Real-Time Reflection Mapping with Parallax

Jingyi Yu* Jason Yang*
Massachusetts Institute of Technology

Leonard McMillan†
University of North Carolina at Chapel Hill

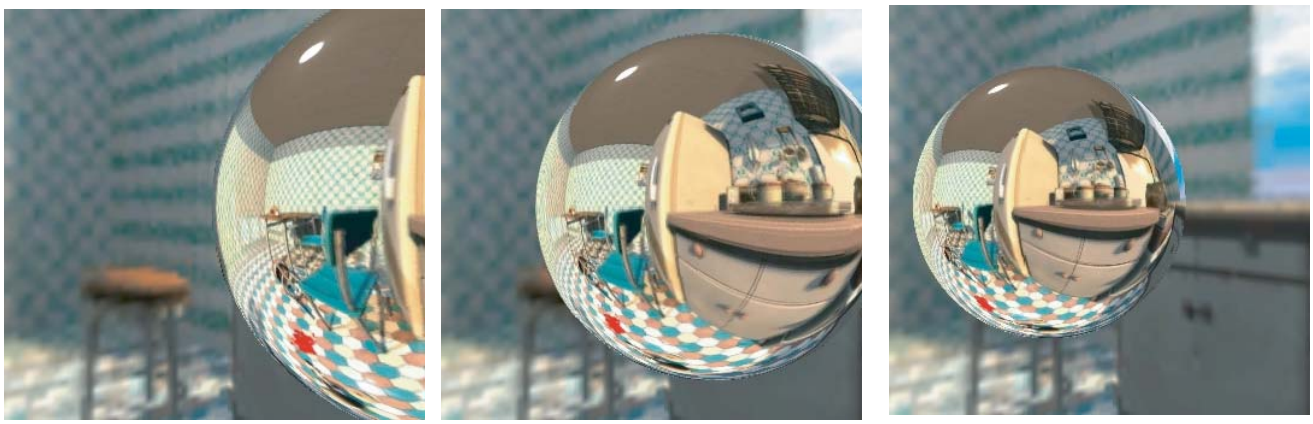

Figure 1: Reflections on a moving sphere rendered using our method. Notice the change of parallax: the table is partially occluded by the green chair (left) and turns gradually visible (middle and right).

Abstract

We present a novel algorithm to efficiently render accurate reflections on programmable graphics hardware. Our algorithm overcomes problems that commonly occur in environment mapping such as the lack of motion parallax and inaccuracies when objects are close to the reflectors. In place of a 2D environment map, which only represents points infinitely far away from the reflector, we use six 4D light field slabs to represent the surrounding scene. Each reflected ray is rendered by indexing into these precaptured environment light fields. We are able to render accurate reflections with motion parallax at interactive frame rates independent of the reflector geometry and the scene complexity. Furthermore, we can move the reflectors within a constrained region of space and guarantee that the environment light field provides the necessary rays. We benefit from the programmability of existing graphics hardware to efficiently compute the reflected rays and transform them into the appropriate light field index. We also take advantage of the large texture memories and memory bandwidth available in today's graphics card to store and query hardware-compressed light fields.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

Keywords: reflections, light fields, pixel shader

*e-mail: {jingyi, jcyang}@graphics.lcs.mit.edu
†e-mail: mcmillan@cs.unc.edu

1. Introduction

The depiction of accurate reflections is still an earmark of ray tracing lying just beyond the reach of interactive rendering. Environment mapping is the most commonly used method for rendering approximate reflections interactively [Blinn and Newell 1976; Greene 1986]. An implicit assumption in environment mapping is that all scene elements are located infinitely far away from the reflecting surface. Equivalently, it models the reflector as a single point. When scene elements are relatively close to the reflectors, the results of environment mapping are inaccurate.

In place of a 2D environment map, we use 4D light fields to represent the surrounding scene. Each reflected ray is rendered by indexing into a precaptured environment light field. Light fields are image-based representations that describe the transmitted radiance along a sampling of rays without using the scene geometries. Its 4D nature enables us to render accurate reflections with motion parallax independent of the scene complexity. Our reflection mapping method is enabled by recent advances in programmable graphics hardware and the availability of large texture memories. A fragment shading program is used to dynamically select, sample, and interpolate approximate reflections upon any rendered surface. Correct parallax is observed as either the surface or viewpoint is changed. For convex objects, our reflection maps are a close approximation to a ray traced image.

2. Previous Work

Classic environment mapping [Blinn and Newell 1976; Greene 1986] has been widely used to approximate reflection in interactive computer graphics. Common variants include a sphere map [Blinn and Newell 1976] in which the entire reflection is described in a single parametrization, and a cube map [Greene 1986], which maps all reflection directions onto one of six cube faces. There are two fundamental shortcomings of the environment mapping method. First, the infinitely faraway environment assumption is not always valid. When it is violated, significant inaccuracies appear on the rendered reflectors, as shown in Figure 10. Second,

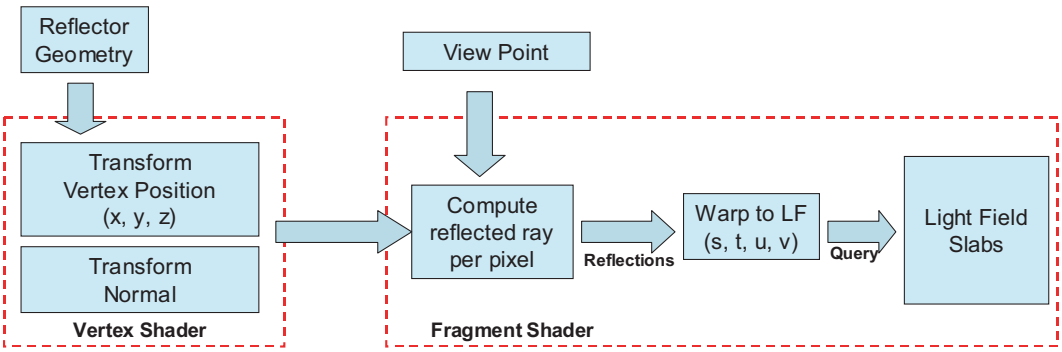

Figure 2: The pipeline of our method to render reflectors. First, the position and the normal at each vertex is transformed in the vertex shader. The rasterizer then interpolates both values for each pixel. Finally, the reflected ray is computed for each pixel and queried into the light field slabs in the pixel shader.

environment mapping loses crucial cues for 3D geometries, and in particular, cannot present motion parallax when the eye point or the reflector moves. Cabral et al [Cabral et al. 1999] suggest using multiple 2D environment maps to obtain view-dependency. Hakura et al [Hakura et al. 2001] proposed using location and view-dependent environment maps to capture local reflections. However, the transitions between different environment maps are difficult to control and may lead to discontinuity.

Ofek and Rappaport [Ofek and Rappoport 1998] developed an alternative method for computing interactive reflections called the explosion map. Their method warps the surrounding scene geometry such that it appears as a correct virtual image when drawn over top of the reflector. They also tessellate the reflectors and compute a reflection subdivision for each cell of the reflector to efficiently index these scene objects. Since their method transforms all scene geometry, and usually requires fine tessellation, it can be computational expensive.

Accurate reflection can also be achieved by ray tracing [Whitted 1980]. With advances in programmability at both the vertex and fragment (pixel) level [ATI 2001, Nvidia 2001], current graphics hardware can be modified to accommodate ray tracing [Purcell et al. 2002]. However, since both vertex and fragment shaders only support limited simple instructions, it is still difficult to accomplish sophisticated illumination effects at an interactive speed.

Alternative approaches to photo-realistic rendering are studied by researchers in image-based modelling and rendering. A light field [Levoy and Hanrahan 1996; Gortler et al. 1996] stores regularly sampled views looking at an object on a 2D sampling plane. These views form a 4D ray database. New views are synthesized by querying and blending existing rays. A sufficiently sampled light field can support view-dependent effects such as varying illumination, specular highlights, and even reflections. Lischinski and Rappoport [Lischinski and Rappoport 1998] used layered light fields (LLF) to render fuzzy reflections on glossy objects and layered depth images (LDI) were used to ray-trace sharp reflections. Hendrich et al [Heidrich et al. 1999] used two light fields to simulate accurate reflections. The first, like ours, models the radiance of the environment. The second maps viewing rays striking the object to outgoing reflection rays, thus allowing for inter-reflections. Compared to our method, theirs requires more storage, which they address with compression. Their method would also be enhanced by our light field slab approach described in Section 5. Most recently, Masselus et al [Masselus et al. 2003] use precaptured light fields to relight objects.

Alternatively, surface light fields [Wood et al. 2000] can also be used to render reflections. However, in practice, realistic rendering without aliasing artifacts requires extremely high sampling rate, and therefore, these methods are usually used to produce low frequency illumination effects rather than sharp reflections. Light field mapping[Chen et al. 2002] treats the outgoing radiance function like a light field to allow object motion and uses PCA to compress the ray database. Sloan [Sloan et al. 2002; Sloan et al. 2003] uses compressed precomputed radiance transfer functions to model inter-reflections between incident and exit rays and uses environment mapping to relight objects. Most of these methods do not handle motion parallax since they rely on environment mapping to model the scene's illumination and are limited to recovering low frequency illuminations due to compression.

Our method employs light fields in a different way. Instead of using them to represent the reflecting object, we use them to represent the environment. Unlike 2D environment maps, light fields are sufficient descriptions of the local environment. By using a two-plane parametrization (2PP), the environment light fields can be efficiently compressed and stored on today's graphics hardware as high dimensional texture maps.

3. Environment Light Field Map

Light fields are simple image-based representations using rays in place of geometry. Light fields capture all the necessary rays within a certain sub-space so that every possible view within a region can be synthesized. In a light field, a sequence of images are captured on a regularly sampled 2D plane, as shown in Figure 3. Rays are indexed as (s,t,u,v) where (s,t) and (u,v) are the intersection of each ray with the st plane $z = z_{st}$ and the uv plane $z = z_{uv}$. In practice, a light field is stored as a 2D array of images. Each pixel in the image can be indexed as an integer 4-tuple (s',t',u',v'), where (s',t') is the image index in the array and (u',v') is the pixel index in the texture. This st-uv-index representation, however, requires additional transformations to warp the canonical (s,t,u,v) representation. Fortunately, all images in a light field are regularly sampled, therefore, the transformation from the camera locations (s,t) into the camera indexes (s',t') can be easily computed using scaling and translation. Similarly, the pixel coordinate (u',v') can be calculated by computing the relative coordinate of (u,v) with respect to camera (s,t), as shown in Equation (1).

$$\begin{aligned} s' &= s \cdot s_1 + s_0 \\ t' &= t \cdot s_1 + t_0 \\ u' &= (u - s) \cdot s_2 + u_0 \\ v' &= (v - t) \cdot s_2 + v_0 \end{aligned} \tag{1}$$

where s_1 and s_2 are scaling factors and s_0, t_0, u_0, v_0 are the necessary translations to guarantee that both camera and pixel index start

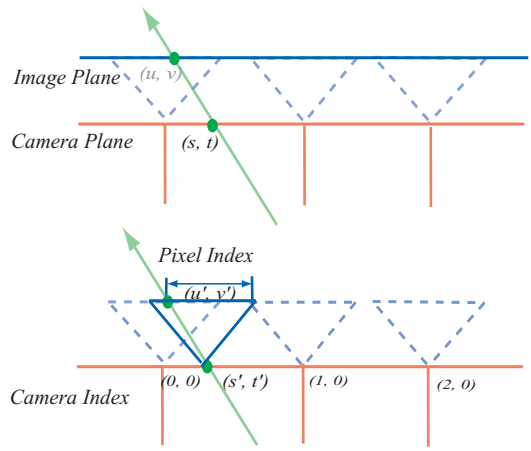

Figure 3: Light field parametrization. Top: a ray can be parameterized as (s,t,u,v) where (s,t) is the intersection with the camera plane and (u, v) is the intersection with the image plane. Bottom: a ray can also be represented by image-pixel coordinate as (s',t',u',v') where (s',t') is the image index, (u',v') is the pixel index.

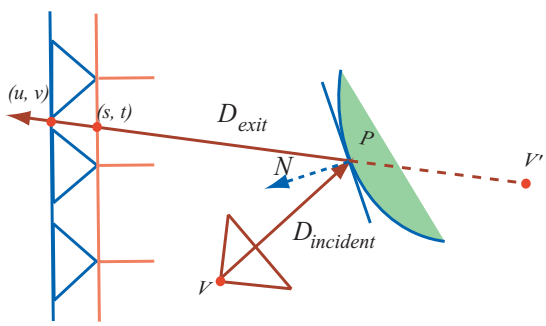

Figure 4: Reflected rays are computed and used to index the light fields. When the reflector is convex, the virtual viewpoint V' is further away from the light field.

from 0. Each pixel index (u',v') represents a common ray direction.

With a 4D *st*-*uv*-index light field parametrization, it is easy to test if a ray can be queried from the light field by checking if it lies within the 4D bounding volume [0, 0, 0, 0] to $[NUM-1, NUM-1, RES_x-1, RES_y-1]$, where NUM is the maximum image index, and RES_x and RES_y are the image resolution. If a ray can be queried, we can then estimate its radiance from its neighboring rays. Bilinear interpolation is often used for blending the sampled rays.

4. Reflection Mapping

The core of our rendering algorithm is to compute reflected rays and transform them into a light field ray parametrization. We take advantage of programmable vertex and fragment shaders to achieve efficient computations. For each vertex of the reflector, we store its position $\mathbf{P}(p_x,p_y,p_z)$ and normal $\mathbf{N}(n_x,n_y,n_z)$. The position and normal map are transformed by the vertex shader. The graphics hardware then automatically interpolates and assigns a position and normal at each pixel.

At the fragment level, for each pixel on the reflector, the reflected ray is computed as follows. Given a viewpoint $\mathbf{V}(v_x,v_y,v_z)$, the incident direction of the ray is $\mathbf{D}_{incident} = \mathbf{P} - \mathbf{V}$, and the exit direction $\mathbf{D}_{exit}$ can be computed as

$$\mathbf{D}_{exit} = 2\cdot\mathbf{N}(\mathbf{D}_{incident}\cdot\mathbf{N}) + \mathbf{D}_{incident} \tag{2}$$

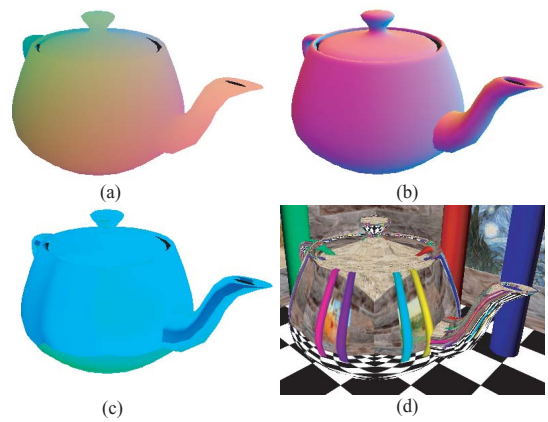

Figure 5: Reflection mapping on graphics hardware. (a) The (x,y,z) position image of the teapot. (b) The normal image of the teapot. (c) The computed (s,t,u,v) image of the teapot. (d) Rendered teapot in a light field environment.

```
float4 main(float4 vert :TEXCOORD0, float3 Normal :TEXCOORD1) : COLOR
{
        //----Calculate Reflected Ray

        float3 EyeToVerT = Vert.xyz - eyepos;
        float3 Reflect = reflect(EyeToVert,normalize(Normal));
        Reflect = Reflect/abs(Reflect.z);

        //----Calculate STUV

        float4 ST = (Vert.xyxy + (STOffset - vert.z) * Reflect.xyxy) + offset;
        float4 UV = Reflect.xyxy * UVScale + UVCenter;
        float4 outcolor = LFFetch(ST, UV, LFNum, Disparity);

        return outcolor;
}
```

Figure 6: Code fragment for computing reflected rays.

The reflected ray is then warped to the light field parametrization by intersecting it with the *st* and *uv* planes. This intersection can be computed as follows:

$$\begin{aligned} \mathbf{P}+\mathbf{D}_{exit}\cdot\lambda_1 &= (s,t,z_{st}) \\ \mathbf{P}+\mathbf{D}_{exit}\cdot\lambda_2 &= (u,v,z_{uv}) \end{aligned} \tag{3}$$

The (s,t,u,v) coordinate is obtained by solving Equation (3) as shown below:

$$\begin{aligned} (s,t,z_{st}) &= \mathbf{P}+\mathbf{D}_{exit}\cdot\frac{z_{st}-p_z}{D_{exit\,z}} \\ (u,v,z_{uv}) &= \mathbf{P}+\mathbf{D}_{exit}\cdot\frac{z_{uv}-p_z}{D_{exit\,z}} \end{aligned} \tag{4}$$

We warp the ray to *st*-*uv*-index coordinates by Equation (1). All of these computations are per-pixel based vector calculations and, hence, can be efficiently implemented on the fragment shader. Figure 5(a) and (b) shows the colored positions and normals for a sample reflector. Figure 5(c) shows the computed (s',t',u',v') map on the graphics card. The final reflectance image is shown in Figure 5(d). The fragment code is given in Figure 6.

5. Light Field Slabs

We surround the reflectors with light field slabs arranged as six faces of a 3D bounding box, as shown in Figure 7. In order to allow objects to move freely within this region, we need to guarantee that

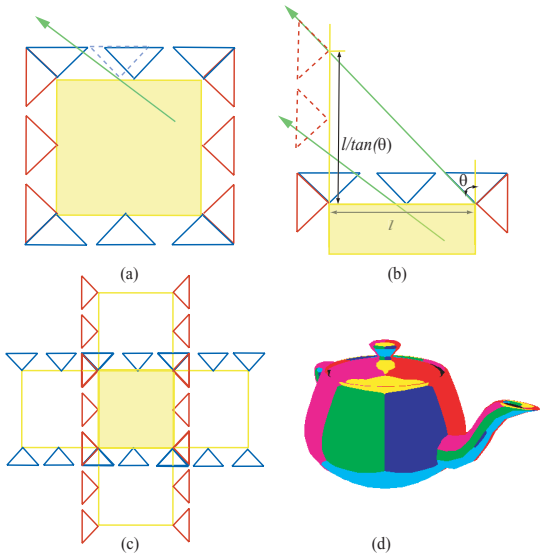

Figure 7: Light field slabs. (a) The green ray cannot be queried from any light field. (b) We can extend the light field slabs to capture missing rays. (c) Light field slabs can be used to guarantee all rays passing through the bounding volume can be queried. (d) A sample image rendered using light field slabs. Each color represents one of the six slab faces.

all rays going through the box are captured by one of the six light fields. It is insufficient to only build light fields on the six faces of the box, as in Figure 7(a). This arrangement does not guarantee that all rays passing through the box will be captured. For instance, the green ray in Figure 7(a) intersects the box but is not contained in any of the six light fields. One way to solve this problem is to extend the range of the light fields on each face.

For simplicity, we assume all the light field sampling cameras are of the same field-of-view 2θ. We first build up a bounding box of the reflector with length l, as shown in Figure 7(b). To guarantee no ray passing through the bounding box is missing, we extend the vertical slab with length $l_{extend} = \frac{l}{tan\theta}$. It is easy to verify that when $\theta \geq \frac{\pi}{4}$, such an extension can capture all rays passing through the bounding box. In practice, we prefer cameras with smaller field-of-view to increase the uv resolution. Therefore, we choose a very simple setup of $\theta = \frac{\pi}{4}$ with orthogonal slabs of length $3l$, as shown in Figure 7(c).

Under this construction, the eye point can move freely without constraints to observe accurate reflections. The reflector can also move freely within the cell without missing a ray. When it moves out of a cell, we need to further extend the light field slabs.

We could store the six light field slabs as individual texture maps. In order render each ray, it is necessary to select the appropriate slab for each query. However, the dynamic branching operation is not supported on the current generation of graphics cards on the GPU and all branches are executed. Therefore, the hardware will perform 24 bilinear texture fetches before it selects the proper color to render. We avoid this problem by combining all six slabs into a single texture volume. Then we adapt a commonly used technique in cubic environment mapping - we choose the light field slab using the largest absolute directional component of the reflected ray. Knowing the proper slab allows us to determine the corresponding slice in the texture volume.

```
float4 LFFetch( float4 ST, float4 UV, float LFnum, float disparity )
{
        //-----Calculate closest integer ST coords
        int4 vST = floor(ST)+int4(0,0,1,1);

        //-----Find interpolaion coeff
        float4 delta = frac(ST)-float4(0,0,1,1);

        //-----Claculate UV with disparity offset
        float4 UVD = UV - disparity * delta;

        //-----Calculate Texture coords
        float4 fcoord1 = vST%dimface + UVD;
        fcoord1 *= TexScale;

        //----Find LF slice

        int4 idx = vST/dimface;
        float4 fdepth = 1/128.0f * (idx.yyww * numgroup + idx.xzxz)
                        + 1/256.0f + LFnum * LFoffset;
        float4 color1 = tex3D(LightField, float3( fcoord1.x, fcoord1.y, fdepth.x  ));
        float4 color2 = tex3D(LightField, float3( fcoord1.z, fcoord1.y, fdepth.y  ));
        float4 color3 = tex3D(LightField, float3( fcoord1.x, fcoord1.w, fdepth.z  ));
        float4 color4 = tex3D(LightField, float3( fcoord1.z, fcoord1.w, fdepth.w  ));

        //----Interpolate
        float4 temp1 = lerp(color1, color2, delta.x);
        float4 temp2 = lerp(color3, color4, delta.x);

        return lerp(temp1, temp2, delta.y);
}
```

Figure 8: Partial fragment code for querying the rays.

6. Rendering

The final stage in rendering the reflection is to query the light field. Conventional light field rendering algorithms select 16 neighboring rays and blend them using quadralinear interpolation. In order to reduce storage overhead, the st camera space are often sparsely sampled. When undersampled, light field rendering usually exhibits aliasing artifacts. The aliasing artifacts can be significantly reduced by using simple geometric proxies like a focal plane [Isaksen et al. 2000; Chai et al. 2000].

Our system allows users to interactively adjust the disparity to achieve satisfactory rendering quality. Given a warped ray $r(s',t',u',v')$, we first round (s',t') to compute the four neighboring integer image indices.

$$\begin{aligned} (s_1,t_1) &= (\lfloor s' \rfloor, \lfloor t' \rfloor) \\ (s_2,t_2) &= (\lfloor s' \rfloor, \lceil t' \rceil) \\ (s_3,t_3) &= (\lceil s' \rceil, \lfloor t' \rfloor) \\ (s_4,t_4) &= (\lceil s' \rceil, \lceil t' \rceil) \end{aligned} \quad (5)$$

We then compute the pixel index at each of the four camera using user specified disparity.

$$(u_i,v_i) = (u',v') + disparity \cdot (s_i - s, t_i - t), i = 1,2,3,4 \quad (6)$$

For each (s_i,t_i) pair from Equation (5) we fetch and bilinearly interpolate the four neighboring pixels to (u_i,v_i). Therefore, it requires 16 texture fetches to render a single ray. To reduce the texture fetch overhead, we take advantage of bilinear texture interpolation capabilities on the graphics hardware. By storing a light field as textures, we only need perform four hardware bilinear texture fetches for each (s_i,t_i) pair and a final bilinear interpolation on the results of those queries.

Once the reflectors are rendered, we can render the background scene using the standard graphics pipeline. However, since we already have the necessary light fields around the reflector, we can also use the same light fields to render the background, so long as

Table 1: Comparison between Environment Map, Explosion Map, Light field map [Heidrich et al. 1999] and our method.

Method	Reflection Type[1]				Texture Storage	GPU Computation		Requires Environment Geometry	Preprocessing Cost[2]	
	D	N	T	S		Per Vertex	Per Pixel		Dynamic Reflector	Dynamic Environment
Environment Map	Yes	No	No	No	Low	Low	Low	No	None	Low
Explosion Map	Yes	Yes	Yes	No	Low	High	Low	Yes	High	Moderate
Light field Map	Yes	Yes	No	Yes	High[3]	Low	Moderate[3]	No	High	High
Our method	Yes	Yes	No	No	High	Low	Moderate	No	Low	High

[1] Reflection type includes distance reflection (D), near reflection (N), touching reflection (T), and self-reflection (S).
[2] Preprocessing overhead cost is incurred per rendered frame for dynamic reflector and dynamic scenes.
[3] Because Light field map stores a second light field to map viewing rays to outgoing reflection rays to support self-reflection, it requires additional texture storage and per pixel (ray) processing overhead compared to our method.

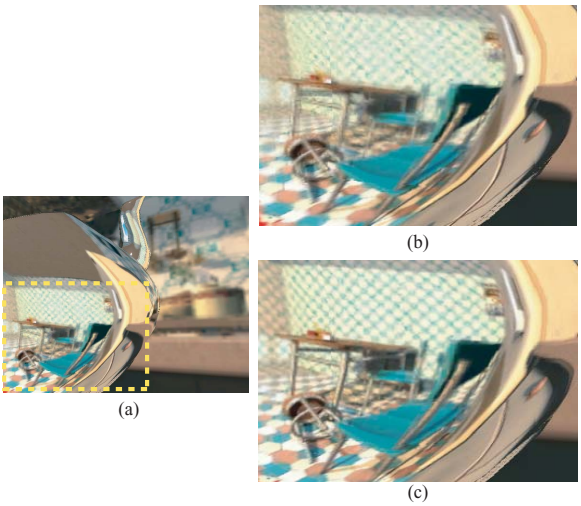

Figure 9: Aliasing artifacts can be reduced using a focal plane. (a) A sample image rendered using the kitchen light field. (b) The focal plane is on the front chair and we observe aliasing artifacts on the background table.(c) Aliasing artifacts can be reduced using the optimal focal plane.

the viewpoint does not move out of the bounding slabs. In fact, it is easy to modify the fragment program to render the background by replacing the reflected direction with the viewing direction at each pixel and taking the viewpoint as the origin of the ray. We can then use the same rendering pipeline to warp and query the ray from the light fields.

In Figure 8, we compare the rendering quality of our algorithm using different disparity values. The aliasing artifacts on the reflectors are significantly reduced when the optimal disparity is used [Chai et al. 2000]. In general, aliasing artifacts on convex reflectors are much less significant than the background, if both are rendered using light fields. This is because the "virtual viewpoint", shown as V' in Figure 4, is farther away from the light fields.

7. Results

We have implemented our algorithm using DirectX9 and HLSL on an ATI Radeon 9700 Pro with 128MB texture memory. We pre-captured the light fields for different scenes and scaled the reflected objects to fit them into the bounds of the light field slabs.

The museum environment with colored columns is prerendered using DirectX as six slabs of 32x32x128x128 light fields. These light field slabs are stored as a single 64MB volume texture with DXT1 compression. These volume textures are of dimension 1024x1024x128 with each light field slab occupying a space of 1024x1024x16. The total volume texture size is actually the size of 8 light fields slabs due to power-of-two texture dimension requirements. The environment is rendered as regular geometries. In Figure 10, we compare our method with environment mapping. Since environment mapping assumes that the scene geometry is located at infinity, its rendered columns are smaller than they should actually appear. In addition, our method renders accurate motion parallax while environment mapping does not. To show this, we rotate the viewpoint around the sphere. The change in occlusions between the foreground columns and the background paintings can be clearly observed. There is no such motion parallax if we use environment mapping, as shown in the bottom row of Figure 10. Better comparisons can be seen in the supplementary video.

The kitchen scene is prerendered using POVRay, where six slabs of 32x32x128x128 light fields are captured. Both the reflector and the background are rendered using light fields. Because the light fields are undersampled, we observe aliasing artifacts on the background. These artifacts are reduced when using the optimal disparity, as shown in Figure 8 and the video. We also observe less aliasing on the reflectors, because the resolution of the reflector is lower and the virtual eye point is farther away from the light fields.

Currently, due to instruction limitations in the shaders, we use a two pass implementation. In the first pass, we render environment geometry into the backbuffer and calculate *stuv* coordinates for the reflector and write them into an off screen buffer. In the second pass we re-render the reflector by using the previously computed *stuv* values to query the light field from the volume texture. This implementation, however, will not be necessary in the next generation of graphics hardware.

The rendering time of our algorithm is independent of the reflector complexity since it does per-pixel based computation on GPU. The major cost of our algorithm comes from two parts: the fragment shader program which computes and warps the reflected ray and four bilinear texture fetches into the light field per pixel. In Table 2, we compare the cost of these two components for different models and scenes.

8. Conclusions

We have presented a novel algorithm to efficiently render accurate reflections on programmable graphics hardware. Our algorithm overcomes many of the problems inherent to environment mapping by six 4D light field slabs to represent the surrounding scene. However, our approach does not solve the problem of self-reflection, inter-reflection between the objects, or dynamic environments.

In Table 1, we compare our method with conventional environment mapping and explosion maps. While explosion maps also render accurate reflections, it is expensive for complex scenes, and, hence, is not scalable. Our method can both render accurate reflections and

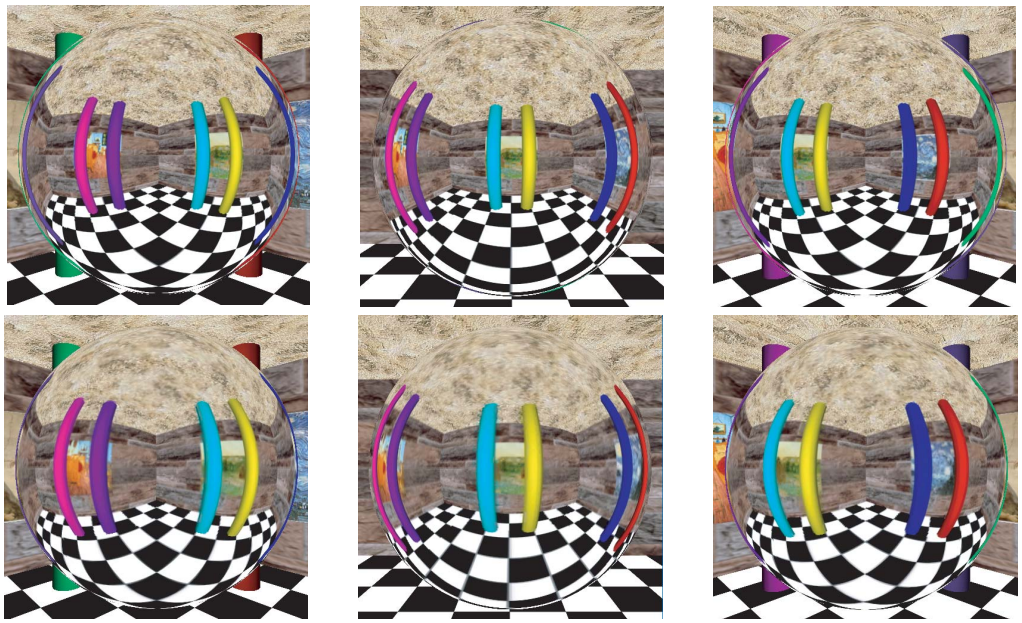

Figure 10: When the viewpoint changes, parallax is visible using our method. Top row: three views rendered using environment mapping. Bottom row: three views rendered using our method. Notice the front columns exhibit occlusion variations around the background paintings but remain static with environment mapping.

Model	Vertices	Faces	RM fps	EM fps
Teapot	1178	2256	80	400
Sphere	9902	19800	90	340
Hippo	18053	51584	70	260
Skull	31076	60339	70	200

Table 1: Comparing Reflection Map (RM) and Environment Map (EM) on ATI Radeon 9700 Pro. All models are rendered at 512x512 resolution using six 32x32x128x128 light fields slabs.

is scalable. However, our method requires large memory storage for light fields while the other two do not. With upcoming generations of graphics hardware supporting larger texture memories, higher memory bandwidth, and more efficient high dimensional texture maps, our method has the potential to be used in both realistic rendering and computer games as an alternative to environment maps.

Acknowledgement

We would like to thank Jaime Vives Piqueres for the PovRay kitchen model.

References

BLINN, J. F., AND NEWELL, M. E. 1976. Texture and reflection in computer generated images. *Commun. ACM 19*, 10, 542–547.

CABRAL, B., OLANO, M., AND NEMEC, P. 1999. Reflection space image based rendering. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, 613–620.

CHAI, J.-X., CHAN, S.-C., SHUM, H.-Y., AND TONG, X. 2000. Plenoptic sampling. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 307–318.

CHEN, W.-C., BOUGUET, J.-Y., CHU, M. H., AND GRZESZCZUK, R. 2002. Light field mapping: efficient representation and hardware rendering of surface light fields. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, 447–456.

GORTLER, S. J., GRZESZCZUK, R., SZELISKI, R., AND COHEN, M. F. 1996. The lumigraph. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 43–54.

GREENE, N. 1986. Environment mapping and other applications of world projection. *IEEE Comput Graphics Appl*.

HAKURA, Z. S., SNYDER, J. M., AND LENGYEL, J. E. 2001. Parameterized environment maps. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, ACM Press, 203–208.

HEIDRICH, W., LENSCH, H., COHEN, M., AND SEIDEL, H.-P. 1999. Light field techniques for reflections and refractions. In *Eurographics Rendering Workshop*.

ISAKSEN, A., MCMILLAN, L., AND GORTLER, S. 2000. Dynamically reparametrized light fields. In *Proc. ACM SIGGRAPH '00*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 297–306.

LEVOY, M., AND HANRAHAN, P. 1996. Light field rendering. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 31–42.

LISCHINSKI, D., AND RAPPOPORT, A. 1998. Image-based rendering for non-diffuse synthetic scenes. In *Eurographics Rendering Workshop*, 301–314.

MASSELUS, V., PEERS, P., DUTR, P., AND WILLEMS, Y. D. 2003. Relighting with 4d incident light fields. In *Proc. ACM SIGGRAPH '03*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 613–620.

OFEK, E., AND RAPPOPORT, A. 1998. Interactive reflections on curved objects. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, 333–342.

PURCELL, T. J., BUCK, I., MARK, W. R., AND HANRAHAN, P. 2002. Ray tracing on programmable graphics hardware. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, 703–712.

SLOAN, P.-P., KAUTZ, J., AND SNYDER, J. 2002. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, 527–536.

SLOAN, P.-P., HALL, J., HART, J., AND SNYDER, J. 2003. Clustered principal components for precomputed radiance transfer. *ACM Trans. Graph. 22*, 3, 382–391.

WHITTED, T. 1980. An improved illumination model for shaded display. *Commun. ACM 23*, 6, 343–349.

WOOD, D. N., AZUMA, D. I., ALDINGER, K., CURLESS, B., DUCHAMP, T., SALESIN, D. H., AND STUETZLE, W. 2000. Surface light fields for 3d photography. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 287–296.

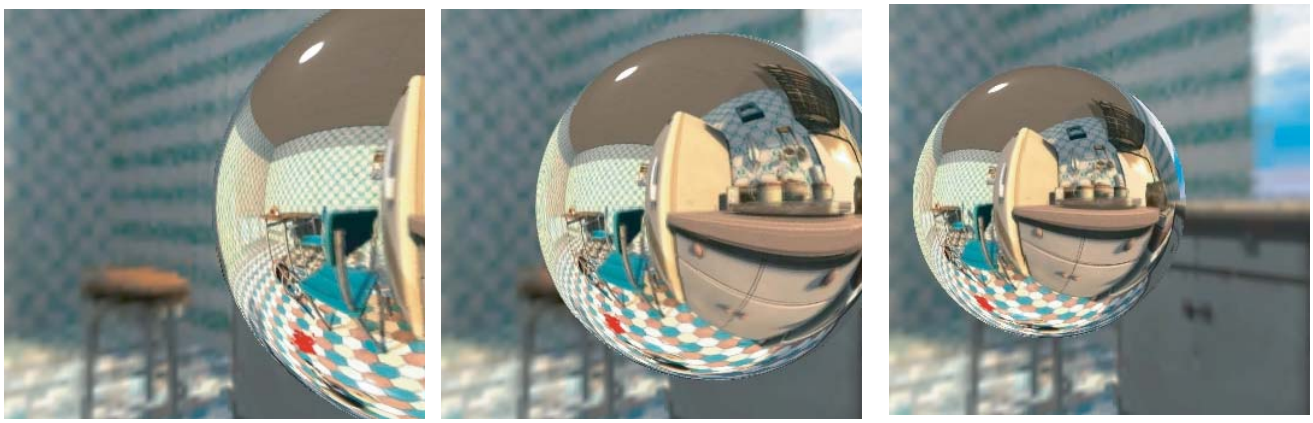

Figure 1: Reflections on a moving sphere rendered using our method. Notice the change of parallax: the table is partially occluded by the green chair (left) and turns gradually visible (middle and right).

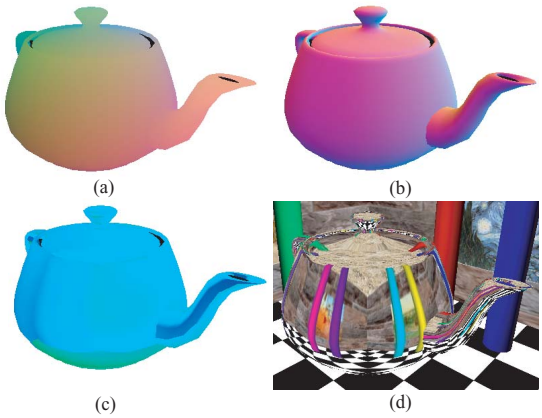

Figure 2: Reflection mapping on graphics hardware. (a) The (x,y,z) position image of the teapot. (b) The normal image of the teapot. (c) The computed (s,t,u,v) image of the teapot. (d) Rendered teapot in a light field environment.

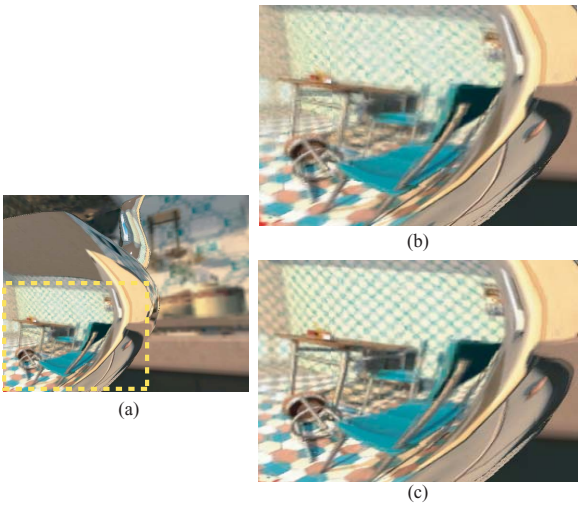

Figure 3: Aliasing artifacts can be reduced using a focal plane. (a) A sample image rendered using the kitchen light field. (b) The focal plane is on the front chair and we observe aliasing artifacts on the background table.(c) Aliasing artifacts can be reduced using the optimal focal plane.

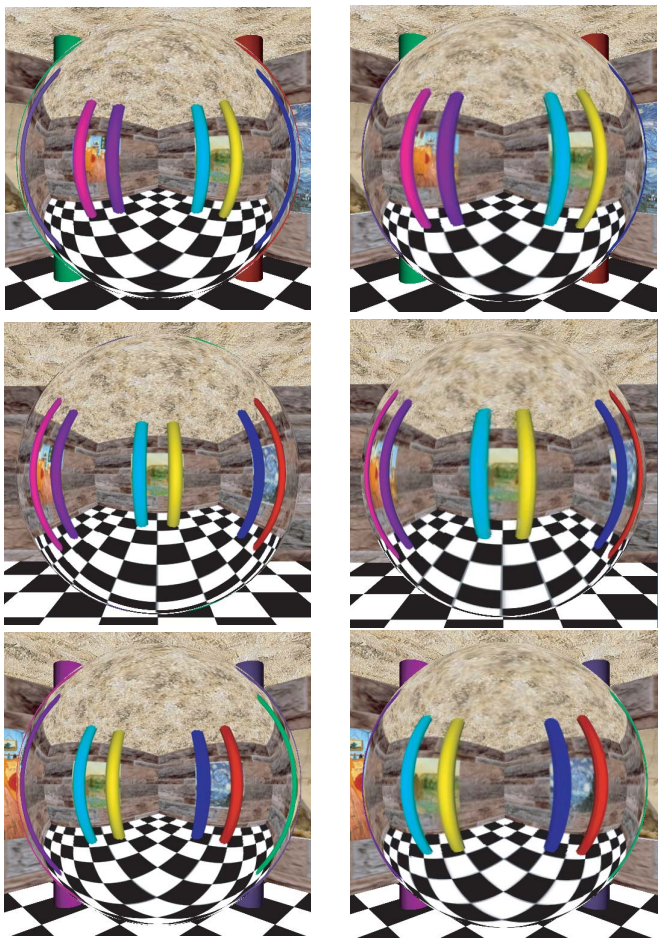

Figure 4: When the viewpoint changes, parallax is visible using our method. Left column: three views rendered using environment mapping. Right column: three views rendered using our method. Notice the front columns exhibit occlusion variations around the background paintings but remain static with environment mapping.