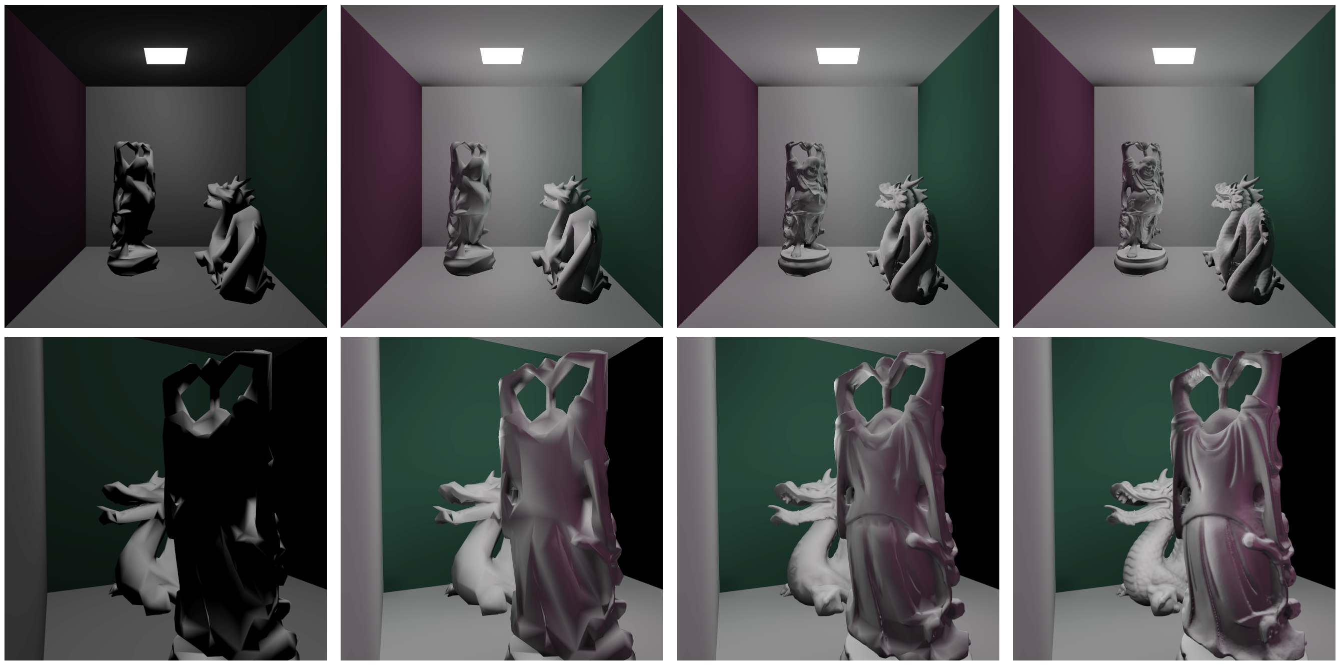
Efficient Streaming of 3D Scenes with Complex Geometry and Complex Lighting

Romain Pacanowski*[1,2] Mickaël Raynaud*[1] Xavier Granier*[1] Patrick Reuter*[1] Christophe Schlick*[1] Pierre Poulin†[2]
[1] INRIA Bordeaux University
Bordeaux (France)
[2]Dép. I.R.O., Université de Montréal
Montréal (Canada)



(a)
0.2% mesh, no IVG

(b)
0.2% mesh, 25% IVG

(c)
2% mesh, 25% IVG

(d)
100% mesh, 100% IVG

Figure 1: *Our irradiance vector grid structure is independent of the geometric complexity and allows our client/server visualization system to stream alternatively geometry data and lighting data to provide interactive rendering on the client side. (a) A scene with 0.2% of the geometry transferred and only direct illumination (without shadows). (b) The same amount of geometry with 25% indirect illumination transferred: color bleeding effects are now included, like on the surface of the buddha oriented toward the red wall that appears more red. (c) Further refinement of the geometry (2%). (d) Full-resolution geometry (50 MB) and full-resolution (1 MB) irradiance vector grid. This scene runs at 50fps on a NVIDIA GeForce Go 7800 GTX.*

Abstract

Streaming data to efficiently render complex 3D scenes in presence of global illumination is still a challenging task. In this paper, we introduce a new data structure based on a 3D grid of irradiance vectors to store the indirect illumination appearing on complex and detailed objects: the Irradiance Vector Grid (IVG). This representation is independent of the geometric complexity and is suitable for quantization to different quantization schemes. Moreover, its streaming over network involves only a small overhead compared to detailed geometry, and can be achieved independently of the geometry. Furthermore, it can be efficiently rendered using modern graphics hardware. We demonstrate our new data structure in a new remote 3D visualization system, that integrates indirect lighting streaming and progressive transmission of the geometry, and study the impact of different strategies on data transfer.

CR Categories: I.3.2 [Computer Graphics]: Graphics Systems—Distributed/network graphics; C.2.4 [Computer-Communication Networks]: Distributed Systems—Client/server; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Shading; Radiosity; RayTracing;

Keywords: 3D Streaming, Global Illumination

1 Introduction

With the recent increase of collaborative Virtual Reality applications, realistic online games, and other remote 3D graphics appli-

*{pacanows | raynaud | granier| reuter | schlick}@labri.fr
†poulin@iro.umontreal.ca

cations, improved realism calls upon global illumination effects. In a remote 3D visualization system, a straightforward solution is to implement a full real-time simulation on the client side. The main advantage of such a solution is the low requirement of data transfer, since only the standard description of the scene is needed. Unfortunately, real-time simulation techniques are limited in the number of indirect light-bounces and heavily depend on the geometric complexity both for speed and accuracy. Therefore, in a client/server context, it can thus be preferable to keep the computation of global illumination on the server side, and to stream the computed indirect illumination to the client. However, if the illumination is directly linked with the geometry, the overhead would be proportional to the size of geometry. In order to limit the transmission overhead, an illumination data structure independent of the geometric complexity would be valuable. This also allows one to select independently the relative quality of the illumination and of the geometry.

As a solution to this problem, we propose a new approach for remote visualization of 3D scenes. We consider that scenes are stored on a server with all the required information: meshes for the 3D geometry, description of material properties, and indirect illumination for each object. To reach our goal, we introduce the following contributions:

- A new structure for indirect illumination based on a volumetric grid of irradiance vectors [Arvo 1994]. Our structure is independent of the geometric complexity and can be easily rendered using modern GPUs.
- A streaming technique for the progressive transmission of this indirect illumination combined with a number of compression schemes.
- A client/server visualization system built around this structure, with independent streaming of geometry and indirect illumination. The less-demanding direct illumination is computed on the client side.

After a brief presentation of some previous work on structures for global illumination, we introduce our new representation (cf. Section 3) and its associated schemes for compression, streaming, and interactive rendering. Then, we present in detail our client/server visualization system (cf. Section 4), and analyze the resulting different data transfers, rendering quality, and framerate (cf. Section 5).

2 Previous Work

Global Illumination (e.g., [Dutré et al. 2006]) has been extensively studied in computer graphics. With the recent increase of computational power of graphics hardware, it is now possible to develop interactive techniques that incorporate global illumination effects. Radiosity techniques can be implemented on modern GPUs [Nijasure et al. 2003; Coombe et al. 2004], capturing the computed indirect lighting into environment maps. Based on a reformulation of the rendering equation [Kajiya 1986], the introduction of *anti-irradiance* [Dachsbacher et al. 2007] removes the problem of visibility estimation, but requires a number of passes proportional to the scene depth complexity. Stochastic solutions have also been implemented on GPUs, such as those based on *Photon-Mapping* [Purcell et al. 2003] or radiance caching [Gautron et al. 2005]. Real time is only achieved with Instant Radiosity [Keller 1997; Segovia et al. 2007] combined with incremental techniques [Laine et al. 2007], but its visual quality strongly depends on the geometric accuracy. In a streaming context, this is hardly achievable.

The common approach in previous work has been to precompute lighting and to store it as a diffuse lightmap. However, a highly detailed geometry requires a highly detailed map to capture all the lighting variations. For almost-planar surfaces or slowly varying geometric details, one can use directional light maps [Hey and Purgathofer 2002] (a texture that stores for each texel a directional light source).

For more general lighting effects, *Precomputed Radiance Transfer* (PRT) [Sloan et al. 2002] has emerged as an efficient representation. PRT encodes precomputed light transport effects of a static scene by projection onto a pre-defined basis. Even though the viewpoint can be changed in recent techniques [Liu et al. 2004; Wang et al. 2004; Ng et al. 2004; Tsai and Shih 2006], PRTs are still limited to distant lighting. Moreover, although the resulting data can be reduced [Sloan et al. 2003] or adapted to normal-mapping [Sloan 2006], its size is still large and strongly depends on the geometric complexity.

Extending work on PRT, recent approaches compute a more accurate global illumination. The Precomputed Radiance Transfer Field [Pan et al. 2007] allows interactive rendering of interreflections in-between dynamic objects. However, the required data size and computation time are huge, even for small undetailed geometry. Another solution is to precompute the direct-to-indirect transfer (e.g., Wang *et al.* [Wang et al. 2007]), and let the hardware efficiently compute the direct lighting. Unfortunately, the final data size is still dependent on the geometric complexity.

The work most related to ours is the Irradiance Volume [Greger et al. 1992] and its extension used by different game engines (e.g. [Mitchell et al. 2006; Mitchell et al. 2007b]). The original Irradiance Volume is a bilevel grid that stores irradiance values at each vertex position. However, game engines use regular grids that can be more easily implemented on GPU. Furthermore, the spatial interpolation scheme, to ensure a smooth reconstruction of the irradiance, is more complex than a classical trilinear interpolation scheme used with a regular grid. Finally, storing irradiance values at vertex positions has two drawbacks. First, irradiance is a geometric dependent value; we prefer to use irradiance vectors [Arvo 1994] that are more robust to geometric variation. Second, in order to reconstruct the irradiance for any normal, many irradiance values have to be stored, increasing furthermore the storage cost of the irradiance volume. Instead we propose a new basis and reconstruction scheme (related to [Mitchell et al. 2007a]) that allows smooth interpolation of the irradiance and requires less storage.

Overview. Our system exploits the same kind of separation between direct and indirect illumination. On the server side, the 3D scene is stored together with the indirect illumination associated with its embedded light sources and material properties. During the streaming of the 3D scene, we also transfer the indirect illumination for complex objects, using a new data structure, called Irradiance Vector Grid (IVG, for short) On the client side, the hardware accelerated direct lighting is combined with the transferred indirect lighting to produce an interactive and accurate global illumination solution.

3 Representation of Indirect Lighting

The Irradiance Vector Grid is an axis-aligned uniform rectangular 3D grid structure where each grid vertex stores six irradiance vectors (cf. Section 3.1). The grid is used by the graphics hardware on the client side to compute the indirect illumination. This is accomplished by interpolating (cf. Section 3.2) spatially and directionally the irradiance vectors of the grid. Our representation can be efficiently compressed (cf. Section 3.3) and easily uploaded on the client graphics hardware.

3.1 Irradiance Vector

For a given wavelength, the *irradiance vector* $\boldsymbol{I_n}(\boldsymbol{p})$, as introduced by Arvo [Arvo 1994], is defined for a point $\boldsymbol{p}$ with normal $\boldsymbol{n}$ as

$$\boldsymbol{I_n}(\boldsymbol{p}) = \int_{\Omega_n} L(\boldsymbol{p} \leftarrow \boldsymbol{\omega}_i)\, \boldsymbol{\omega}_i \, d\boldsymbol{\omega}_i \qquad (1)$$

where $L(\boldsymbol{p} \leftarrow \boldsymbol{\omega}_i)$ represents the incident radiance at $\boldsymbol{p}$ from direction $\boldsymbol{\omega}_i$, $d\boldsymbol{\omega}_i$ the differential solid angle sustained by $\boldsymbol{\omega}_i$ and $\Omega_{\boldsymbol{n}}$ the hemisphere centered at $\boldsymbol{p}$ oriented toward $\boldsymbol{n}$. The irradiance vector stores radiometric and geometric information; it is directly related to the diffusely reflected radiance:

$$L_r(\boldsymbol{p} \rightarrow \boldsymbol{\omega}_o) = \frac{\rho_D}{\pi} \left(\boldsymbol{I_n}(\boldsymbol{p}) \cdot \boldsymbol{n} \right) \qquad (2)$$

where ρ_D is the diffuse BRDF at point $\boldsymbol{p}$ and $\cdot$ denotes the dot product. The main benefit of irradiance vectors compared to irradiance is that for a local variation of a surface normal, the reflected radiance can be adjusted, making this representation more **geometrically robust**. To evaluate Equation 2, one may use any global illumination algorithm such as Photon-Tracing or Path-Tracing. Intuitively, an irradiance vector represents the intensity of the incident lighting and the mean direction where it comes from.

Bear in mind that we want to compute the reflected radiance $L_r(\boldsymbol{p})$, where the normal at $\boldsymbol{p}$ may be along any direction. Therefore, we need an efficient representation to store the incident illumination for any direction. We thus subdivide the direction space with six overlapping hemispheres, where each hemisphere is oriented toward a main direction $\boldsymbol{\delta} = \pm\boldsymbol{x}|\pm\boldsymbol{y}|\pm\boldsymbol{z}$. We precompute an irradiance vector $I_{\boldsymbol{\delta}}$ for each of the six hemispheres to represent the incident illumination. With an appropriate interpolation scheme, we combine the different values of $I_{\boldsymbol{\delta}}$ to evaluate $L_r(\boldsymbol{p})$ for any normal.

3.2 Interpolation of Irradiance Vectors

In order to compute smooth indirect illumination, we interpolate an irradiance vector for each point $\boldsymbol{p} = (p_x, p_y, p_z)$ with normal $\boldsymbol{n} = (n_x, n_y, n_z)$ that needs to be shaded. This interpolation is performed in two successive steps: a spatial interpolation according to $\boldsymbol{p}$ and then a directional interpolation according to $\boldsymbol{n}$. In the first step, the irradiance vector $\boldsymbol{I_\delta}(\boldsymbol{p})$ is obtained by performing either a trilinear or a tricubic spatial interpolation of the irradiance vectors stored at the grid vertices surrounding point $\boldsymbol{p}$. The interpolation is only done for three out of the six possible directions of $\boldsymbol{\delta}$. The choice between $\pm\boldsymbol{x}$ (resp. $\pm\boldsymbol{y}$ and $\pm\boldsymbol{z}$) is done according to the sign of n_x (resp. n_y and n_z). In the second step, the final interpolated irradiance vector $\boldsymbol{I_n}(\boldsymbol{p})$ is obtained by remapping the three spatially interpolated irradiance vectors according to the normal direction $\boldsymbol{n}$ at point $\boldsymbol{p}$:

$$\boldsymbol{I_n}(\boldsymbol{p}) = \boldsymbol{I_x}(\boldsymbol{p})\, n_x^2 + \boldsymbol{I_y}(\boldsymbol{p})\, n_y^2 + \boldsymbol{I_z}(\boldsymbol{p})\, n_z^2 \,.$$

Notice that our directional interpolation is exact when the normal $\boldsymbol{n}$ is aligned with $\boldsymbol{\delta}$. To achieve real-time performance, these interpolations are directly executed on the client GPU once the irradiance vector grid is uploaded on it. However, to reduce the footprint on video memory, we need to compress our grid, as detailed in the next section.

3.3 Irradiance Vector Compression and GPU Rendering

Remember that Equation 1 defines an irradiance vector (requiring three floats) for a single wavelength. Since, in computer graphics chrominance is defined by three primary colors (R, G, B), we need, for each $\boldsymbol{\delta}$, three irradiance vectors stored in a 3×3 matrix $\boldsymbol{M} = (\boldsymbol{I}^R_{\boldsymbol{\delta}} | \boldsymbol{I}^G_{\boldsymbol{\delta}} | \boldsymbol{I}^B_{\boldsymbol{\delta}})$. For a given grid vertex and a given $\boldsymbol{\delta}$, we compress $\boldsymbol{M}$ as the product of a direction $\boldsymbol{d}$ and a color $\boldsymbol{c}$ defined as follows:

$$\boldsymbol{d} = \frac{\boldsymbol{I}^R + \boldsymbol{I}^G + \boldsymbol{I}^B}{\|\boldsymbol{I}^R + \boldsymbol{I}^G + \boldsymbol{I}^B\|} \qquad \boldsymbol{c} = \left[\frac{\boldsymbol{I}^R \cdot \boldsymbol{\delta}}{\boldsymbol{d} \cdot \boldsymbol{\delta}},\ \frac{\boldsymbol{I}^G \cdot \boldsymbol{\delta}}{\boldsymbol{d} \cdot \boldsymbol{\delta}},\ \frac{\boldsymbol{I}^B \cdot \boldsymbol{\delta}}{\boldsymbol{d} \cdot \boldsymbol{\delta}} \right].$$

This guarantees that when the normal $\boldsymbol{n}$ is aligned with $\boldsymbol{\delta}$, we preserve the original RGB intensity: $\boldsymbol{Mn} = \boldsymbol{c}(\boldsymbol{d} \cdot \boldsymbol{n})$. We have tested experimentally that this compression does not introduce any artifact in the indirect lighting interpolation. To reduce the required bandwidth usage even further, the direction can be quantized on 24 bits (classical quantization with 8 bits for each coordinate) and the color on 32 bits, using the GPU-compatible R9_G9_B9_E5 format[1] similar to the RGBE format [Ward 1991]. This compression can be done before transmitting the grid data (cf. Section 4.1).

Finally, the great benefit of using a 3D regular grid is that the data structure can be straightforwardly uploaded on the GPU as a 3D texture. In our case, the interpolated irradiance vectors are simply used by the fragment shader as additional light sources that are meant to encode indirect illumination. These two vectors are encoded in two 3D textures, and therefore the information for the six $\boldsymbol{\delta}$ directions requires 12 3D textures. To reconstruct the indirect lighting, one may use trilinear interpolation natively provided by the hardware, or adapt a tricubic interpolation technique [Sigg and Hadwiger 2005].

4 Our Remote Visualization System

Our visualization system is based on a client/server architecture. The server precomputes and stores the lighting structures and the level of detail (LOD) of each complex geometry represented as a progressive mesh. The server sends either new geometric (cf. Section 4.2) or lighting (cf. Section 4.1) level of detail depending on the client requests. After each data reception, the client performs some processes on the illumination structure and on the progressive mesh before uploading them on the GPU. Moreover, our approach allows to interleave geometric and lighting data when transmitting the scene from the server to the client. This offers a very smooth progressive visualization until the desired quality is reached.

4.1 Irradiance Vector Grid Streaming

A classical solution for the progressive transfer of the texture is to use a hierarchical decomposition based on recursive basis functions such as wavelets. However, to reconstruct each hierarchical level, this decomposition techniques require waiting for the reception of all corresponding detail coefficients. Thus, the time required to obtain each new resolution level is growing with its size. We propose here an alternative approach, which allows the transmission of constant size bundles of voxels.

The transmission is initialized by transferring the eight corners of the grid. Then, each client request consists of a constant number of irradiance vectors. Notice that the number of irradiance vectors per request can be dynamically set depending on the client GPU/CPU capabilities as well as the network bandwidth and reliability. To get a smooth global update of the indirect illumination, we have implemented a stratified random sampling of the grid. In our current implementation, the grid is divided in a set of slices along its longest axis. Then, at each client request, the server sends the requested number of irradiance vectors. The locations of the irradiance vectors are randomly distributed on each slice.

[1]This HDR color compression is supported in DirectX 10 and in OpenGL through the extension GL_EXT_texture_shared_exponent.

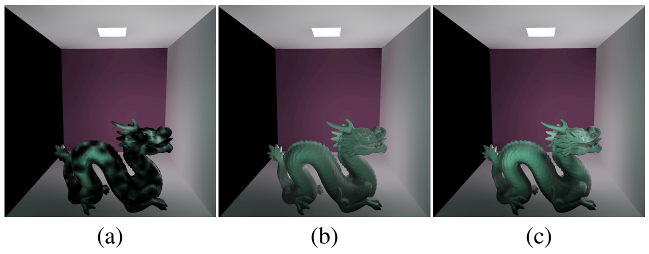


Figure 2: *Comparison of indirect illumination reconstructed on the right side of the dragon (a) without our push-pull algorithm, (b) with and (c) with a push-pull algorithm without smoothing. The grid dimension is* $16 \times 16 \times 16$ *and half of the vertices have been transferred. The push-pull process has filled the darker regions of the dragon with smooth indirect illumination.*

Unfortunately, when streaming the IVG, some illumination holes may appear until the grid has been fully transferred. This comes from the fact that the incomplete grid holds invalid irradiance vectors at some locations. We have adapted a 3D push-pull algorithm to fill the missing irradiance vectors with smooth data interpolation (cf. Figure 2). Notice that our push-pull algorithm does not modify any received data. To get smoother results, we apply after each push step a pyramidal filter to the current grid level. Finally, the push-pull process can be skipped if the client has limited CPU capabilities. Therefore, the minimal hardware requirement for our client is programmable graphics hardware with 3D texture support.

The direction and color of each irradiance vector is sent either in floating point format or quantized. Our experimentations have shown that using quantization reduces the transfer time by a factor of about 2.5 without introducing any visible artefact. Indeed, for the scene presented in Figure 1, the mean difference between an image reconstructed with and without quantization is only 0.008% in *Lab* color space. Obviously, the dequantization process must be performed on the client side in order to perform the push-pull steps, but the resulting overhead is very small (cf. the chart of Figure 6).

4.2 Geometry Streaming

Since our illumination technique is independent of the geometry, any encoding scheme could be used. Inspired by Hoppe's seminal work on encoding progressive meshes by successively applying edge collapse operators [Hoppe 1996], we adopted a streaming technique that progressively refines a coarse mesh into a detailed mesh by the inverse operation: the vertex split. For a memory-efficient geometry streaming, we trade a slight loss in quality against a very compact multiresolution mesh representation: instead of successively collapsing the less-significant edge (determined by some energy function [Garland and Heckbert 1997]) onto an additional vertex along the edge, we simply collapse the edge onto one of its end vertices. Consequently, the entire multiresolution mesh can be encoded from the original mesh as an indexed face set, with a very small overhead. This overhead corresponds to the vertex lookup table that indicates the vertex index for every vertex to which it is collapsed [Melax 1998].

For a progressive transmission of the multiresolution mesh, we reorder the vertices so that the most significant vertices (that are present in the coarser meshes) can be streamed first. The progressive transmission then consists of alternating sequences of vertices, of faces, and of small vertex lookup tables. Whereas the received vertices are directly transferred to the GPU, the vertex indices involved in the faces are first updated recursively by using the vertex lookup tables. This process offers $n-2$ different level of detail (where n is the total number of vertices) and guarantees that only valid faces are streamed. Our technique is somehow similar to [Guéziec et al. 1999].

5 Results and Discussion

We have tested our remote 3D visualization system with an Intel Q6600 with 4GB memory as a server and an Intel Pentium M 2.26Ghz with 2GB memory as a client. We have measured all network transmission times on a 802.11g WiFi network. Each measurement has been repeated and averaged.

5.1 Independence of Geometry and Lighting

As a demonstration of the independence of geometry and lighting, we have tested our remote visualization system with two extreme streaming strategies. In the first one (cf. Figure 3), we first transfer a low-resolution geometry with a full-resolution IVG. We then progressively transfer all the remaining details of the geometry. Thanks to our vectorial representation, the illumination adapts smoothly to local variations of the geometry during its refinement without requiring more information.

In the second strategy (cf. Figure 4), we transfer a full-resolution geometry with low-resolution IVG, that is progressively refined. This strategy is useful when the server refines the illumination in a dynamic environment by using an incremental solution for global illumination (e.g., [Dmitriev et al. 2002]). This is also useful if we want the server to remotely compute the illumination and progressively transfer the computed IVG nodes using a parallel version of our algorithm.

One main advantage of our separation between the illumination structure and the geometry structure is that the client adapts its data refinement demands according to both hardware and network capabilities. The classical strategy that offers the smoothest progressive visualization is to use an interleave streaming of illumination and of geometry (cf. Figure 1). Our tests show that the client framerate remains constant, when updating either the geometry or the illumination grid on the GPU. Therefore our system provides a real-time and continuous feedback to the client. Moreover, for a given scene, we did not measure any framerate performance penalty when introducing the illumination grid.

5.2 Transfer Time

We have also tested separately the required streaming times for the geometry and the illumination grid. In order to test the influence of the streaming process on the transmission time, we have compared the time required to download the full structure with the time required to transfer the geometry/grid without the overhead of the streaming (cf. cyan horizontal curve of Figure 5). On the geometry side, as illustrated in Figure 5, the maximum overhead was 70%, but only for small packet sizes. When increasing the packet size, we quickly reduce this overhead to only 10%. This is a very good trade-off for a streaming solution compared to a classical LOD mechanism of VRML where the overhead can reach 100%. The overhead of our streaming solution is due to the transfer of the vertex lookup table, the edge collapses, and the transfer to the GPU.

On the illumination side, we have tested both the streaming of quantized and floating point grids. As expected, the network transfer time is reduced when using a quantized grid (cf. black curve on the two graphs of Figure 6). Moreover, the comparison of the two red curves of Figure 6 demonstrates that the data dequantization process is very small (approximately 4%). This dequantization step

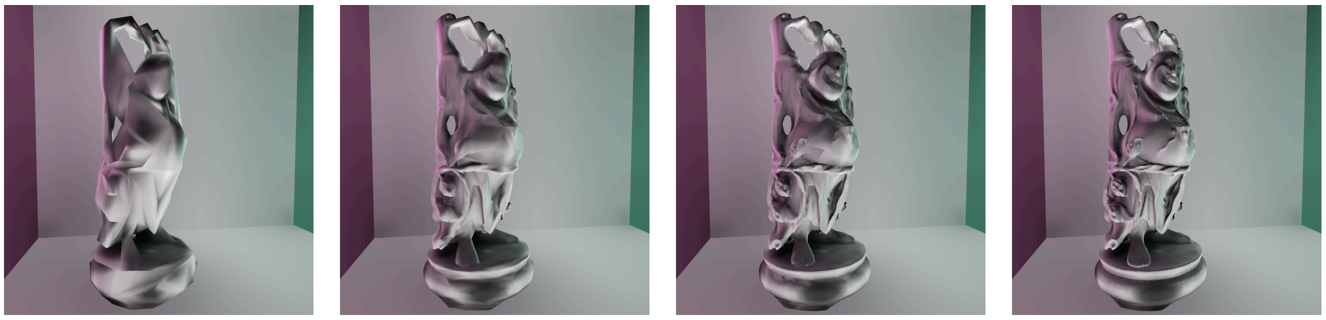


Figure 3: *Streaming the geometry within a constant illumination* $16 \times 16 \times 16$ *grid (563 KB).* ***Left to right****: the indirect illumination adapts itself when the geometry is refined by using a buffer size of 250 vertices. For each image, the timing indicated below represents the total time required to reach the indicated geometry size from one on its immediate left. The faces oriented toward the red wall are always red. The indirect illumination and color bleeding effects are coherently represented.*

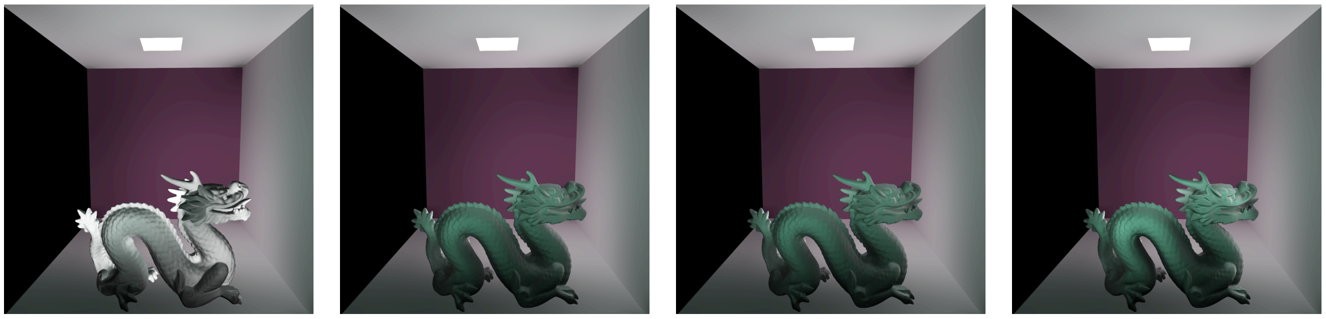


Figure 4: *Streaming a* $16 \times 16 \times 16$ *illumination grid (dequantized) for the same geometry. (a) The initial illumination grid with only 8 irradiance vectors (ivs) is refined (b)-(d) with 64 samples per slice and per request. For each image, the timing indicated below represents the total time required to reach the indicated geometry size from one on its immediate left.*

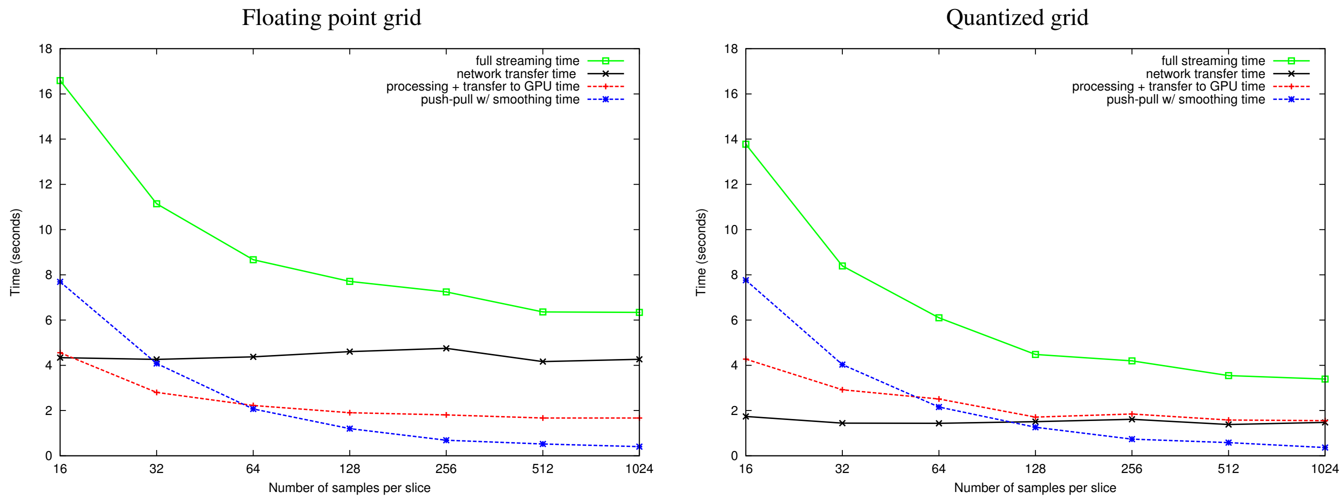


Figure 6: *Downloading time for the complete illumination grid (32×32×32) with different buffer sizes. The size of the floating point grid is* ***3 MB*** *and the size of the quantized grid is* ***1.3 MB****. The processing includes the dequantization process and the copy to CPU memory.*

is required to perform the push-pull algorithm without introducing numerical errors. The time spent on the push-pull algorithm is constant per grid size, but the number of push-pull processes depends on the number of client requests. Therefore, the packet size

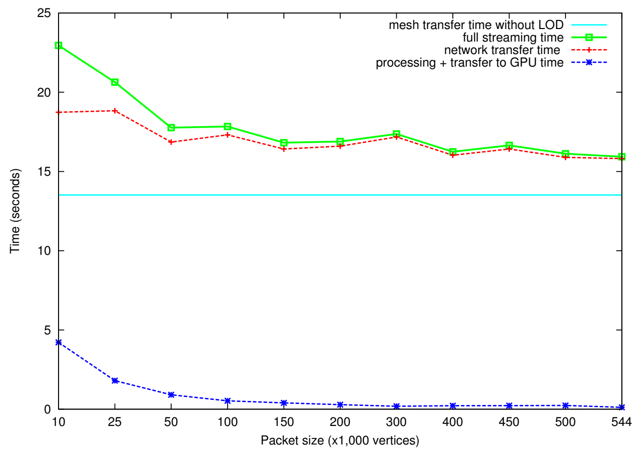


Figure 5: *Downloading time for the complete buddha geometry of Figure 1 with different packet sizes (a packet corresponds to a fixed number of vertices). The full geometry size is* ***1.1 M*** *polygons. The processing includes the update of the vertex indices by using the vertex lookup table.*

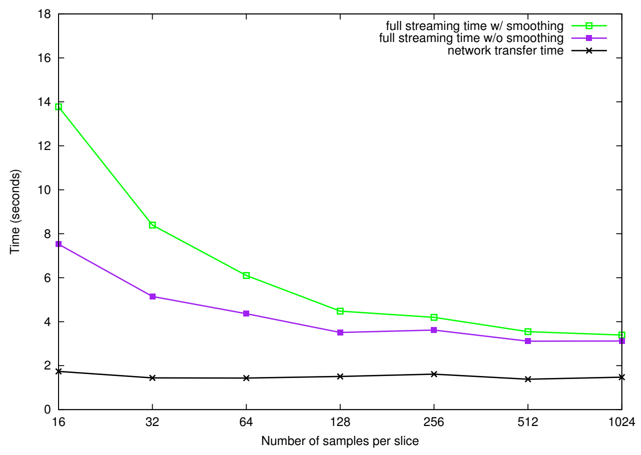


Figure 7: *Comparison of downloading time with and without smoothing during the push-pull step for a quantized* $32 \times 32 \times 32$ *grid. The smoothing computation represents 80% of the total time spent for the push-pull process.*

has to be chosen carefully depending on client/server and network capacities. According to Figure 6, when using a $32 \times 32 \times 32$ quantized illumination grid, the appropriate packet size is reached when asking approximately 100 samples per slice (the corresponding abscissa when the black curve crosses the blue curve). Finally, as illustrated in Figure 7, most of the computation overhead is due to the smoothing pass of the 3D push-pull algorithm. However, depending on client capabilities, this step can be skipped and as shown in Figure 2, the penalty on quality remains small.

6 Conclusion and Future Work

We have introduced a new structure to efficiently represent and transfer the indirect illumination of a 3D scene in a remote rendering context: the irradiance vector grid. This structure is based on a 3D grid of irradiance vectors. The main advantage of our IVG structure is to be independent of the geometric complexity and to be small compared to the geometry size. Our structure integrates easily with geometry streaming techniques in a remote visualization system to quickly provide global illumination effects for complex geometries. Furthermore, the transfer time overhead induced by our structure is very small as well as the overhead on the performance at rendering time.

For future work, we would like to improve both the server and the client of our system. On the server side, we want to develop new precomputation schemes that take advantage of cluster architectures for on-line computation of requested illumination. We also want to provide a fast update mechanism for dynamic 3D scenes, with a localized computation in regions of important changes. In order to reduce the processing time on the client side for the illumination, new algorithms for the push-pull process need to be developed.

References

ARVO, J. 1994. The irradiance jacobian for partially occluded polyhedral sources. In *Proc. SIGGRAPH '94*, ACM, 343–350.

COOMBE, G., HARRIS, M. J., AND LASTRA, A. 2004. Radiosity on graphics hardware. In *Proc. Graphics Interface 2004*, Canadian Human-Computer Communications Society, 161–168.

DACHSBACHER, C., STAMMINGER, M., DRETTAKIS, G., AND DURAND, F. 2007. Implicit visibility and antiradiance for interactive global illumination. *ACM Trans. Graph. 26*, 3.

DMITRIEV, K., BRABEC, S., MYSZKOWSKI, K., AND SEIDEL, H.-P. 2002. Interactive global illumination using selective photon tracing. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 25–36.

DUTRÉ, P., BALA, K., AND BEKAERT, P. 2006. *Advanced Global Illumination*. A. K. Peters, Ltd.

GARLAND, M., AND HECKBERT, P. S. 1997. Surface simplification using quadric error metrics. In *Proceedings of ACM SIGGRAPH 1996*, ACM, 209–216.

GAUTRON, P., KŘIVÁNEK, J., BOUATOUCH, K., AND PATTANAIK, S. 2005. Radiance cache splatting: a gpu-friendly global illumination algorithm. In *Proc. Eurographics Symposium on Rendering*, 55–64.

GREGER, G., SHIRLEY, P., HUBBARD, P., AND GREENBERG, D. 1992. Irradiance volume. *IEEE Comp. Graph. and Appl. 18*, 2, 32–43.

GUÉZIEC, A., TAUBIN, G., HORN, B., AND LAZARUS, F. 1999. A framework for streaming geometry in vrml. *IEEE Comput. Graph. Appl. 19*, 2, 68–78.

HEY, H., AND PURGATHOFER, W. 2002. Real-time rendering of globally illuminated soft glossy scenes with directional light maps. Tech. Rep. TR-186-2-02-05, Institute of Computer Graphics and Algorithms, Vienna University of Technology.

HOPPE, H. 1996. Progressive meshes. In *Proceedings of ACM SIGGRAPH 1996*, ACM, 99–108.

KAJIYA, J. T. 1986. The rendering equation. In *Comp. Graph. (Proc. ACM SIGGRAPH '86)*, vol. 20, 143–150.

KELLER, A. 1997. Instant radiosity. In *Proc. SIGGRAPH '97*, ACM Press/Addison-Wesley Publishing Co., 49–56.

LAINE, S., SARANSAARI, H., KONTKANEN, J., LEHTINEN, J., AND AILA, T. 2007. Incremental instant radiosity for real-time indirect illumination. In *Proc. Eurographics Symposium on Rendering 2007*, 277–286.

LIU, X., SLOAN, P. P., SHUM, H. Y., AND SNYDER, J. 2004. All-frequency precomputed radiance transfer for glossy objects. In *Proc. Eurographics Symposium on Rendering*, 337–344.

MELAX, S. 1998. A simple, fast and effective polygon reduction algorithm. *Game Developer Magazine* (November), 209–216.

MITCHELL, J., MCTAGGART, G., AND GREEN, C. 2006. Shading in valve's source engine. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses*, ACM, New York, NY, USA, 129–142.

MITCHELL, J., FRANCKE, M., AND ENG, D. 2007. Illustrative rendering in team fortress 2. In *Symposium on Non-Photorealistic Animation and Rendering*, ACM, 71–76.

MITCHELL, J. L., FRANCKE, M., AND ENG, D. 2007. Illustrative rendering in team fortress 2. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, ACM, New York, NY, USA, 19–32.

NG, R., RAMAMOORTHI, R., AND HANRAHAN, P. 2004. Triple product wavelet integrals for all-frequency relighting. *ACM Trans. Graph. 23*, 3, 477–487.

NIJASURE, M., PATTANAIK, S., AND GOEL, V. 2003. Interactive global illumination in dynamic environments using commodity graphics hardware. In *Proc. Pacific Conference on Computer Graphics and Applications*, IEEE Computer Society, 450.

PAN, M., WANG, R., LIU, X., PENG, Q., AND BAO, H. 2007. Precomputed radiance transfer field for rendering interreflections in dynamic scenes. *Comp. Graph. Forum 26*, 3, 485–493.

PURCELL, T. J., DONNER, C., CAMMARANO, M., JENSEN, H. W., AND HANRAHAN, P. 2003. Photon mapping on programmable graphics hardware. In *Proc. ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, 41–50.

SEGOVIA, B., , IEHL, J.-C., AND PEROCHE, B. 2007. Metropolis instant radiosity. *Comp. Graph. Forum 26*, 3, 425–434.

SIGG, C., AND HADWIGER, M. 2005. *GPU Gems 2*. Addison-Wesley, ch. Fast Third-Order Texture Filtering, 313–330.

SLOAN, P.-P., KAUTZ, J., AND SNYDER, J. 2002. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Trans. Graph. 21*, 3, 527–536.

SLOAN, P.-P., HALL, J., HART, J., AND SNYDER, J. 2003. Clustered principal components for precomputed radiance transfer. *ACM Trans. Graph. 22*, 3, 382–391.

SLOAN, P.-P. 2006. Normal mapping for precomputed radiance transfer. In *Proc. Symposium on Interactive 3D Graphics and Games*, ACM, 23–26.

TSAI, Y.-T., AND SHIH, Z.-C. 2006. All-frequency precomputed radiance transfer using spherical radial basis functions and clustered tensor approximation. *ACM Trans. Graph. 25*, 3 (July), 967–976.

WANG, R., TRAN, J., AND LUEBKE, D. 2004. All-frequency relighting of non-diffuse objects using separable brdf approximation. In *Proc. Eurographics Symposium on Rendering*, 345–354.

WANG, R., ZHU, J., AND HUMPHREYS, G. 2007. Precomputed radiance transfer for real-time indirect lighting using a spectral mesh basis. In *Proc. Eurographics Symposium on Rendering*, 967–976.

WARD, G. 1991. *Graphics Gems 2*. Morgan Kaufman Publisher, ch. Real Pixels, 80–83.