Interactive Display of Global Illumination Solutions for Non-Diffuse Environments

Wolfgang Heidrich

Department of Computer Science
The University of British Columbia
2366 Main Mall
Vancouver, B.C. V6T 1Z4 CANADA
heidrich@cs.ubc.ca

Abstract

In recent years there has been a lot of work on interactively displaying global illumination solutions for non-diffuse environments. This is an extremely active field of research, in which a lot of different approaches have been proposed recently. In this State-of-The-Art-Report, we will discuss and compare these. This will hopefully lay the ground for systematically addressing the open questions in the future.

1. Introduction

Indirect illumination is an important visual effect that contributes significantly to the realism of computer generated imagery. Thus, it is not surprising that several methods for including these effects in interactive applications have been developed over the past few years.

For example, a common technique for displaying the indirect illumination in a diffuse environment is to use Gouraud shading and texture mapping for displaying the results of a Radiosity simulation[1, 2, 3, 4]. Due to the view-independent nature of illumination in a diffuse environment, the expensive precomputation step can easily be separated from the inexpensive display step, which allows for high frame rates using graphics hardware.

The case of non-diffuse reflection is more challenging and interesting, because it requires either precomputing and storing a large amount of data, or the execution of relatively expensive operations on the fly. The purpose of this State-of-The-Art-Report is to show the current status quo of methods related to interactive display of global illumination solutions, and to compare the different approaches to each other.

In practice, we find that the different approaches constitute a smooth transition from precomputed illumination with large storage requirements to on-the-fly computations consuming little or no memory. For the purposes of this report, we have grouped the different methods in the following categories:

- Image-based methods using precomputed environment maps (Section 2). These methods ignore parallax effects due to objects that are large compared to their environment. Environment maps allow for an efficient reconstruction based on texture mapping.
- Image-based methods using precomputed light fields (Section 3). As in the case of environment maps, this allows for efficient reconstruction that is mostly a simple table lookup or a lookup combined with some interpolation.
- Reconstruction from sparse or scattered light field information (Section 4). Like the methods in Section 3, these techniques also work on precomputed light fields, but they usually use fairly sparse representations that require a significant amount of work during a reconstruction phase to be performed on the fly.
- On-the-fly methods (Section 5). Here, the global illumination is completely computed during rendering time. This means that dynamic scenes can trivially be handled, but on the other hand the complexity of the scene or the illumination effects that can be dealt with are often seriously limited for performance reasons.

2. Environment Maps

In this part of the tutorial we are going to discuss environment maps[5] and their applications to interactive rendering. Environment maps are particular textures that describe for all directions the incoming or outgoing light at one point in space. The main use of these maps is to simulate mirror reflections in curved objects, but they can do much more than that. In particular in hardware-accelerated renderers, environment maps are often used to store precomputed directional information that is too expensive to compute on the fly.

The basic idea of environment maps is that, if a reflecting object is small compared to its distance from the environment, the incoming illumination on the surface really only depends on the direction of the reflected ray. Its origin, that is the actual position on the surface, can be neglected. Therefore, the incoming illumination at the object can be precomputed and stored in a 2-dimensional texture map.

If the parameterization for this texture map is cleverly chosen, the illumination for reflections off the surface can be looked up very efficiently. Of course, the assumption of a small object compared to the environment often does not hold, but environment maps are a good compromise between rendering quality and the need to store the full 4-dimensional radiance field on the surface.

Both offline[6] and interactive, hardware-based renderers[7] have used this approach to simulate mirror reflections, often with amazing results.

In this section, we first discuss the issue of parameterizations (or representations) for environment mapping. Afterwards, we compare techniques for using environment maps for matte reflections and different reflection models.

2.1. Parameterizations for Environment Maps

Since environment maps represent directional information as a 2D texture, it is necessary to decide for a mapping from directions to texture coordinates in order to define a concrete representation. This mapping, which is also called the *parameterization* of the environment map, should fulfill a couple of properties in order to be useful for interactive rendering:

- for walkthroughs of static environments, it should not be necessary to create a new environment map every frame. This means that
  - the computation of the texture coordinates is possible for all viewing directions.
  - all light directions need to be represented equally well in the environment map. Although some light directions are more important than others for a certain viewing direction, all directions are equally important for a walkthrough, where the viewing direction is not previously known. This property is called the *uniformity* of the parameterization.
- for interaction with dynamic environments, it should be easy and inexpensive to create a new environment map from perspective images of the scene (because this is what the hardware can generate).
- the method for computing the texture coordinates should be simple and efficient, and it should be easy to implement in hardware. This means that complicated and expensive mathematical functions line trigonometric functions should not be necessary.

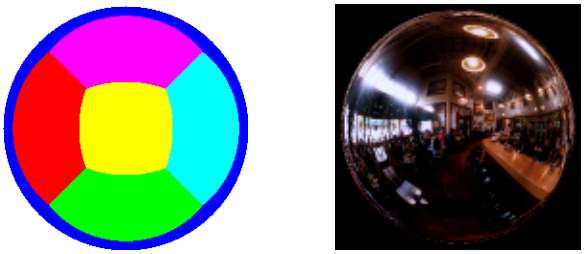

Figure 1: *Left: A spherical environment map from the center of a colored cube. Note the bad sampling of the cube face directly in front of the observer (black). Right: a spherical map of a real scene.*

In the following, we will discuss the three parameterizations for environment maps that have gained some importance in interactive and hardware-accelerated rendering.

Spherical Maps. The parameterization traditionally used in computer graphics hardware is the *spherical environment map*[8]. It is based on the analogy of a small, perfectly mirroring metal ball centered around the object. The image that an orthographic camera sees when looking at such a ball from a certain viewing direction is the environment map. An example environment map from the center of a colored cube is shown on the left of Figure 1, a map of a real scene is shown on the right.

The major reason why spherical maps are used is that the lookup can be computed efficiently with simple operations in hardware (see Figure 2 for the geometry): for each vertex compute the reflection vector $\vec{r}$ of the per-vertex viewing direction $\vec{v}$. A spherical environment map which has been generated for an orthographic camera pointing into direction $\vec{v}_o$, stores the corresponding radiance information for this direction at the point where the reflective sphere has the normal $\vec{h} := (\vec{v}_o + \vec{r})/||\vec{v}_o + \vec{r}||$. If $\vec{v}_o$ is the negative z-axis in viewing coordinates, then the 2D texture coordinates are simply the x and y components of the normalized halfway vector $\vec{h}$. For environment mapping on a per-vertex basis and a reference viewing direction v_o identical to the negative z-axis in eye space, these texture coordinates are automatically computed by the texture coordinate generation mechanism of OpenGL.

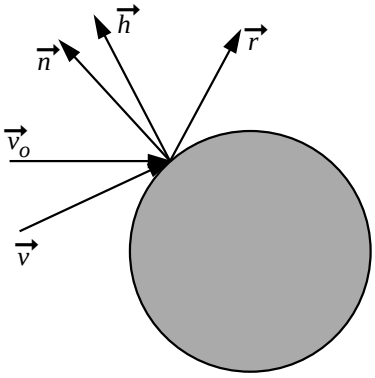

Figure 2: *The lookup process in a spherical environment map.*

The sampling rate of spherical maps reaches its maximum for directions opposing the viewing direction (that is, objects behind the viewer), and goes towards zero for directions close to the viewing direction, because these correspond to the tangential areas of the virtual metal ball used to generate the map. Because of this singularity in the viewing direction, it is clear that this parameterization is not suitable for viewing directions other than the original one, especially since the automatic texture coordinate mode does not support this case. Thus, maps using this parameterization have to be regenerated for each change of the view point, even if the environment is otherwise static. The creation of a spherical map requires a texture mapping step in which perspective images are warped into the spherical form.

Despite these disadvantages, the spherical map is very useful if the only interaction with the scene is rotating an object in front of the screen. This is the case, for example, in design and CAD applications.

Cube Maps. The second parameterization, *cubical environment maps* or *cube maps*[9, 10] consist of six independent perspective images from the center of a cube through each of its faces. From this description it is clear that the generation of such a map simply consists of rendering the six perspective images. A warping step as required for spherical maps is not necessary. The sampling of these maps is fairly good. It can be shown that the sampling rates for all directions differ by a factor of $3\sqrt{3} \approx 5.2$.

The calculation of the texture coordinates proceeds as follows:

- compute eye-space reflection vector $\vec{r}_e$.
- transform $\vec{r}_e$ to object space, yielding $\vec{r}_o$ (cube faces are aligned with main axes in object space).
- the component of $\vec{r}_o$ with the largest absolute value and the sign of this component determine the cube face. The other two components divided by the largest one are the texture coordinates. For example, if $y = -.7$ is the largest absolute value, then the cube face at $y = -1$ is used, and $s := x/y$ and $t := z/y$ are the texture coordinates within that face.

Obviously, this parameterization is suitable for arbitrary viewing directions, and several current PC graphics boards support it via a specific OpenGL extension. One problem here is the use of six independent textures, which requires some special mechanisms in the texture specification. Also, the separation into six textures may produce seams between the cube faces. In particular, this is the case if mip-mapping is used, because then each face is downsampled individually. It would be possible to overcome these problems by adding a border of several pixels to each of the faces, and to replicate some information from neighboring faces there.

Parabolic Maps. Finally, *Parabolic maps*[11, 12], sometimes also called *dual paraboloid maps*, are based on an analogy similar to the one used to describe spherical environment maps. Assume that the reflecting object lies at the origin, and that the viewing direction is along the negative z axis. The image seen by an orthographic camera when looking at a metallic, reflecting paraboloid contains the information about the hemisphere facing towards the viewer. The complete environment is stored in two separate textures, each containing the information of one hemisphere. The geometry is depicted in Figure 3.

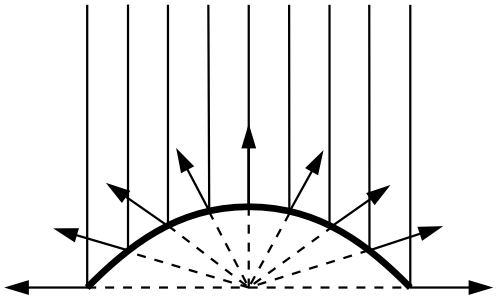

Figure 3: *The rays of an orthographic camera reflected off a paraboloid sample a complete hemisphere of directions.*

This parameterization has also been introduced by Nayar[13] in a different context. He actually built a lens and camera system that is capable of capturing this sort of image from the real world. Besides ray-tracing and warping of cubical environment maps, this is actually one way of acquiring maps in the proposed format. Since two of these cameras can be attached back to back, it is possible to create full 360° images of real world scenes.

The geometry described above has some interesting properties. Firstly, the reflected rays in each point of the paraboloid all originate from a single point, the focal point of the paraboloid, which is also the origin of the coordinate system (see dashed lines in Figure 3). This means that the resulting image can indeed be used as an environ-

ment map for an object in the origin. Spherical environment maps do not have this property; the metal spheres used there have to be assumed small.
Secondly, the sampling rate of a parabolic map varies by a factor of 4 over the complete image (see Heidrich[14] for a proof). Pixels in the outer regions of the map cover only $1/4$ of the solid angle covered by center pixels. This means that directions perpendicular to the viewing direction are sampled at a higher rate than directions parallel to the viewing direction. Depending on how we select mip-map levels, the factor of 4 in the sampling rate corresponds to one or two levels difference, which is quite acceptable. In particular this is somewhat better than the sampling of cubical environment maps.
It also turns out[11, 12] that the texture coordinates can be computed by calculating the reflection vector in eye space, transforming it back into object space, adding it to the (constant) vector $(0,0,1)$, and finally dividing by the z component of the resulting vector. This is equivalent to a projective transformation of the reflected viewing vector, and can be performed easily in graphics hardware by using the texture matrix stack.

Anti-aliasing is of particular importance for environment maps, since, depending on the surface geometry, reflections can occur both magnified and minified, and therefore have a large range of possible scales.

In hardware, anti-aliasing of textures is done with mip-mapping[15], or, more recently, with anisotropic filtering methods like footprint assembly[16]. If available with the specific hardware in use, anisotropic filtering is highly recommendable, because most of the time some parts of objects will be seen at grazing angles.

Both for mip-mapping and footprint assembly, it is necessary to compute a hierarchy of texture maps at different resolutions. For normal texture mapping, these are most often generated by simply averaging a 2×2 block of pixels to obtain one pixel value for the next level. For environment maps, however, this is not the right way. Rather than that, each pixel should be weighted by the solid angle it covers to account for the non-uniformity of the used environment map parameterization.

Furthermore, in order to avoid seams for representations that use multiple 2D textures for one environment map, there should be a border that is several pixels wide for each of the textures. This border should replicate information from the other textures in the environment map, so that the mip-mapping uses cross-texture information.

2.2. Complex Reflection Models and Environment Map Prefiltering

Once an environment map is available, it can be used to add a mirror reflection term to an object. Using multi-pass rendering and alpha blending, this mirror reflection term can be added to local illumination terms that are generated using hardware lighting. In order to incorporate global illumination with other reflection models than a perfect metallic mirror, we need to perform some precomputations, since real-time calculations are typically not possible due to the high computational cost. The two fundamental techniques for using environment maps with more general reflection models are

- Decomposition. The reflection model is decomposed into simpler contributions, which can be treated separately. For example, a reflection model may be separated into a diffuse and a specular term, where the specular term is additionally multiplied with an angular dependent term (Fresnel term).
- Prefiltering. For certain reflection models, the reflection of an environment map can be analytically precomputed and stored into a new map. The latter is called *prefiltered environment map* or *reflection map*.

In the following we will describe these two techniques and demonstrate some applications for them.

2.2.1. Decomposition

As stated above, decomposition of a reflection model means separating its terms into simpler expressions that can be handled individually. The most fundamental example is a separation into diffuse and specular contributions. We will show in below that the diffuse term as well as certain specular terms can be treated with prefiltering. Another term could be if we had a reflection model with a term for retro-reflection (light that is reflected back into the direction of incoming light). Also a very interesting example is the factorization of the specular component into a standard environment map and an angular dependent term (Fresnel term), as described in the following.

Generalized Mirror Reflections using a Fresnel Term

The Fresnel term is a physical term describing the reflectivity of a material depending on its optical density n ("index of refraction") and the angle of incoming light. It is given as

$$F = \frac{(g-c)^2}{2(g+c)^2}\left[1+\frac{(c(g+c)-1)^2}{(c(g-c)+1)^2}\right], \qquad (1)$$

with $c = <\vec{n},\vec{v}>$ and $g^2 = n^2 + c^2 - 1$.
A regular environment map without prefiltering describes the incoming illumination at a point in space. If this information is directly used as the outgoing illumination, as is described above, and as it is state of the art for interactive applications, only metallic surfaces can be modeled. This is because for metallic surfaces (surfaces with a high optical density) the Fresnel term is almost constant one, independent of the angle between light direction and surface normal. Thus, for a perfectly smooth (i.e. mirroring)

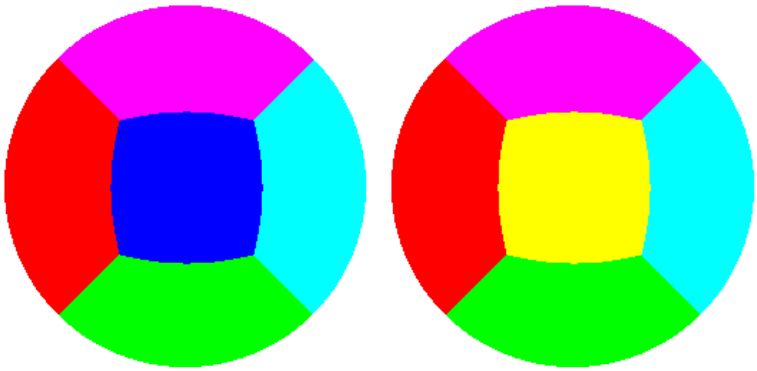

Figure 4: *The two textures comprising an environment map for an object in the center of a colored cube.*

surface, incoming light is reflected in the mirror direction with a constant reflectance.

For non-metallic materials (materials with a small optical density), however, the reflectance strongly depends on the angle of the incoming light. Mirror reflections on these materials should be weighted by the Fresnel term for the angle between the normal and the reflected viewing direction $\vec{r}_v$, which is, of course, the same as the angle between normal and viewing direction $\vec{v}$.

For any given material, the Fresnel term $F(\cos\theta)$ for the mirror direction $\vec{r}_v$ can be stored in a 1-dimensional texture map, and rendered to the framebuffer's alpha channel in a separate rendering pass. The mirror part is then multiplied with this Fresnel term in a second pass, and a third pass is used to add the diffuse part. If we have a reflection model consisting of a mirror component L_m and a diffuse component L_d, this yields an outgoing radiance of $L_o = F \cdot L_m + L_d$.

In addition to simply adding the diffuse part to the Fresnel-weighted mirror reflection, we can also use the Fresnel term for blending between diffuse and specular: $L_o = F \cdot L_m + (1-F)L_d$. This allows us to simulate diffuse surfaces with a transparent coating: the mirror term describes the reflection off the coating. Only light not reflected by the coating hits the underlying surface and is there reflected diffusely.

Figure 5 shows images generated using these two approaches. In the top row, the Fresnel-weighted mirror term is shown for indices of refraction of 1.5, 5, and 200. In the center row, a diffuse term is added, and in the bottom row, mirror and diffuse terms are blended using the Fresnel term. Note that for low indices of refraction, the object is only specular for grazing viewing angles, while for a high indices of refraction we get the original metal-like reflection.

2.2.2. Prefiltered Environment Maps

Generally speaking, prefiltered environment maps capture all the reflected exitant radiance towards all directions $\vec{v}$ from a fixed position $\mathbf{x}$:

$$L_o(\mathbf{x};\vec{v},\vec{n},\vec{t}) = \int_\Omega f_r(\vec{w}(\vec{v},\vec{n},\vec{t}),\vec{w}(\vec{l},\vec{n},\vec{t}))L_i(\mathbf{x};\vec{l}) < \vec{n},\vec{l} > d\vec{l}, \quad (2)$$

where $\vec{v}$ is the viewing direction and $\vec{l}$ is the light direction in world-space, $\{\vec{n},\vec{t},\vec{n}\times\vec{t}\}$ is the local coordinate frame of the reflective surface, $\vec{w}(\vec{v},\vec{n},\vec{t})$ represents the viewing direction and $\vec{w}(\vec{l},\vec{n},\vec{t})$ the light direction relative to that frame, f_r is the BRDF, which is usually parameterized via a local viewing and light direction.

A prefiltered environment map stores the radiance of light reflected towards the viewing direction $\vec{v}$, which is computed by weighting the incoming light L_i from all directions $\vec{l}$ with the BRDF f_r. Note, that L_i is stored in the unfiltered original environment map. As you can see, in the general case we have a dependence on the viewing direction as well as on the orientation of the reflective surface, i.e. the local coordinate frame $\{\vec{n},\vec{t},\vec{n}\times\vec{t}\}$.

This general kind of environment map is five-dimensional. Two dimensions are needed to represent the viewing direction $\vec{v}$ (a unit vector in world coordinates) and three dimensions are necessary to represent the coordinate frame $\{\vec{n},\vec{t},\vec{n}\times\vec{t}\}$; e.g. three angles can be used to specify the orientation of an arbitrary coordinate frame.

Of course, five-dimensional textures have enormous memory requirements, which is why the prefiltered environment maps which we will examine drop some dependencies (e.g. the tangent $\vec{t}$) and are often reparameterized (e.g. indexing is not done with the viewing direction $\vec{v}$, but the reflected viewing direction). Because this reduction in dimensionality also removes some to the generality of the approach, the decomposition method is often required to combine several of these simplified models.

If the original environment map is given in a high-dynamic range format[17], then the prefiltering technique allows for effects similar to the ones described by Debevec[18].

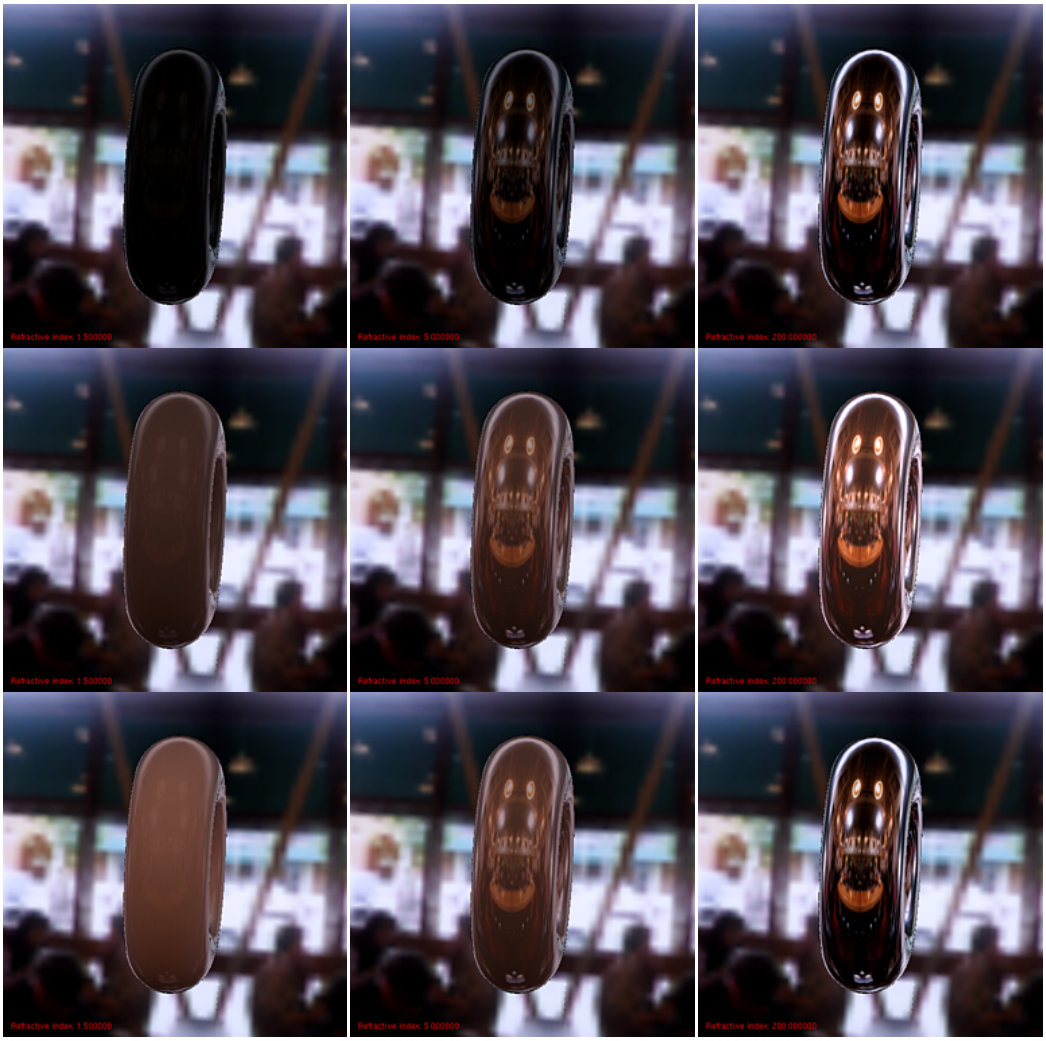

Figure 5: *Top row: Fresnel weighted mirror term. Center row: Fresnel weighted mirror term plus diffuse illumination. Bottom row: Fresnel blending between mirror and diffuse term. The indices of refraction are (from left to right) 1.5, 5, and 200.*

Diffusely Prefiltered Maps As we have seen, we can combine a mirror reflection term using an environment map with local illumination terms that are generated using hardware lighting. It is also possible to add a diffuse global illumination term through the use of a precomputed texture. For the generation of such a texture, there are two methods. In the first approach, a global illumination algorithm such as Radiosity is used to compute the diffuse global illumination of every surface point.

The second approach is purely image-based, and uses a prefiltered environment map[19, 9]. The environment map used for the mirror term contains information about the incoming radiance $L_i(\mathbf{x}, \vec{l})$, where $\mathbf{x}$ is the point for which the environment map is valid, and $\vec{l}$ the direction of the incoming light. The outgoing radiance for a diffuse BRDF is then:

$$L_o(\mathbf{x}, \vec{n}) = k_d \cdot \int_{\Omega(\vec{n})} L_i(\mathbf{x}, \vec{l}) \cdot \cos(\vec{n}, \vec{l}) d\omega(\vec{l}). \qquad (3)$$

Due to the constant BRDF of diffuse surfaces, L_o is only a function of the surface normal $\vec{n}$ and the illumination L_i stored in the environment map, but not of the outgoing direction $\vec{v}$. Thus, it is possible to precompute a map containing the diffuse illumination for all possible surface normals. For this map, like for the mirror map, any parameterization from Section 2.1 can be used. The only difference is that diffusely prefiltered maps are always referenced via the normal of a vertex in environment map space, instead of via the reflection vector. Figure 6 shows

such a prefiltered map, a torus with diffuse illumination only as well as a torus with diffuse and mirror illumination.

Glossy Prefiltering of Environment Maps A simplification similar to the one used for diffuse materials is also possible for certain specular reflection models[12, 19], most notably the Phong model. Voorhies et al.[10] used a similar approach to implement Phong shading for directional light sources.

As shown by Lewis[20], the Phong BRDF is given by

$$f_r(\mathbf{x}, \vec{l} \rightarrow \vec{v}) = k_s \cdot \frac{<\vec{r}_l, \vec{v}>^{1/r}}{\cos\alpha} = k_s \cdot \frac{<\vec{r}_v, \vec{l}>^{1/r}}{\cos\alpha}, \quad (4)$$

where $\vec{r}_l$, and $\vec{r}_v$ are the reflected light- and viewing directions, respectively, and $\cos\alpha = <\vec{n}, \vec{l}>$. Thus, the specular global illumination using the Phong model is

$$L_o(\mathbf{x}, \vec{r}_v) = k_s \cdot \int_{\Omega(\vec{n})} <\vec{r}_v, \vec{l}>^{1/r} L_i(\mathbf{x}, \vec{l})\, d\omega(\vec{l}), \quad (5)$$

for some roughness value r. This is only a function of the reflection vector $\vec{r}_v$ and the environment map containing the incoming radiance $L_i(\mathbf{x}, \vec{l})$. As for diffuse illumination, it is therefore possible to take a map containing $L_i(\mathbf{x}, \vec{l})$, and generate a filtered map containing the outgoing radiance $L_o(\mathbf{x}, \vec{r}_v)$ for a glossy Phong material.

Figure 7 shows such a map, as well as a glossy sphere and torus textured with this map.

A Fresnel weighting of these prefiltered environment maps similar to the way it is described above is only possible with approximations. The exact Fresnel term for the glossy reflection cannot be used, since this term would have to appear inside the integral of Equation 5. However, for glossy surfaces with a low roughness, the Fresnel term can be assumed constant over the whole specular peak (which is very narrow in this case). Then the Fresnel term can be moved out of the integral, and the same technique as for mirror reflections applies.

The use of a Phong model for the prefiltering is somewhat unsatisfactory, since this is not a physically valid model. However, this method works for all reflection models having lobes that are rotationally symmetric about the reflected viewing direction, and whose shape does not depend on the angle to the surface normal.

Approximations of General Isotropic BRDFs Based on this concept, Kautz and McCool[21] extended the Phong environment maps idea to other isotropic BRDFs by approximating them with a special class of BRDFs:

$$f_r(\vec{v}, \vec{l}) \;:=\; p(<\vec{n}, \vec{r}_v(\vec{n})>, <\vec{r}_v(\vec{n}), \vec{l}>),$$

where p is an approximation to a given isotropic BRDF, which is not only isotropic, but also radially symmetric about $\vec{r}_v(\vec{n}) = 2(\vec{n}\cdot\vec{v})\vec{n} - \vec{v}$, and therefore only depends on two parameters.

Now consider Equation 2 using this reflectance function:

$$L_o(\mathbf{x}; \vec{v}, \vec{n}, \vec{t}) = \int_{\Omega(\vec{n})} p(<\vec{n}, \vec{r}_v(\vec{n})>, <\vec{r}_v(\vec{n}), \vec{l}>) \cdot L_i(\mathbf{x}; \vec{l}) <\vec{n}, \vec{l}>\, d\omega(\vec{l}). \quad (6)$$

The authors then make the assumption that the used BRDF is fairly specular, i.e. the BRDF close to zero almost everywhere, except for $\vec{r}_v(\vec{n}) \approx \vec{l}$. Using this assumption they reason that $<\vec{n}, \vec{r}_v(\vec{n})> \approx <\vec{n}, \vec{l}>$. Now the equation can be reparameterized and rewritten the following way:

$$L_o(\mathbf{x}; \vec{r}_v, <\vec{n}, \vec{r}_v>) = <\vec{n}, \vec{r}_v> \int_{\Omega(\vec{n})} p(<\vec{n}, \vec{r}_v>, <\vec{r}_v, \vec{l}>) \cdot L_i(\mathbf{x}; \vec{l})\, d\omega(\vec{l}), \quad (7)$$

which is three dimensional. The third dimension is used to vary the diameter of the lobe with the angle between reflection vector and surface normal. This way, it is possible to have materials that are almost mirror-like at grazing viewing angles, while they are matte if looked at perpendicularly. This is a behavior that can be seen quite often with real materials.

In addition to this, Kautz and McCool also proposed an approximation technique that generates a BRDF with rotationally symmetric lobes from an arbitrary BRDF. This is done by averaging the lobes for different viewing directions.

This technique has the advantage that it can use approximations of arbitrary isotropic BRDFs and achieves interactive frame rates. Off-specular peaks can also be incorporated into this technique. An additional Fresnel factor like Miller[19] and Heidrich[12] proposed is not needed because it can be incorporated into the dependency on the viewing angle, i.e. the third dimension of the map. On the down side, 3D textures are quite space consuming and are not supported by most current low-end hardware.

Depending on the BRDF, the quality of the approximation varies. For higher quality approximations Kautz and McCool also propose to use a multilobe approximation, which basically results in several prefiltered environment maps which have to be added.

For instance, if a BRDF is to be used, which is based on several separate surface phenomena (e.g. has retro-reflections, diffuse reflections, and glossy reflections) each part has to be approximated separately, since no radially symmetric approximation can be found for the whole BRDF. This again means a decomposition of the reflection model into several parts.

Warping for Environment Maps with Isotropic BRDFs A different technique which makes similar assumptions (isotropic and radially symmetric BRDF) was presented by Cabral et al.[22]. They prefilter an environment map

Figure 6: *Left: diffusely prefiltered environment map of the cafe scene. Center: diffusely illuminated torus. Right: same torus illuminated with both a diffuse and a mirror term.*

Figure 7: *A prefiltered version of the map with a roughness of 0.01, and application of this map to a reflective sphere and torus.*

for different fixed viewing directions, resulting in view-dependent, spherical environment maps. An alternative to the prefiltering process is to take photographs from different viewing directions of a sphere made of the same material one would like to represent.

In contrast to the previous approach, this is actually a four-dimensional environment map

$$L_o(\mathbf{x};\vec{v},\vec{n}) = \int_{\Omega(\vec{n})} p(<\vec{n},\vec{r}_v>,<\vec{r}_v,\vec{l}>)\cdot L_i(\mathbf{x};\vec{l})<\vec{n},\vec{l}>\, d\omega(\vec{l}), \tag{8}$$

but the two dimensions representing the viewing direction $\vec{v}$ are only sampled very coarsely. A different two dimensional spherical map is extracted from this four-dimensional map for every new viewpoint. This map corresponds to one specific viewing direction and is generated using warping. The new view-dependent environment map is then applied to an object. The warping compensates for the undersampled viewing directions, and minimizes the visible artifacts. Although the warping requires high-end graphics hardware to achieve interactive frame rates, the final rendering can be done with standard sphere mapping, which is major the reason for generating the intermediate spherical map.

Warping is done based on an assumption what the central reflection direction of the BRDF is (the reflected viewing direction and the surface normal are mentioned as examples in the original paper[22]). For example, if a specular highlight is assumed, then the warping is performed such that the location of the highlights are located in the same position after warping to the destination direction. The assumption of a single, predominant reflection direction fails for BRDFs that have off-specular reflections like strong diffuse components or retro-reflection. Similarly, since radially symmetric BRDFs are used, this method has the same difficulties with complex BRDFs as the previous method. To overcome these problems, the method can be combined with a decomposition approach.

As mentioned before the generated two dimensional environment map is view-dependent, so the reflective object needs to be viewed with an orthographic projection or otherwise the reflections are incorrect, since the reflection directions are computed based on an infinite viewer. For

example, if the material contains a strongly varying Fresnel term, it cannot be represented in this form, because the spherical map does not depend on the angle between normal and light direction.

Hardware Accelerated Prefiltering For interactive applications it would be nice if environment map prefiltering could be done on the fly. This means that if the scene changes, glossy reflections change accordingly. Here, we will describe a method to perform hardware-accelerated Phong filtering[23] of a given environment map.

In a prefiltered environment map, every texel is a weighted sum of all pixels in a source environment map. This means, we can think of the filtering process as applying a (BRDF-dependent) filter kernel to some unfiltered source map. We would like to map this filtering operation to the operations provided by a graphics hardware pipeline. The OpenGL imaging subset only supports shift-invariant two dimensional filters of certain sizes, and we would like to use this feature to perform the filtering. Hence, for hardware accelerated prefiltering we have to choose an environment map technique that uses only two dimensional environment maps with a BRDF which results in a shift-invariant filter over the hemisphere, and an environment map representation that keeps the filter shift-invariant.

The Phong model has a shift-invariant filter kernel over the hemisphere, since its cosine lobe is constant for all reflected viewing directions $\vec{r}_v$. It is also radially symmetric about $\vec{r}_v$. The filter size can also be decreased if smaller values are clamped to zero (this will be necessary due to the restricted filter size of the graphics hardware). The filter shape is obviously circular, since it is radially symmetric. Therefore Phong environment maps fulfill the necessary requirements for hardware accelerated prefiltering. We still need to find an environment map representation that maps the shift-invariant circular filter kernel on the hemisphere to a shift-invariant circular filter kernel in texture space. It turns out[23] that the parabolic maps come close to this desired property. A circular filter kernel which is mapped from the parabolic environment map back to the hemisphere is also (almost) circular.

Unfortunately a shift-invariant filter kernel on the sphere does still not completely map to a shift-invariant filter in the parabolic space. Besides the slight distortion, the size of the filter kernel varies with the distance d to the center of the map. The ratio between the smallest filter radius and largest filter radius is $1:2$, since the ratio for the areas is 1:4, as shown above. To adjust for this, we generate two prefiltered environment maps, one with the smallest and one with the largest necessary filter size. Then we blend between both prefiltered environments. The value with which we need to blend between both maps is different for different pixels in the parabolic environment map, but it depends only on the distance of the pixel from the center of the map.

Using the same arguments as above, we can not only use Phong materials for this hardware prefiltering, but any BRDF with radially symmetric lobes.

The problem of the environment map approach is that all objects are assumed infinitely small and distant to the environment. This means that the parallax that can be observed for large objects, and especially large planar reflectors, cannot be correctly simulated with environment maps. Nonetheless, environment maps as well as decomposition and prefiltering are the most commonly used approach for rendering non-diffuse global effects in highly interactive applications today. This includes both games and a variety of virtual reality systems.

3. Light Fields and Lumigraphs

In order to correctly account for the parallax effects that the environment map approach cannot handle, we require one environment map for each point in space (or at the very least for each point on the surface of the reflective object).

This is the fundamental idea of light fields. A light field[24] is a 5-dimensional function describing the radiance at every point in space in each direction. It is closely related to the *plenoptic function* introduced by Adelson[25], which in addition to location and orientation also describes the wavelength dependency of light.

In the case of a scene that is only to be viewed from outside a convex hull, it is sufficient to know what radiance leaves each point on the surface of this convex hull in any given direction. Since the space outside the convex is assumed to be empty, and radiance does not change along a ray in empty space, the dimensionality of the light field can be reduced by one, if an appropriate parameterization is found. The so-called two-plane parameterization fulfills this requirement. It represents a ray via its intersection points with two parallel planes. Several of these pairs of planes (also called *slabs*) are required to represent a complete hull of the object. Since each of these points is characterized by two parameters in the plane, this results in a 4-dimensional function that can be densely sampled through a regular grid on each plane (see Figure 8).

One useful property of the two-plane parameterization is that all the rays passing through a single point on the (s,t)-plane form a perspective image of the scene, with the (s,t) point being the center of projection. Thus, a light field can be considered a 2-dimensional array of perspective projections with eye points regularly spaced on the (s,t)-plane. Other properties of this parameterization have been discussed in detail by Gu et al.[26].

Since we assume that the sampling is dense, the radiance along an arbitrary ray passing through the two planes can be interpolated from the known radiance values in nearby grid points. Each such ray passes through one of the grid cells on the (s,t)-plane and one on the (u,v)-plane. These

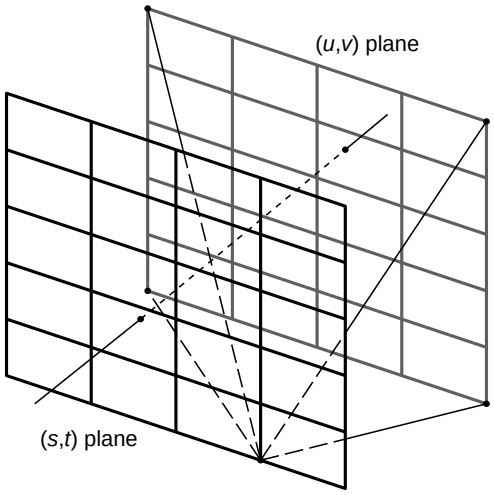

Figure 8: *A light field is a 2-dimensional array of images taken from a regular grid of eye points on the (s,t)-plane through a window on the (u,v)-plane. The two planes are parallel, and the window is the same for all eye points.*

are bounded by four grid points on the respective plane, and the radiance from any of the (u,v)-points to any of the (s,t)-points is stored in the data structure. This makes for a total of 16 radiance values, from which the radiance along the ray can be interpolated quadri-linearly. As shown in by Gortler et al[27] and Sloan et al.[28], this algorithm can be considerably sped up by the use of texture mapping hardware.

Other parameterizations for the light field have been proposed by several authors[29, 30] in order to achieve a better sampling uniformity. In practice, however, these are not of great importance since the reconstruction time for the radiance along any given ray can no longer be done in constant time as with the regular grid in the two-plane parameterization.

3.1. Lumigraphs: Light Fields Plus Geometry

The quadri-linear interpolation in the light field data works well as long as the resolution of the light field is high. For low resolutions, the interpolation only yields a sharp image for objects in the (u,v)-plane. The further away points are from this plane, the more blurred they appear in the interpolated image.

The Lumigraph [27] extends the concept of a light field by adding some geometric information that helps compensating for this problem. A coarse polygon mesh is stored together with the images. The mesh is used to first find the approximate depth of the object along the ray to be reconstructed, and then this depth is used to correct the weights for the interpolation. This will reduce the ghosting artifacts for the geometry itself. However, in the case of very shiny materials, the reflections will still exhibit some amount of ghosting, since the depth correction is performed for the geometry and not for the apparent depth of the object seen in the reflection. This is a problem that none of the light field or Lumigraph methods can solve at the moment.

Heidrich et al.[31] take a similar, but purely sampling-based approach. Instead of a polygon mesh, the depth of each pixel in the light field is stored. This information is then used to refine the light field with warped images until the rendering quality is satisfactory. This decouples the more expensive depth correction from the efficient quadri-linear interpolation, and thus can be used to achieve higher frame rates. This basic idea of a sampling-based representation for the geometry has since been used for adaptive acquisition of light fields[32] and a high-quality, warping-based reconstruction instead of the quadri-linear interpolation[33].

3.2. Surface Light Fields

If we take the concept of combining geometry and light field data to its extremes, we arrive at the original geometric model of the object. Instead of parameterizing the ray space using intersections with virtual geometry (such as a light slab in the case of the two-plane parameterization), we can then reparameterize the light field to use the 2D surface position on the original geometry as two of the four parameter values. The remaining two directions would then parameterize the hemisphere of directions over the tangent plane of that surface point. This concept is called a *surface light field*[34].

In other words, a surface light field is a 4D data structure which describes for every point on a surface parameterized over a 2D parameter domain u,v, which radiance leaves that surface point for all possible directions (parameterized over s,t). In order to render an image of the object, we then have to first compute the intersection of the ray with the original geometry, and then reconstruct the radiance for that intersection point and viewing direction by quadri-linear interpolation. Miller et al.[34] describe both a specific parameterization for the directions, as well as a way of partially exploiting the graphics hardware for doing the interpolation.

While Miller et al. use synthetically generated surface light fields for parametric objects, Wood et al.[35] describe in a recent paper how to acquire surface light fields from real world objects. They generate a set of so-called *lumispheres* for a dense set of surface points by using techniques from mesh simplification and surface fairing. They also describe how to compress the resulting data set, which will be described in more detail below. In order to arrive at a parameterization for the geometry, which may be a polygonal model of arbitrary topology, Wood et al. use the MAPS algorithm by Lee et al.[36].

The obvious advantage of the surface light field approach is that images rendered in this fashion will always show

sharp geometry, although reflections may still exhibit ghosting as described above. The downside of the surface light field approach is that the rendering time is no longer independent of geometric complexity, which is generally considered one of the most interesting features of image-based rendering in general.

3.3. Illumination From Light Fields

Instead of using light fields or Lumigraphs directly for viewing, it is also possible to use them for illumination purposes only. An example of such an algorithm is the *canned light source* approach[37], where complex luminaries with strongly varying spatial and directional light distributions are stored in a light field data structure. Such a canned light can then later be used for illuminating arbitrary objects, for example in a ray-tracing step, but also in hardware-accelerated rendering.

Another example is the rendering of refractive objects using light field representationd for storing geometric information[38]. In this work, a two-plane parameterized light field of a refractive object stores the refracted ray leaving the object for every incident viewing ray. In this way, the geometry-dependent visibility information is precomputed and stored in a light field data structure. At rendering time, the object is displayed using the normal light field rendering algorithm, yielding an image representing the refracted ray for every pixel. For each of these pixels, the illumination along that ray is then looked up either from an environment map or from another light field. In this way, the geometry is decoupled from the illumination, and both can be exchanged independently.

3.4. Compression of Light Fields

Since the size of light field data sets can easily exceed several Gigabytes, several researchers have worked on compression schemes to make light fields more practical. For an efficient rendering it is usually desirable to have the possibility for a random access, constant time reconstruction of the radiance along a given ray. This is the case for the two-plane parameterized light field, but difficult to achieve in combination with a good compression scheme.

The first compression scheme for light fields was vector quantization, and was presented in the original paper by Levoy and Hanrahan[24]. Later, a hardware-accelerated form of reconstruction for vector-quantized light fields was presented[38]. In the compression scheme proposed by Levoy and Hanrahan, blocks of adjacent light field samples (e.g. blocks of 4^4 pixels) are concatenated to form vectors. Each block is then replaced by a 16-24 bit index into a vector table describing the radiance along all rays between points in the vector. This gives moderate compression ratios of about 24:1, depending on the exact size of the table and the blocks, but constant reconstruction time is maintained.

Wood et al.[35] have extended the vector quantization approach to surface light fields. Here, the reconstructed lumispheres (see Section 3.2) are quantized after some transformations that try to increase the similarity between the different spheres. Since the lumispheres are actually continuous functions rather than vectors, Woo et al. call their approach *function quantization* rather than vector quantization. Similarly, they propose a method they call *principal function analysis*, which is a generalization of principal component analysis.

Better compression ratios can be achieved by block based coding with motion prediction similar to the MPEG and H263 methods for video compression. Such methods have been introduced by Miller et al.[34] for surface light fields, and by Magnor and Girod[39] for two-plane parameterized light fields. This maintains a constant reconstruction time, but since a whole block is decompressed at once, a sequential reconstruction is faster than random access.

Finally, Lalonde and Fournier[40] have worked on wavelet compression for light fields. The compression ratios reported for their scheme are interesting, but reconstruction time is logarithmic rather than constant, and in absolute rendering times the reconstruction takes significantly longer than the other methods.

4. Light Field-Like Representations

The illumination representations used by the methods described in this section can be interpreted as light fields although the authors of the work have not originally described their algorithms in this fashion. In contrast to the techniques from Section 3, the methods described here use a sparser, less regular representation of the light field information. This results in significantly reduced storage costs, but also in a more expensive reconstruction step. In addition, the sparse sampling of the light field typically means that less spatial or directional detail can be preserved.

4.1. View-Dependent Vertex Colors

A simple method for interactive viewing of precomputed global illumination solutions is to extract the view-dependent color for each vertex of a polygonal model from the illumination solution, and then use graphics hardware to render the model with Gouraud shading. In order to facilitate the view-dependent per-vertex computations, the illumination should be stored in a fashion that makes this operation as efficient as possible.

Stamminger et al.[41] propose the *illumination sample* method for generating the global illumination solution in the first place. This algorithm is a radiance clustering method that efficiently represents incoming illumination at patches. After the solution has been computed, this representation is transformed into a directional representation of the illumination at the vertices of the polygonal model. A Haar wavelet

basis was chosen by Stamminger et al. for storing the directional information. This approach can be interpreted as a surface light field where the spatial samples are located at vertices only, and the directional samples are projected in a wavelet basis.

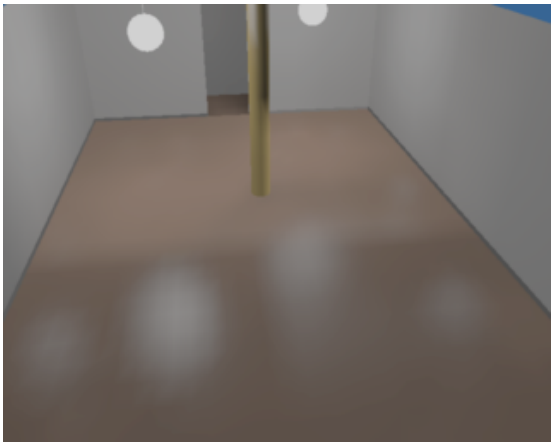

Figure 9: *An example of the algorithm by Stamminger et al.*[41]. *Image courtesy of Marc Stamminger.*

At rendering time, the color at each vertex has to be determined for the current viewing direction by reconstructing the outgoing radiance towards the viewer from the wavelet representation. Since this can be quite costly to do for every vertex, a threshold for the change in viewing direction is introduced, below which the color from the previous frame is reused. Since the lighting is per-vertex, the spatial resolution of the reconstructed illumination is typically fairly low, especially compared to the light field methods from Section 3.

4.2. Interactive Display of Photon Maps

A scattered representation of the light arriving at a surface has been proposed by Shirley et al.[42]. Photons are traced from the light sources, and reflected or refracted via specular surfaces. The density of the photon hits is later used to estimate the local radiance distribution at a surface point (density estimation). Since the photons are traced stochastically, both the points where they hit an object, and their incoming direction direction at that point is random. This corresponds to a scattered data representation of the light field arriving at the surfaces in the scene.

Stürzlinger and Bastos[43] proposed an interactive viewing algorithm for displaying the results of this simulation, which we will describe in the following. Like Shirley et al.[42], Stürzlinger and Bastos use splatting of the individual photons for the density estimation step. To this end, they choose an object-space filter kernel, which they store in a 2-dimensional texture map. Then they render each individual photon as an object-space triangle with that texture applied. In order to account for view-dependent effects, the splat for each photon has to be weighted by the BRDF value for the given viewing direction and the direction of the incoming photon.

To improve the visual quality of the rendering method, Stürzlinger and Bastos[43] extend this basic algorithm such that direct illumination is performed by hardware lighting combined with shadow maps, and the photons are only used for rendering the indirect light contributions. This method produced near-interactive frame rates for moderately complex scenes on high-end graphics hardware.

4.3. Virtual Lights

Another branch of research for interactively displaying indirect illumination is the fitting of virtual lights to previously obtained global illumination solutions. Different sets of virtual lights are computed for each of the objects in the scene or even for different parts of an object. Each set of lights is only active while the respective object or object part is rendered. These virtual lights allow for the use of graphics hardware combined with a Phong reflection model for approximating the true indirect illumination of the object. One could interpret this approach as a highly compressed representation for surface light fields, where each virtual light corresponds to a beam of incident light on the surface. Therefore, the spatial resolution of the illumination is high (the reflection can be computed at every pixel of the final image), but the directional sampling is limited by the number of light sources used.

The first work based on this general idea was presented by Stamminger et al.[44]. They obtained their global illumination solution with a Wavelet Radiance method[45], a generalization of Wavelet Radiosity[46] to non-diffuse environments. The link structure of the Wavelet Radiance solution gives an indication which objects or patches in the scene send the most energy towards any given object. The brightest of these patches are then replaced by virtual point lights whose brightness is set according to the energy exchanged between the patch and the object. An example of this method is shown in Figure 10.

The second approach, presented by Walter et al.[47] starts with a global illumination solution provided in the form of a surface light field. For a collection of points or vertices on the object, for which the outgoing radiance is known, the authors greedily fit a number of Phong lobes corresponding to the effect of directional light sources. Since Phong lobes are positive everywhere, it is not possible to subtract energy that has once been added. This complicates the fitting process.

Both the approach from Stamminger et al.[44] and Walter et al.[47] suffer from the fact that graphics hardware supports only a very small number of light sources, typically 8. This means that detailed reflection patterns cannot be represented with this method, except when the set of virtual light sources

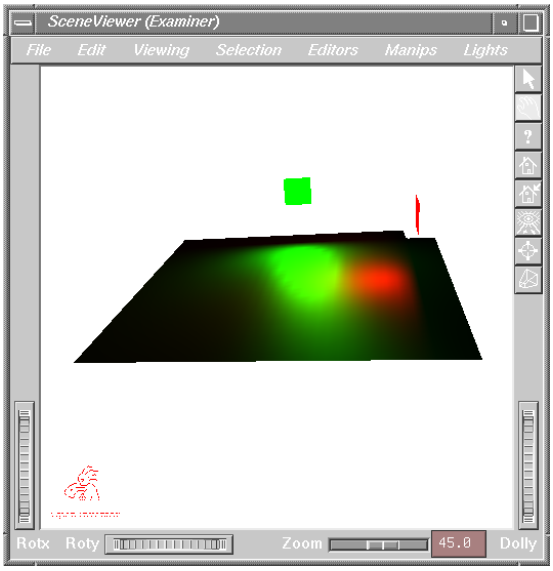

Figure 10: *An example of the virtual light fitting method by Stamminger et al.*[44]*. Image courtesy of Marc Stamminger.*

is chosen for very small geometric entities, such as polygons. In that case, however, the parameters of the light sources have to be changed frequently, which is a slow operation on graphics hardware, and is thus only feasible for small scenes.

4.4. Instant Radiosity

Instant Radiosity[48] is an approach that also uses virtual light sources, but on a global scale rather than different lights for each of the objects in the scene. As the name suggests, the basic method has been developed for diffuse scenes, but the paper also describes an extension to glossy environments.

The core of the method is to trace the path of photons emitted from a light source with a quasi-random walk. In a diffuse scene, a virtual point light is placed wherever the photon hits a surface. This light represents the light reflected from that point in all directions. In a specular scene, light should only be reflected in a specific direction. This is achieved by reflecting the origin of the ray from which the photon arrived at the surface that was hit by the photon, and placing a light source there. The effect of the light source is then clamped to the region contained in the generalized pyramid from the light position through the boundaries of the polygon.

Since virtual lights affect all objects in the scene in an Instant Radiosity implementation, shadowing effects have to be taken into account. Keller[48] solves this by implementing a shadow map approach described in[49]. To account for dynamic environments, Keller suggests to replace a subset of the photon paths for every frame. This is a relatively cheap operation since only a few rays have to be traced.

5. On-The-Fly Computation

The methods described so far have all been based on precomputed global illumination solutions. As a consequence, most of them are not well suited for dynamic environments (with the exception of the hardware-accelerated prefiltering of environment maps, Section 2.2.2, as well as Instant Radiosity, Section 4.4). In this section, we describe methods for computing certain light paths in real time. This includes both specialized methods for very specific geometry and materials, as well as general solutions based on ray-tracing.

5.1. Mirror Reflections in Planar or Slightly Curved Geometry

A commonly used technique for rendering mirror reflections on planar objects is given by Diefenbach[50, 51]: with a simple affine model/view matrix, the scene is mirrored at the planar reflector. This mirrored scene is rendered at every pixel where the reflector is visible in the current view. This is typically achieved in two rendering passes. First, the original scene is rendered, and all pixels of the planar reflector are marked in the stencil buffer. Then, the model/view matrix is modified to accommodate for the reflection. The scene is now rendered again, but only pixels marked in the stencil buffer are set. If the stencil buffer has more than one bit, it is also possible to realize multiple reflections by recursing the procedure[51].

A similar effect can be achieved using texture mapping. Instead of mirroring the scene, the eye point $\mathbf{p}$ is mirrored, yielding a reflected eye point $\mathbf{p}'$, as depicted on the left side of Figure 11. Rendering the scene from this eye point with the reflector as an image plane yields the texture image to be applied to the reflector as seen from the eye. Note that this approach has two major disadvantages relative to the one from Diefenbach. Firstly, the rendered image from the first pass needs to be transferred from the framebuffer into texture memory, which requires additional bandwidth, and secondly, the texturing represents a resampling step that results in reduced image quality.

5.1.1. Glossy Reflection for Planar Reflectors

For these reasons, the modified algorithm is an inferior choice for implementing reflections on planar surfaces, but it is useful for approximating glossy reflections. Bastos et al. [52, 53] propose to convolve the texture (which they generate with an image-based warping step rather than a geometry-based rendering) with a space-invariant filter kernel corresponding to the BRDF.

This is similar in spirit to the prefiltering approach described in Section 2.2.2, but since the filter is space-invariant, a orthographic viewer is implicitly assumed for the prefiltering. This will produce artifacts for large reflectors or wide angle cameras. In addition to prefiltering, Bastos et al. [52, 53] also use decomposition, as explained in Section 2.2.1, since they also separate the Fresnel term from the

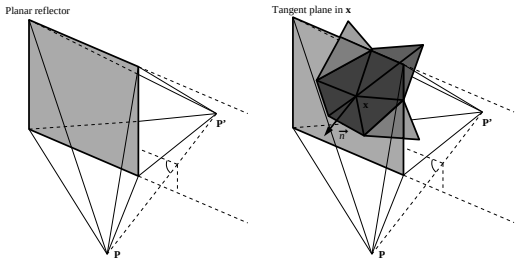

Figure 11: *Multi-pass mirror reflections in planar and curved objects. While a single reflected eye point* $\mathbf{p}'$ *exists for planar reflectors, curved reflectors do not have such a uniquely defined point. If a curved object is approximated by a triangle mesh with per-vertex normals, one reflected point can be defined for each vertex in the mesh using the tangent plane in that vertex.*

BRDF and apply it in a separate rendering pass using texture mapping.

5.1.2. Texture-Based Rendering of Curved Mirrors

Based on the texture-based rendering approach for planar mirrors described above, we can also develop a naive (and inefficient) method for generating reflections on curved surfaces represented as triangle meshes: for curved surfaces the problem is that the reflected eye point is not constant, but varies across the surface. However, if the surface is reasonably smooth, then it suffices to compute the reflected eye point only at some discrete points on the surface, say the vertices of the triangle mesh, and to interpolate the radiance for each point inside a triangle from the textures obtained for each of the three vertices. Note that each of the textures corresponds to a dynamically generated, 2-dimensional slice of a light field describing the incoming illumination around the reflector, and the interpolation step is nothing but the reconstruction of a novel view from this light field information.

The complete algorithm would then work as follows (see right side of Figure 11): For each vertex in the triangle mesh the tangent plane is determined, the eye point is reflected in that plane, and the reflection texture for that tangent plane is rendered. Then, for each vertex, the triangle fan surrounding it is rendered with the reflection applied as a projective texture. During this rendering, the alpha value for the center vertex of the fan is set to 1, the alpha values for all other vertices are set to 0, and the result from texturing is multiplied by the alpha channel. This way, the alpha channel contains the basis functions for the Barycentric coordinates in each pixel. The final image results from adding up all the contributions from the different triangle fans in the frame buffer. This is exactly the interpolation scheme used for the hardware implementation of light fields and Lumigraphs.[27, 28]

Clearly, this approach is not feasible for real time or interactive applications, since the number of vertices (and thus the number of rendering passes) on typical reflectors are often in the order of tens of thousands. On the other hand, for a static scene the incoming light field at the object does not change. Therefore, it is not necessary to rerender the geometry multiple times for each frame in order to generate the 2D slices used as textures. Instead, a practical light field-based method could rely on some amount of precomputation to achieve interactive frame rates.

5.1.3. Geometry-Based Rendering of Curved Mirrors

A geometry-based method for reflections on curved surfaces has recently been introduced by Ofek and Rappoport[54]. For each frame, all vertices of the reflected geometry are individually transformed in software to form a virtual reflected object. To this end, it is necessary to determine the point on the reflector, on which the reflection of the vertex is visible. This is done by testing all triangles on the reflector. For every eye point, each reflector triangle with per-vertex normal defines a 3D region, called *reflected cell*, in which 3D geometry reflected via this triangle can reside. Once the triangle on which the reflection occurs has been determined, the location of the exact reflection point can be found, and finding the virtual vertex position from that point and its normal is easy. After all vertices of the surrounding environment have been mirrored this way, the resulting virtual object is simply rendered using graphics hardware.

In the case of a convex reflector, the reflected cells of the different reflector triangles do not overlap, so that each vertex of the surrounding geometry maps to exactly one vertex on the virtual object. This corresponds to the fact that only one copy of the surrounding environment can be visible in a convex reflector. Reflections in concave objects can be achieved in a similar fashion, but mixed reflectors of convex and concave regions have to be partitioned to the simples cases first. For each convex or concave region, there can be one virtual object corresponding to the complete geometry of the surrounding environment.

This approach only works at interactive performance for relatively smooth objects that are either concave or convex. Like the texturing method described above, this geometry-based approach also quickly becomes infeasible for more complex scenes.

5.2. Interactive Ray-Tracing and Ray Caching

The most serious restriction of the methods described so far is that the illumination effects that can be captured are very limited. For every new effect, a completely new, specialized algorithm has to be developed. In contrast, ray-tracing is a very general method that can simulate a wide variety of different illumination effects, especially when combined with stochastic sampling (e.g. distribution ray tracing[55] or bidirectional path tracing[56]).

In the past few years, CPU performance has grown to a

level where it has become possible to perform interactive ray-tracing of non-trivial scenes on large multiprocessor systems. Parker et al.[57] describe how to make such a system work by carefully optimizing the ray-tracer for the specific caching architecture of the multiprocessor machine, and by applying efficient tests of whether a surface point is in the penumbra or not. Only for points in the penumbra, lot of light source samples are required, while points in the umbra or completely lit points don not require any sampling.

Despite these optimizations and a large number of processors, there will clearly be scene sizes that cannot be handled at interactive frame rates any more. Therefore, Parker et al.[57] employ frameless rendering[58] to incrementally update the pixels in a random order instead of updating all pixels in the image, and then displaying the whole image at once. Using these methods, they achieve several frames per second on a 60 processor Onyx2 for fairly complex scenes.

5.2.1. The Holodeck

The holodeck algorithm[59] combines ray-tracing with a caching of the previously computed rays, so that these can be reused for different views. Rays are generated on the fly and stored in data base. A specific grid data structure is used to store *beams* (rays of light passing through the same cell in the same direction). Over time, the holodeck algorithm builds up the complete information of a light field. The algorithm as proposed by Larson does not have a notion of moving objects, and continues to reuse rays for an indefinite period of time. Thus, the original algorithm only deals with static scenes, but it has the advantage over light field rendering[24, 27] that the data is built up incrementally rather than requiring a lengthy precomputation phase.

The rendering using the holodeck algorithm works as follows. For new viewpoint, the display process determines which beams are required to render the image. Then, all rays inside this beam are searched for, first in main memory, then on disk (if rays have been swapped out of a smaller RAM cache), and finally, new sample rays are generated. From all the samples obtained in a certain time budget, an image is reconstructed by rendering the individual samples, and filling the inbetween holes with a Voronoi diagram of the samples.

5.2.2. The Render Cache

A similar caching scheme for dynamic scenes has been proposed by Walter et al.[60] The entities cached in their system are individual samples of illumination, composed of a 3D point location, color, object and image id (the latter can be used to compute the viewing direction), as well as an age.

The first step in rendering a new frame is to reproject the individual samples to the new viewpoint. In a second phase the image is traversed, and depth culling and hole filling are performed. This process uses heuristics based on strong differences in the depth buffer and object id, and removes distant objects shining through closer geometry. The resulting image can be displayed to the user.

During this second phase, a sampling priority is generated for each pixel in the destination image. This is used to determine the set of rays to be traced or re-traced for the current view. The age of a sample is also taken into account during generation of the priority value. Since only a subset of pixels can be sampled for each frame, an error-diffusion dithering algorithm is used to thin out the samples and to distribute new rays across the image according to the local sampling priority.

5.3. Combined Ray-Tracing and Hardware Rendering

Because pure ray-tracing is still quite expensive and because interactive frame rates can today only be achieved on large multiprocessor machines, a natural solution is to use graphics hardware for rendering those parts it can deal with, and then only using ray-tracing for filling in those parts that the hardware cannot handle. These approaches will be summarized in the following.

5.3.1. Hybrid Hardware Rendering and Ray-Tracing

Udeshi and Hansen[61] propose a system where OpenGL hardware is used to render the direct illumination on diffuse surfaces including shadows (a shadow volume algorithm[62, 50] is used for this purpose). Furthermore, one-bounce indirect illumination for diffuse surfaces is computed on the fly with a hemicube-style approach[63] for which the graphics hardware is also used.

Finally, the hardware renders an item buffer that is used to spawn primary rays for all surfaces that are not diffuse. A parallel ray-tracer working on several CPUs of a shared memory system is then used to fill in these non-diffuse parts. Frame rates of several frames per second are achieved with this method, where the rendering of the polygons generated for the shadow volume dominates the rendering times. Replacing the shadow volume method with a shadow map algorithm[49, 14] could probably remove this bottleneck.

5.3.2. Corrective Texture Mapping

Stamminger et al.[64] have recently proposed a texture based approach for hybrid ray-tracing and hardware rendering. A scene is first rendered by means of graphics hardware. This rendering can include global illumination effects, e.g. shadows. Although this approximate rendering contains all geometric features of the scene (which is important for navigation), it will in general not cover the whole range of lighting effects, as for example multiple reflections and refractions, or complex reflection characteristics.

In order to improve the quality of these interactive renderings, high-quality samples are acquired asynchronously by ray-tracing. The resulting error values, that is, the differences between these samples and the interactive solution,

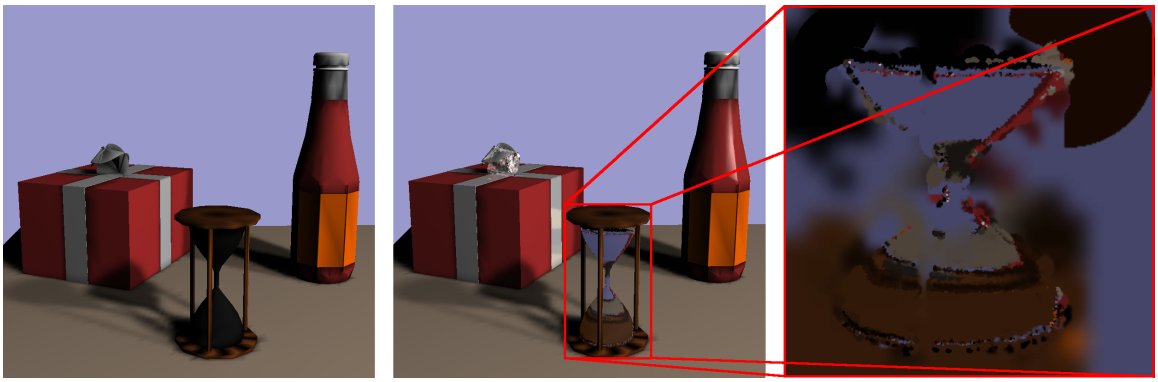

Figure 12: *Left: hardware rendering of a scene with diffuse direct illumination only. Center: high-quality solution generated by corrective texture mapping. Right: the corrective texture used for one of the objects. Images courtesy of Marc Stamminger.*

are stored in *corrective textures* which are mapped onto the corresponding object during the interactive display process. Figure 12 shows the result of the method along with one corrective texture.

As new samples for one specific object are ray-traced, they are splatted into the corrective texture. The area influenced by this new sample depends on the age of the samples already in the texture, as well as on the difference in viewing direction, under which the sample has been generated. Furthermore, the error between the ray-traced sample and the OpenGL rendering is used to guide the placement of new sample rays, so that highly specular regions are updated more frequently than mostly diffuse ones, and dynamic parts of the scene more frequently than static ones.

Figure 13: *Left: the object structure of a sample scene. Each red box corresponds to one object that shares one corrective texture. Right: a resulting rendering exhibiting reflections, refractions, and a caustic. Images courtesy of Marc Stamminger.*

6. Conclusion

In this State-of-The-Art report we have reviewed the work of many researchers on interactive display of global illumination solutions. We have seen that many of the methods rely on precomputed global illumination solutions stored in a form that at least loosely resembles a light field. This is not too surprising, since the problem of storing directionally dependent illumination on 2D manifolds naturally leads to a 4D data structure of some kind, which can then be interpreted as a light field. Unfortunately, four dimensions are not sufficient to extend the approaches to participating media if the viewer is to be allowed to stand inside the medium. An additional dimension, however, would further amplify the storage problems of these methods.

It is interesting to note that those representations used most frequently in interactive and realtime applications (and especially in games) make further simplifying assumptions and reduce the 4D light field to 2D environment maps. This certainly has something to do with the tradeoff between storage costs and efficiency of reconstruction for the light field methods, as discussed in Sections 3 and 4. We can expect these disadvantages of light fields to become less important as more research goes into more efficient compression schemes and adaptive acquisition techniques.

Finally, there needs to be more research on methods that can deal with dynamic scenes by recomputing the global illumination on the fly. The work that has been done on interactive ray-tracing and hybrid ray-tracing/hardware rendering is promising, but not feasible for many practical applications at the moment. Since many of these techniques are actually bound by memory bandwidth rather than CPU performance, and bandwidth does not improve as quickly as CPU speed, more research is required to overcome this problem.

7. Acknowledgments

This document was written while the author was working at the Max-Planck-Institute for Computer Science in Saarbrücken, Germany. Many thanks to Marc Stamminger for providing images for several techniques described in this report.

References

1. Michael F. Cohen and John R. Wallace. *Radiosity and Realistic Image Synthesis*. Academic Press, 1993.
2. Karol Myskowski and Tosiyasu. L. Kunii. Texture mapping as an alternative for meshing during walkthrough animation. In *Photorealistic Rendering Techniques*, pages 389–400. Springer, June 1994.
3. Francois X. Sillion and Claude Puech. *Radiosity & Global Illumination*. Morgan Kaufmann, 1994.
4. Rui Bastos. Efficient radiosity rendering using textures and bicubic reconstruction. In *Symposium on Interactive 3D Graphics*, 1997.
5. James F. Blinn and Martin E. Newell. Texture and reflection in computer generated images. *Communications of the ACM*, 19:542–546, 1976.
6. Pat Hanrahan and Jim Lawson. A language for shading and lighting calculations. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, pages 289–298, August 1990.
7. Mark Segal and Kurt Akeley. *The OpenGL Graphics System: A Specification (Version 1.2)*, 1998.
8. Paul Haeberli and Mark Segal. Texture mapping as a fundamental drawing primitive. In *Fourth Eurographics Workshop on Rendering*, pages 259–266, June 1993.
9. Ned Greene. Applications of world projections. In *Proceedings of Graphics Interface '86*, pages 108–114, May 1986.
10. D. Voorhies and J. Foran. Reflection Vector Shading Hardware. In *Computer Graphics (SIGGRAPH '94 Proceedings)*, pages 163–166, July 1994.
11. Wolfgang Heidrich and Hans-Peter Seidel. View-independent environment maps. In *Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 39–45, 1998.
12. Wolfgang Heidrich and Hans-Peter Seidel. Realistic, hardware-accelerated shading and lighting. In *Computer Graphics (SIGGRAPH '99 Proceedings)*, August 1999.
13. Shree K. Nayar. Catadioptric omnidirectional camera. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 482–488, June 1997.
14. Wolfgang Heidrich. *High-quality Shading and Lighting for Hardware-accelerated Rendering*. PhD thesis, University of Erlangen-Nürnberg, April 1999.
15. Lance Williams. Pyramidal parametrics. In *Computer Graphics (SIGGRAPH '83 Proceedings)*, pages 1–11, July 1983.
16. Andreas Schilling, Günter Knittel, and Wolfgang Straßer. Texram: A smart memory for texturing. *IEEE Computer Graphics and Applications*, 16(3):32–41, May 1996.
17. Paul E. Debevec and Jitendra Malik. Recovering high dynamic range radiance maps from photographs. In *Computer Graphics (SIGGRAPH '97 Proceedings)*, pages 369–378, August 1997.
18. Paul E. Debevec. Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *Computer Graphics (SIGGRAPH '98 Proceedings)*, pages 189–198, July 1998.
19. Gene Miller and Robert Hoffman. Illumination and Reflection Maps: Simulated Objects in Simulated and Real Environments. In *SIGGRAPH '84 Course Notes – Advanced Computer Graphics Animation*, July 1984.
20. Robert R. Lewis. Making shaders more physically plausible. In *Fourth Eurographics Workshop on Rendering*, pages 47–62, June 1993.
21. Jan Kautz and Michael McCool. Approximation of glossy reflection with prefiltered environment maps. In *Proc. of Graphics Interface*, May 2000.
22. Brian Cabral, Marc Olano, and Paul Nemec. Reflection space image based rendering. In *Computer Graphics (SIGGRAPH '99 Proceedings)*, pages 165–170, August 1999.
23. Jan Kautz, Pere-Pau Vázquez, Wolfgang Heidrich, and Hans-Peter Seidel. Unified approach to prefiltered environment maps. In *submitted*, 2000.
24. Marc Levoy and Pat Hanrahan. Light field rendering. In *Computer Graphics (SIGGRAPH '96 Proceedings)*, pages 31–42, August 1996.
25. E. H. Adelson and J. R. Bergen. *Computational Models of Visual Processing*, chapter 1 (The Plenoptic Function and the Elements of Early Vision). MIT Press, Cambridge, MA, 1991.
26. Xianfeng Gu, Steven J. Gortler, and Michael F. Cohen. Polyhedral geometry and the two-plane parameterization. In *Rendering Techniques '97 (Proceedings of Eurographics Rendering Workshop)*, pages 1–12, June 1997.
27. Steven J. Gortler, Radek Grzeszczuk, Richard Szelinski, and Michael F. Cohen. The Lumigraph. In *Computer Graphics (SIGGRAPH '96 Proceedings)*, pages 43–54, August 1996.
28. Peter-Pike Sloan, Michael F. Cohen, and Steven J. Gortler. Time critical Lumigraph rendering. In *Symposium on Interactive 3D Graphics*, 1997.
29. Emilio Camahort, Apostolos Lerios, and Donald

Fussell. Uniformly sampled light fields. In *Rendering Techniques '98 (Proceedings of Eurographics Rendering Workshop)*, pages 117–130, March 1998.

30. Glenn Tsang, Sherif Ghali, Eugene L. Fiume, and Anastasios N. Venetsanopoulos. A novel parameterization of the light field. In *Proceedings of the Image and Multi-dimensional Digital Signal Processing Workshop (IMDSP)*, 1998.

31. Wolfgang Heidrich, Hartmut Schirmacher, and Hans-Peter Seidel. A warping-based refinement of lumigraphs. In *Proceedings of WSCG*, 1999.

32. Hartmut Schirmacher, Wolfgang Heidrich, and Hans-Peter Seidel. Adaptive acquisition of Lumigraphs from synthetic scenes. In *Computer Graphics Forum (Proceedings of Eurographics '99)*, pages 151–160, September 1999.

33. Hartmut Schirmacher, Wolfgang Heidrich, and Hans-Peter Seidel. High-quality interactive lumigraph rendering through warping. In *Graphics Interface 2000*, May 2000.

34. Gavin Miller, Steven Rubin, and Dulce Ponceleon. Lazy decompression of surface light fields for precomputed global illumination. In *Rendering Techniques '98 (Proceedings of Eurographics Rendering Workshop)*, pages 281–292, March 1998.

35. Daniel N. Wood, Daniel I. Azuma, Ken Aldinger, Brian Curless, Tom Duchamp, David H. Salesin, and Werner Stuetzle. Surface light fields for 3D photography. In *Computer Graphics (SIGGRAPH 2000 Proceedings)*, July 2000.

36. Aaron W. F. Lee, Wim Sweldens, Peter Schröder, Lawrence Cowsar, and David Dobkin. Maps: Multiresolution adaptive parameterization of surfaces. In *Computer Graphics (SIGGRAPH '98 Proceedings)*, pages 31–42, July 1998.

37. Wolfgang Heidrich, Jan Kautz, Philipp Slusallek, and Hans-Peter Seidel. Canned lightsources. In *Rendering Techniques '98 (Proceedings of Eurographics Rendering Workshop)*, 1998.

38. Wolfgang Heidrich, Hendrik Lensch, Michael F. Cohen, and Hans-Peter Seidel. Light field techniques for reflections and refractions. In *Rendering Techniques '99 (Proceedings of Eurographics Rendering Workshop)*, 1999.

39. Marcus Magnor and Bernd Girod. Adaptive block-based light field coding. In *3rd International Workshop on Synthetic and Natural Hybrid Coding and Three-Dimensional Imaging*, pages 140–143, September 1999.

40. Paul Lalonde and Alain Fournier. Interactive rendering of wavelet projected light fields. In *Graphics Interface '99*, pages 107–114, June 1999.

41. Marc Stamminger, Annette Scheel, Xavier Granier, Frederic Perez-Cazorla, George Drettakis, and Francois Sillion. Efficient glossy global illumination with interactive viewing. In *Graphics Interface '99*, pages 50–57, June 1999.

42. Peter Shirley, Bretton Wade, Philip Hubbard, David Zareski, Bruce Walter, and Donald P. Greenberg. Global illumination via density estimation. In *Eurographics Rendering Workshop 1995*, pages 219–231, June 1995.

43. Wolfgang Stürzlinger and Rui Bastos. Interactive rendering of globally illuminated glossy scenes. In *Rendering Techniques '97*, pages 93–102, 1997.

44. Marc Stamminger, Philipp Slusallek, and Hans-Peter Seidel. Interactive walkthroughs and higher order global illumination. In *Modeling, Virtual Worlds, Distributed Graphics*, pages 121–128, November 1995.

45. Per Christensen. *Hierarchical Techniques for Glossy Global Illumination*. PhD thesis, University of Washington, 1995.

46. Peter Schroeder. *Wavelet Algorithms for Illumination Computations*. PhD thesis, Princeton University, 1994.

47. Bruce Walter, Gün Alppay, Eric Lafortune, Sebastian Fernandez, and Donald P. Greenberg. Fitting virtual lights for non-diffuse walkthroughs. In *Computer Graphics (SIGGRAPH '97 Proceedings)*, pages 45–48, August 1997.

48. Alexander Keller. Instant radiosity. In *Computer Graphics (SIGGRAPH '97 Proceedings)*, pages 49–56, August 1997.

49. Marc Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli. Fast shadow and lighting effects using texture mapping. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, pages 249–252, July 1992.

50. Paul J. Diefenbach and Norman Badler. Pipeline Rendering: Interactive refractions, reflections and shadows. *Displays: Special Issue on Interactive Computer Graphics*, 15(3):173–180, 1994.

51. Paul J. Diefenbach. *Pipeline Rendering: Interaction and Realism Through Hardware-based Multi-Pass Rendering*. PhD thesis, University of Pennsylvania, 3401 Walnut Street, Suite 400A, Philadelphia, PA 19104-6228, June 1996.

52. Rui Bastos, Keneth Hoff, William Wynn, and Anselmo Lastra. Increased photorealism for interactive architectural walkthroughs. In *Symposium on Interactive 3D Graphics*, pages 183–190, 1999.

53. Rui Bastos. *Superposition Rendering: Increased Realism for Interactive Walkthrough*. PhD thesis, University of North Carolina at Chapel Hill, 1999.

54. Eyal Ofek and Ari Rappoport. Interactive reflections on curved objects. In *Computer Graphics (SIGGRAPH '98 Proceedings)*, pages 333–342, July 1998.

55. Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, pages 134–145, July 1984.

56. Eric Veach and Leonidas Guibas. Optimally combining sampling techniques for monte carlo rendering. In *"Computer Graphics (SIGGRAPH '95 Proceedings)*, pages 419–428, August 1995.

57. Steven Parker, William Martin, Peter-Pike J. Sloan, Peter Shirley, Brian Smits, and Charles Hansen. Interactive ray tracing. In *1999 ACM Symposium on Interactive 3D Graphics*, pages 119–126, April 1999.

58. Gary Bishop, Henry Fuchs, Leonard McMillan, and Ellen J. Scher Zagier. Frameless rendering: Double buffering considered harmful. In *Computer Graphics (SIGGRAPH '94 Proceedings)*, pages 175–176, July 1994.

59. Gregory Ward Larson and Maryann Simmons. The holodeck ray cache: An interactive rendering system for global illumination in non-diffuse environments. *ACM Transactions on Graphics*, 18(4):361–368, October 1999.

60. Bruce Walter, George Drettakis, and Steven Parker. Interactive rendering using the render cache. In *Eurographics Rendering Workshop*, June 1999.

61. Tushar Udeshi and Charles Hansen. Towards interactive, photorealistic rendering of indoor scenes: A hybrid approach. In *Rendering Techniques '99*, pages 63–76, June 1999.

62. Franklin C. Crow. Shadow algorithms for computer graphics. In *Computer Graphics (SIGGRAPH '77 Proceedings)*, pages 242–248, July 1977.

63. Michael F. Cohen, Shenchang Eric Chen, John R. Wallace, and Donald P. Greenberg. A progressive refinement approach to fast radiosity image generation. In *Computer Graphics (SIGGRAPH '88 Proceedings)*, pages 75–84, August 1988.

64. Marc Stamminger, Jörg Haber, and Hartmut Schirmacher. Walkthroughs with corrective texturing. In *Rendering Techniques 2000*, pages 377–388, June 2000.