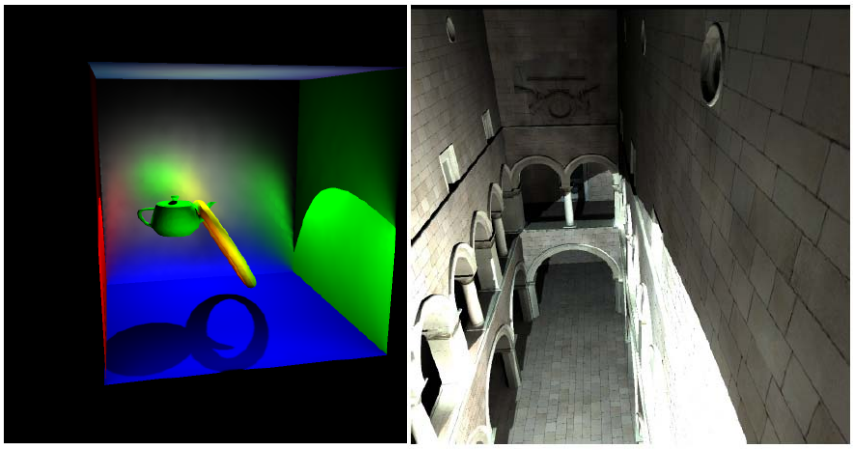
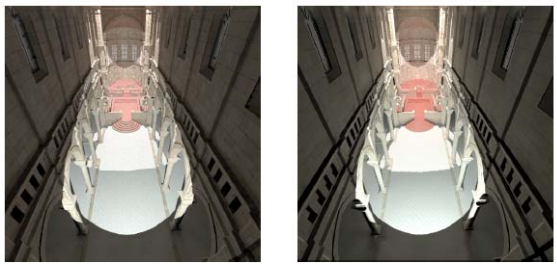


Figure 1: *The Castle scene (58K triangles) illuminated by an environment map. Our renderer computes first-bounce glossy global illumination in 10.1 s at resolution* 1000×1000.



(b) Cornell Box (a) Sponza Atrium

Figure 9: *Images obtained with our renderer. The Sponza Atrium (66K triangles) contain only diffuse surfaces. The Cornell Box (1K triangles) contains a glossy back wall.*



(a) Radiance (b) Our renderer

Figure 8: *The Sibenik Cathedral scene (80K triangles). The images show first bounce global illumination computed with Radiance (a) and our renderer (b)*