A Virtual Light Field for Propagation and Walkthrough of Globally Illuminated Scenes

Pankaj Khanna Mel Slater Jesper Mortensen Insu Yu
Department of Computer Science
University College London
Gower Street
London WC1E 6BT
UK
www.cs.ucl.ac.uk/vlf
p.khanna | m.slater | j.mortensen | i.yu @ cs.ucl.ac.uk

Abstract

This paper describes an algorithm that provides real-time walkthrough for globally illuminated scenes comprising of ideal diffuse and specular polygonal surfaces. A type of light field data structure is used for propagating radiance outward from light emitters through the scene and accounts for all L(S|D) light paths. 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 creating a rectangular grid of parallel rays. Each rectangular grid of rays, called a 'parallel subfield' 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 belonging to that tile. Following propagation, this structure is used for final image rendering. Propagation times are currently very long and the memory requirements 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 a (partial) solution to the global illumination problem that supports real-time walkthrough. 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 (including caustics). 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 the idea of 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 scene.

For illumination propagation an approach is used that is similar to the 'ray bundle' method for stochastic global illumination as introduced in [12]. Light is propagated through bundles of parallel rays in successive iterations. However, here the approach is deterministic, and the ray bundles are fixed sets of rays with their origins in 2D square tile grids. Standard polygon rasterisation is used to compute ray-polygon intersections rather than a Painters' Algorithm.

The ideas presented in this paper may also be thought of as a combination of light field and illumination network [7]. Both employ a fixed ray based data structure that is a discretisation of the distribution of radiance. The illumination network maintains 'links' from object to object, where two objects are linked if there is an unoccluded ray that joins them, and the link is a pointer from one object connecting to the other along such a ray. Objects are subdivided into patches, and the illumination network determines the radiance of the patches. It is finally the objects which are rendered. The virtual light field (VLF) approach described here does not need to render objects at all (though it can do so with some advantages). Rather the objects illuminate the rays, and the rendering is based on the rays. The present paper embodies the first practical realization of the ideas first presented in [8].

In the next Section we discuss further background information locating the new approach relative to other approaches to global illumination that attempt to achieve real-time walkthrough. 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 are discussed in Section 5, with results including images, timing, and memory requirements in Section 6. Further work and conclusions are given in Section 7.

2. Background

Radiosity was the first algorithm that made possible real-time walkthrough with realistic illumination but for scenes with only diffusely reflecting surfaces [13][14]. This requires a relatively extensive view independent iterative propagation phase that eventually produces radiosity values at the vertices of surface patches. Then standard hardware rasterisation including smooth shading interpolation can be used for a real-time walkthrough rendering [15]. Real-time rendering is problematic once glossy and specular surfaces are included, owing to the view dependent nature of the required global illumination solution in this case. Radiosity was extended to non-diffuse environments, for example as in [16][17], though not all light paths could be simulated and walkthrough was unattainable. Combining progressive refinement radiosity with a Monte Carlo and light tracing phase was an early attempt at a relatively fast, but not interactive time, global illumination solution including diffuse and glossy surfaces [19]. Hierarchical radiosity [18] greatly speeded up the radiosity propagation phase, and was extended to include glossy reflection [20]. Hierarchical radiosity is a fundamental approach that has been extended to include a line-space hierarchy to support rapid computation of a new solution when the scene changes [26], including glossy illumination [27][28].

There are 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 [21][22][23][24][25]. Precompute algorithms compute a global illumination solution and then approximate this in some way for rapid rendering. For example [29][30] compute virtual point light sources that produce direct illumination approximating the global solution. As another example, in [31] photon tracing [32] is used to compute a global illumination solution, and then splatting is used at rendering time together with viewing direction and surface properties to rapidly display an approximate global illumination solution. In [33] hierarchical clustering 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 [34] and path tracing [35], to the point where interactive speed for millions of polygons on clusters of consumer PCs has become possible [36]. This work 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 [37][38]. An excellent summary and overview can be found in [39].

The ‘virtual light field’ approach has similarities to many other approaches: it is like photon mapping [32] since it propagates light from the emitters, but it is a deterministic rather than Monte Carlo solution, employing a fixed set of rays instead of a randomly generated set. In photon mapping a final density estimation phase is needed to compute the radiance from the irradiance stored at the surfaces, and also a final ray trace for accurate specular reflection. Although a final rendering time ray trace can be employed for the virtual light field, it is not inherently necessary, and no final density estimation is needed. It relies on a 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. Although in the current implementation the propagation phase is very long, and the memory requirement is huge, the payoff is that final rendering is very fast. There is no ray-object intersection searching in any phase of the propagation, 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 very fast final rendering.

3. Virtual Light Field Data Structure

The ray space discretisation followed in this research is similar to that in [4] where a uniformly chosen vector at the center of a sphere defines a direction, that direction is treated as the normal to a

plane through the center of the sphere, and then a uniform set of points chosen on the plane intersected by the sphere determines a set of parallel directions. In our case the scene is enclosed by a regular cuboid bound from (-1,-1,-1) to (1,1,1). For example, consider the lower face of this bounding cuboid with corner vertices at (-1,-1,-1) and (1,1,-1). This face is discretised into $N \times N$ pixels. The pixel position (i,j) corresponds to the point:

$$\left(\frac{(2i+1)}{N} - 1, \frac{(2j+1)}{N} - 1, -1 \right) (i, j = 0, ..., N-1) \quad (1)$$

Consider this as the origin of a ray that is parallel to the z-axis, i.e., parallel to the vector (0,0,1). This set of $N \times N$ rays is called the canonical *parallel subfield* (PSF), with (0,0,1) as its direction. If l points with spherical coordinates $\omega_i = (\theta_i, \phi_i)$ are chosen on the positive hemisphere then l PSFs are defined as rotations of the canonical PSF by rotating the direction into the corresponding point.

Once again, consider a 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 $r(t) = r_0 + vt \ \ (t \geq 0)$ where v is the direction vector of the ray (0,0,1), and r_0 is the ray origin, then the intersection points can be characterized as an array of monotonically increasing parametric values $[t_1, t_2, ..., t_k]$. At each of these intersection points additional information can be stored: the surface identifier at that intersection, and eventually the outgoing radiance from the surface at that point. This is possible, and the first implementation [8] followed this approach. However, no use would be made of the very great coherence between neighbouring rays, and the memory costs would be substantial. Instead, the pixel space of the PSF is subdivided into *tiles*, each of resolution $m \times m$, where $1 \leq m \leq N$ and N is a multiple of m. Each tile maintains a sequence of surface identifiers that are intersected by any ray within the tile. Corresponding to each surface identifier there is a visibility map for the surface and a 2D image that will eventually hold the outgoing radiances corresponding to each point that has a non-zero visibility entry. In principle the visibility map is an $m \times m$ bitmap, with entries 1 corresponding to where the surface is within the tile, and 0 elsewhere. In practice this is implemented as an Edge Table, such that for each row (j=0,1,…,m-1) within the tile the boundaries $i = [a_1, a_2], [a_3, a_4], ...$ are stored, where the a's are successive pairs of coordinates such that the surface exists within these bounds. These ideas are illustrated in . The left hand rectangle shows the canonical PSF partitioned into ray origin pixels, and into tiles, with the tile blown up in the middle showing a polygon intersecting it. The Edge Table that is used to efficiently code the visibility map is shown on the right.

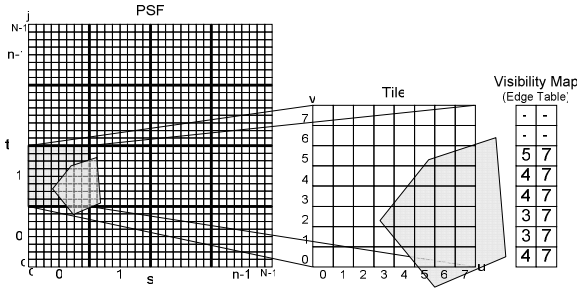


Figure 1. A PSF, Tile, Polygon and associated Visibility Map represented as an Edge Table

The process of finding all the intersections of surfaces with the rays and tiles of the canonical PSF is straightforward. If we consider the special case that all surfaces are planar polygons, then this is equivalent to polygon rasterisation, except that a layered depth image is computed. It is trivial to compute the set of polygon fragments belonging to each tile, and also trivial to construct the Edge Tables – with minor modifications to the standard polygon rasterisation algorithm (e.g., [9]). If the polygons are also convex then an Edge Table entry has either 0 or 2 entries per row.

So far we have only discussed the canonical PSF. Given any other PSF corresponding to direction ω_i, the scene can be rotated such that ω_i is mapped to direction (0,0,1) and then the rasterisation carried out in the canonical space.

The 2D image map that belongs to each surface intersected in a tile is called a *radiance map*. This (after light propagation) will contain the radiance values corresponding to each non-zero entry in the visibility map for the surface. (Note that the t-intersection values are not stored, since these can be rapidly recomputed as needed). Now suppose that all the radiances for all tiles in all PSFs have somehow been computed and the outgoing radiance in direction ω at a particular point (x, y, z) on surface P is required. We first find the direction amongst ω_i (i=0,1,…,l-1) that is closest to ω - suppose that this is ω_j. There will be a rotation matrix $\mathbf{M}_j$ that rotates ω_j into the canonical direction (0,0,1). Then $(x, y, z)\mathbf{M}_j = (x_q, y_q, z_q)$ will be the point in the canonical PSF space that corresponds to (x, y, z) in

scene space. In particular the projection $(x_q, y_q, -1)$ will be closest to some pixel (i, j) which belongs to a particular tile. If we traverse the identifiers in that tile until we find P, then we can look up the required radiance value in the radiance map belonging to P in that tile. Of course, this will only be an approximation to the true value given the discretisation employed. For acceptable accuracy either both n and l must be quite large, or interpolation methods must be employed to improve the approximation.

It is critical to choose a parameterisation over the hemisphere that requires *no searching* to find closest rays – since such ray lookup is a critical operation during both propagation and eventual rendering. The method used is a triangle based subdivision of the hemisphere, with *constant time lookup* for any arbitrary point on the hemisphere in order to find the closest stored point. This is described in [10].

In the next section we discuss how this data structure is employed in the 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 within a tile is referenced by (u,v), $u,v = 0,1,...,m-1$. Hence (ω, s, t, u, v) refers to the ray that is in direction ω and with origin at coordinates given by equation (1) where $(i, j) = (sm + u,\ tm + v)$. We 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 in the tile there are in fact two radiance maps: called 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 appropriate domain.

In addition each surface P in the scene 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 $L(\omega, s, t, u, v, P^C)$, which corresponds 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 (from diffuse or specular senders) 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 on diffuse surface P over all propagation iterations.

Exchange Buffer

Any two surfaces belonging to a tile may exchange energy along a ray, depending on whether or not they can 'see' each other (i.e., whether or not there is an occluder between them). 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. $V(\omega, s, t, u, v, P) = 1$ if and only if when polygon P is rasterised on PSF ω, the pixel (u,v) within tile (ω, s, t) is set, otherwise the 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. It is obvious that $X(\omega, s, t, u, v, P) = Q$ 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 **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). For now we assume that light sources are isotropic and 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*/
            end
        end
    end
end
current texture maps = next texture maps;
next texture maps = 0;
unshot radiances = 0;
```

The pseudo code is explained as follows, referring to the comment numbers at the end of the lines:

(1) The iteration cycle is over each PSF.
(2) Within each PSF each non-empty tile is visited.
(3) Within each tile each surface (P) is visited and treated as a potential receiver of energy. Note that with respect to the surface and the ray direction, if the surface is a polygon then all specular reflection directions are the same for this PSF and tile.
(4) For the current surface (P) all rays in the visibility map are considered, and
(5) The surface (Q, if any) that sends energy to P along this ray is extracted from the exchange buffer.
(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 balance of energy in the scene. This represents the diffuse interreflection from Q to P.
(7) The Unshot Radiance Map for Q along the current PSF is looked up at the ray position and the radiance there is determined.
(8) These two are summed to give the total radiance from Q along this ray that will strike P.
(9) This total radiance is accumulated into the Total Radiance Map for Q at the position corresponding to the ray.
(10) The texel corresponding to the ray for the next diffuse radiance map for P is also updated with this radiance as irradiance that must be distributed in the next propagation cycle.
(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 next 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.

At the end of each iteration cycle, the current texture maps are set to the next texture maps, and the latter are reset to zero. In addition the directional unshot radiance maps are reset to zero.

4.2 Diffuse Surface as Sender

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. If the sender is diffuse, the transfer to the receiver takes place in either one of two methods depending on the material properties of the receiver.

Diffuse to Diffuse transfer

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 2). 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. The 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).

This transfer of radiance to a diffuse receiver is performed in two parts: the temporary radiance tile first accumulates radiance from all senders (both diffuse and specular) and the loaded radiance tile is then mapped onto the Next Texture Map of the receiver P.

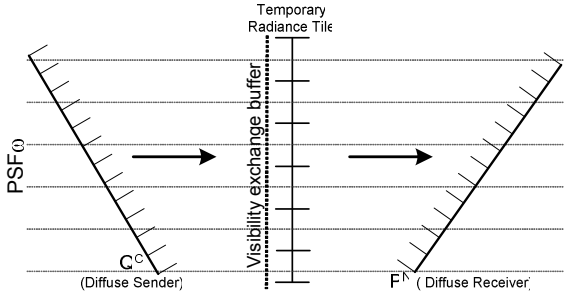


Figure 2. Diffuse to diffuse transfer along a single PSF tile via the temporary radiance tile.

Discrete cells on a sender Q 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 (visible) to total 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 cell of the sender's current diffuse map (Q^C) in the PSF direction and clipping the boundary of that cell against those of the cells on the temporary radiance tile. Once this accumulation of radiance from all senders onto the temporary radiance tile is complete, the temporary radiance tile is mapped onto the next diffuse 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. Less compute-intensive (discrete sampling) methods were attempted but caused significant aliasing and proved to be too inaccurate. The temporary radiance tile, apart from being mapped to the next diffuse map on the receiver (P^N), is also added to the total radiance map of the sender for the PSF tile being propagated $L_T(\omega,s,t,Q)$.

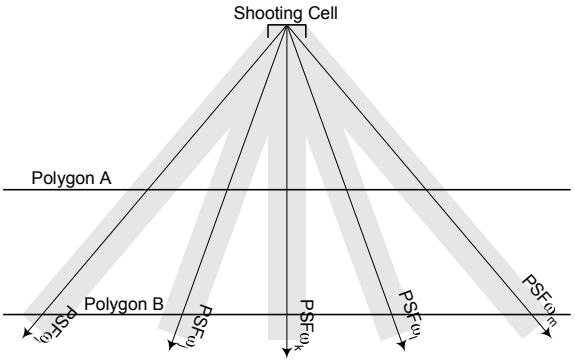


Figure 3. Angular spread of PSF surface hits with distance *r*.

The mapping of radiance between a diffuse sender and a diffuse receiver using the above two-step project and clip process correctly accounts for the terms on the numerator in the integral of the standard form-factor equation in Equation **Error! Reference source not found.**:

$$F_{ij} = \frac{1}{A_i}\int_{A_i}\int_{A_j} \frac{\cos\phi_i \cos\phi_j dA_i dA_j}{\pi r^2} H_{ij} \quad (2)$$

The inverse scaling of irradiance by π during the propagation process is explained in Section 4.4, Equation **(3)**. Also, the integral is performed explicitly by iteratively propagating along all PSF directions. The presence of the r^2 term in the denominator is however not explicitly present, but is accounted for by the very nature of discretisation of the VLF and the propagation process (see Figure 3). This r^2 term accounts for the angular spread of diffuse energy over a distance r. In the VLF, the spread of energy deposited over a surface is inherently coupled with the distance of the sender to the receiver – the VLF method being equivalent to $1/r^2$ in the limit.

Diffuse to Specular transfer

The fundamental principle in diffuse to specular transfer along a PSF direction ω is that energy from a diffuse sender Q towards a specular receiver P is reflected towards a PSF direction ω' by P. If this transfer is performed by a forward mapping from the diffuse sender into the reflected direction, under-sampling artefacts are created on the Unshot Radiance Map $L_U(\omega',s',t',P)$. This is due to the many to one mapping from the sender's unshot diffuse map Q^C into the cells of $L_U(\omega',s',t',P)$. We thus employ a backwards mapping from the reflected direction onto

P to ensure that every cell $\mathbf{r}' = (\omega', s', t', u', v')$ on the Unshot Radiance Map $L_U(\mathbf{r}', P)$ receives energy from the sender Q. Energy from the sender is thus added to the unshot radiance map in the reflected direction as illustrated in Figure 4.

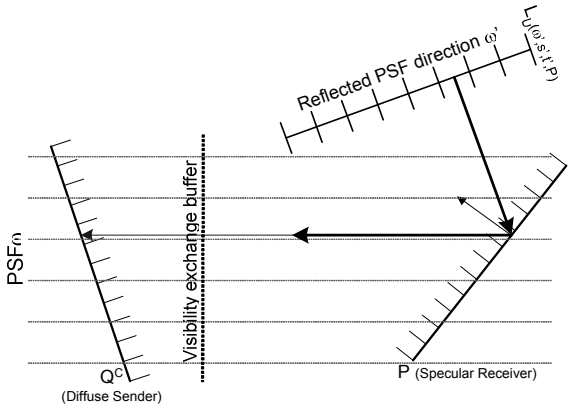


Figure 4. Diffuse to specular transfer along a PSF using backward mapping.

This transfer is performed in two stages. The four corners of tile (s,t) in PSF direction ω are projected onto the reflected PSF direction ω' to find the bounding box of the tile (s,t) on $L_U(\omega', s', t', P)$. Cells outside regions of $L_U(\omega', s', t', P)$ can then be disregarded. The cells inside the boundary potentially receive energy from the sender Q. In the next stage, each candidate cell is projected back onto the PSF ω. If the projected cell is visible from $L_U(\omega', s', t', P)$ as indicated by the visibility exchange buffer then energy is extracted from the current texture map of the sender Q^C.

Due to discretisation of directions in the VLF, 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, and barycentric weights computed to interpolate the actual direction from these three. Energy from the diffuse sender is then transferred along each of these directions appropriately weighted by these barycentric coefficients.

4.3 Specular Surface as Sender

Specular to Specular transfer

Specular to specular energy transfer is very similar to the case of diffuse to specular transfer, following the same principle. When a ray $(\omega', s', t', u', v')$ from the reflected direction is mapped backwards through P onto the specular sender Q, energy from the unshot radiance map $L_U(\omega, s, t, Q)$ is obtained (rather than from Q^C as in the case of diffuse-specular transfer).

Specular to Diffuse transfer

In specular to diffuse transfer, energy in the specular sender's Unshot Radiance Map $L_U(\omega, s, t, u, v, Q)$ (which was gathered in the previous propagation cycle) is pushed into the next diffuse map on receiver P^N. This has the effect of creating caustics on the diffuse surface.

When cells on the specular sender Q are visible from the diffuse receiver P the unshot radiance from the sender $L_U(\omega, s, t, Q)$ is accumulated into the temporary radiance tile. This transfer is a simple one-to-one mapping since $L_U(\omega, s, t, Q)$ and the temporary radiance tile are at the same resolution along the PSF direction. As mentioned earlier, the temporary radiance tile is used to gather energy sent by all senders (specular and diffuse) to the receiver P. The transfer of energy from the temporary radiance tile onto the receiver's next diffuse map P^N and the total radiance map of the sender $Q_T(\omega, s, t)$ is by the same mechanism described earlier for diffuse to diffuse transfer (Section 4.1).

4.4 Energy balance

In this section we show that light transport in the VLF is carried out correctly as a radiative transfer – that it obeys the requirement that light energy is conserved in a closed system (the 1st law of thermodynamics). During propagation irradiance is stored on diffuse surfaces in texture maps. Also, radiance leaving a face in a particular direction is stored in the light field tiles in directional radiance maps aligned to the tiles of the corresponding PSF direction. Thus (during propagation) energy in this system consists of a manifold of directional and non-directional elements. In its current state, the VLF supports only specular and diffuse surfaces and the modes of transport that can occur under this constraint are[2]: $D \rightarrow D$, $D \rightarrow S$, $S \rightarrow D$ and $S \rightarrow S$. However, since the propagation in the VLF is based on radiance flow between radiance maps this can be broken down to the following steps, sequences of which map to the modes of transport above: *texture map → radiance map, radiance map → texture map* and *radiance map → radiance map*.

Diffuse texture map to radiance map transport

The radiant exitance M stored at texel (u_q, v_q) of a diffuse texture map Q^C needs to be uniformly distributed over the hemisphere above the texel. Normally in radiant transfer the hemisphere is divided into a disjoint set of solid angles corresponding to a set of chosen directions, each of which receive a fraction of M. In the discretised light field this is (implicitly) achieved by projecting the texel into each PSF.

For any PSF ω the texel (u_q, v_q) projects to a cell (ω, s, t, u, v). This cell corresponds to a ray leaving Q. Due to the underlying discretisation, a ray represents a beam of constant radiance, and given the many-one mapping from texel to PSF space, the texel will normally fall within the beam of a single ray. However, as shown in Figure 5 in the worst case the texel might overlap the beams of several rays (such as texel X in the figure). In such cases clipping will ensure that the correct radiance fraction will be distributed to each ray. Therefore, without loss of generality we can assume for this discussion that a diffuse texel maps to a single ray.

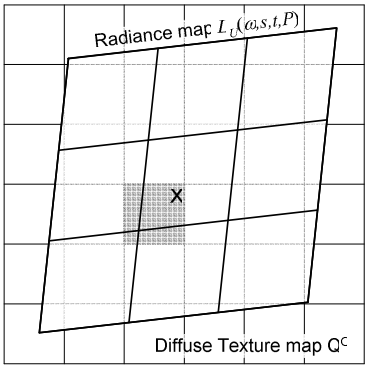


Figure 5. Projection of a radiance map onto a texture map.

The ray leaving Q will be carrying radiance $L_U(\omega, s, t, u, v, Q)$, and will additionally receive a fraction of M contained in the texel. This fraction is determined by the area of the projection of the texel divided by the sum of the areas formed by projecting the texel onto all PSFs. Therefore, the radiance carried along the ray is

$$L_U(\omega, s, t, u, v, Q) + M(Q, u_q, v_q) \times \frac{A(\omega, Q, u_q, v_q)}{\sum_{i=0}^{l-1} A(\omega_i, Q, u_q, v_q)} \quad (3)$$

where $A(\omega, Q, u_q, v_q)$ is the area of the projection of texel (u_q, v_q) on face Q into PSF ω, and $M(Q, u_q, v_q)$ is the *radiant exitance* of face Q at texel (u_q, v_q). The ratio on the right hand side of the equation essentially estimates the solid angle needed to compute the radiance traveling in that particular direction.

Due to the uniform nature of the light field the area computation can be implemented efficiently by precomputing a single value for the given light field discretisation that is the sum of the projected area of a unit diffuse texel into all PSFs. This '*total projected unit area*' multiplied by the area of a given texel yields the denominator of the division in (3).

This transport essentially 'loads up' the radiance maps for a (diffuse) face with the radiant exitance of that face, whether it is emission for an emitter or reflected irradiance for a non-emitter.

Radiance map to diffuse texture map transport

When energy is sent from a radiance map $L_U(\omega, s, t, Q)$ to a texture map P^N it is stored at some texel (u_p, v_p). This texel will by the end of each propagation cycle have summed the irradiance due to incident radiances $L_U(\omega, s, t, Q)$ over the hemisphere at the texel.

In a closed scene take the set of rays (in fact, beams) induced by the VLF discretisation emanating from a face Q then each of those beams will intersect another face P (and thus a set of texels on P). Clearly the overall area of the intersected texels projected into the PSF corresponding to the beam that intersected it will equal the denominator in the division in (3), which means that the 'outgoing' projected area equals the 'incoming' projected area overall (see Figure 6).

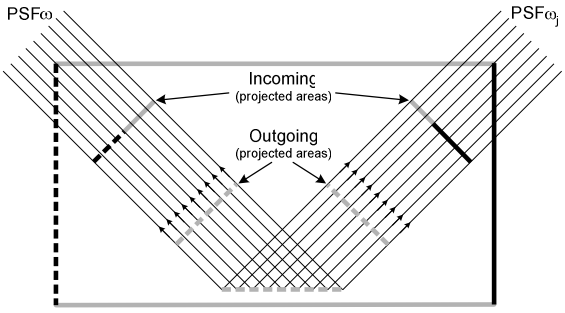


Figure 6. "Outgoing" projected area (dashed grey) equals "incoming" projected area for 3 enclosing polygons (grey, black and dashed black) in a simple VLF with only two PSFs

The texture map → radiance map step followed by a radiance map → texture map step for all PSFs fully describes for a given face the *diffuse to diffuse* transport mode.

Radiance map to radiance map transport

This mode of transport represents specular reflection and does not involve diffuse texture maps. Given a PSF ω_i sending radiance towards a specular reflector, ω_i is reflected about the surface normal of the specular surface to yield the reflected direction ω'_i - this direction is looked up among the PSFs in the light field yielding a PSF ω_j. Because the direction is mirrored the projection of the surface into PSFs ω_i and ω_j will induce a 1-1 mapping over which the radiance stored at the rays is transferred. Only ideal specular materials are supported. Let the specular reflection coefficient be ρ_S then the reflected radiance will be $\rho_S L(\omega, s, t, u, v, Q)$ (where $\rho_S \in [0,1]$) and the absorbed radiance $(1-\rho_s)L$, so there is no loss or gain of radiance due to such a reflection. In practise however $\omega'_i \neq \omega_j$ and further steps are taken in transferring of specular radiance as explained earlier – energy balance is still maintained by using appropriate weighting in that transfer.

This step completes the modes of transport by including specular reflections. A reflection step followed by a radiance map to texture map transfer describes the *specular→diffuse* transport mode, and a texture map to radiance map step followed by specular reflection describes the *diffuse→specular* transport mode. Finally and obviously, the *specular→specular* transport mode is represented by two specular reflection steps.

4.5 Deferred Propagation

As mentioned earlier, the project and clip stage required in transferring radiance is the most compute-intensive stage. A progressive propagation stage can be introduced that prevents a surface from being an emitter of radiance for a particular iteration if it does not meet certain requirements. At the start of a propagation iteration, the total unshot radiance in the system is known. We also know the average reflectance per unit area of the scene expressed below:

$$\rho_S = \frac{\sum_i (A_i \rho_i)}{\sum_i A_i} \qquad (4)$$

where the summation is over all surfaces in the scene and A_i and ρ_i are the area and reflectance of surface i respectively. The product of the unshot radiance in the current iteration with this constant provides an estimate of the unshot radiance at the start of the next iteration. Based on this estimate, a per-surface unshot radiance threshold is determined and Next and Current diffuse texture maps are swapped only if the unshot radiance at the surface is above this threshold. If the radiance maps are not swapped, the surface is marked as deferred and is not selected as an emitter till it exceeds the unshot radiance threshold in a later iteration. The surface will however continue to act as an accumulator of radiance. The scaling used in determining the radiance cut-off threshold for a surface is chosen conservatively. Deferred propagation can also be disabled for the last few iterations to ensure that all surfaces emit their unshot radiance a certain minimum number of times.

5. Implementation Issues

Though the Virtual Light Field allows for correct propagation of illumination in the scene in the limit (with a large *N* and *l*), the discrete representation necessitates several forms of filtering to reduce aliasing and better model the propagation. The problem arises from the fact that the low sampling density (both in the number of directions used, and radiance maps on surfaces) leads to considerable aliasing both in the propagation and walkthrough stages. A simple solution would be to increase the sampling rates – this however is not practical due to memory and computational limitations. An alternative approach is to extract/estimate additional information (using spatial and directional data) at suitable stages and use the results in the given low-sampled representation.

The selection of PSF directions in the VLF is based on the recursive subdivision of a regular tetrahedron [10]. The subdivision so obtained is not ideally uniform for the VLF – for the VLF, an ideal subdivision of the hemisphere for directions requires each direction's solid-angle to be the same; also, the 'shape' of each solid angle should be similar. Neither of these requirements is met by the current subdivision scheme, with as much as a 60% variance in solid-angle. The resulting partition also has unequal ray density (due to varying solid-angle shape) in different parts of the hemisphere. There appears to be no solution in the

literature that provides an alternative subdivision scheme that solves these problems while allowing a constant-time lookup. The problem of varying solid-angles leads to unequal radiance being propagated in different directions over a diffuse surface. To counteract this, the radiance sent along a direction is normalised by a weight based on its corresponding solid-angle.

Another consequence of discretising the directions is the creation of 'holes' during propagation (see Figure 3). Due to discretisation each direction actually represents a beam (containing the set of rays that lie closest to that discretised ray). The problem occurs because individual beams rather than being conical (and widening with distance r) are taken to be cylindrical[1]. This problem cannot be solved when energy is propagated towards a specular surface, as these are represented by directional radiance maps which only exist for the given set of PSF directions. However, in the case of energy propagated towards diffuse surfaces we can use the surface's diffuse texture map to fill in the holes. For diffuse surfaces texture maps are used to propagate radiance whereas directional radiance maps only keep track of total energy propagated in a particular direction. For diffuse receivers, additional directions (beyond those used for 'actual' PSFs) are simulated by perturbing the VLF using a stratified sampling of solid angles. Dimensions of the average solid angle (for the discretised set of PSF directions) are computed at start-up, and a number of stratified samples are randomly selected within this solid angle for each propagation iteration. 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.

The additional directions of propagation in the VLF introduced by the perturbation and the radiance they carry are filtered by the geometric information of the scene in the visibility exchange buffers. Perturbation is not required for all cycles of the propagation process; the number of iterations to perturb is scene dependent and perturbing the initial few is generally adequate. The number of perturbations required per propagation iteration depends on the level of discretisation used for the VLF and the nature of the scene. When enumerating senders for a receiver during a perturbed transfer, visibility lists in the tiles are no longer valid and senders for a receiver in the tile are recomputed from the visibility exchange buffer. All senders then push radiance into the temporary radiance tile which is subsequently mapped to the receiver's next diffuse 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 (directional total radiance maps are updated only during perturbation 0 involving the 'true' PSF direction). Thus, the result of perturbation is brought into effect only in the next iteration when the jittered (and filtered) unshot diffuse texture maps (Q^C) are propagated into the scene.

Following perturbation, Gaussian filtering is required on the next diffuse texture maps to remove the high frequencies that were introduced – this is normally performed with a small σ and filter size to minimise blurring of caustic and shadow boundaries. This action is performed just before the current and next diffuse maps are swapped at the end of an iteration.

During propagation, energy transfer between a sender and receiver within a tile is dictated by the information in the visibility exchange buffer. This buffer is computed using OpenGL false-colour rendered images on a per-receiver, per-PSF basis. The problem with this approach is that though it is much faster than its alternatives, it introduces error and aliasing at polygon boundaries. The solution to this is to use super-sampled visibility exchange buffers to examine and compute the transfer of radiance. This allows for more detailed and accurate project and clip operations. Though this increases the computation required, it is acceptable as it is more expensive only at polygon boundaries and rectifies radiance propagation along edges.

Unlike the perturbation method for diffuse surfaces, we are unable to increase the number of directions of propagation and representation for specular surfaces without actually increasing the number of PSFs being used. In the case of diffuse receivers, geometric information about spatial position and orientation was an adequate filter for a perturbed transfer of radiance; while the (view-independent) diffuse texture maps allowed for correct representation of received radiance. Specular surfaces being view-dependent are not open to a similar scheme as their radiance transfer is not adequately described by just spatial position and orientation. Once the propagation is completed we can however re-sample the directional radiance maps on specular surfaces using backwards ray-tracing, following S^*D paths only.

[1] The choice of cylindrical beams over conical ones is due to both computational complexity and also due to the fact that unequal solid angle 'shapes' lead to complex shaped conical beams.

This backwards ray-tracing is performed by following real ray paths (not only paths dictated by PSF directions) and thus allows for a more accurate representation during walkthrough in the same data structures. We can thus obtain directional radiance maps which are geometrically more accurate and free of any propagation 'holes'. More specifically suppose P is a specular surface, and consider $L_T(\omega,s,t,P)$ the radiance map for P for a particular tile. Each $(u,v) \in V(\omega,s,t,P)$ is a ray that leaves P carrying radiance as computed from the propagation. Now find the specularly reflected direction ω' emanating from P, and trace this ray back until it hits another surface. If this is specular then the same is repeated recursively until a diffuse surface is reached.

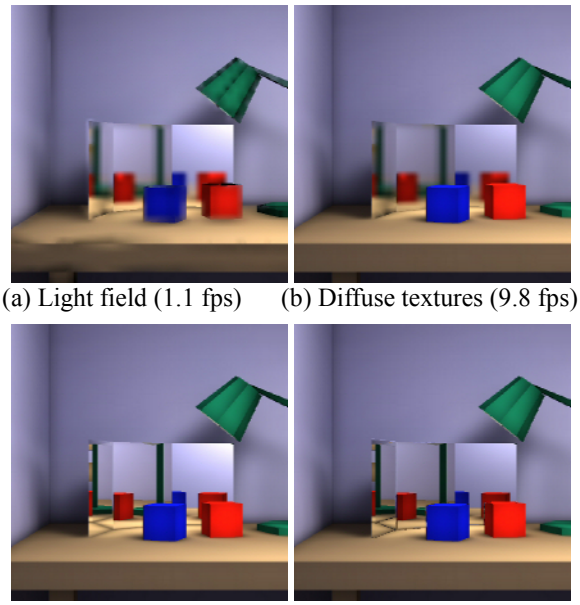
Backwards ray-traced re-sampling of the total radiance maps is performed at the end of the propagation iterations, prior to the final rendering and is performed only once.

6. Rendering

For the final rendering of the light field we have used a variety of methods that allow for different speed for quality improvements. Each is now described.

6.1 Rendering from the rays

An image can be rendered from the Total Radiance Maps stored in the tiles. First a false colour OpenGL rendering is carried out to identify the nearest surfaces intersected by the primary rays. For any primary ray $\mathbf{r}$ we can then lookup the identifier of the nearest intersected face P along the ray, and also compute the intersection point (x,y,z). Given the triangular subdivision of the hemisphere the ray direction of $\mathbf{r}$ will intersect exactly one such hemispherical triangle with vertices ω_i, ω_j and ω_k corresponding to three PSFs. We can now retrieve the radiances in these directions from face P. For each $h \in (i,j,k)$ rotate the intersection point into canonical PSF coordinates $(x,y,z)\mathbf{M}_h = (x_h,y_h,z_h)$. The projection $(x_h,y_h,-1)$ on the corresponding PSF ω_h maps to a pixel $(p,q) = (s_h m+u_h,\ t_h m+v_h)$. We find the face P in the tile list of (ω_h,s_h,t_h) and the radiance is interpolated from the 8-neighbourhood around (p,q) following the same approach as in [7]. This is carried out for each of the three PSFs yielding three radiance values. Then spherical interpolation weights [40] of $\mathbf{r}$ in relation to the three PSF directions are then used to interpolate these radiances.



(a) Light field (1.1 fps) (b) Diffuse textures (9.8 fps)

(c) Ray tracing (0.2 fps) (d) RT ray tracing (20.2 fps)

Figure 7. The results of different rendering methods. Propagated with *N* = 128, *m* =16 and *l* =2049. Propagation time 36.64 hours, 908MB memory.

With this approach, direction and radiance lookups are almost constant time, the only searching being locating the tile for a face, which is logarithmic in the average number of surfaces intersected by a tile. An example is shown in Figure 7(a).

6.2 Texture mapping for diffuse faces

False colour rendering can be wholly avoided for those parts of the image that represent diffuse surfaces. In order to do this we render all diffuse surfaces P textured with their Total texture map P^T, and for the specular surfaces the approach in Sections 6.1 or 6.3 can be applied. In practice this can be done by first rendering the textured diffuse faces, followed by a pass that generates a viewport sized texture, containing the colours of the specular radiances in pixels that strike specular surfaces and colours with an alpha value of zero set in all other pixels. Then this texture is applied to the viewport and the alpha test makes sure that the diffuse surfaces are visible in the appropriate places (Figure 7(b)).

6.3 Backwards ray tracing for specular faces

Usually the PSF directional resolution is too low to provide adequate geometric sampling. In these cases the image exhibits ghosting or blurring of specular reflections. However, in the case of ideal specular surfaces, ray tracing can be used to resolve this. For each ray through a pixel on the image plane which strikes a specular surface, trace its unique reflected ray into the scene recursively until it strikes a diffuse surface P and using bilinear interpolation pick up a value from its Total Texture Map P^T. Applying this to all specularly reflected rays to the eye yields correct geometrical sampling and thus produces visually pleasing reflections (Figure 7(c)).

The downside of this approach is that it is computationally intensive for scenes with many large specular reflectors, due to the fact that many rays must be intersected with the scene. The hardware accelerated OpenGL false colour rendering can only be applied to primary rays (*ES*), so generally this cannot be done in real-time. However, advances in ray tracing are approaching a level where this may become possible [36], especially given that in our approach only *ES*D* rays need be computed, and no shadow rays, nor sampling of area light sources or sampling of diffuse BRDFs are required. An approach similar to [36] has been implemented allowing for real-time walk-through of the VLF (see Figure 7(d)).

Of course a progressive refinement approach can be used, for example, that uses 6.1 or 6.2 while the camera is moving, but when it is stationary a backwards ray tracing pass can be completed as described here. This is best implemented on a dual processor machine so that both the faster, lower detailed image and the slower, more detailed image can be computed simultaneously.

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 were propagated on dual Xeon 1.7Ghz workstations though only one processor was used as the propagation is not yet multi-threaded.

We look at the effect of three parameters: number of PSF directions, the resolution of the PSFs and tiles, and the number of polygons, on propagation time and on memory.

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.

PSF Directions

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 analysis the scene used is shown in Figure 7 and 11. For generating the data, the PSF resolution used was N=64 and m=8. This setup was used for l = 9, 33, 129, 513, 2049, and 8193. The number of propagation cycles was 4.

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.

Numbers of Polygons

Figure 8 shows the scene used for testing the impact of the number of polygons on memory and propagation time. The scene has one emitter, and from 224 to 1736 polygons. The ratio of diffuse to specular surfaces is 5:1 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.

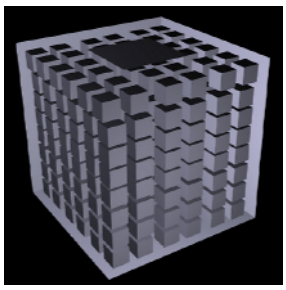


Figure 8. Complexity test scene (OpenGL)

PSF and Tile Resolution

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.

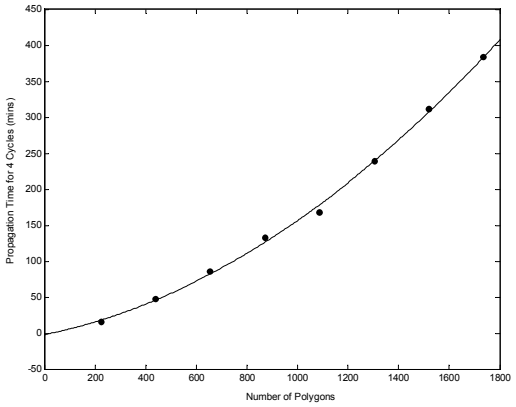


Figure 9. Propagation time by Number of Polygons.

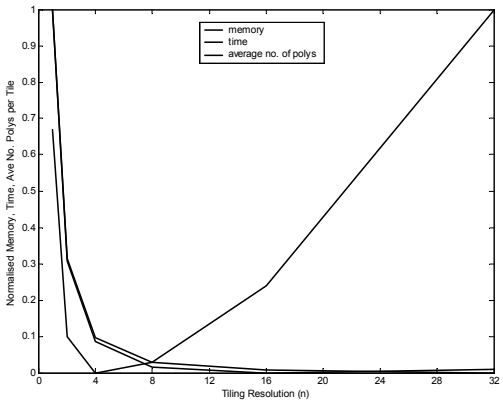


Figure 10. Normalised Memory, Propagation Time, and Average No. of Polygons per Tile, for varying Tile Sizes.

Figure 10 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 tiling 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. The scene used for this test is shown in Figure 11.

The propagation used 513 directions, with $N = 64$ and $m = 8$ throughout. shows that propagation time varies 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.

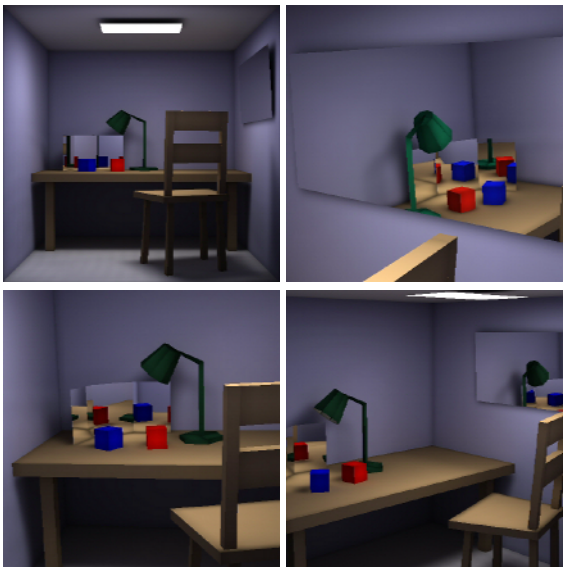


Figure 11. The 'Office' test scene with 137 polygons. The VLF used here is the same as that for Figure 7.

7.2 Images

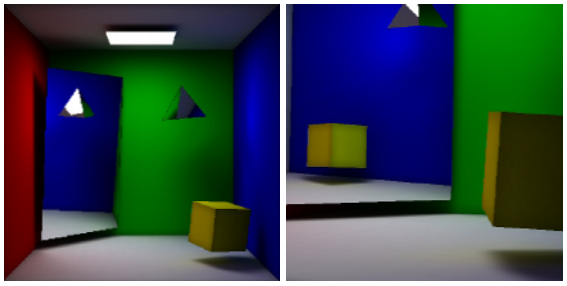
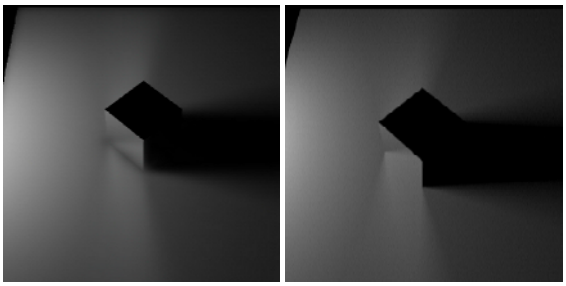


Figure 12. A simple Cornell type scene with 6 specular reflectors. 22 fps, *N* =128, *m* =16, *I* =2049, Propagation time = 16 hours

Figure 12 shows a Cornell type room, which has a large specular mirror and a specular tetrahedron. The scene is simple enough to show that correct radiance is produced.

In order to demonstrate production of caustics via $L(D|S)^{+}SD$ paths, we compare a test scene from [41] with the same scene rendered by the light field using the backwards ray tracing approach described in 6.3. This is shown in Figure 13. A further illustration of caustics is shown in Figure 14. This exhibits some aliasing on the ground plane resulting from the choice of a very small Gaussian kernel for filtering perturbations. Ideally different filtering steps should be carried out for detailed areas such as caustics and shadow boundaries and other diffuse areas.



(a) VLF render (b) Original render

Figure 13. From Smits and Jensen [41]. The light field image has properties 27 fps, N= 256, m= 16, l= 2049, Propagation time= 33.53 hours

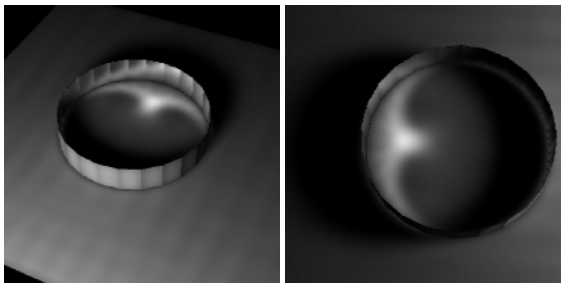


Figure 14. Ring Caustics. This has 50 polygons. 26 fps, *N*= 256, *m*= 16, *l*= 2049, Propagation time= 25.45 hours.

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 7(a) and 7(d) 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 depth complexity of the scene (relative to the tile size).

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 even on laptops is becoming common. Moreover, the Unshot Radiance Maps can be deleted after propagation.

The method is offered as an additional paradigm in the range of solutions for rapid walkthrough with global illumination. 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. First, instead of using perturbation to remove holes we will use an estimation method based on a wavelet representation of the distribution of energy on a surface. This same representation applied to the radiance maps will help with 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.

Figure 15. A study with ceiling and lamp emitters. The two images were produced by the progressive render method.

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.

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 Proceeings, 23(3) 89-98..

[8] Slater, M. (2000) A Note on Virtual Light Fields, Department of Computer Science Research Note RN/00/26/ April 5th 2000, University College London. www.cs.ucl.ac.uk/staff/m.slater/publications.htm.

[9] Foley, J.D., van Dam, A., Feiner, S.K., Hughes, J.F. (1996) Computer Graphics: Principles and Practice in C (2nd Edition), Addison-Wesley.

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

[11] 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.

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

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

[14] Cohen, M.F. and Greenberg, D.P. (1985) The Hemi-Cube: A Radiosity Solution for Complex Environments, Computer Graphics (SIGGRAPH)19(3), 31-40.

[15] 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.

[16] Immel, D.S., Cohen, M.F. and Greenberg, D.P. (1986) A radiosity method for non-diffuse environments, Computer Graphics (SIGGRAPH) 20(4), 133-142.

[17] Wallace, J.R., Cohen, M.F., and Greenberg, D.P. (1987) A two-pass solution to the rendering equation: A synthesis of ray tracing and radiosity methods, Computer Graphics (SIGGRAPH), Proceedings of the 14th annual conference, p311-320.

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

[19] Chen, S. E., Rushmeier, H. E., Miller, G. and Turner, D. (1991) A progressive multi-pass method for global illumination, Computer Graphics, 25(4), 165–174.

[20] Aupperle, L. and Hanrahan, P. (1993) A hierarchical illumination algorithm for surfaces with glossy reflection, Computer Graphics Proceedings, Annual Conference Series, 155–162.

[21] 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.

[22] 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.

[23] Simmons, M. and Séquin, C. H. (2000) Tapestry: A dynamic mesh-based display representation for interactive rendering, in Rendering Techniques 2000 (Proceedings of the Eleventh Eurographics Workshop on Rendering), (eds: B. Peroche and H. Rushmeier, 329–340.

[24] Stamminger, M., Haber, J., Schirmacher, H. and Seidel, H.-P. (2000) Walkthroughs with corrective texturing, Rendering Techniques 2000 (Proceedings of the Eleventh Eurographics Workshop on Rendering), (eds: B. Peroche and H. Rushmeier, 377–388.

[25] 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.

[26] Drettakis G. and Sillion, F. X. (1997) Interactive update of global illumination using a line-space hierarchy, Computer Graphics, 31(Annual Conference Series, SIGGRAPH), 57–64.

[27] Stamminger, M., Scheel, A., Granier, X. , Perez-Cazorla, F. , Drettakis, G., and Sillion, F. X. (2000) Efficient glossy global illumination with interactive viewing, Computer Graphics Forum, 19(1), pp. 13–25 (2000).

[28] Granier X. and Drettakis, G. (2001) Incremental updates for rapid glossy global illumination, Computer Graphics Forum (Eurographics 2001) 20(3).

[29] 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.

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

[31] Stürzlinger, W. and Bastos, R. (1997) Interactive rendering of globally illuminated glossy scenes, Eurographics Rendering Workshop 1997 (J. Dorsey and P. Slusallek, eds.), 93–102.

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

[33] 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.

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

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

[36] 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.).

[37] Wald, I., Kollig, T., Benthin, C., Keller, A., Slusallek, P., (2002) Interactive Global Illumination Using Fast Ray Tracing, Thirteenth Eurographics Workshop on Rendering, P. Debevec and S. Gibson (eds).

[38] Benthin, C., Wald, I., Slusallek, P. (2003) A Scalable Approach to Interactive Global Illumination, Eurographics 2003 22(3) P. Brunet and D. Fellner (eds).

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

[40] Alfeld, P., Neamtu, M. and Schumaker, L.L. (1996) Bernstein-Bézier Polynomials on Spheres and Sphere-Like Surfaces., CAGD 13, 333--349.

[41] Smits, B. and Jensen, H.W. (2000) Global Illumination Test Scenes; Tech. Rep. UUCS-00-013, Computer Science Department, University of Utah, June 2000.
http://www.cs.utah.edu/~bes/papers/scenes/