1. Introduction

In this chapter we show how to achieve 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 (Heckbert, 1990) can be simulated (including caustics). The method exploits the idea of light fields (Levoy and Hanrahan, 1996) (or lumigraphs, Gortler et al., 1996), though the particular type of light field representation used is similar to that in (Camahort et al, 1998) and also similar to a data structure used for visibility culling in (Chrysanthou, Cohen-Or and Lischinski, 1998). It exploits the idea of Layered Depth Images (Shade et al., 1998) 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 (Szirmay-Kalos, 1999). 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 tiles. Standard polygon rasterisation is used to compute ray-polygon intersections rather than a Painters’ Algorithm.
The ideas presented in this chapter may also be thought of as a combination of light field and illumination network (Buckalew and Fussell, 1989). 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.

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 (Goral et al., 1984; Cohen and Greenberg, 1985). 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 (Cohen, Shenchang, Wallace, and Greenberg, 1988). 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 (Immel et al., 1986; Wallace et al., 1987), 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 (Chen et al., 1991). Hierarchical radiosity (Hannahan et al., 1991) greatly speeded up the radiosity propagation phase, and was extended to include glossy reflection (Aup-
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 (Drettakis and Sillion, 1997), including glossy illumination (Stamminger et al., 2000; Granier and Drettakis, 2001).

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 (Ward and Simmons, 1999; Walter et al., 1999; Simmons and Séquin, 2000; Stamminger et al., 2000; Tole and Pellacini, 2002). Precompute algorithms compute a global illumination solution and then approximate this in some way for rapid rendering. For example (Walter et al., 1997; Keller, 1997) compute virtual point light sources that produce direct illumination approximating the global solution. As another example, in (Stürzlunger et al., 1997) photon tracing (Jensen, 1996) 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 (Gobbetti et al., 2003) 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 (Whitted, 1980) and path tracing (Kajiya, 1986), to the point where interactive speed for millions of polygons on clusters of consumer PCs has become possible (Wald et al., 2003). 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 (Wald et al., 2002; Benthin et al., 2003). An excellent summary and overview can be found in (Dutre et al., 2003).

The ‘virtual light field’ approach has similarities to many other approaches: it is like photon mapping (Jensen, 1996) 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 particular ray space discretisation followed in this research is similar to that in (Camahort et al, 1998) 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 a scene (or part of a larger scene) can be well enclosed, for example, by a regular cuboid. Suppose this is a cuboid bound by (-1,-1,-1) to (1,1,1). Consider the lower face (at z = -1) bounded by (-1, -1, -1) to (1,1,-1). This is discretised into $N \times N$ pixels. The pixel position $(i,j)$ corresponds to the point

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

Consider this as the origin of a ray that is parallel to the z-axis, i.e., parallel to the vector (0,0,1). The set of all $N \times N$ rays is called the canonical parallel subfield (PSF), and (0,0,1) is its direction. If $l$ points with spherical coordinates $\omega_i = (\theta_i, \phi_i)$ are chosen on the unit hemisphere that is over the $z=0$ plane then $l$ PSFs are defined as rotations of the canonical PSF by rotating the direction into the corresponding point. The rotation can be achieved in any manner that is consistent throughout (for example, first rotate by $\theta_i$ about the $y$-axis to bring the vector to the $zx$ plane and then $\phi_i$ about $z$, to bring the vector into $\omega_i$).
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 
\[ r(t) = r_0 + vt \] 
where \(v\) is the direction vector \((0,0,1)\), and \(r_0\) is the ray origin, then the intersection points can be characterized as an array of non-decreasing parametric values \([t_1,t_2,...,t_k]\). At each one of these intersection points additional information will be stored: the identifier of the surface at that intersection, and eventually the outgoing radiance from the surface at that intersection point. This is possible, and the first implementation (Slater, 2000) 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 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 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 Figure 1. A PSF, Tile, Polygon and Visibility Map Representation as an Edge Table. 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.
Virtual Light Fields

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 very small modifications to the standard polygon rasterisation algorithm (e.g., Foley et al., 1996). If the polygons are also convex then each ET entry will only have either 0 or 2 entries per row.

So far we have only discussed the canonical PSF. Given any other PSF, corresponding to direction \( \omega_i \), the scene can be rotated such that \( \omega_i \) is mapped to the (0,0,1) direction 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. Suppose the outgoing radiance in direction \( \omega \) at a particular point \((x, y, z)\) on surface \( P \) is required. Find the direction amongst \( \omega_i \) \( (i=0,1,\ldots,L-1) \) that is closest to \( \omega \) and suppose that this is \( \omega_j \). There will be a rotation matrix \( M_j \) that rotates \( \omega_j \) into (0,0,1). Then \((x, y, z)M_j = (x_q, y_q, z_q)\) will be the point in the

![Figure 1. A PSF, Tile, Polygon and Visibility Map Representation as an Edge Table](image)
4. Propagation

This section discusses propagation while abstracting from practical implementation issues, which are considered in Section 5. First we provide an overview, and then go into detailed considerations regarding propagation through particular light paths.

4.1 Overview

Notation. The (finite) set of given PSF directions is denoted $\Omega_l$ 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 $(\omega,s,t)$. The coor-
ordinate system 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 \(\omega\) and with origin at coordinates given by Equation (EQ 79) where \((i,j) = (sm + u,tm + v)\). Then for surface \(P\), \(L(\omega,s,t,u,v,P)\) is the radiance from \(p\) along the specified ray. Suppose \(p'\) refers to a texture map for \(p\) then the ray will intersect a particular texel within that map, and \(L(\omega,s,t,u,v,P')\) is the radiance picked up from that texel. We will sometimes use the abbreviation \(r = (\omega,s,t,u,v)\).

**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 are in fact two radiance maps: called the total radiance map and unshot radiance map. A radiance map is a 2D array of radiance values of size \(m \times m\) (later we will relax the requirement that the radiance maps are the same resolution as the tile resolution). 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 \((\omega,s,t,u,v)\) from surface \(P\) in the direction going out on the same side of \(P\) as its outward normal. Obviously this is radiance for \(P\) in tile \((\omega,s,t)\) in position \((u,v)\) within the tile. \(L(\omega,s,t,P)\) is a radiance map for \(P\) in the tile \((\omega,s,t)\). 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 \(p\) has two associated texture maps \(p^C\) (current) and \(p^N\) (next) to store radiance values due to diffuse reflection. Any ray \((\omega,s,t,u,v)\) 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 this surface from the area corresponding to the texel. New radiance due to diffuse reflection that is generated within the current cycle is stored in the next radiance map \(p^N\) and will be distributed in the next cycle.
4. Propagation

**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 any 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 $\omega$ the pixel $(u, v)$ within tile $(\omega, s, t)$ is set, otherwise the value is 0. We will use the notation $(u, v) \in V(\omega, s, t, P)$ 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 that ray with direction corresponding to the PSF. The exchange buffer is a 2D array of identifiers of the surface that is visible to $p$ at each $(u, v)$ within the visibility map of $p$. It is constructed on the fly as surface $p$ is processed for incoming energy, and deleted after use. $EB(\omega, s, t, P)$ is the exchange buffer for polygon $p$ in the tile $(\omega, s, t)$. $EB(\omega, s, t, u, v, P) = Q$ if and only if the ray $(\omega, s, t, u, v)$ 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 $EB(\omega, s, t, u, v, P) = Q$ if and only if $EB(\omega, s, t, u, v, Q) = P$.

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 and each tile within each PSF, and each surface within each tile. In propagation cycle 0 the light source current texture maps 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 $1 \times 1$ arrays. The following pseudo code illustrates how each propagation cycle unfolds.
for each $\omega \in \Omega_1$ /*1*/
for each $s,t = 0,...,n-1$ /*2*/
for each surface $P \in (\omega,s,t)$ /*3*/
$\omega' \in \Omega_i$ approx specular reflection direction;
$EB = $ exchange buffer for $P$ ;
for each $(u,v) \in V(P,\omega,s,t)$ /*4*/
$\rho = (\omega,s,t,u,v)$ ;
$Q = EB(\rho,P)$ ; /*5*/
$L_d = L(\rho,Q^C)$ ; /*6*/
$L_s = L_U(\rho,Q)$ ; /*7*/
$L = L_d + L_s$ ; /*8*/
$L_T(\rho,Q)^+ = L$ ; /*9*/
$L(\rho,P^N)^+ = L$ ; /*10*/
$\rho' = (\omega',s',t',u',v')$ spec. reflected ray;
$L_U(\rho',P)^+ = \rho L$ ; /*11*/
end
end
end
next texture maps = current current 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.
4. Propagation

(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 all rays in the visibility map are considered, and
(5) the surface \((Q,\text{ if any})\) that sends energy to \(P\) along this ray is extracted from the exchange buffer.
(6) The texel in the current texture map for \(Q\) intersected by the ray is looked up and an appropriate fraction of the radiance there is extracted.
(7) The unshot radiance map for \(Q\) is looked up at the ray position and the radiance there is extracted.
(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 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 \((Q)\) 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 \( P \) and sender \( Q \) 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 (see Figure 2. Diffuse to diffuse transfer along a single PSF tile via the temporary radiance tile.). 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 which along with a continuous radiance mapping approach using an area-overlap method allows for proper sampling and transfer of energy. With this framework, 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 \( P^N \)).

![Figure 2. Diffuse to diffuse transfer along a single PSF tile via the temporary radiance tile.](image)

The 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); the loaded radiance tile is then mapped onto the next texture map of the receiver. The mapping is performed with reference to the visibility exchange buffer. Discrete cells on a sender are mapped onto a receiver using a polygon clipping algorithm so that contributions from a single shooting cell can be correctly distributed amongst the receiver cells. The ratio of intersected versus total area of
4. Propagation

the sender cell is used in determining the amount of radiance to be 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 diffuse map (\(Q^C\)) in the PSF direction and clipping against 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 about 85-90% 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 \(L_r(\omega,s,t,Q)\).

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 (EQ 80):

\[
F_\gamma \equiv \frac{1}{A_i A_j} \int \int \frac{\cos \phi_i \cos \phi_j dA_i dA_j}{\pi r^2} H_{ij}
\]

(EQ 80)

The inverse scaling of irradiance by \(\pi\) during the propagation process is explained in Section 4.4, Equation (EQ 81). 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 the VLF and the propagation process (see Figure 3. Angular spread of PSF surface hits with distance \(r\)). 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 tied up 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 $\omega$ is that energy from a diffuse sender $Q$ towards a specular receiver $P$ is reflected towards a PSF direction $\omega'$ by $P$. If this transfer is performed by a forward mapping from the diffuse sender into the reflected direction, artifacts (seen as holes) are created on the unshot radiance map $L_U(\omega', s', t')$ on the receiver $P$. 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 $r' = (\omega', s', t', u', v')$ on the unshot radiance map $L_U(r', P)$ receives energy from the sender $Q$. Energy from the sender is thus added to the unshot radiance map in the reflected direction as illustrated below (Figure 4. Diffuse to specular transfer along a PSF using backward mapping.).
4. Propagation

This transfer is performed in two stages. The four corners of a tile \((s,t)\) in PSF direction \(\omega\) are first projected onto the reflected PSF direction \(\omega'\) 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 get energy from the sender \(Q\). In the next stage, each candidate cell is projected inversely onto the current PSF \((\omega,s,t,u,v)\). If the projected cell is visible from \(L_U(\omega',s',t',P)\) as indicated by the visibility exchange buffer then energy is pulled from the current texture map of the sender \(Q^C\).

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. Then the energy of the diffuse sender is transferred along each of these directions weighted by the 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 \(P\) as indicated by the visibility exchange buffer, the unshot radiance from the sender \(L_{U} (\omega, s, t, Q)\) is accumulated into the temporary radiance tile (as shown in Figure 5. Specular to Diffuse transfer along a PSF.). This transfer is a simple one-to-one mapping since \(L_{U} (\omega, s, t, Q)\) and the temporary radiance tile have the same resolution and are along the PSF direction.
4. Propagation

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 $Q_f(\omega, s, t)$ is by the process described for the second stage of diffuse to diffuse transfer (Section 4.1).

4.4 Energy balance

In this section we show that light transport 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 faces in texture maps and radiance leaving a face in a particular direction is stored in the light field tiles in directional radiance maps the tiles of a PSF, corresponding to the PSF direction. When equilibrium is reached the texture maps are no longer needed and could be discarded. The light field itself has all the necessary information from which to render images (although the texture maps can be used for rendering as discussed later). The current version of the algorithm only support diffuse emitters so the propagation stage always starts from light source texture maps loaded up with irradiance corresponding to the isotropic emission of the light source(s). (It is trivial to add anisotropic emitters). Currently, we ensure that we have a many-to-one mapping between elements of the texture maps and the tile

Figure 5. Specular to Diffuse transfer along a PSF.

Lighting - the Radiance Equation 83

Great Events of the Twentieth Century 83
directional radiance maps in order to satisfy the sampling frequency of the light field. This is especially important for high-frequency phenomena such as shadow boundaries and caustics. Thus (during propagation) energy in this system consists of a manifold of directional and non-directional elements.

We need to ensure that the modes of transport in our system obey the energy conservation law. All modes of transport can occur within the constraint that we support only ideal specular and diffuse materials (using Heckbert’s notation (Levoy and Hanrahan, 1996): \( D \rightarrow D \), \( D \rightarrow S \), \( S \rightarrow D \) and \( S \rightarrow S \). However, since the propagation is based on estimating radiance flow between radiance maps this can be broken down to the following steps, sequences of which map to the modes of transport listed above: texture map \( \rightarrow \) radiance map, radiance map \( \rightarrow \) texture map and radiance map \( \rightarrow \) radiance map.

Figure 6. This shows the projection of a radiance map onto a texture map. The centers of the quadrilaterals (projected ray beams) are the ray origins. Each texel on \( Q^C \) typically maps to inside one ray within a tile for the radiance map.
4. Propagation

\[ L_e(\alpha, s, t, P) \]. However, the worst case is that a texel intersects the boundary of 4 neighbouring rays, such as at texel X.

**Diffuse map to radiance map transport.** The radiant exitance \( M \) stored at texel \((u_x, v_x)\) 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 \( \omega \) the texel will project to a cell \((\alpha, s, t, u, v)\). This cell corresponds to a ray leaving \( Q \). Due to the discrete nature of the representation 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 6. This shows the projection of a radiance map onto a texture map. The centers of the quadrilaterals (projected ray beams) are the ray origins. Each texel on typically maps to inside one ray within a tile for the radiance map. However, the worst case is that a texel intersects the boundary of 4 neighbouring rays, such as at texel X. 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.

The ray leaving \( Q \) will be carrying radiance \( L_x(\alpha, s, t, u, v, Q) \), 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 contribution to the radiance of the ray becomes:

\[
L(\alpha, s, t, u, v, Q) + = M(P, u_x, v_x) \times \frac{A(\alpha, Q, u_x, v_x)}{\sum_{i=0}^{\infty} A(\alpha, Q, u_x, v_x)}
\]  

(EQ 81)
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 $Q$, and $M(Q, u_q, v_q)$ is the radiant exitance of texel $(u_q, v_q)$ on face $Q$. 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 discretation 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 (EQ 81).

This step 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 map transport.** When energy is sent from a radiance map $L_U(\omega, s, t, Q)$ to a texture map $p^V$ it is stored in some texel $(u_p, v_p)$. This texel will, by the end of each propagation cycle, have summed the irradiance $E$, due to incident radiances $L_U(\omega, s, t, Q)$ over the hemisphere at the texel. This is a simple sum of the radiance carried by rays in the light field that intersect that texel.

In a closed scene take the set of rays (in fact, beams) induced by the light field 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 (EQ 81), which means that the ‘outgoing’ projected area equals the ‘incoming’ projected area overall (see ).
4. Propagation

Figure 7. Illustrates how "outgoing" projected area (dashed grey) equals "incoming" projected area (grey, black and dashed black) overall in a simple VLF with only two PSFs

So since the radiant exitance is distributed over an area equal to the area over which it is gathered again and the environment is closed the energy conservation law will be obeyed.

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

**Radiance map to radiance map transport.** This mode of transport describes specular reflection. It takes place entirely within the light field and does not involve the texture maps at all. Given a PSF $\omega_i$ sending radiance towards an ideal specular reflector, $\omega_j$ is reflected about the surface normal of the specular surface to yield the reflected direction $\omega_j'$ which is then looked up among the PSFs in the light field yielding a PSF $\omega_j$. Because the direction is mirrored the projection of the surface into PSFs $\omega_i$ and $\omega_j$ will induce a 1-1 mapping over which the radiance stored at the rays is transferred (see Figure 8. Illustrates how a specular reflection step is carried out, the radiance travelling along PSFwi is specularly reflected via a specular surface (dashed grey) into PSFwj).
Only ideal specular materials are supported. Let the specular reflection coefficient be $\rho_s$, 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 so there will be no loss or gain of radiance due to such a reflection.

This step completes the modes of transport by including specular reflections. A reflection step followed by a radiance map to texture map, will describe the specular to diffuse transport mode, and a texture map to radiance map step followed by a specular reflection will describe the diffuse to specular transport mode. Finally and obviously, two such specular reflection steps will describe the specular to specular transport mode.

5. Implementation Issues

Though the Virtual Light Field allows for correct propagation of illumination in the scene in the limit (large $N$ and $l$), the discrete representation necessitates several different forms of filtering to achieve accuracy and avoid aliasing. The problem arises from the fact that the low sampling density (both in the number of directions...
5. Implementation Issues

used, and radiance maps on surfaces) leads to considerable aliasing both in the propagation and the final rendering 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 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 (Slater, 2002). 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 seems to be no solution in the literature that provides a suitable alternative subdivision scheme that solved these problems and that was compatible to the constant-time lookup that is required. The problem of varying solid-angles leads to unequal radiance being propagated in directions over a diffuse surface. To counteract this, the radiance sent in a direction is normalised by a weight based on its corresponding solid-angle.

Another major consequence of the discretisation of directions is that ‘holes’ in propagation are created (see Figure 3. Angular spread of PSF surface hits with distance \( r \)). Due to the discretisation of directions, 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 set of actual PSF directions. However, in the case of energy propagated towards diffuse surfaces (from both specular and diffuse senders) we can use the surface’s texture map to fill in the holes. Texture maps are used to propagate radiance while 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 by a stratified sampling of solid angles. Dimensions of the average solid angle

---

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.
(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; 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 also on the nature of the scene. When enumerating senders for a receiver during a perturbation, 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 maps \(Q^C\) are propagated into the scene.

Following perturbation, Gaussian filtering is required on the next diffuse maps to remove the high frequencies that were introduced – this is normally performed with a suitably small \(\sigma\) 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 a propagation 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
only occurs at polygon boundaries and rectifies radiance propagation at polygon 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 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.

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 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 $L_f(\omega,s,t,P)$ 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 $\omega'$ 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 following the normal practice of ray tracing.

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, trading off speed for quality. Each is now described.
6.1 Rendering from the rays

This method uses only the radiance information stored in the light field to render an image. Given a camera pointed at the scene we compute a ray $r$ going from the eye $e$ through a pixel $p$ on the image plane (see Figure 9. (a) Camera setup and (b) closeup of ray striking back wall with three nearest PSF directions). Suppose $r$ has direction vector $q$. Now, in order to determine the radiance travelling through $p$ we need to know which $P$ is intersected by $r$. We use OpenGL accelerated false-colour rendering to determine this. This simply renders the scene with z-buffering using the viewing parameters given by the virtual camera, colouring each face in an unique colour with a 1-1 mapping from colours to face identifiers. The frame is read back from the framebuffer and the mapping is applied, yielding a matrix of face identifiers of the intersected faces.

For any primary ray $r$ 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 $r$ will intersect exactly one such hemispherical triangle (see Figure 9). At the corners of this triangle are three directions $\omega_k, \omega_l$ and $\omega_m$ corresponding to three PSFs. We can now retrieve the radi-
6. Rendering

ances in these directions from face \( P \). For each \( i \in (k, l, m) \) rotate the intersection point into canonical PSF coordinates \((x, y, z) = (x_i, y_i, z_i)\). Now the projection \((x_i, y_i, -1)\) on the corresponding PSF \( \omega_i \) maps to a pixel \((i, j) = (sm + u, tm + v)\).

We find the face \( P \) in the tile list of \((\omega_j, s, t)\) and the radiance is bilinearly interpolated from the 8-neighbourhood around \((i, j)\) following the same approach as in (Buckalew and Fussell, 1989). This is carried out for each of the three PSFs yielding three radiance values. Then spherical interpolation weights ((Note that these references are not exactly in alphabetical order).) of \( r \) in relation to the three PSF directions is used to interpolate these radiances.

This rendering method is by far the fastest, the major bottleneck being the OpenGL false colour rendering and associated framebuffer readback. Direction and radiance lookups are 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.

6.2 Backwards ray tracing for specular faces

It is more often than not the case that the PSF directional resolution is too low to provide adequate sampling for a camera with \( UxV \) samples spread within its field-of-view. In these cases the image will exhibit ghosting effects on specular reflections, due to the poor sampling of the geometry. However, in the case of ideal specular surfaces extrapolation to resolve this can be applied. The basic idea is for each ray travelling 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 diffuse texture map \( p^\tau \). This does not limit the light paths sampled by the light field propagation step. The propagation step is \( L(D|S)^* \) and the rendering step will be \( ES*D \) yielding \( ES*D(D|S)^*L \). The total diffuse texture maps will form a correctly illuminated boundary between the propagation step and the rendering step. The 1-1 ray mapping over the specular reflections ensure that the only energy that can travel across a \( ES^* \) path must come from a diffuse surface or an emitter (also considered as a diffuse surface).

Applying this method to all specularly reflected rays to the ey yields correct geometrical sampling and thus produces visually pleasing reflections, and combined with the previous approach applied to the diffuse rays good results are obtained.
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 first hits (ES), so generally this cannot be done in real-time. However, advances ray tracing is approaching a level where this may become possible (Wald et al., 2003), especially given that in our approach only ES*D rays must be computed, no shadow rays, sampling of area light sources or sampling of diffuse BRDFs are required.

6.3 Texture mapping for diffuse faces

As discussed in section 6.1 above, the bottlenecks of that approach is the false colour rendering and the tile searching. These steps can be wholly avoided for those parts of the image that directly display diffuse surfaces, though sacrificing some of the elegance and generality of the light field approach. In order to do this we render all diffuse surfaces \( P \) textured with their total diffuse texture map \( P^{T} \), and for the specular surfaces the approach in Sections 6.1 or 6.2 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 0.0 set in all other pixels. Then this texture is applied to the viewport and the alpha test will make sure that the diffuse surfaces are visible in the appropriate places.

6.4 Progressive method

At the time of writing a combination of the method described in Section 6.3 for diffuse and the method described in Section 6.1 for specular achieves the best frame-rate. In this approach the specular reflections exhibit some ghosting and/or blurring, although these are not important while the viewpoint is moving. However, in a typical interactive walkthrough the viewer will typically pause when inspecting some particular part of the scene, and in these moments progressive refinement can be used. It is possible to exchange the specular method with the one described in Section 6.2 progressively enhancing the image when the viewer pauses. Given a dual CPU computer this method can be realised by having two threads one apply-
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 scene 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. The test scenes were propagated on several workstations which varied only in the type of graphics card installed – the results obtained were thus subject to small random variation depending on which machine the propagation was performed on. In order to determine the scalability of the various aspects of the VLF, options such as perturbed transfers and super-sampled visibility exchange buffers were disabled. It was however noted that an iteration involving five perturbations (one along the original PSF directions and four stratified samples) depending on the scene, took between 2 to 3 times as long as a non-perturbed iteration.

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.

PSF Directions. The higher the number of directions the greater accuracy there will be 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 requires more memory and longer propagation times. For this analysis we use a Cornell type of scene, which contains diffuse walls, a specular mirror, and box and tetrahedron. The scene has 1 emitter, 14 diffuse polygons, and 5 specular polygons. 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 3. Figure 10. Propagation time by Number of PSF Directions, shows the plot of propagation time against the number of PSF directions. The resulting trend line is clearly linear over the range shown. The same is true for Figure 11. Memory by Number of PSF Directions regarding memory.
7. Results

Figure 10. Propagation time by Number of PSF Directions.

Figure 11. Memory by Number of PSF Directions
Numbers of Polygons. Figure 12. Test Scene for Performance Analysis shows the scene used for testing the impact of the number of polygons on memory and propagation time. The scene has one emitter, and numbers of polygons ranging from 187 through to 1087. The ratio of diffuse to specular surfaces is close to 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.
7. Results

Figure 13. Propagation time by Number of Polygons.

The propagation used 513 directions, with $N = 64$ and $m = 8$ throughout. Figure 13. Propagation time by Number of Polygons. shows that propagation time varies quadratically with the number of polygons. The actual least squares equation is shown in (EQ 82), and the quadratic term is significant.

$$time = -13.78 + 0.3296 \cdot \text{NumPoly} + 0.0003361 \cdot \text{NumPoly}^2$$  \hspace{1cm} (EQ 82)
Figure 14. Memory by Number of Polygons shows the plot for memory requirements. This is linear over the range considered.

**PSF and Tile Resolution.** The resolution of the PSF determines the accuracy with which the geometry and radiance is sampled. The size of the tiles relative to the PSF resolution is important in determining overall speed of propagation and rendering (when the light field itself is used for 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. In this section we look at the impact of tile size and PSF resolution.

8. Conclusions

This chapter has discussed a different approach to the problem of global illumination using light fields. 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. The
8. Conclusions

biggest drawbacks to the approach in its current version are the very large propagation times and the large memory requirements. On the other hand, a scene needs only to be propagated once, and gigabytes of memory even on laptops is becoming common. Some examples are shown below.
Virtual Light Fields

![Diagram of light field](image1)

![Diagram of light field](image2)

![Diagram of light field](image3)

![Diagram of light field](image4)

Lighting - the Radiance Equation

Great Events of the Twentieth Century