A Virtual Light Field Approach to Global Illumination

Mel Slater Jesper Mortensen Pankaj Khanna Insu Yu
Department of Computer Science
University College London, Gower Street, London WC1E 6BT, UK
www.cs.ucl.ac.uk/research/vr/Projects/VLF

Abstract

This paper describes an algorithm that provides real-time walkthrough for globally illuminated scenes that contain mixtures of ideal diffuse and specular surfaces. A type of light field data structure is used for propagating radiance outward from light emitters through the scene, accounting for any kind of L(S|D) light path. The light field employed is constructed by choosing a regular point subdivision over a hemisphere, to give a set of directions, and then corresponding to each direction there is a rectangular grid of parallel rays. Each rectangular grid of rays is further subdivided into rectangular tiles, such that each tile references a sequence of 2D images containing colour values corresponding to the outgoing radiances of surfaces intersected by the rays in that tile. This structure is then used for final image rendering. Propagation times can be very long and the memory requirements very high. This algorithm, however, offers a global illumination solution for real-time walkthrough even on a single processor.*

1. Introduction

The main contribution of this paper is to offer a partial solution to the problem of real-time walkthrough for global illumination. The solution is partial in the sense that only ideal specular and diffuse surfaces are supported. Nevertheless any kind of *L(S|D)** light path [1] can be simulated. The method exploits the idea of light fields [2] (or lumigraphs [3]), though the particular type of light field representation used is similar to that in [4] and also similar to a data structure used for visibility culling in [5]. It exploits Layered Depth Images [6] where each ray in the light field maintains radiance information about each of the surfaces that it intersects rather than just the first surface. In this way a projected image can be reconstructed from any viewpoint and direction.

In the next Section we discuss further background information. In Section 3 the main data structure is presented, and the propagation of light through the data structure is discussed in Section 4. Implementation details and rendering are discussed in Sections 5 and 6, with results including images, timing, and memory requirements are presented in Section 7. Conclusions are given in Section 8.

2. Background

Radiosity was the first algorithm that supported real-time walkthrough with global illumination but only for scenes with diffusely reflecting surfaces [11]. With the addition of glossy and specular surfaces real-time walkthrough becomes problematic, owing to the view dependent nature of the required global illumination solution in this case. There are, however, several different classes of algorithm that attempt to provide interactive time rendering for globally illuminated scenes. Caching schemes rely on reusing elements of a global illumination solution across several views [13][14][15]. Precompute algorithms compute a global illumination solution and then approximate this in some way for rapid rendering. For example [16][17] compute virtual point light sources that produce direct illumination approximating the global solution. In [19] hierarchical clustering [12] is extended by partitioning the models into areas where global illumination is well approximated based on a set of basis functions for the irradiance over the patches, and then interactive time rendering is achieved for moderately glossy surfaces.

The exponential growth in processor speed, and advances in graphics hardware have supported a massive speed up in ray tracing [20] and path tracing [21], to the point where interactive speed for millions of polygons on clusters of consumer PCs has become possible [22]. Such research has exploited space subdivision schemes for fast ray-intersection solutions, careful organization of the overall algorithm to fit the

needs of the hardware, together with exploitation of graphics card processing, and parallel implementation across PC clusters [23].

The 'virtual light field' approach has similarities to many other approaches. For example, it is like photon mapping [18] since it propagates light from the emitters, and it is superficially similar to the 'ray bundle' method for stochastic global illumination [10] since light is propagated through bundles of parallel rays in successive iterations. It may also be thought of as a combination of light field and illumination network [7]. However, it is a deterministic rather than Monte Carlo solution, employing a fixed set of rays for propagation. It relies on this pre-computed global illumination solution stored in a massive data structure, and then uses lookups into the data structure for determining radiance to be assigned to primary rays in the rendering phase. There is no significant ray-object intersection searching in any phase of the propagation or rendering, everything is carried out by rasterisation or by direct lookup. The lookup is typically into a very small list of surface identifiers, and the only 'search' required is to find a matching element in the list. This approach therefore sacrifices propagation time and memory to the goal of fast final rendering.

3. VLF Data Structure Overview

A scene (or part of a larger scene) can be well enclosed, by a regular cuboid, for example, bounded by (-1,-1,-1) to (1,1,1). Consider the lower face (at z=1) (-1, -1, -1) to (1,1,-1). This is discretised into a regular grid of $N \times N$ 'cells', (i,j), (i,j = 0,1,…,N-1) each of which is the origin of a ray that is parallel to the z-axis. The set of all $N \times N$ rays is called the *canonical parallel subfield* (PSF), and k = (0,0,1) is its direction. If l points with spherical coordinates $\omega_i = (\theta_i, \phi_i)$ are chosen on the unit hemisphere over the z=0 plane then l PSFs are defined as rotations of the canonical PSF. The rotation can be achieved in any manner that is consistent throughout (for example, first rotate by θ_i about the y-axis to transform k to the zx plane and then by ϕ_i about z, to bring it into ω_i).

Consider any ray (i,j) in the canonical PSF. This will intersect a number of surfaces in the scene. If we parameterise the ray in the form $r(t) = r_0 + kt \ \ (t \geq 0)$, and r_0 is the ray origin, then the intersection points can be characterized as an array of non-decreasing parametric values $[t_1, t_2, ..., t_k]$. With each one of these intersection points additional information could be stored: the identifier of the surface at that intersection, and eventually the outgoing radiance from the surface at that intersection point. Although this method is possible, no use would have been made of the great coherence between neighbouring rays, and the memory costs would be substantial. Instead, the cell space is subdivided into tiles, each of resolution $m \times m$, where $1 \leq m \leq N$ and N/m is integral. Each tile maintains a sequence of surface identifiers that are intersected by any ray which has its origin within the tile.

The process of finding all the intersections of surfaces with the rays and tiles of the canonical PSF can be achieved by a simple and fast rasterisation algorithm in the case when the surfaces are all polygons, except that a layered depth image is computed. Given any other PSF, corresponding to direction ω_i the scene can be rotated such that ω_i is transformed to the (0,0,1) direction and then the rasterisation carried out in the canonical space.

It is essential to choose a parameterisation over the hemisphere that *does not require searching* in order to find closest rays – since such ray lookup is a critical operation during both propagation and eventual rendering. The method employed uses a recursive subdivision of a regular tetrahedron, which partitions the hemisphere into triangles. However, fast *constant time lookup* is attained for any arbitrary point on the hemisphere in order to find the closest stored point. This is described in [8]. In the next section we discuss this data structure in more detail and how it is employed in energy propagation.

4. Propagation

4.1 Overview

Notation

The (finite) set of given PSF directions is denoted Ω_l and $\omega \in \Omega_l$ refers to a particular direction. The tiling coordinate system is referenced by (s,t), $s,t = 0,1,...,n-1$ where $n = N/m$. Hence a tile is referenced as (ω, s, t). The coordinate system of cells within a tile is referenced by (u,v), $u,v = 0,1,...,m-1$. Hence (w,s,t,u,v) refers to the ray that is in direction ω and with origin corresponding to the cell $(sm+u, tm+v)$. We will sometimes use the abbreviation $\mathbf{r} \equiv (\omega, s, t, u, v)$.

Data Structures

For each PSF, each tile contains a set of surface identifiers, corresponding to the surfaces that are

intersected by any ray within the tile. Associated with each surface fragment are two 2D radiance arrays that we refer to as radiance maps. These are referred to as the *Total Radiance Map* and *Unshot Radiance Map*. In general L is a radiance function – its domain depends on context. L_U refers to unshot radiance, L_T refers to total or accumulated radiance. $L(\omega,s,t,u,v,P)$ is the radiance for ray (ω,s,t,u,v) from surface P in the direction that is on the same side of P as its outward normal. Obviously this is radiance for P in tile (ω,s,t) in position (u,v) within the tile. $L(\omega,s,t,P)$ is a radiance map for P in the tile (ω,s,t). The individual elements of this radiance map are $L(\omega,s,t,u,v,P)$ as (u,v) vary over the coordinates of the tile.

The radiance maps clearly represent directional energy. In addition each diffuse surface P has two associated *texture maps* P^C (*Current*) and P^N (*Next*) to store radiance values due to diffuse reflection. Any ray (ω,s,t,u,v) that passes through a texel of such a texture map picks up a radiance value $L(\omega,s,t,u,v,P^C)$, corresponding to the amount of accumulated radiance that is to be distributed diffusely from the area corresponding to the texel. New radiance due to diffuse reflection that is generated within the current propagation cycle is stored in the *Next Texture Map* P^N and will be distributed in the next cycle. For one kind of rendering technique it is also useful to compute a *Total Texture Map* P^T which stores the radiance accumulated over all propagation cycles for diffuse reflection

Visibility Map and Exchange Buffer

When treating surface P within a tile as a receiver of energy, the *visibility map*, $V(\omega,s,t,P)$, for P provides information about where P is located within the tile. $V(\omega,s,t,u,v,P)=1$ only when polygon P is rasterised on the cell (u,v) within tile (ω,s,t) for PSF ω, otherwise this value is 0. We will use the notation $(u,v)\in V(P,\omega,s,t)$ to mean that $V(\omega,s,t,u,v,P)=1$.

For each set position (u,v) within the visibility map for P there will be at most one other surface in the tile that is visible along the ray corresponding to the PSF. The *exchange buffer* is a 2D array of *identifiers* of the surfaces that are visible to P within the visibility map of P. It is constructed on the fly as surface P is processed for incoming energy, and deleted after use. $X(\omega,s,t,P)$ is the exchange buffer for polygon P in the tile (ω,s,t). $X(\omega,s,t,u,v,P)=Q$ if and only if the ray (ω,s,t,u,v) intersects both P and Q, their outward normals point towards each other, and there is no intervening polygon in the ray segment joining P and Q. Clearly $X(\omega,s,t,u,v,P)=Q$ if and only if $X(\omega,s,t,u,v,Q)=P$.

In this discussion directions are always interpreted to correspond to the front-facing normals of the surfaces involved. So ray $\mathbf{r}$ is considered in one direction from Q to P and in the opposite direction from P to Q.

The Propagation Cycles

The propagation cycles run through each PSF, each tile within each PSF, and each surface within each tile. In propagation cycle 0 the Current Texture Maps of the light sources are loaded up with their given energy (their Next Texture Maps are never needed - assuming that lights do not additionally reflect energy, but only emit it). We assume that light sources are isotropic and are diffuse, and so the light source Current Texture Maps are always 1×1 arrays. The following pseudo code illustrates how each propagation cycle unfolds.

```
for each ω∈Ω_l {                                        /*1*/
    for each s,t = 0,...,n−1 {                          /*2*/
        for each surface P∈(ω,s,t) {                    /*3*/
            ω'∈Ω_l approx specular reflection direction;
            X = exchange buffer for P;
            for each (u,v)∈V(P,ω,s,t) {                 /*4*/
                r = (ω,s,t,u,v);
                Q = X(r,P);                             /*5*/
                L_d ∝ L(r,Q^C);                         /*6*/
                L_s = L_U(r,Q);                         /*7*/
                L = L_d + L_s;                          /*8*/
                L_T(r,Q) += L;                          /*9*/
                L(r,P^N) += L;                          /*10*/
                r' = (ω',s',t',u',v') spec. reflected ray;
                L_U(r',P) += ρL;                        /*11*/
            }}}}
current texture maps = next texture maps;
next texture maps = 0;
unshot radiances = 0;
```

Note that in (6) an appropriate fraction of the energy is extracted from the texels on the Current Texture Map of Q that are intersected by the current ray. The fraction of energy extracted and sent to P ensures a proper balance of energy in the scene. In (11) the ray corresponding to the specific specular direction of reflection from P is computed, and the nearest PSF direction and ray to this is found. This is used to

identify the cell in the Unshot Radiance Map for P along this reflection direction, and the radiance value in that cell is updated by the appropriate fraction of incoming radiance.

4.2 Diffuse Surface as Receiver

The above outlines the overall propagation process; we now consider some of its elements in more detail. As discussed earlier, the flow of radiance to a receiving face (P) is performed by identifying all surfaces (Q) that send radiance to it along a single PSF tile at a time.

When the receiver and sender are both diffuse, energy transfer takes place through a temporary radiance tile (a 2D array of radiances) aligned with the current PSF tile being propagated (Figure 1). This temporary radiance tile is used as an accumulator for radiance propagating towards the diffuse receiver within that PSF tile. The mapping of the texture maps on the sender (Q^C) and receiver (P^N) to the temporary radiance tile is a many-to-one mapping. This transfer of energy is transformed from a discrete representation (on the sender in Q^C) into a continuous one from which it is then correctly re-sampled to another discrete representation (on the receiver in P^N).

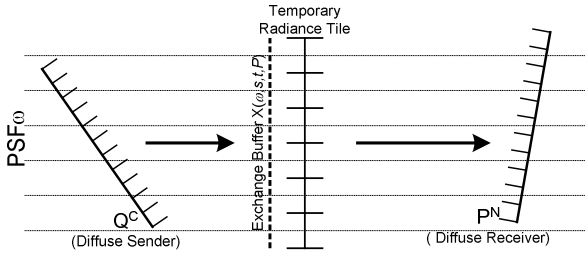


Figure 1. Diffuse to diffuse transfer along a tile.

Discrete cells on a sender are mapped onto a receiver using polygon clipping so that contributions from a single shooting cell can be correctly distributed amongst the receiver cells. The ratio of intersected to total (visible) area of the sender cell determines the amount of radiance attributed to the receiver cell. For all senders in the tile, radiance is accumulated onto the temporary radiance tile by projecting the cells of the sender's Current Texture Map (Q^C) in the PSF direction and clipping against the cells on the temporary radiance tile.

In specular to diffuse transfer, energy in the specular sender's Unshot Radiance Map $L_U(\omega,s,t,Q)$ is pushed into the Next Texture Map on the receiver (P^N) via the temporary radiance tile and shows up as caustics. This transfer is a simple visibility-based one-to-one mapping since $L_U(\omega,s,t,Q)$ and the temporary radiance tile have the same resolution along the PSF direction.

Once the accumulation of radiance from all senders onto the temporary radiance tile is complete, the temporary radiance tile is mapped onto the Next Texture Map on the receiver (P^N) using a similar projection and clipping process. Clipping is the most expensive process during propagation; accounting for up to 80-85% of overall compute time. However, this 'continuous' clipping algorithm is a required step for proper mapping of energy without aliasing which is introduced by other less compute intensive methods. The temporary radiance tile is also added to the Total Radiance Map of the sender for the PSF tile being propagated ($L_T(\omega,s,t,u,v,Q)$).

4.3 Specular Surface as Receiver

The fundamental principle in diffuse to specular transfer along a PSF direction ω is that energy from a diffuse sender Q sent to a specular receiver P is reflected towards a PSF direction ω' by P. To avoid artifacts due to the many to one mapping from the sender's Current Texture Map on to the specular receiver's radiance map, the transfer is performed by a backwards lookup from P onto Q. This lookup of Q^C on the sender from $L_U(\omega',s',t',u',v',P)$ on the receiver is shown below in Figure 2.

Candidate receiving cells of the receiver's unshot radiance map in PSF ω' are determined by the bounding box of the four projected corners of tile (s,t) in PSF ω. Each candidate cell ($L_U(\omega',s',t',u',v',P)$) in PSF ω' is projected backwards into PSF ω, allowing a transfer of radiance from Q^C as determined by the exchange buffer $X(\omega,s,t,P)$.

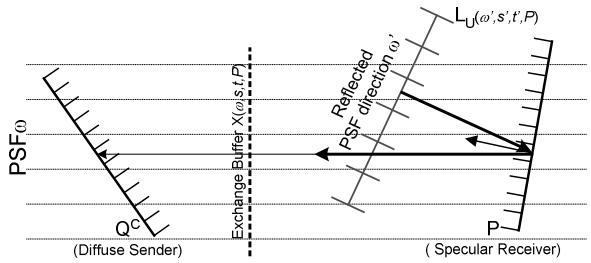


Figure 2. Diffuse to specular transfer along a PSF using backward mapping.

The specularly reflected ray will probably not correspond in direction to any of the actual PSF directions. Instead, the three nearest rays corresponding to three PSFs are found along with barycentric weights to interpolate the actual direction from them. The

energy of the diffuse sender is then transferred along each of these directions weighted by the barycentric coefficients.

Specular to specular energy transfer is very similar to the case of diffuse to specular transfer. The transfer of energy is via a backwards lookup from $L_U(\omega',s',t',u',v',P)$ of the sender's unshot radiance map $L_U(\omega,s,t,u,v,Q)$ (rather than from Q^C as in the above case).

5. Implementation Issues

A major consequence of the discretisation of directions is that 'holes' in propagation are created (Figure 3). This problem cannot be solved when energy is propagated towards specular surfaces, since these are represented by directional radiance maps which only exist for the set of actual PSF directions. However, in the case of energy propagated towards diffuse surfaces (from all senders) we can use the surface texture maps to fill in the holes.

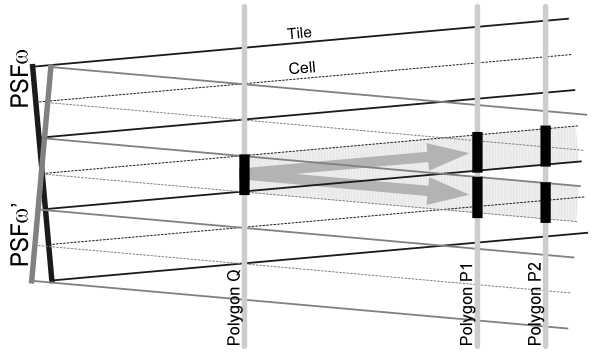


Figure 3. Angular spread of surface hits with distance.

For diffuse receivers, additional directions (beyond those used for 'actual' PSFs) are simulated by perturbing the VLF by a stratified sampling of the canonical PSF's solid angle. At each perturbation, the entire scene is rotated such that the canonical PSF (aligned with the *z*-axis) is oriented along the new sampled direction. Once a perturbation angle has been selected, the propagation is repeated ensuring that the sum of radiance propagated into the scene over the perturbations is equivalent to the total unshot radiance for that propagation iteration. Perturbation is not required for all cycles of the propagation process, perturbing the initial few is generally adequate. The number of perturbations required per propagation iteration depends on the number of PSF directions in the VLF and the nature of the scene.

When enumerating senders for a receiver during a perturbation, visibility maps in the tiles are no longer valid and senders for a receiver in the tile are recomputed from the exchange buffer $X(\omega,s,t,P)$. All senders push radiance into the temporary radiance tile which is subsequently mapped to the receiver's Next Texture Map (P^N). During a perturbed transfer, the sender's directional Total Radiance Maps (L_T) are not updated as the perturbed transfer is in a direction other than that of any actual PSF (Total Radiance Maps are updated only during perturbation 0 which involves the true PSF direction). Thus, the result of perturbation is brought into effect only in the next iteration when the perturbed (and filtered) Current Texture Maps (Q^C) are propagated into the scene.

Following perturbation, Gaussian filtering is required on the Next Texture Maps to remove the high frequencies that were introduced. This is performed with a suitably small σ and filter size to minimise blurring of caustic and shadow boundaries.

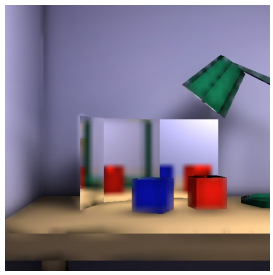
During propagation, energy transfer between a sender and receiver within a tile is dictated by the information in the exchange buffer $X(\omega,s,t,P)$. This buffer is computed using OpenGL false-colour rendered images on a per-receiver, per-PSF basis and introduces rasterisation error and aliasing at polygon boundaries. This is solved by super-sampling the visibility exchange which allows for more detailed and accurate project and clip operations. The computational overhead of this process only occurs at polygon boundaries where it rectifies radiance propagation.

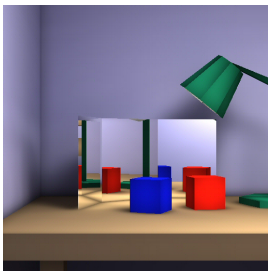
Unlike the perturbation method for diffuse surfaces, the number of directions of propagation and representation for specular surfaces cannot be increased without actually increasing the number of PSFs used. Once the propagation is completed we can however re-sample directional radiance maps on specular surfaces using backwards ray tracing, following S^*D paths only. This backwards ray tracing is performed by following true ray paths (not only paths dictated by fixed PSF directions) and thereby allows for a more accurate representation in the same data structures. We can thus obtain directional radiance maps which are geometrically more accurate and free of any 'holes'. Backwards ray traced re-sampling of the Total Radiance Maps is performed at the end of all the propagation iterations, prior to the final rendering and is performed only once.

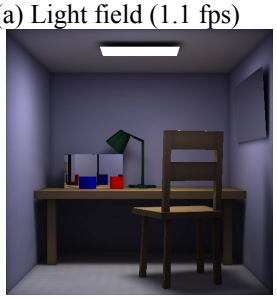
6. Rendering Methods

Images can be rendered directly from the Total Radiance Maps stored in the tiles. First a false colour

rendering is carried out to identify the nearest surfaces intersected by the primary rays. For any primary ray **r** this yields the identifier P of the intersected face along the ray, from which the intersection point (x, y, z) can be computed. The ray direction of **r** will intersect exactly one hemispherical triangle with vertices ω_k, ω_l and ω_m corresponding to three PSFs. For each $h \in (k, l, m)$ map the intersection point into canonical PSF coordinates (x_h, y_h, z_h). Now the projection $(x_h, y_h, -1)$ on the corresponding PSF ω_h maps to a cell $(i, j) = (s_h m + u_h, t_h m + v_h)$. We find the face P in the tile list of (ω_h, s_h, t_h) and the radiance is bilinearly interpolated from the 8-neighbourhood around (i, j) [7]. Finally spherical interpolation weights of **r** in relation to the three PSF directions are used to interpolate the three radiances. With this approach, direction and radiance lookups are almost constant time, the only searching being locating the face within a tile, which is logarithmic in the average number of surfaces intersected by a tile. An example is shown in Figure 4(a) rendered on a single 2.8Ghz Xeon.


(a) Light field (1.1 fps)


(b) RT ray tracing (20fps)


(c) The Office scene

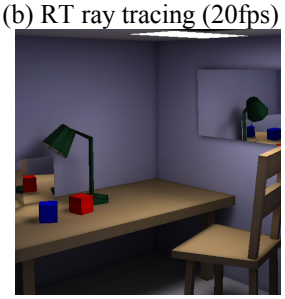

(d) *ES*D* paths

Figure 4. Rendering methods (*N*=128, *m*=16 and *l*=2049, propagation time=36 hrs, size=908MB)

In cases where PSF directional resolution is too low the image will exhibit ghosting effects on specular reflections. However, backwards ray tracing can be used to render ideal specular pixels [20]. The downside of this approach is that it is computationally intensive for scenes with many large specular reflectors. However, advances in ray tracing are approaching a level where this may become possible [22], especially given that only *ES*D* rays must be computed, and no shadow rays, nor sampling of area light sources are required. Also those parts of the image showing diffuse surfaces directly are rendered using OpenGL and texture mapping with the view independent Total Texture Map P^T. An approach similar to [25] has been implemented allowing faster walk-through of the VLF (see Figure 4(b) rendered on two 2.8Ghz Xeon CPUs).

7. Results

7.1 Performance

In order to determine the scalability of the algorithm, both in terms of memory requirements and computation time, a number of scenes were propagated under various conditions. Scalability data was gathered varying the number of PSF directions in a VLF, PSF size, and the number of polygons in a scene. All scenes in this section were propagated on a 1.7Ghz Xeon workstation. Although the scenes we use have few polygons, the same scenes rendered with radiosity would render many tens of thousands of patches in order to obtain the same level of accuracy.

An artificial scene consisting of axis aligned non-intersecting boxes uniformly distributed in a cubical space was used for testing the impact of the number of polygons on memory and propagation time. The scene had one emitter, and the number of polygons ranged from 224 through to 1736. The ratio of diffuse to specular surfaces was 5 to 1 throughout. This scene is the worst case for this algorithm, because along every tile there will be a relatively large number of polygons stored, unlike 'normal' interior scenes, which for the most part have surfaces that are sparsely distributed throughout the space. The propagation used 513 directions, with $N = 64$ and $m = 8$ throughout.

The results showed that propagation time varied quadratically with the number of polygons, as would be expected. The memory grows linearly with the number of polygons, the minimum being 75MB and the maximum 458MB.

The higher the number of directions the greater accuracy particularly in specular surfaces and the less need there will be for perturbations in order to overcome the problem of holes. On the other hand more directions require more memory and longer propagation times. For this and subsequent analyses the scene used is shown in Figure 4. For generating the data, the PSF resolution was N=64, m = 8, with l = 9, 33, 129, 513, 2049, and 8193. The number of propagation cycles was 4.

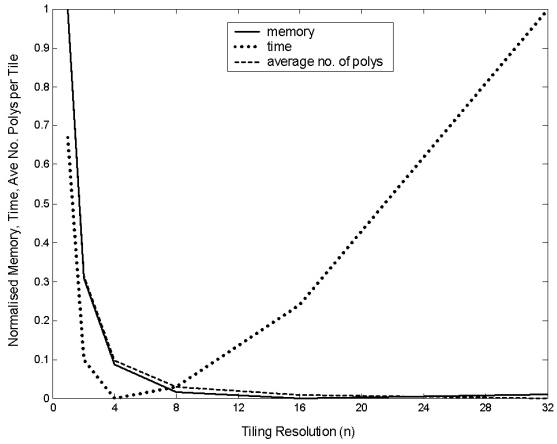


Figure 5. Normalised memory, propagation time, and average polygons per tile, for varying tile sizes.

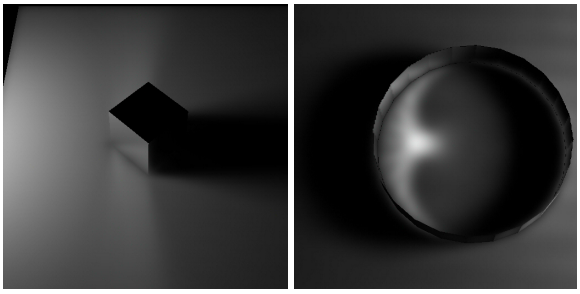
The relationship between propagation time and the number of directions is almost exactly linear. For example, with 129 directions the time is 8.5 minutes, for 513 directions it is 34 minutes and for 8193 directions, 489 minutes. There is also a linear relationship between the number of directions and memory. The three corresponding memory figures are 11MB, 45MB and 733MB respectively.

The size of the tiles relative to the PSF resolution is important in determining overall speed of propagation and rendering. Other things being equal, the smaller the tiles the fewer the number of surfaces intersected by a tile, so that less time is spent on the final search for an identifier in a tile to match a given one. However, decreasing the tile size will result in more memory and eventually in greater propagation time. Figure 5 shows plots of the memory, propagation time, and the average number of polygons in a tile for various tiling resolutions (n) (1×1, 2×2, 4×4, 8×8, 16×16, 32×32) for a 64×64 PSF size. The vertical axis is on a normalized 0–1 scale. Memory ranged from 937MB for 1×1 tiling resolution (i.e., the whole of the PSF was the tile) to 42MB for a resolution of 32×32. Timing ranged from 111 minutes for 1×1 to 150 minutes for 32×32, with the lowest being 61 minutes at 4×4. Finally, the average number of polygons per PSF ranged from 121±10 at 1×1 through to 1.7±2.1 at 32×32 which was the lowest.

7.2 Images

Figure 4 illustrates a scene that has several specularly reflecting surfaces together with diffuse surfaces. The scene is simple enough to show that correct radiance is produced.

In order to demonstrate production of caustics, we rendered a test scene from [24] using the light field with the backwards ray tracing approach. This is shown in Figure 6(a) and compares well with the original test image. A further illustration of caustics is shown in Figure 6 (b).



(a) Smits and Jensen [24] (b) Ring Caustic

Figure 6. (a) FPS =27, *N* =128, *m* = 16, *l* =2049, Propagation time =33.53 hours. (b) Ring Caustic. 50 polygons. FPS =26, *N* =256, *m* =16, *l* =2049, Propagation time =25.45 hours.

8. Conclusions

A different approach to the problem of global illumination has been introduced. The goal has been to achieve real-time walkthrough for globally illuminated scenes, relying mainly on fast lookups at the final rendering stage. This has been achieved by using a light field for energy propagation. Figure 4 (a) and (b) suggests that a final backwards ray tracing approach is faster than rendering from the light field itself. However, the ray tracing approach has complexity logarithmic in the number of polygons, whereas the VLF is logarithmic in the average number of surfaces in a tile.

The biggest drawbacks to the approach in its current version are the large propagation times and memory requirements. On the other hand, a scene needs only to be propagated once, and gigabytes of memory on desktop PCs is becoming common. Moreover, the Unshot Radiance Maps can be deleted after propagation.

In the implementation discussed in this paper, we have deliberately sacrificed memory and propagation time for interaction. However, there are clear advances that can be made in both through density estimation, and compression. The vast amount of time in propagation is caused by clipping. However, we will parallelize clipping by using streaming SIMD. Finally, the method is being extended to treat glossy surfaces.

Acknowledgements

This research is funded by the UK EPSRC, grant number GR/R13685/01, 'The Virtual Light Field'. Mel Slater is supported by an EPSRC Senior Research Fellowship. Thanks to Ingo Wald and Carsten Benthin for helpful suggestions on real time ray tracing, and to Yiorgos Chrysanthou for helpful comments on an earlier draft.

References

[1] Heckbert, P.S. (1990) Adaptive Radiosity Textures for Bidirectional Ray Tracing, Computer Graphics (SIGGRAPH) 24, 145-154.

[2] Levoy M, Hanrahan, P. (1996) Light Field Rendering, Computer Graphics (SIGGRAPH), Annual Conference Series, 31-42.

[3] Gortler, S., Grzeszczuk, R., Szeliski, R., Cohen, M. (1996) The Lumigraph, Computer Graphics (SIGGRAPH), Annual Conference Series, 43-52.

[4] Camahort, E., Lerios, A., Fussell, D. (1998) Uniformly Sampled Light Fields, Rendering Techniques 1998: 117-130.

[5] Chrysanthou, Y., Daniel Cohen-Or and Dani Lischinski (1998) Fast Approximate Quantitative Visibility for Complex Scenes, Computer Graphics International '98, Hannover, Germany, June 1998, 220-227.

[6] Shade, J., Gortler, S.J., He, L. and Szeliski, R. (1998) Layered Depth Images, Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH), 231-242.

[7] Buckalew, C. and Fussell, D. (1989) Illumination networks: Fast realistic rendering with general reflectance functions. Computer Graphics (SIGGRAPH) Conference Proceedings, 23(3) 89-98.

[8] Slater, M. (2002) Constant Time Queries on Uniformly Distributed Points on a Hemisphere, Journal of Graphics Tools, 7(1):33-44.

[9] Cohen, M.F., Shenchang, Ec., Wallace, J.R. and Greenberg, D.P. (1988) A Progressive Refinement Approach to Fast Radiosity Image Generation, Computer Graphics (SIGGRAPH) 22(4), 75-84.

[10] Szirmay-Kalos, L. (1999) Stochastic Iteration for Non-Diffuse Global Illumination, Computer Graphics Forum (Eurographics '99) 18(3) 233-244.

[11] Goral, C., Torrance, K.E., Greenberg, D. (1984) Modeling the Interaction of Light Between Diffuse Surfaces, Computer Graphics (SIGGRAPH), 18(3), 213-222.

[12] Hanrahan, P., Saltzman, D. and Aupperle, L. (1991) A rapid hierarchical radiosity algorithm. Computer Graphics (SIGGRAPH), 25(4) 197-206.

[13] Ward, G. and Simmons, M. (1999) The Holodeck Ray Cache: An Interactive Rendering System for Global Illumination in Nondiffuse Environments, ACM Transactions on Graphics, 18(4):361-98.

[14] Walter, B., Drettakis, G. and Parker, S. (1999) Interactive rendering using render cache, Rendering Techniques '99, Eurographics, (D. Lischinski and G.W. Larson, eds.), 19–30.

[15] Tole, P., Pellacini, F., Walter, B. and Greenberg, D. P. (2002) Interactive global illumination in dynamic scenes, ACM Transactions on Graphics, 21(3), 537–546.

[16] Walter, B., Alppay, G., Lafortune, E.P.F., Fernandez, S. and Greenberg, D. P. (1997) Fitting virtual lights for non-diffuse walkthroughs, SIGGRAPH 97 Conference Proceedings, Annual Conference Series, 45–48.

[17] Keller, A. (1997) Instant Radiosity Computer Graphics Proceedings, Annual Conference Series, SIGGRAPH 97 Conference Proceedings, Annual Conference Series, 49-56.

[18] Jensen, H.W. (1996) Global Illumination Using Photon Maps, Rendering Techniques '96, Proceedings of the 7th Eurographics Workshop on Rendering, 21-30.

[19] Gobbetti, E., Spanò, L. and Agus, M. (2003) Hierarchical Higher Order Face Cluster Radiosity for Global Illumination Walkthroughs of Complex Non-Diffuse Environments. Computer Graphics Forum, 22(3), (Eurographics 2003, eds P. Brunet and D. Fellner) September 2003.

[20] Whitted, T. (1980) An Improved Illumination Model for Shaded Display, Communications of the ACM, 23(6), 343-349.

[21] Kajiya, J.T. (1986) The Rendering Equation, Computer Graphics (SIGGRAPH), 20(4), 143-150.

[22] Wald, I., Schmittler, J., Benthin, C., Slusallek, P., Purcell, T.J. (2003) Realtime Ray Tracing and its use for Interactive Global Illumination, STAR, Eurographics 2003 22(3) P. Brunet and D. Fellner (eds.).

[23] Dutre, P., Bekaart, P., Bala, K. (2003) Advanced Global Illumination, A.K. Peters, Chapter 8.

[24] Smits, B. and Jensen, H.W. (2000) Global Illumination Test Scenes; Tech. Rep. UUCS-00-013, Computer Science Department, University of Utah, June 2000.

[25] Wald, I., Slusallek, P., Benthin, C., Wagner, M., (2001) Interactive Rendering with Coherent Ray Tracing; Eurographics 2001 20(3) A. Chalmers and T. M. Rhyne eds.