Interactive Global Illumination with Precomputed Radiance Maps

László Szécsi, László Szirmay-Kalos, and Mateu Sbert

Introduction

This article presents a real-time global illumination method for static scenes illuminated by arbitrary, dynamic light sources. The method consists of a preprocessing phase and a real-time rendering phase. The real-time rendering algorithm obtains the indirect illumination caused by the multiple scattering of the light from partial light paths that are precomputed and stored in the preprocessing phase. These partial light paths connect two points on the surface either directly or via one or more reflection points. Unlike Precomputed Radiance Transfer [Sloan02], the method of this article requires moderate preprocessing time, does not assume low-frequency hemispherical lighting, and can also work well for small light sources that may get close to the surfaces. The implemented version considers only diffuse reflections for the indirect illumination. To deal with complex scenes, a clustering scheme is also introduced, which trades storage space for high frequency details in the indirect illumination.

Problem statement

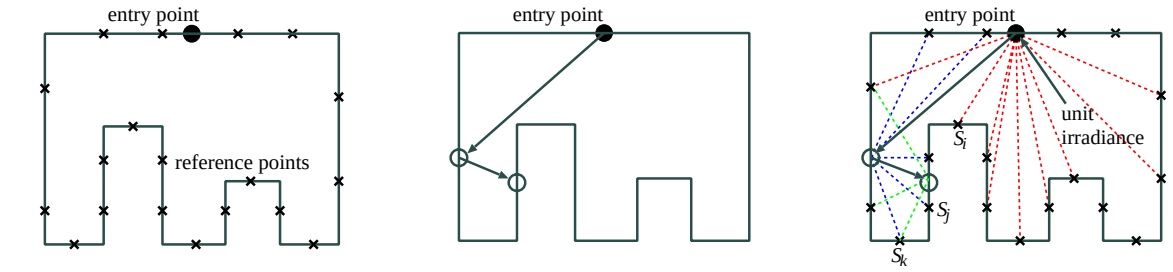
Rendering requires the identification of those light paths that connect light sources to the eye via reflections and then the computation of the sum of the path contributions. Accurate results require a high number of light paths, which are generally impossible to evaluate in real-time. However, in static scenes we can exploit the fact that those parts of light paths which connect surface points do not change when lights or the camera may move. This recognition allows us to precompute these light paths and combine the prepared data with the actual lighting conditions during real-time rendering [Szecsi06]. This means that having preprocessed the scene we can obtain global illumination results paying the cost of local illumination rendering.

Overview of the method

The proposed method consists of a preprocessing phase and a fast rendering phase.

Preprocessing

The preprocessing phase determines the self illumination capabilities of the static scene. This information is computed for finite number of *exit points* on the surface, and we use interpolation for other points. Exit points are depicted by symbol $\times$ in Figure 1. Exit points are defined as points corresponding to the texel centers of a texture atlas.



1. Exit points are defined and entry points are sampled

2. Path generation from the entry points

3. S is the illumination of exit points PRM = {(●, ×, S)}

Figure 1. Overview of the preprocessing phase. Entry points are depicted by ●, and exit points by ×. The PRM is a collection of (entry point ●, exit point ×, radiance S_k) triplets, called items.

The first step of preprocessing is the generation of certain number of *entry points* on the surface. These entry points are the samples of first hits of the light emitted by moving light sources. Entry points are depicted by symbol ● in Figure 1. Entry points are sampled randomly and are used as the start of a given number of light paths. A light path is a random or a quasi-random walk [Keller97] on the surfaces, which is terminated randomly according to Russian-roulette.

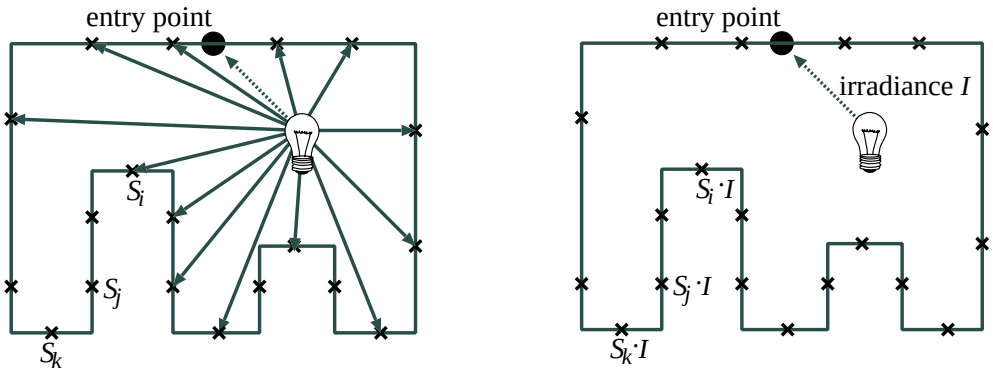
The power arriving at the points visited by a random walk is computed assuming that the particular entry point has unit irradiance. Then these visited points are considered as *virtual point lights* [Keller97,Wald02], which may illuminate all other points of the scene. While generating random walks is best done on the CPU, the GPU is better in computing the effect of virtual lights on exit points. Since the exit points are the texels of a texture map, the virtual light source algorithm should be implemented in a way that it renders into a texture map.

A virtual light source illuminates all those exit points that are visible from them, where the reflected radiance is obtained. The direct illumination caused by these virtual lights divided by the probability of the path is the Monte Carlo estimate of the global, i.e. direct and indirect illumination of the light source put at the entry point. The average of the Monte Carlo estimates of several paths associated with each entry and exit point pair is stored. We call this data structure the *Precomputed Radiance Map*, or *PRM* for short. Thus a PRM contains *items* corresponding to entry and exit point pairs. Items that belong to the same entry point constitute a PRM *pane*. A PRM pane is an array or a 2D texture of exit point radiances computed with the assumption that the corresponding entry point has unit irradiance while all other entry points have zero irradiance.

Rendering

The rendering step is realized completely on the GPU. During real-time rendering, PRM is taken advantage of to speed up the global illumination calculation. Lights and the camera are placed in the virtual world (Figure 2). The direct illumination effects are computed by standard techniques, which usually include some shadow algorithm to identify those points that are visible from the light source. PRM can be used to add the indirect illumination. This step requires visibility calculations,

which is for free, since this visibility information was already obtained during the direct illumination computation when shadows were generated [Dachsbacher05].



1. Direct illumination + entry point visibility

2. Weighting irradiance I with items S_i

Figure 2. Overview of the rendering phase. The irradiance of the entry points are computed, from which the radiance of the exit points is obtained by weighting according to the PRM.

A PRM pane (Figure 3) stores the indirect illumination computed for the case when the respective entry point has unit irradiance. During the rendering phase, however, we have to adapt to a different lighting environment. The PRM panes associated with entry points should be weighted in order to make them reflect the actual lighting. Computing the weighting factors involves a visibility check that can effectively be done in a shader using the shadow map already computed for direct illumination. This shader renders into a one-dimensional texture of weights. Although these values would later be accessible via texture reads, they can be read back and uploaded into constant registers for efficiency. Furthermore, zero weight textures can be excluded, sparing superfluous texture accesses.

In order to find the indirect illumination at an exit point, the corresponding PRM items should be read from the textures and their values summed having multiplied them by the weighting factors and the light intensity. We can limit the number of entry points to those having the highest weights. Selection of the currently most significant texture panes can be done on the CPU before uploading the weighting factors as constants.

Having obtained the radiance for each exit point, the scene is rendered in a standard way with linear interpolation between the exit points. Since exit points correspond to texel centers, the required linear interpolation is automatically provided by the bi-linear filtering of the texturing hardware.

Storing and compressing PRMs

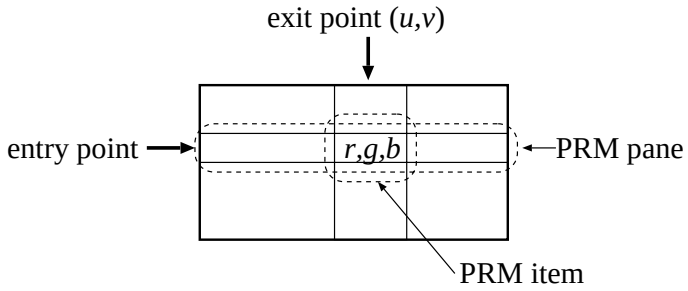


Figure 3. Representation of a PRM as an array indexed by entry points and exit points. A single element of this map is the PRM item, a single row is the PRM pane.

PRMs are stored in textures for real-time rendering. A single texel stores a PRM item that represents the contribution of all paths connecting the same entry point and exit point. A PRM can thus be imagined as an array indexed by entry points and exit points, and storing the radiance on the wavelengths of red, green, and blue (Figure 3). Since an exit point itself is identified by two texture coordinates, a PRM can be stored either in a 3D texture or in a set of 2D textures (Figure 4), where each 2D texture represents a single PRM pane (i.e. a row of the table in Figure 3), which includes the PRM items belonging to a single entry point.

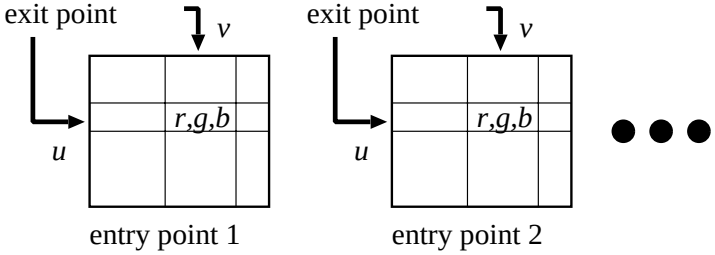


Figure 4. PRM stored as 2D textures.

The number of 2D textures is equal to the number of entry points. However, the graphics hardware has just a few texture units. Fortunately, this can be sidestepped by tiling the PRM panes into one or more larger textures. Tiling allows us to render indirect illumination interactively with a typical number of 256 entry points. While this figure is generally considered sufficient for a medium complexity scene, difficult geometries and animation may emphasize virtual light source artifacts as spikes or flickering, thus requiring even more samples. Simply increasing the number of entry points and adding corresponding PRM panes would quickly challenge even the latest hardware in terms of texture memory and texture access performance. To cope with this problem, we can apply an approximation (a kind of lossy compression scheme), which keeps the number of panes under control when the number of entry points increases.

The compression algorithm clusters those entry points that are close and are on a similarly oriented surface. Contributions of a cluster of entry points are added and stored in a single PRM pane. As these *clustered entry points* cannot be separated during rendering, they will all share the same weight when the entry point

contributions are combined. This common weight is obtained as the average of the individual weights of the entry points.

The key recognition behind clustering is that if two entry points are close and lay on similarly aligned surfaces, then their direct illumination will be probably very similar during the light animation. Of course this is not true when a hard shadow boundary separates the two entry points, but due to the fact that a single entry point is responsible just for a small fraction of the indirect illumination, these approximation errors can be tolerated and do not cause noticeable artifacts. This property can also be understood if we examine how clustering affects the represented indirect illumination. Clustering entry points corresponds to a low-pass filtering of the indirect illumination, which is usually already low-frequency by itself, thus the filtering does not cause significant error. Furthermore, errors in the low frequency domain are not disturbing for the human eye. Clustering also helps to eliminate animation artifacts. When a small light source moves, the illumination of an entry point may change abruptly, possibly causing flickering. If multiple entry points are clustered together, their average illumination will change smoothly. This way clustering also trades high-frequency error in the temporal domain for low-frequency error in the spatial domain.

Results

Figure 5 shows a marble chamber test scene consisting of 3335 triangles, rendered on 1024×768 resolution. We used 4096 entry points. Entry points were organized into 256 clusters. We set the PRM pane resolution to 256×256, and used the 32 highest weighted entry clusters. In this case the peak texture memory requirement was 128 Mbytes.

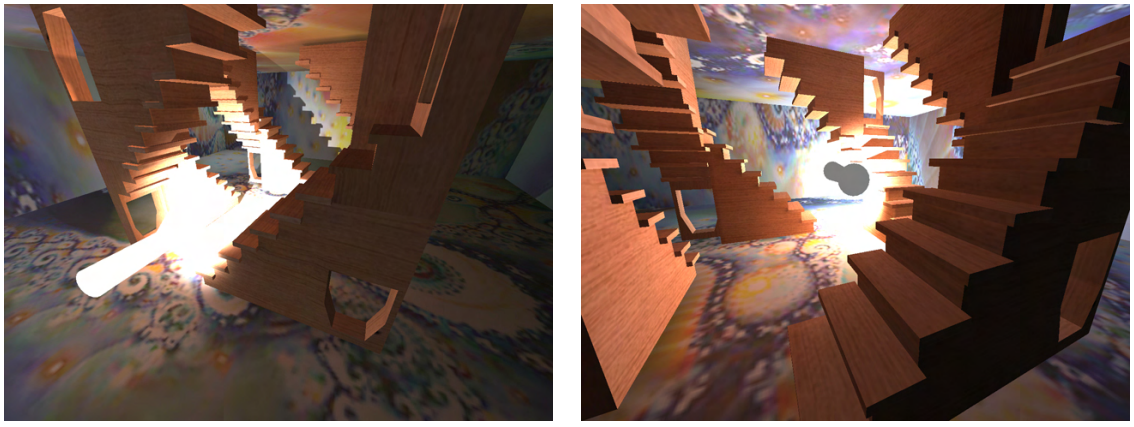
Figure 5. Marble chamber scene. The left images shows the distribution of randomly generated entry points. Middle and right images compare local illumination and the proposed global illumination rendering methods. The lower half of these images has been rendered with local illumination, while the upper half with the discussed global illumination method

For this scene, the preprocessing took 8.5 sec, which can further be decomposed as building the kd-tree for ray casting (0.12 sec), light tracing with ray casting (0.17 sec), and PRM generation (8.2 sec). Having obtained the PRM, we could run the global illumination rendering interactively changing the camera and light positions. Figure 5 also includes screen shots where half of the image was rendered with the new algorithm, and the other half with local illumination to allow

comparisons. The effects are most obvious in shadows, but also notice color bleeding and finer details in indirect illumination that could not be achieved by fake methods like using an ambient lighting term. We could measure 40 FPS and 200 FPS in full screen mode on NVidia GeForce 6800 GT and 8800 GTX graphics cards, respectively. The chairs scene of Figure 6 was also rendered with the same speed.

Figure 6. The chairs scene lit by a rectangular spot light and rendered with the proposed method.

Figure 7 shows a scene inspired by the stairs of Escher. The scene consists of 9 object, each having 32 entry point clusters. This scene is rendered at 30 FPS and at 180 FPS on NVidia GeForce 6800 GT and 8800 GTX graphics cards, respectively.



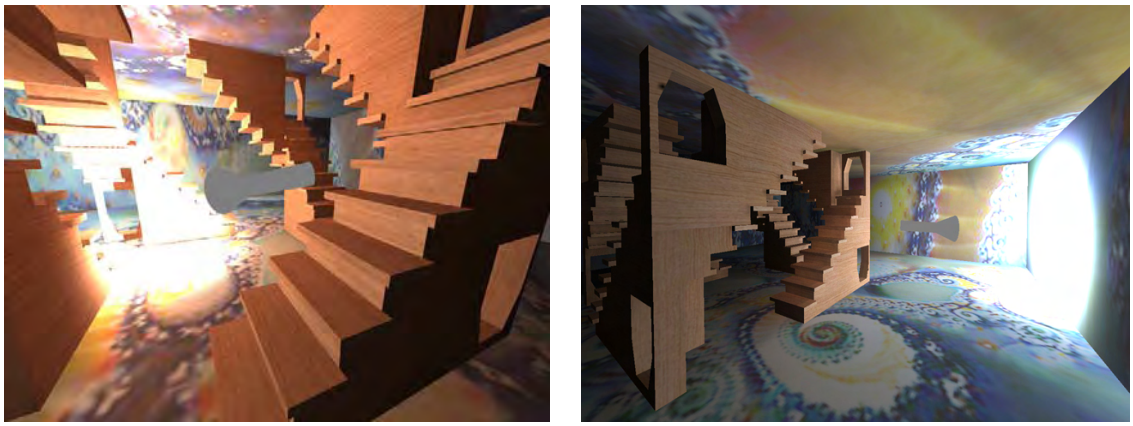


Figure 7. Escher type stairs in a globally illuminated room.

Figure 8 shows two screen shots of a demo game called Space Station developed in the framework of the GameTools project. This game does not use ambient light, but the indirect illumination of the static scene (the space station itself) and of dynamic characters is computed by the discussed Precomputed Radiance Maps method, and by localized environment maps [Lazanyi06], respectively.

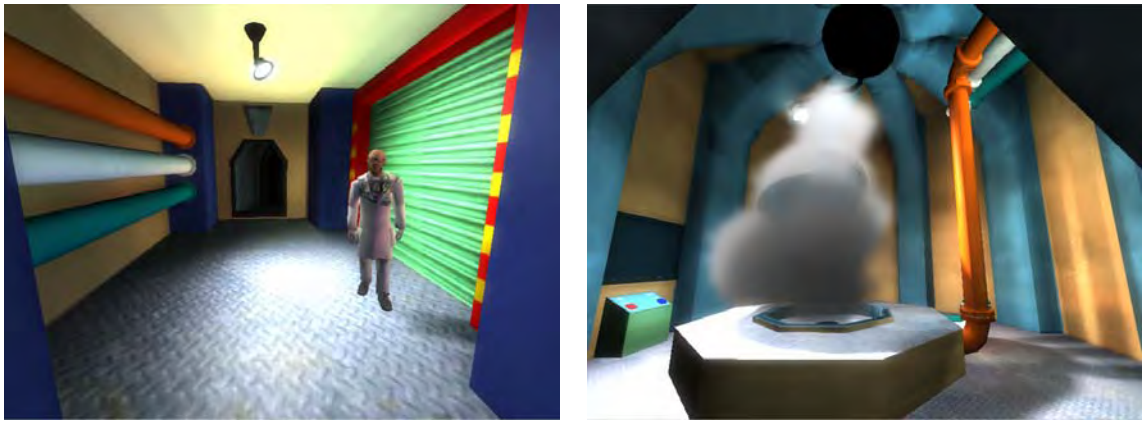


Figure 8. Integration of the Precomputed Radiance Maps method into a game called Space Station [GameTools07].

Conclusions and future work

The role of the presented method in games and interactive applications is similar to that of *light maps*. It also renders indirect lighting of static geometry, but unlike light maps, it allows for dynamic lighting and updates indirect shadows and color bleeding effects when light sources move.

Global illumination computations are performed in a precomputing step, using ray casting and the virtual light sources method. The contributions of virtual light samples are computed with depth mapping on the GPU. However, instead of computing a single light map, multiple texture atlases (constituting the PRM) are generated for the scene objects, all corresponding to a cluster of indirect lighting samples. These atlases are combined according to actual lighting conditions.

Weighting factors depend on how much light actually arrives at the sample points used for PRM generation, which is also computed in a GPU pass. The final result will be a plausible rendering of indirect illumination. Indirect shadows and color bleeding effects will appear, and illumination will change as the light sources move.

The presented algorithm makes a clear separation between PRM generation, which is done during preprocessing, and application, which is executed during run time. PRMs could also be computed incrementally adding or deleting entry points. Such incremental approaches have been proposed in [Sbert04] and [Laine07]. However, incremental updates have significant overhead and prohibits the application of compression schemes, thus our static approach is more suitable for game applications.

As a future work we plan to extend the method for scenes that has a larger static part and smaller dynamic objects. In this case, lighting invalidated by occlusions of dynamic objects should be removed, which is possible without recomputing the PRMs from scratch by working with negative light, called *antiradiance* [Dachsbacher07].

Demo

A real-time demo implemented using DirectX shader Model 3.0.

References

[Dachsbacher03] C. Dachsbacher and M. Stamminger. “Reflective shadow maps. *SI3D '05: Proc. of the 2005 Symp. on Interactive 3D Graphics and Games*, pages 203-231, 2005.

[Dachsbacher07] C. Dachsbacher, M. Stamminger, G. Drettakis, and F. Durand. “Implicit Visibility and Antiradiance for Interactive Global Illumination”, in SIGGRAPH 2007 Proceedings, 2007.

[GameTools07] http://www.gametools.org

[Keller97] A. Keller. “Instant radiosity”. In *SIGGRAPH '97 Proceedings*, 1997, pp 49-55.

[Lazanyi06] I. Lazányi and L. Szirmay-Kalos. “Indirect Diffuse and Glossy Illumination on the GPU”, in ShaderX 5 (edited by Wolfgang Engel), 2006, pp 345-358.

[Laine07] S. Laine, H. Saransaari, J. Kontkanen, J. Lehtinen, and T. Aila. “Incremental Instant Radiosity for Real-Time Indirect Illumination”, Eurographics Symposium on Rendering, 2007, pp 49-55.

[Sbert04] M. Sbert, L. Szécsi, and L. Szirmay-Kalos. “Real-time Light Animation”, Computer Graphics Forum (Eurographics 04), Volume 23, Number 3, 2004, pp 291-300.

[Sloan02] P. Sloan, J. Kautz, and J. Snyder. “Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments”. In *SIGGRAPH 2002 Proceedings*, 2002, pp 527-536.

[Szecsi06] L. Szécsi, L. Szirmay-Kalos, and M. Sbert. “Light animation with precomputed light paths on the GPU”, in *Graphics Interface 2006 Proceedings*, 2006, pp 187-194.

[Wald02] I. Wald, T. Kollig, C. Benthin, A. Keller, and P. Slussalek. “Interactive global illumination using fast ray tracing”, in *13th Eurographics Workshop on Rendering*, 2002.