Light Field Techniques for Reflections and Refractions

Wolfgang Heidrich[†], Hendrik Lensch[†], Michael F. Cohen[*], Hans-Peter Seidel[†]

[†]Max-Planck-Institute for Computer Science
{heidrich,lensch,seidel}@mpi-sb.mpg.de

[*]Microsoft Research
mcohen@microsoft.com

Abstract. Reflections and refractions are important visual effects that have long been considered too costly for interactive applications. Although most contemporary graphics hardware supports reflections off curved surfaces in the form of environment maps, refractions in thick, solid objects cannot be handled with this approach, and the simplifying assumptions of environment maps also produce visible artifacts for reflections.
Only recently have researchers developed techniques for the interactive rendering of true reflections and refractions in curved objects. This paper introduces a new, light field based approach to achieving this goal. The method is based on a strict decoupling of geometry and illumination. Hardware support for all stages of the technique is possible through existing extensions of the OpenGL rendering pipeline. In addition, we also discuss storage issues and introduce methods for handling vector-quantized data with graphics hardware.

1 Introduction

Reflections and refractions are important visual effects that have not been handled appropriately in interactive applications. Highly reflective or transparent objects can be thought of as a lens that transforms incoming rays into outgoing rays in some new direction. Consider a glass pitcher half full of water. A ray of light entering the pitcher may be reflected and/or refracted many times, eventually leaving the pitcher in some new direction. (We will ignore the fact that some of the energy may be absorbed and the ray may split into a tree of subrays.) Unfortunately, it is not possible to compute this mapping from rays in to rays out in real time for complex objects. However, for static objects, this map can be computed offline and reused at each frame at render time if we can find a way to efficiently represent this mapping and provide fast lookup methods.

The light field [13] and Lumigraph [6] image based rendering approaches faced a similar problem. In this work it was recognized that one could precompute all the light leaving the convex hull of an object and then quickly lookup appropriate values to create any image of the object. This 4D mapping from rays out from an object to colors was the central structure of these methods.

A key idea in the current work is to replace the mapping from rays to colors as found in the Lumigraph with a mapping from rays to rays. In other words, replace the RGB color triplet with four numbers representing a new ray direction. Storing this mapping in the form of a Lumigraph allows us to use all the methodologies developed in [13] and [6].

This completely decouples the illumination from the geometry, and allows us to exchange the environment and the refracting/reflecting object independently of each other. Such a separation also has the advantage that some extra blurring for the indirect illumination will in most cases be tolerable since direct illumination is visually much more important than indirect illumination for almost all scenes. This means that one can get away with a lower resolution light field, which saves memory and reduces the acquisition cost.

After a discussion of previous work, we first introduce some notation for light fields, and then describe our technique for rendering reflections and refractions in Section 4. We then turn to the hardware rendering of vector-quantized light fields [13] in Section 5, and discuss our approaches in Section 6.

2 Previous Work

Producing approximate reflections and refractions at interactive rates has been done with environment maps [2]. The inherent assumption is that what is seen in a reflection is sufficiently far away from the reflector. This assumption breaks down in many scenes.

Environment maps can also be used for refractions at thin surfaces. A thin surface is one, for which the thin lens approximation holds (see [3]). This approximation assumes that the entry and exit points of each ray coincide. It does not hold for objects like glass balls, but it can be used for windows or spectacle lenses.

Only recently have researchers started to develop algorithms that do not have these restrictions. A commonly used technique for rendering mirror reflections in planar objects is given in [5]: 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. It is possible to realize multiple reflections by repeating this process as explained in [5].

For curved objects the geometric transformations yielding the reflected geometry are more complex. Rather than transforming the complete object with a single affine transformation, a different transformation is required for each point on the reflected object. In [16] an algorithm is introduced, in which all vertices of the reflected geometry are individually transformed in software. This approach only works at interactive frame rates for relatively smooth objects that are either concave or convex.

As mentioned above, the light field [13] and Lumigraph [6] approaches are based on a dense sampling of the plenoptic function [1], which describes the flow of light in a scene. This way, the light field outside the convex hull of an object in empty space can be represented as a 4-dimensional function in the parameters u, v, s, and t. The parameterization of this function is a topic of ongoing research [4], but typically the 2-plane parameterization explained in Section 3 is used [6, 13]. The advantage of the light field approach is that images of the scene from new camera positions can simply be computed by interpolating radiance values in 4 dimensions. This is a very inexpensive operation that can be further accelerated through the use of OpenGL-compliant graphics hardware, as has been shown in [6] and [19].

The disadvantage of this method is that it involves large memory requirements to store the 4D data structures. A light field of moderate resolution can easily have a size of multiple Gigabytes. The authors of [13] have proposed to use vector quantization (VQ) to compress the data. VQ has some interesting advantages over other compression methods that make it attractive for compressing light fields. Firstly, decompression is a constant time operation, and secondly the different entries can be decompressed independently of each other and in arbitrary order (random access). In Section 5 we describe how the hardware accelerated rendering algorithm for light fields [6] can be extended to the direct rendering of vector-quantized light fields. This means that the light fields do not have to be decompressed before hardware rendering, which is particularly interesting since texture RAM is typically a scarce resource.

The Lumigraph [6] extends the concept of a light field by adding some geometric information which helps compensating for artifacts that arise from the use of quadrilinear interpolation for the reconstruction. A coarse polygon mesh is stored together with the images. The mesh is used to first find the approximate depth along the ray to be reconstructed, and then this depth is used to correct the weights for the interpolation. This depth corrected rendering can also be accelerated with graphics hardware [6].

Miller [15] attempts to overcome blurring artifacts by introducing a parameterization in which the u and v parameters of the light field are the surface parameters of the reflecting object, and s and t parameterize the hemisphere of directions over the surface point (u, v). The authors call this a *surface light field* because the (u, v) parameters are directly attached to the surface of the reflector. This is particularly well suited to mostly diffuse objects with well defined surface parameters.

In addition to this different parameterization, [15] also introduces a block-based compression scheme which achieves higher compression ratios than VQ, but is also more complicated and requires the decompression of a complete block of values. This decompression has to be performed in software during the resampling process used for generating the texture maps.

Both light field representations reach their limits when it comes to mirror reflections and narrow specular highlights from light sources. These will in both approaches still result in some amount of unwanted blurring, due to the limited (s, t) resolution of the light field.

This problem is fixed by another approach for using light fields to render reflections [14]. There, the authors introduce the concept of *image-based ray-tracing* to render mirror reflections. In this work, a light field not only stores radiance values, but also geometric information such as depth and normal. The method proceeds by tracing rays through a light field of layered depth images [18]. This method is computationally expensive and cannot be performed in real time on contemporary hardware.

3 Light Fields

Our work builds strongly on the light field [13], and the Lumigraph [6]. A light field is a dense sampling of the 5-dimensional plenoptic function [1], which describes the radiance at every point in space in every direction. Since radiance does not change along a ray in empty space (e.g., outside the convex hull of an object), the dimensionality can be reduced by one, if an appropriate parameterization is found, that reflects this property.

The so-called 2-plane parameterization used by [13] and [6] representation fulfills this requirement. It represents a ray via its intersection points with two parallel planes. Since each of these points is characterized by two parameters in the plane, this results in a 4-dimensional function that is sampled through a regular grid on each plane (see Fig. 1).

One useful property of the 2-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 images with eye points regularly spaced on the (s, t)-plane.

Moreover, 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 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 [6], this can also be done in hardware.

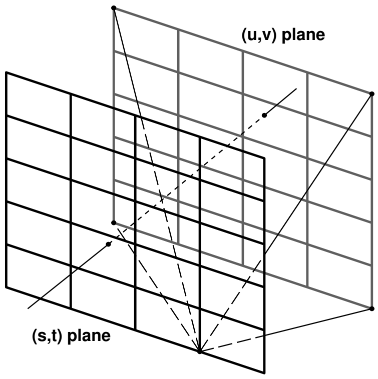

Fig. 1. A light field can be parameterized as a 2-dimensional array of images taken from a regular grid of eye points on the (s, t)-plane through a window on the (u, v)-plane.

For each eye point on the (s, t)-plane, the hardware rendering algorithm draws the grid cells surrounding the eye point using graphics hardware. The alpha values of the polygon vertices are set to 1 at the eye point and to 0 everywhere else. Furthermore, each rendered polygon is textured with the (u, v)-slice corresponding to the eye point, and these textures are weighted by the alpha values of the polygons. This combination of alpha blending and texture mapping with bi-linear interpolation yields the desired reconstruction.[1] This hardware algorithms is many times faster than a purely software-based approach. For example, on an SGI O2 the hardware renderer can achieve approximately 25 frames per second in full screen resolution. This number is almost independent of light field resolution.

The possibility to use graphics hardware for rendering is also what distinguishes the 2-plane parameterization from most other parameterizations that have been proposed for light fields (see, for example [4]). For us, this is the reason to use the 2-plane parameterization.

4 Decoupling Geometry and Illumination

As stated above, the core of the proposed method is to separate the geometry and the illumination of an object into two distinct image-based data structures. The first data structure is a 2-plane parameterized “light field” containing a mapping from incoming rays to outgoing rays based on the geometry and refractive properties of the object. The outgoing rays are stored in the form of a color coded direction. The illumination corresponding to this outgoing ray can either be provided in the form of an environment map or in the form of another light field.

Thus, to render a complete image, first the geometry light field is rendered, yielding an image of color coded ray directions. These can then be used to look up the illumination from an environment map, to trace new rays into a scene, or lookup a color from a second light field describing the surrounding scene.

When using environment maps, the geometry light field can directly contain the 2D texture coordinates for the entry in the environment map that corresponds to the

[1] As pointed out in [6] quadri-linear interpolation is not possible due to the use of Gouraud shading for the interpolation of the alpha values across polygons. However, the described algorithm yields a close approximation.

refracted ray direction. The format for these values depends on the specific parameterization used for the environment map. For example, for a spherical environment map [7], the texture coordinates are given as the x and y components of the normalized halfway vector $\vec{h} = (h_x, h_y, h_z)^T$ between the refracted viewing ray $\vec{v}$ and the z-axis (see Fig. 2).

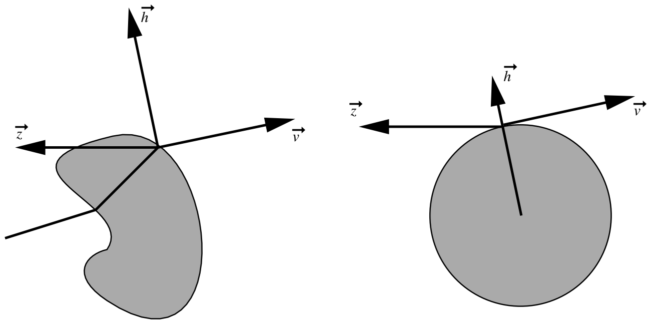

Fig. 2. The texture coordinates that correspond to a certain ray direction $\vec{v}$ in a spherical environment map are given by the normalized halfway vector $\vec{h}$ between $\vec{v}$ and the z-axis (left). This is because a spherical map is the image of a small, reflective sphere as seen by an orthographic camera looking into the negative z-axis. The vector $\vec{h}$ corresponds to both the point on the sphere where $\vec{v}$ is the reflection of $\vec{z}$, and to the sphere's surface normal in that point.

In our implementation we use the parabolic parameterization presented in [10]. It has the advantage of providing a more uniform sampling of the hemisphere of directions than the spherical parameterization, and the texture coordinates are less expensive to compute. The texture coordinates of the parabolic parameterization are given as h_x/h_z and h_y/h_z, which means that a simple division suffices to compute the texture coordinates instead of a vector normalization. While the advantage of a uniform sampling is essential, the performance benefits are not significant in our case.

Independent of how the illumination is represented, the first step in the algorithm is the hardware-based rendering of the geometry light field as described in [6] and [19]. Afterwards, the framebuffer contains an image with the color coded texture coordinates for the environment map holding the illumination information. Now the framebuffer contents are read back to main memory and for each pixel the texture coordinates need to be used to look up the illumination from the environment map, before the resulting image is written back to the framebuffer, yielding the final image.

The texture lookup can be achieved in one of the following two ways. Firstly, it can be performed in software. This is a relatively inexpensive operation, that is feasible even on low end platforms, as shown below. Alternatively, the graphics hardware can be employed for the texture lookup, using an extension that is available from SGI. The so-called *pixel texture extension* [17, 8] modifies the rendering pipeline in such a way that the color values of the pixels in an image can be interpreted as texture coordinates, which are then in turn used to texture each individual pixel. This feature can be applied during so-called *pixel transfer operations*, that is, during the transfer of images to and from the framebuffer, and is exactly the functionality required for our algorithm. Although pixel textures are currently a proprietary extension from SGI, their usefulness is demonstrated by several applications, and will hopefully result in a more widespread availability of this extension.

Fig. 3 shows images that were generated using this approach. The left image represents the color coded texture coordinates reconstructed from the geometry light field. The light field itself was generated using ray-tracing. Center and right represent the final result after the application of the environment map. This method is very fast, and achieves between 15 and 20 fps. on an Octane MXE using the pixel texture extension. The same images can be rendered with the software method at about 10 frames per second on an SGI O2 with a 175 MHz R10k (images are of size 400×400 pixels).

Fig. 3. Light fields rendering with decoupled geometry and illumination, the latter being provided through an environment map. Left: color coded texture coordinates for the environment map, as extracted from the geometry light field. Center/right: final renderings.

4.1 Illumination from Light Fields

Instead of using an environment map to store the illumination, it is also possible to use a second light field. The advantage of this approach is the additional dependency on the surface location, which allows for the proper handling of objects close to the refractor. As stated above, the use of environment maps for illumination is typically not of interest for implementing reflections. If the illumination is stored in light fields, however, the situation is different, because the near field effects that can be achieved with this method exceed the possibilities of traditional geometry-based rendering with environment maps.

To store the illumination in a light field, the geometry light field has to contain the correct 4-dimensional ray direction u, v, s, and t referencing into the illumination light field. The ray direction can be color coded into the R, G, B and A channels of the geometry light field. The exact coordinates, of course, again depend on the parameterization of the illumination light field.

The rendering algorithm proceeds similarly to the case where the illumination is stored in an environment map. First, the geometry light field is rendered using hardware acceleration. Then, the framebuffer contents are read back to main memory. The ray lookup for determining the illumination is either performed in software, or using pixel textures and 4-dimensional texture mapping. The latter method will be described in more detail in Section 5.

4.2 Realistic Materials

So far, we have only considered either reflections or refractions, which is mandated by the fact that the geometry light field can only store either the coordinates for the reflection or for the refraction part. However, it is possible to combine the two parts using multi-pass rendering, and also to develop models for more complicated materials, such as glossy reflections and the inclusion of a Fresnel term. These multi-pass techniques have been introduced for geometry-based rendering in [11] and [9], but they directly translate to the image-based rendering techniques presented in this paper. Please refer to the listed publications for the details. Fig. 4 gives an example of these techniques.

Fig. 4. The glossy torus on the left is the result of a geometry rendering with a prefiltered environment map. The torus in the center represents the refraction part. Both parts have been weighted by the appropriate Fresnel term. The right image shows the combination of the two parts.

5 Hardware Rendering of Vector-quantized Light Fields

One of the fundamental problems of light fields and Lumigraphs is the large memory consumption. For practical applications of these techniques it is therefore mandatory to compress the data sets. The problem with current implementations of compression schemes is that the hardware cannot directly deal with the compressed data. As a consequence, the required parts of the light field have to be decompressed before the hardware can be used. This is unfortunate since the texture RAM is a particularly scarce resource in many systems. If it is too small to hold the required parts of the light field, these need to be swapped in and out for every frame, which consumes large amounts of bandwidth, and results in serious performance penalties. The major reason why the hardware rendering of light fields performs so well on O2s (see Section 3), is that on this platform the texture RAM resides in main memory and is thus almost unlimited.

In the following we describe two slightly different methods for rendering vector-quantized light fields directly in hardware without the need to decompress them beforehand. VQ compresses a light field by replacing the color values of an adjacent block of pixels by a single index into a lookup table containing the color values. An often used configuration is to choose the block size as 2^4 and the table size as 2^{16}, meaning that 2 samples in each of the parametric directions u, v, s, and t are subsumed in a block, and all 16 RGB colors in the block are replaced by a single 2 Byte color index. This yields a compression ratio of $1 : 24$ (neglecting the size of the lookup table). The advantage of VQ over other compression techniques is that the basic structure of the data set is preserved: a light field is a 4-dimensional array of samples. A vector-quantized light field with the above parameters is a 4-dimensional array of color indices, but the resolution is reduced by a factor of 2 in each parametric direction. Instead of having a single lookup table that yields the complete 2^4 block of color values for each index, it is also possible to use 16 different tables, where each only returns a single color value. This is one table for all combinations of odd/even sample locations in the four parametric directions u, v, s, and t.

This property can be used for two related algorithms of rendering compressed light fields, using a hardware extension that is currently available. This extension is the so-called *texture color table*, which, in contrast to all the other color tables in the OpenGL pipeline, is *not* a pixel transfer operation, but takes place *after* the texture lookup and filtering. This extension is available on all SGI systems, and has in the past been used primarily for volume rendering (see, e.g. [20]).

This extension allows us to store the light field as a set of one-component textures. Then, multiple rendering passes with different color lookup tables are used to reconstruct the colors. Special care has to be taken for the interpolation of these color values. The difficulty is that the color in each pixel needs to be interpolated from the colors resulting from lookups in all the color tables. This can be achieved through alpha blending, as described below.

Let us assume for a moment that only the resolution on the (s, t)-plane, the plane of eye points, has been halved by the VQ algorithm. Then, the rendering is a trivial extension of the algorithm described in Section 3. The only difference to the algorithm there is that the correct texture color table needs to be loaded before polygons surrounding the (s, t)-sample point are rendered. Since in this situation there are only four different color tables, the (s, t)-samples can be ordered in such a way that only 4 changes of the color table are required for every frame (all samples with even s and even t have one table, all samples with even s and odd t another, and so forth).

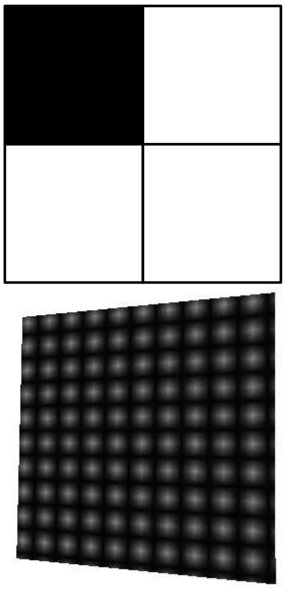

Fig. 5. Top: the 2×2 texture used to render the basis functions into the alpha buffer. Bottom: The set of basis functions for one color table. This image was generated by applying bilinear texture filtering to several replicated copies of the texture from the left.

Now, if the resolution on the (u, v)-plane is also halved, then it is necessary to interpolate the color values for a pixel in the (u, v)-plane between images generated by four different table lookups. This is achieved by rendering the basis functions for a bi-linear interpolation into the alpha channel, and then multiplying this basis function with the result of a nearest-neighbor sampled image resulting from a single table lookup. Repeating this process four times with four different lookup tables and the four corresponding basis functions, and summing up the results in the framebuffer, yields the correctly reconstructed image.

To render the basis functions into the alpha channel, we specify a single 2×2 texture in which exactly one pixel is one, and the others are zero. By replicating this texture according to the resolution of the (u, v)-plane in the original light field, and by using bi-linear texture mapping, the basis functions for, say, all odd u and all even v samples can be rendered. If the same texture is shifted by one texel in subsequent rendering passes, the other basis functions can be rendered as well. Fig. 5 illustrates this method. The cost of this technique is about four times the cost of rendering an uncompressed light field, so that the method only pays off if the texture RAM is too small to hold the complete data set. The color plates show a comparison of the uncompressed light field rendering with vector-quantized data sets.

Another hardware-accelerated algorithm for rendering light fields that might become interesting in the future, is 4-dimensional texture mapping. Currently, this feature is supported on some high-end SGI machines. With 4D texture mapping the rendering of 2-plane parameterized light fields becomes extremely simple. The (s, t) grid is projected down on the (u, v)-plane to compute the texture coordinates for the corner vertices of the (u, v)-grid. Then (u, v)-plane is rendered as a single polygon with the precomputed 4D texture coordinates, and the light field as a texture.

4D texture mapping is also interesting in combination with the pixel texture extension, and can be applied in the reflection and refraction algorithm from Section 4.1. Here, the R, G, B, and A channels of each pixel in the image are interpreted as 4D texture coordinates u, v, s, and t.

The disadvantage of 4D texture mapping in the context of light field rendering is, that the whole light field needs to fit into the texture RAM. For practical applications this mandates the use of a compression technique, and VQ is again a good choice that allows us to exploit graphics hardware.

Since 4D texture mapping does not treat the (u, v)-plane differently from the (s, t)-plane like the algorithm from [6], the implementation is somewhat different than described above. What remains is the concept of using the alpha channel to render basis functions for the different lookup tables. This time, however, the basis functions are generated by 4D textures of resolution 2^4. In each of these textures, exactly two pixels are one and the others are zero. Fig. 6 illustrates this for a 2-dimensional "light field".

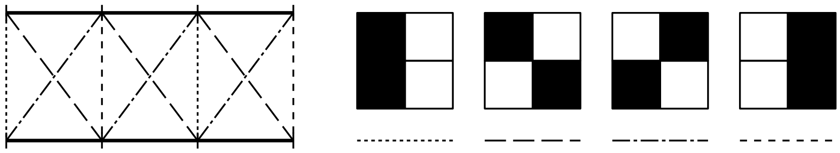

Fig. 6. The textures that comprise the basis functions for a 2D "light field", and the rays that each basis function corresponds to.

6 Discussion and Conclusion

In this paper we have explored practical techniques for applying light fields to the rendering of reflections and refractions on curved objects. Firstly, our method separates geometric and illumination information into two independent, image-based representations. The advantage of this approach is an increased flexibility for modeling. Image-based representations of objects can be positioned in a scene and lit by image-based representations of the illumination in that scene. Existing graphics hardware can be efficiently utilized for the implementation of the algorithm.

Secondly, we have described a technique for rendering vector-quantized light fields directly with graphics hardware. This approach is not limited to the rendering of reflections and refractions, but is a natural extension of the hardware algorithm presented in [6], and can therefore be used with all hardware light field renderers.

The techniques presented here rely on some OpenGL extensions that have not yet become mainstream features in low-end graphics hardware. However, all the extensions used have proven to be useful in several other applications as well [20, 12], so that it seems likely that some of these features will find their way into future graphics standards.

This idea of storing geometric information instead of simply color values in a light field can be extended even further. For example, if normal vectors are stored in the light field data structure, the local illumination for the object can be computed using the very same hardware techniques also applied to normal maps in [11] and [9]. With these techniques, it is possible to recompute the illumination for each point using complex materials models. This allows for image-based rendering with changing illumination, an area that has recently been of increased interest in the research community.

7 Acknowledgments

The environment map used in this paper is a resampled version of the cafe used in [7]. The Buddha light field is from the Stanford collection of light field data sets.

References

1. 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.
2. J. F. Blinn and M. E. Newell. Texture and reflection in computer generated images. *Communications of the ACM*, 19:542–546, 1976.
3. M. Born and E. Wolf. *Principles of Optics*. Pergamon Press, Oxford, 6 edition, 1993.
4. E. Camahort, A. Lerios, and D. Fussell. Uniformly sampled light fields. In *Rendering Techniques '98*, pages 117–130, March 1998.
5. P. J. Diefenbach. *Pipeline Rendering: Interaction and Realism Through Hardware-based Multi-Pass Rendering*. PhD thesis, University of Pennsylvania, June 1996.
6. S. J. Gortler, R. Grzeszczuk, R. Szelinski, and M. F. Cohen. The Lumigraph. In *SIGGRAPH '96 Proceedings*, pages 43–54, August 1996.
7. P. Haeberli and M. Segal. Texture mapping as A fundamental drawing primitive. In *Fourth Eurographics Workshop on Rendering*, pages 259–266, June 1993.
8. P. Hansen. Introducing pixel texture. In *Developer News*, pages 23–26. SGI, May 1997.
9. W. Heidrich. *High-quality Shading and Lighting for Hardware-accelerated Rendering*. PhD thesis, University of Erlangen-Nürnberg, April 1999.
10. W. Heidrich and H.-P. Seidel. View-independent environment maps. In *Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 39–45, 1998.
11. W. Heidrich and H.-P. Seidel. Realistic, hardware-accelerated shading and lighting. In *SIGGRAPH '99 Proceedings*, August 1999. See http://www.mpi-sb.mpg.de/˜heidrich.
12. W. Heidrich, R. Westermann, H.-P. Seidel, and Th. Ertl. Applications of pixel textures in visualization and realistic image synthesis. In *Symposium on Interactive 3D Graphics*, 1999.
13. M. Levoy and P. Hanrahan. Light field rendering. In *SIGGRAPH '96 Proceedings*, pages 31–42, August 1996.
14. D. Lischinski and A. Rappoport. Image-based rendering for non-diffuse synthetic scenes. In *Rendering Techniques '98*, pages 301–314, June 1998.
15. G. Miller, S. Rubin, and D. Ponceleon. Lazy decompression of surface light fields for precomputed global illumination. In *Rendering Techniques '98*, pages 281–292, March 1998.
16. E. Ofek and A. Rappoport. Interactive reflections on curved objects. In *SIGGRAPH '98 Proceedings*, pages 333–342, July 1998.
17. SGI. *Pixel Texture Extension*, December 1996. Specification document, available from http://www.opengl.org.
18. J. W. Shade, S. J. Gortler, L. He, and R. Szeliski. Layered depth images. In *SIGGRAPH '98 Proceedings*, pages 231–242, July 1998.
19. P.-P. Sloan, M. F. Cohen, and S. J. Gortler. Time critical Lumigraph rendering. In *Symposium on Interactive 3D Graphics*, 1997.
20. R. Westermann and Th. Ertl. Efficiently using graphics hardware in volume rendering applications. In *SIGGRAPH '98 Proceedings*, pages 169–178, July 1998.

Light field rendering with decoupled geometry and illumination, the latter being provided through an environment map. Top: color coded texture coordinates for the environment map, as extracted from the geometry light field. Bottom: final renderings. Compare Fig. 3.

The glossy torus on the left is the result of a geometry rendering with a prefiltered environment map. The torus in the center represents the refraction part. Both parts have been weighted by the appropriate Fresnel term. The right image shows the combination of the two parts. Compare Fig. 4.

Rendering of vector-quantized light fields. Left: uncompressed, center: reduced resolution on the (u, v)-plane, right: reduced resolution in all 4 dimensions.