Real-Time Global Illumination for VR Applications

Jesper Mortensen, Insu Yu, and Pankaj Khanna ▪ *University College London*

Franco Tecchia and Giuseppe Marino ▪ *Scuola Superiore Sant'Anna*

Bernhard Spanlang ▪ *Universitat Politècnica de Catalunya*

Mel Slater ▪ *ICREA—University of Barcelona*

Real-time global illumination in VR systems enhances scene realism by incorporating soft shadows, reflections of objects in the scene, and color bleeding. The Virtual Light Field (VLF) method enables real-time global illumination rendering in VR. The VLF has been integrated with the Extreme VR system for real-time GPU-based rendering in a Cave Automatic Virtual Environment.

This article presents a solution for the rendering of immersive VR using real-time global illumination. One of the important reasons for using VR in an application is that participants should respond realistically to virtual objects and events—for example, in applications concerned with training or rehabilitation. Therefore, investigating the factors that might be critical in producing such realistic responses is an important area of scientific investigation, with implications for the engineering of successful VR applications.[1]

An obvious factor to consider is the realism of the illumination. While a static globally illuminated scene is well within the grasp of current methods, it places limits on the range of environments and tasks that can be effectively represented. When scenes are rendered with global illumination, they not only look more realistic but also critically have dynamic shadows and reflections of objects, most importantly of the virtual body of the participant interacting in the VR. In other words, as participants move through the VR they would see real-time correlations of their activities through not only changing object reflections but also dynamic changes of the shadows and reflections of their body. This can have an anchoring effect that profoundly situates the participant within the virtual environment.[2]

The problems faced in rendering with real-time global illumination, however are twofold: First, the computational complexity of such a real-time rendering system while achieving an acceptable frame rate is daunting. Second, we are faced with a choice of how this is to be achieved—one option is to construct an ad hoc system with support for related tasks such as tracking, display management, and synchronization. Alternatively, a general solution requires integration of the global illumination method within a more general rendering system, such as Performer–CAVELib (an API for the Cave Automatic Virtual Environment), DIVE (Distributed Interactive Virtual Environment), or XVR (Extreme VR), which can again be a complex task.

In this article, we present the Virtual Light Field (VLF) paradigm as a solution to this problem. We discuss its potential and advantages in such an application. Finally, we present details of our integration of the VLF rendering method within XVR[3] to provide a practical real-time global illumination solution.

Background

The key elements for visual realism are accurate geometric models and their realistic illumination. In this research we consider only the illumination aspect of visual realism. Ray tracing[4] and radiosity[5] provide a partial global illumination solution and have both been considered for VR rendering. Whereas ray tracing extends easily to real-time dynamics including shadows and reflections, performance is limited by the complexity of the scene and its elements as well as the required screen resolution and update rate. The number of primary rays for a CAVE (Cave Automatic Virtual Environment) application running in stereo at a resolution of 1,024 × 768 at 30 frames per second is 47 million rays (Mrays) per second. The state of the art in efficient ray tracing[6] can achieve roughly 40 Mrays/sec., so even a single shadow ray per pixel would exceed the ray budget.

Global illumination techniques such as path tracing and photon mapping, among others, offer a more complete illumination solution, supporting a larger range of materials and light interactions, but they do not easily extend to real-time rendering. At least 20–40 rays per pixel are needed to achieve global illumination;[7] this exceeds the ray budget we just mentioned by an order of a magnitude and will not be applicable for CAVE rendering.

In VR applications the frame rate must be real-time and constant; even temporary drops in frame rate can cause the participant to lose orientation or cause simulator sickness. A nonstable frame rate is a weakness of many caching algorithms, where a sudden change in viewpoint can produce a view that is not fully represented in the cache, causing a temporary drop in fidelity or frame rate. Similarly, dynamic techniques such as ray tracing for global illumination can also exhibit variable frame rates when the viewpoint changes from a complex region to a less complex region in terms of illumination.

Precomputed Radiance Transfer[8] offers an approximation to global illumination for static scenes and has been applied to rendering in a CAVE.[9] The preprocessed static scene is illuminated by dynamic environment maps for realistic rendering. The light field presents an image-based approach for representing and rendering radiance information from real or virtual scenes.[10,11] The advantage of such a representation is that rendering is independent of scene complexity—in both the number of polygons and surface materials. Pankaj Khanna and his colleagues[12] utilize a direction-and-point parameterized (DPP) light field data structure[13] storing visibility for accelerating ray tracing, and subsequently Peijie Huang and his colleagues employed a surface light field for that purpose, supporting rigid dynamics.[14] Similarly, Zhong Ren and his colleagues precompute and store visibility for fast global illumination computation of low-frequency lighting at interactive frame rates.[15]

The Virtual Light Field uses a 5D DPP light field to propagate and represent global illumination in a scene for real-time rendering.[16] Unlike many current techniques in virtual and augmented/mixed reality applications that approximate physically based rendering, the VLF provides a true solution, representing all $L(S|D) * E$ light paths. After propagation, this radiance information is available for rendering and relighting. We believe such a representation has significant potential for allowing realistically illuminated virtual environments, although the rendering method can equally be used to represent a "real-world" light field for VR applications at high, stable frame rates.

A Brief Overview of the VLF

We have previously presented the fundamental VLF data structure and algorithm.[16] A 2D grid of rays all parallel to the z-axis is called the canonical parallel subfield (PSF). This canonical PSF is intersected with all objects in the scene (which requires only a 2D rasterization algorithm). Multiple rotations of the canonical PSF are formed, each again intersected with the scene. Thus each ray in the data structure belongs to one and only one PSF and has a (possibly) empty sequence of surface intersections along it. Any arbitrary ray through the scene will have a set of nearest-neighbor rays in the PSF data structure. In practice, if we take any rectangular grid of rays, we can exploit coherence by dividing these into smaller rectangular tiles, and keep a set of surface identifiers within each tile indicating the set of surfaces intersected by at least one ray in that tile. This also helps in visibility calculations and results in a massive reduction in memory requirement.

A general solution requires integration of the global illumination method within a more general rendering system, such as Performer–CAVELib.

$L_s(\omega, s, t, u, v, p)$, where ω indicates PSF_ω, (s, t) is a tile for face p, and (u, v) is a cell within this tile. Our approach uses texture atlases for both radiance and irradiance maps for improved efficiency and compact representation.

Once the VLF data structure is built, propagation is in principle a straightforward Neumann expansion of the rendering equation. Radiance is emitted from light sources following the paths provided by fixed bundles of parallel rays in the PSFs, which are used as approximations for true ray directions. Coherence is exploited by following parallel bundles of rays rather than dealing with individual rays. This method maps well to the GPU, providing a very efficient light transport step. The method can provide solutions with tens of thousands of polygons with millions of radiance or irradiance elements in minutes.[17]

Rendering the VLF in the CAVE

When the VLF propagation step has converged, the GPU can render novel views from the data structure by interpolating between samples stored in the diffuse textures and nondiffuse view-dependent radiance tiles. Diffuse surfaces can be rendered directly using texturing with the diffuse textures available in the irradiance texture atlases. The GPU performs interpolation efficiently in this case.

Flat specular faces can be rendered with ray tracing by recursively following a view ray reflected in the specular face until it strikes a diffuse face where the visible radiance can be collected. A similar idea, often used in real-time VR applications, is to use the stencil buffer to render a reflected view of the scene as seen through the specular face and then paste this onto the face with texturing.[18] These methods are efficient only if few specular surfaces are present in the scene and do not apply to, for example, glossy bidirectional reflectance distribution functions (BRDFs).

A more general method is to resample images from the directionally dependent radiance stored in the nondiffuse radiance tiles. As we described in the section "A Brief Overview of the VLF," the data structure can be formalized as $L_s(\omega, s, t, u, v, p)$. This effectively references a radiance value in direction ω, from a point on p described by the intersection of the canonical ray (s, t, u, v) with p. Owing to the discrete representation, a PSF matching exactly the direction ω is rarely available. The three PSFs $(\omega_i, \omega_j, \omega_k)$ at the vertices of the spherical triangle in which ω falls are used with barycentric weights $(\alpha_i, \alpha_j, \alpha_k)$ for an interpolated value:

$$\begin{aligned} L_s(\omega, s, t, u, v, p) = \; & \alpha_i * L_s(\omega, s, t, u, v, p) \\ & + \alpha_j * L_s(\omega_j, s, t, u, v, p) \\ & + \alpha_k * L_s(\omega_k, s, t, u, v, p) \end{aligned} \quad (1)$$

In order to compute the values necessary to index into Equation 1, four off-screen passes are rendered. A fifth and final pass performs the final shading, producing the globally lit image. In order to identify nondiffuse pixels in the image plane, an optional stencil image can be produced by rendering the nondiffuse polygons to an off-screen target. This can serve to limit the computation performed in each subsequent pass to only nondiffuse pixels.

In Pass 1 the camera is placed at the center of the unit sphere, and the spherical triangles are rendered in false color to a texture. This produces the indices of the three nearest PSFs $(\omega_i, \omega_j, \omega_k)$ for each pixel.

This is repeated in Pass 2, this time setting vertex colors for each spherical triangle to (1, 0, 0), (0, 1, 0), and (0, 0, 1). The GPU interpolates this over each triangle, resulting in a texture with three barycentric weights for each pixel.

Pass 3 serves to determine p, this time rendering the scene geometry in false color, yielding a texture with a face identifier for each visible nondiffuse pixel.

Pass 4 renders the scene geometry again, where each vertex is colored with its world coordinate (WC) position; interpolation across the geometry produces a texture with the WC position of the intersection of the viewing ray for that pixel with the face p. Note that ray casting could easily replace these last two passes. A fifth and final pass renders the final radiances to the image. For each pixel this is achieved by mapping the hit position to each of the three PSFs by applying the respective $\mathbf{M}_{WC \rightarrow PSF}$ matrix, which maps from WCs to PSF coordinates to the hit position, producing an (x, y, z) value in canonical PSF coordinates where (x, z) trivially maps to a tile/cell pair (s, t, u, v). The tiled data structure is then looked up, and a radiance value for each PSF is weighted by its corresponding barycentric weight and written to the image.

Performance is dependent on the time taken to resolve visibility (Pass 3); the remaining passes and radiance retrieval involve a small constant time per pixel. Either ray tracing or rasterization can be used to resolve the visibility; here, we use the latter. One of the main points of the VLF approach is that global illumination values can be retrieved directly from the data structure; no further shadow rays or sampling is necessary. This results in stable, predictable frame rates, which is of great utility in VR applications.

Dynamics Integration

Integrating dynamic elements in a global illumination solution is a difficult task. The computational resources needed to solve the rendering equation numerically at real-time frame rates are currently not readily available. Popular approaches attempting this are ray tracing[7] and hierarchical finite-element approaches.[19] At the time of writing, these cannot deliver the frame rates and resolution needed for VR applications. Making some simplifying assumptions can, however, make the problem tractable. If we separate the scene geometry into dynamic and static elements we can precompute the global illumination for the static elements using the VLF and focus on the dynamic elements at runtime.

Dynamic elements undergoing only rigid-body animation can be easily integrated using Precomputed Radiance Transfer,[8] where the VLF can provide the input radiance. This approach does not apply to elements such as virtual characters (avatars) using skinned animation, which are a crucial element of many VR applications. The mesh of an avatar is typically made up of thousands of triangles essentially undergoing unstructured motion, making it virtually impossible to accelerate through a precompute approach. However, by breaking up the problem and attacking the modes of transport that contribute most to the image, we can achieve real time and still support significant global illumination effects.

We can separate the problem into three main modes of transport contributing to the image and focus on solving them:

- field radiance scattered off the avatar toward the eye,
- soft shadows cast by the avatar, and
- specular reflections of the avatar.

This does not solve for diffuse reflections of the avatar; thus, color bleeding caused by the avatar will not be accounted for. However, the magnitude of illumination that has undergone multiple diffuse reflections is generally low and will add little to the image.

In order to solve the field radiance problem, we need to be able to rapidly provide the irradiance at an arbitrary spatial position; a shader program can then use this to calculate the surface shading at points on the mesh. In order to provide irradiance calculations at real-time frame rates, we introduce another precomputed data structure derived from the VLF. The bounding volume of the scene is subdivided into a set of voxels. A voxel stores irradiance retrieved from the VLF projected to a spherical harmonic; owing to the properties of spherical harmonics, they can be calculated at arbitrary positions by trilinear interpolation of the eight nearest voxels. Such irradiance volume was suggested by Ravi Ramamoorthi and Pat Hanrahan[20] and by Gene Greger and his colleagues,[21] albeit in a different form.

Physically correct soft shadows are notoriously difficult to calculate. However, perceptually correct soft shadows can be rendered in real time using the GPU. Percent Closer Soft Shadows samples a standard shadow map stochastically to provide approximate umbra and penumbra regions of a shadow due to an area light source.[22] In order to combine this with the physically correct soft shadows (cast by static elements) already present in the scene, the visibilities of shadow-mapped light sources are also stored as texture maps in the VLF. This information is trivially available during the VLF precompute.

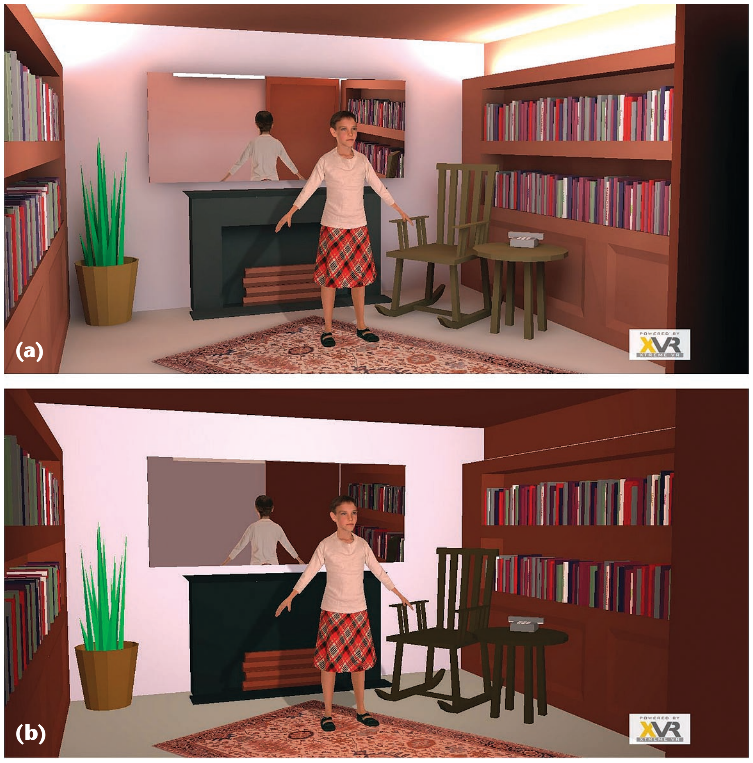

Figure 1. Integrating dynamic elements into a static scene: (a) a virtual character in a globally lit scene showing color bleeding, caustics, and soft shadows, as well as specular reflection; and (b) an OpenGL rendering. Note the realistic appearance due to global illumination effects, in contrast to standard OpenGL rendering, which is commonplace in VR research.

Reflections (and caustics) are already appropriately accounted for in the VLF for the static parts of the scene, but this obviously does not include the dynamic elements. Also, depending on the resolution of the VLF and the BRDF of the surface, they might benefit from reconstruction using the scene geometry. This can easily be achieved in real time using a reflection rendering pass, rendering the visible scene onto a reflective surface.[18] This can be extended to curved surfaces using traditional environment-mapping techniques.

The effect of these techniques used in conjunction is quite striking. The dynamic elements merge well with the surrounding scene, featuring impinging color bleeding and caustics and casting soft shadows, as well as being visible in reflective surfaces (see Figure 1a). In contrast, to obtain the OpenGL rendering (see Figure 1b), the ambient terms and other direct-lighting coefficients had to be painstakingly adjusted to obtain the overall warm look of the globally lit image. Even then, the image looks flat owing to missing soft shadows and color-bleeding effects.

Implementation

A system for real-time rendering in a VR system such as a CAVE requires more than a suitable rendering algorithm. Operating system, networking, synchronization, and tracking issues must all be combined into one overall application that affords not just real-time rendering but also the creation of a coherent VR. The VLF rendering method has been integrated into XVR,[3] which is a stand-alone integrated development environment for the rapid development of complex VR applications. The XVR Network Renderer module has been used in order to distribute the graphic load on a local-area network (LAN).

We chose XVR because, in addition to real-time graphics rendering, it includes the ability to handle many collateral aspects of VR programming.

The XVR Framework

We chose XVR as our implementation framework because in addition to real-time graphics rendering, it includes the ability to handle many collateral aspects of VR programming, such as sound, haptics, and interaction. To allow for CAVE applications development, XVR has a dedicated module called the Network Renderer.[23] This module allows for cluster-based rendering of XVR applications following a "sort-first" approach.[24] By using a cluster of workstations, the rendering load is distributed among several machines. In particular, we let each PC of the cluster manage the rendering of a different screen of the CAVE system.

The Network Renderer is totally transparent to the original XVR application: each of the OpenGL calls performed by the master application is intercepted by the module, which catches all the information about the calls and stores them into an internal memory buffer. Each time that it is necessary, and according to the rules of an internal synchronization protocol, XVR's Network Driver module sends the content of the buffer to a set of remote executables, called graphic slaves, which run on the machines composing the cluster.

In order to minimize the network load, data is sent through broadcast or multicast datagrams. In addition, before sending, all data is compressed using the LZO (Lempel/Ziv/Oberhumer) algorithm.[25] Each of the slaves then executes the OpenGL calls received from the master. In order to assure the consistency of the master application state, the OpenGL calls are performed on the master side too, once they are intercepted and information about them is collected. It should be noted that following this approach, the output resolution of the OpenGL context on the master machine is completely unrelated to the resolution of the slaves.

Network traffic generated by the Network Driver has soft real-time requirements and is not tolerant of data loss. Consequently, proper functioning of the Network Renderer requires a fast LAN, limited network delay, and a reliable transport layer with guaranteed in-order arrival. Typical XVR network rendering uses isolated, monohop Ethernet networks, where the MAC (media access control) data-link protocol provides the required guarantees against data loss; in this scenario, UDP (User Datagram Protocol) datagrams can be used to transport the network data.

In order to manage the high-level data transmission, two application layer protocols were developed, each of them dealing with a different aspect of the communication. NOGLP (Network OpenGL Protocol), the higher layer, manages the information exchange, per-frame synchronization, and data compression. FDP (Fragmented Datagram Protocol), the lower layer, handles the fragmentation of those NOGLP packets exceeding UDP's maximum transmission unit (MTU); FDP also prevents data loss due to slave-side buffer overflow, through the introduction of acknowledgement messages. In order to obtain a consistent visualization and avoid inconsistencies between CAVE wall images, per-frame synchronization of the graphic slaves has to be performed by the Network Renderer. The per-frame synchronization includes the Master Node too; that is the only computer actually running the whole application.

The immersive capabilities provided by a CAVE system include three main factors: stereo graphics rendering, head tracking, and the fact that the participant is surrounded by the visual display (apart typically from the ceiling and back wall, although there do exist six-sided CAVEs). All these factors must be considered in order to make the visualization system work properly and in a consistent way. In particular, it is necessary to calculate the proper perspective projection matrix for each of the screens, according to the position and the orientation of the participant's viewpoint and head direction. Performing all these operations is a task independent of the specific application running, and it can be exactly defined given the specifications of the particular CAVE in use.

For these reasons, the Network Renderer relieves the application programmer from being concerned with these issues. The programmer needs to render only monoscopic images, and XVR takes care of properly rendering in stereo and onto the CAVE screens. The conversion from mono to stereo is achieved by buffering the commands composing a frame during the execution of the left eye and executing them again for the right eye with the specified distance between the eyes. The value of the projection matrix takes into consideration the data coming from the head-tracking device. In our CAVE setup, four rendering clients drive the front, left, right, and floor projections, respectively.

Virtual Characters in the VLF

As we stated in the introduction, one of the potential advantages of real-time global illumination in a VR is that the virtual body of the participant can itself have shadows and reflections. This requires that virtual characters, including their movements and deformations, must be rendered in real time (for example, in a CAVE), which is a challenging task in itself. This is particularly true if the realistic appearance of characters is important and if they are therefore represented by a large number of polygons.

The description of our virtual characters is based on the Cal3D (3D character animation library, https://gna.org/projects/cal3d) XML file format. Cal3D enables us to describe a bone weighted mesh in which each vertex has a weighted influence of one or more bones of the character's skeleton. In addition to weights, the mesh vertices contain material, normal, and texture coordinate information. The skeletal structure, material, and animation are also described in separate Cal3D files for each character. Cal3D provides exporter plug-ins, in source code, for Autodesk (www.autodesk.com) Maya and 3D Studio Max and for the open source 3D modeler Blender (www.blender.org). The Autodesk filmbox format can be used to transfer animations created within Autodesk's Motionbuilder system to Cal3D via Max or Maya. At the moment, we are using hand-rigged characters from aXYZ design (www.axyz-design.com) that are represented by about 5,000 to 10,000 polygons.

We developed a dynamic link library, which we embedded into the XVR framework. Different GLSL (OpenGL Shading Language) shaders can be loaded from the XVR scripting language (S3D) to perform the virtual-character skin deformation by using standard matrix or dual-quaternion blending techniques[26] on the GPU. Once the information of each character's vertex weighted mesh is distributed to vertex buffer objects of each GPU, character animation is performed by transmitting only the transformation matrix or dual quaternion of each bone of the characters' skeletons for every animation update. Because skin deformations are computed on the GPU and only skeletal transformation information and no vertex information is transmitted by the network renderer, animating our characters is highly efficient, requiring little bandwidth.

For motion capture or keyframe animation, we extend the abstract Cal3D mixer class, which enables us to blend and loop different sequences. To perform interactive animations based on real-time tracking data, we can change the transformations of every bone of a character from within S3D. We developed a simple inverse-kinematics (IK) function that enables characters to perform shoulder and elbow rotations of the left and the right arm so that, for example, a character's hand position, if it is within reach, coincides with the positional information of real-time tracking data. By using this IK function we can achieve realistic-looking arm movements based only on the positions read from the trackers. Head rotations and body positions are directly applied from the appropriate tracking data in order to mirror the gaze and location of the participants in our immersive dynamically and globally illuminated virtual environments.

One of the potential advantages of real-time global illumination in a VR is that the virtual body of the participant can itself have shadows and reflections.

For CAVE applications the virtual character is used only for rendering of shadows and reflections because the participant's own body is visible when using shutter glasses. In HMD applications this is not the case, and the virtual character's body but not the head must also be rendered for direct viewing.

Results

Timings were obtained on an Intel Core 2 Quad (QX6700) 2.66-GHz processor with a GeForce

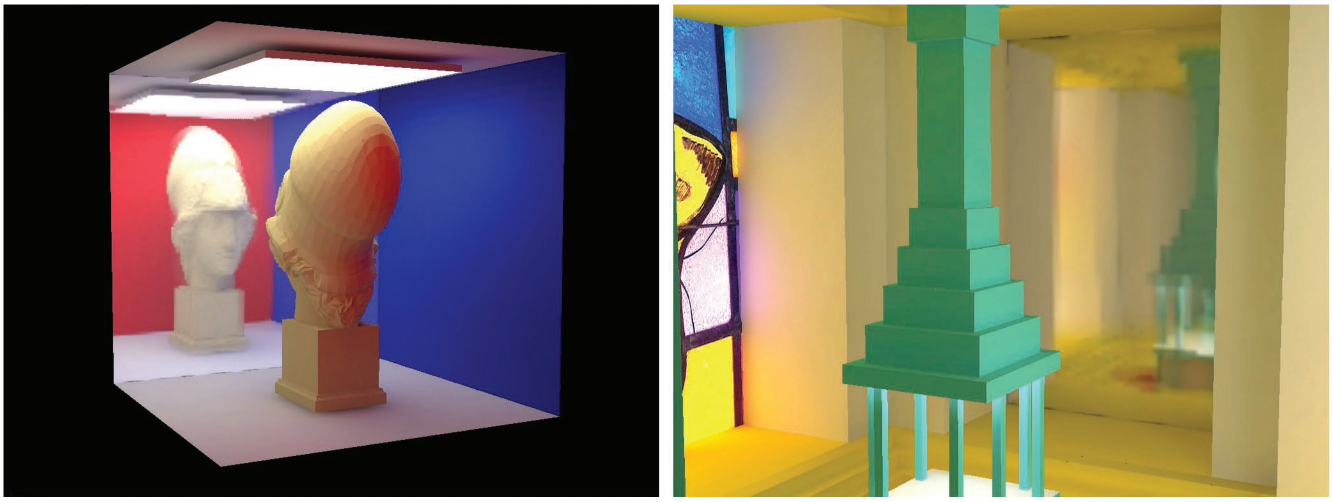
(a) (b)

Figure 2. Scenes used for rendering the performance tests: (a) the Atenea scene (containing a statuette of Athena), rendered at 120 frames per second mono (60 fps stereo) and (b) the Grotto scene, rendered at 121 fps mono (60.5 fps stereo). These images illustrate real-time performance for rendering scenes with complex illumination involving specular reflections.

8800 GTX, 768 Mbytes of graphics memory, and 4 Gbytes of host memory. Resolution was fixed at 1,024 × 768, with 2,000 directions on the sphere and 256^2 radiance maps. The scenes considered here have a relatively large nondiffuse area, and we are using a perfectly specular BRDF, which is the worst-case scenario for a light field with discrete directions. Other less directionally dependent BRDFs would require fewer directions and would render with fewer visible artifacts. No stencil buffer was used, such that the interpolation computation was always performed for all pixels in the view, yielding a worst-case but stable frame rate.

The Atenea scene (containing a statuette of Athena) in Figure 2a is composed of 9,410 polygons with a single emitter. The wall opposite the statuette is specular with a slight bluish diffuse component. The scene contains approximately 30.6 million nondiffuse elements and approximately 2.4 million diffuse elements and is propagated in approximately 16.3 minutes with six iterations. The data structures consume 22.3 Mbytes of memory compressed on disk or 142 Mbytes of texture memory on the GPU. The Grotto scene in Figure 2b is composed of 318 polygons with three emitters, of which two are textured. The scene contains approximately 9.2 million nondiffuse elements and approximately 1 million diffuse elements and is propagated in approximately 2.5 minutes with six iterations. The data structures consume 8.1 Mbytes of memory on disk or 31 Mbytes of texture memory on the GPU. Although this scene has a small number of polygons, its illumination complexity is high, and it is used as a standard scene in global illumination research.[27]

The frame rates quoted in Figure 2 were sustained from all viewpoints, even when the nondiffuse polygons filled the entire image.

The scene used in Figure 1, including a dynamic character, rendered with all effects enabled at approximately 25 frames per second (fps) in the CAVE. By far the most expensive part of the rendering was the soft-shadowing technique requiring a floating-point shadow map and many texture samples for each visible fragment. The rendering costs of the VLF, stencil reflections, and character relighting were negligible, reducing the frame rate by only a few frames per second. Without the soft-shadow technique enabled, the application ran at approximately 40 fps. We believe that this is limited mainly by synchronization issues with the CAVE hardware.

We have presented a GPU VLF rendering method capable of rendering full global illumination for VR applications at real-time frame rates. Rendering performance is independent of illumination complexity and geometric complexity, assuming that visibility can be resolved in real time. The global illumination shading adds only a small constant time operation per pixel, through accessing the DPP light field stored on the GPU.

We have constructed a complex application that is the basis for further experimentation of the influence of global illumination on people's responses within VR. The application places the participant in a library, which includes a plant, a telephone, many books on shelves, and a large mirror on one wall. The application runs in a four-screen CAVE-like system (specifically, a Trimension ReaCTor) and includes books falling from the shelves with appropriate dynamics. The participant is endowed with a complex virtual body. Because the participant is in a CAVE, his or her own body can be seen, so that the virtual body is not visible, but it is reflected in the mirror (see Figure 3). We had tracking on the head and one hand so that inverse kinematics were used to approximately map the movements of the virtual body from the movements of the participant. The experiment, recently completed, suggests together with the results reported earlier[2] that the critical element for high presence in the virtual scenario is the dynamic shadows and reflections rather than the overall quality of illumination. However, this is a tentative result awaiting further analysis, and is in any case application specific.

Directions for future work include integrating the dynamic objects further by considering also radiance and irradiance reflected from dynamic objects onto the static objects. Also, better support for more complex materials for both the static and dynamic objects would be another avenue worth pursuing. Finally, by moving to an approach where all direct lighting and shadows are computed using graphics hardware, it might be possible to interactively update a low-resolution VLF in the background supporting dynamic light sources and materials, and fully dynamic geometry.

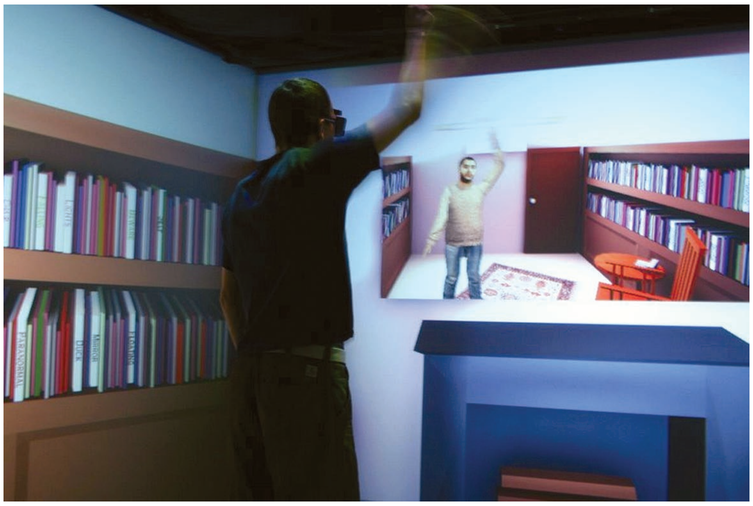

Figure 3. A participant in a CAVE (Cave Automatic Virtual Environment) virtual library application. His virtual body is invisible within the scene but does reflect in the virtual mirror. This illustrates full global illumination with support for tracked dynamic avatars; note the correspondence of the virtual character's pose with the tracked subject.

Acknowledgments

The Engineering and Physical Sciences Research Council funded this research through grant EP/C511824/1, Presence in the Virtual Light Field. The European Sixth Framework Program, Future and Emerging Technologies project Presence: Research Encompassing Sensory Enhancement, Neuroscience, Cerebral-Computer Interfaces and Applications (Presenccia, contract number 27731), together with support from the Spanish Ministry of Science and Innovation, supported the virtual character research.

References

1. M.V. Sanchez-Vives and M. Slater, "From Presence to Consciousness through Virtual Reality," *Nature Reviews Neuroscience*, vol. 6, no. 4, 2005, pp. 332–339.
2. M. Slater et al., 2008. "Visual Realism Enhances Realistic Response in an Immersive Virtual Environment," to be published in *IEEE Computer Graphics and Applications*, www.cs.ucl.ac.uk/staff/m.slater/Papers/pitroom.pdf.
3. M. Carrozzino et al., "Lowering the Development Time of Multimodal Interactive Application: The Real-Life Experience of the XVR Project," *Proc. 2005 ACM SIGCHI Int'l Conf. Advances in Computer Entertainment Technology* (ACE 05), ACM Press, 2005, pp. 270–273.
4. T. Whitted, "An Improved Illumination Model for Shaded Display," *Proc. Siggraph*, ACM Press, 1979, pp. 343–349.
5. C.M. Goral et al., "Modeling the Interaction of Light between Diffuse Surfaces," *Proc. Siggraph*, ACM Press, 1984, pp. 213–222.
6. A. Reshetov, A. Soupikov, and J. Hurley, "Multilevel Ray Tracing Algorithm," *ACM Trans. Graphics* (Proc. Siggraph), vol. 24, no. 3, 2005, pp. 1176–1185.
7. I. Wald et al., "A Ray Tracing Based Virtual Reality Framework for Industrial Design," *Proc. 2006 IEEE Symp. Interactive Ray Tracing*, IEEE Press, 2006, pp. 177–185.
8. P.-P. Sloan, J. Kautz, and J. Snyder, "Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments," *Proc. Siggraph*, ACM Press, 2002, pp. 527–536.
9. K. Dmitriev et al., "A CAVE System for Interactive Modeling of Global Illumination in Car Interior," *Proc. ACM Symp. Virtual Reality Software and Technology* (VRST 04), ACM Press, 2004, pp. 137–145.
10. S.J. Gortler et al., "The Lumigraph," *Proc. Siggraph*, ACM Press, 1996, pp. 43–54.
11. M. Levoy and P. Hanrahan, "Light Field Rendering," *Proc. Siggraph*, ACM Press, 1996, pp. 31–42.
12. P. Khanna et al., "Fast Ray Tracing of Scenes with Unstructured Motion," *ACM Siggraph 2004 Posters*, ACM Press, 2004, p. 35.
13. E. Camahort, A. Lerios, and D. Fussell, "Uniformly Sampled Light Fields," *Proc. Eurographics Rendering Workshop*, Springer, 1998, pp. 117–130.

14. P. Huang et al., "Traversal Fields for Ray Tracing Dynamic Scenes," *Proc. ACM Symp. Virtual Reality Software and Technology* (VRST 06), ACM Press, 2006, pp. 65–74.
15. Z. Ren et al., "Intersection Fields for Interactive Global Illumination," *The Visual Computer*, vol. 21, nos. 8–10, 2005, pp. 569–578.
16. M. Slater et al., "A Virtual Light Field Approach to Global Illumination," *Proc. Computer Graphics Int'l* (CGI 04), IEEE CS Press, 2004, pp. 102–109.
17. J. Mortensen et al., "Real-Time Global Illumination in the CAVE," *Proc. 2007 ACM Symp. Virtual Reality Software and Technology* (VRST 07), ACM Press, 2007, pp. 145–148.
18. M.J. Kilgard, "Improving Shadows and Reflections via the Stencil Buffer," white paper, Nvidia Corp., 2002.
19. C. Dachsbacher et al., "Implicit Visibility and Antiradiance for Interactive Global Illumination," *ACM Trans. Graphics* (Proc. Siggraph), vol. 26, no. 3, 2007, p. 61.
20. R. Ramamoorthi and P. Hanrahan, "An Efficient Representation for Irradiance Environment Maps," *Proc. Siggraph*, ACM Press, 2001, pp. 497–500.
21. G. Greger et al., "The Irradiance Volume," *IEEE Computer Graphics and Applications*, vol. 18, no. 2, 1998, pp. 32–43.
22. Y. Uralsky, *Efficient Soft-Edged Shadows Using Pixel Shader Branching*, Addison-Wesley, 2005, pp. 269–282.
23. G. Marino et al., "Description and Performance Analysis of a Distributed Rendering Architecture for Virtual Environments," *Proc. 17th Ann. Int'l Conf. Artificial Reality and Telexistence* (ICAT 07), IEEE CS Press, 2007, pp. 234–241.
24. S. Molnar et al., "A Sorting Classification of Parallel Rendering," *IEEE Computer Graphics and Applications*, vol. 14, no. 4, 1994, pp. 23–32.
25. M.F.X.J. Oberhumer, *LZO—a Real-Time Data Compression Library*, documentation for LZO version 2.02, 2005; www.oberhumer.com/opensource/lzo.
26. L. Kavan et al., "Skinning with Dual Quaternions," *Proc. 2007 ACM Siggraph Symp. Interactive 3D Graphics and Games*, ACM Press, 2007, pp. 39–46.
27. G. Coombe, M.J. Harris, and A. Lastra, "Radiosity on Graphics Hardware," *Proc. 2004 Conf. Graphics Interface* (GI 04), ACM Press, 2004, pp. 161–168.

Jesper Mortensen *is an R&D programmer at Geomerics Ltd. working on real-time global illumination for the games industry. He's also a completing PhD student on the topic of global illumination at University College London. His research interest is rendering algorithms, particularly for global illumination. Mortensen received his MSc in vision, imaging, and virtual environments from University College London. Contact him at jesper@geomerics.com.*

Insu Yu *is a finishing PhD student at University College London. His interests include real-time global illumination on GPUs, especially based on spherical harmonics. Yu received his MSc in vision, imaging, and virtual environments from University College London. Contact him at i.yu@cs.ucl.ac.uk.*

Pankaj Khanna *is a financial-services consultant and maintains links to academia as an Honorary Research Fellow at University College London. He's also a PhD candidate in computer science at University College London, with interests in global illumination, real-time physically based rendering, and virtual reality. Khanna received his MSc in vision, imaging, and virtual environments from University College London. Contact him at p.khanna@cs.ucl.ac.uk.*

Franco Tecchia *is an assistant professor in computer science with a primary specialization in software engineering and computer graphics, and is the head of the Visualisation Systems Group at the PERCRO (Perceptual Robotics) Laboratory of the Scuola Superiore Sant'Anna. His research activities focus on the design and development of complex VR systems, and include real-time computer graphics, virtual and augmented reality, and software engineering applied to virtual environments. Contact him at franco.tecchia@sssup.it.*

Bernhard Spanlang *is a postdoctoral research fellow at the Universitat Politècnica de Catalunya. His research interests are statistical character animation, human-avatar interaction, and virtual clothing. Spanlang received his EngD in vision, imaging, and virtual environments from University College London. Contact him at bspanlang@lsi.upc.edu.*

Giuseppe Marino *is a PhD student at the PERCRO (Perceptual Robotics) laboratory of the Scuola Superiore Sant'Anna. His technical interests include high-performance cluster rendering and vision-based tracking systems. Marino received his master's degree in computer engineering from the University of Pisa. Contact him at giuseppe.marino@sssup.it.*

Mel Slater *is Institució Catalana de Recerca i Estudis Avançats Research (ICREA) Professor at the University of Barcelona and is Professor of Virtual Environments at University College London. His research interests include global illumination algorithms for virtual reality and scientific understanding of how people respond to VR. Slater received his DSc in computer science from the University of London. Contact him at m.slater@cs.ucl.ac.uk.*