Realistic Reflections and Refractions
on Graphics Hardware With Hybrid Rendering
and Layered Environment Maps

Ziyad S. Hakura[*], John M. Snyder[**]
[*]Stanford University, [**]Microsoft Research

Abstract.

We introduce hybrid rendering, a scheme that dynamically ray traces the local geometry of reflective and refractive objects, but approximates more distant geometry by hardware-supported environment maps (EMs). To limit computation, we use a greedy ray path shading model that prunes the binary ray tree generated by refractive objects to form just two ray paths. We also restrict ray queries to triangle vertices, but perform adaptive tessellation to shoot additional rays where neighboring ray paths differ sufficiently. By using layered, parameterized EMs that are inferred over a set of viewpoint samples to match ray traced imagery, we accurately handle parallax and view-dependent shading in the environment. We increase robustness of EMs by inferring them simultaneously across multiple viewpoints and including environmental geometry that is occluded from the viewpoint sample but is revealed in nearby viewpoints. We demonstrate realistic shiny and glass objects with a user-controlled viewpoint.

1   Introduction

Z-buffer hardware is well-suited for rendering texture-mapped 3D geometry but inadequate for rendering reflective and refractive objects. It rasterizes geometry with respect to a constrained ray set – rays emanating from a point and passing through a uniformly parameterized rectangle in 3D. Reflective and refractive objects create a more general lens system mapping each incoming ray into a number of outgoing rays, according to a complex, spatially-varying set of multiple ray "bounces". Environment maps (EMs) [2] extend hardware to simulate reflections from an infinitely distant environment, but ignore all but the first bounce and so omit self-reflections and refractions. On the other hand, ray tracing [22] generates these effects but is unaccelerated and incoherently accesses a large scene database. Nevertheless, modern CPUs are powerful enough to perform limited ray tracing during real-time rendering.

We combine the benefits of both systems by tracing ray paths through reflective/refractive objects to compute how the local geometry maps incoming rays to outgoing. To encapsulate more distant geometry, we use the outgoing rays as indices into a previously-inferred EM per object, allowing efficient access and resampling by graphics hardware. We call this *hybrid rendering*.

We begin by segmenting reflective/refractive scene geometry into a set of *local lens objects*. Typically each glass or shiny object forms a single local lens object, but multiple objects can be combined if they are close or one contains or surrounds another. Because rays are traced through a system of only one or a few objects, the working set is smaller and memory access more coherent than with traditional ray tracing. To make the approach practical, we also initially limit the number of ray casts to polygon vertices, adaptively shooting additional rays only where necessary. We also prune the ray tree (binary for refractive objects where an incoming ray striking an interface generates child reflection and refraction rays) ignoring all but a few ray paths that still approximate the full ray traced rendering well (see Figure 2).

Each local lens object's EM is different from traditional ones: it is *inferred*, *layered*, and *parameterized*. Inferred means our EMs are computed as a least-squares best match to a pre-computed, ray traced image at a viewpoint *when applied as an EM in a rendering by the target graphics system*. The alternative of sampling a spherical image of incident radiance at a point (typically the lens
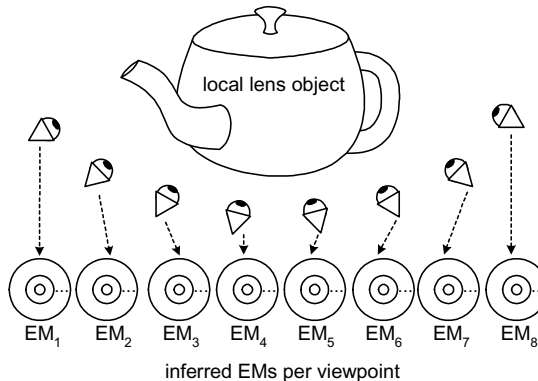

Figure 1: A PEM is a sequence of EMs recorded over a set of viewpoints for each local lens object. Each EM_i consists of layered shells at various distances from the local lens object's center.

object's center) ignores view-dependent shading in the environment and produces a less accurate match at the viewpoint. Layered means that we use multiple environmental shells to better approximate the environment. Parameterized means we compute an EM per viewpoint over a set of viewpoints (Figure 1) to provide view-dependent shading on imaged objects and reduce the number of layers needed for accurate parallax. Our examples use a 1D viewpoint subspace that obtains accurate results over that subspace and plausible results for any viewpoint.

Our contributions include the hybrid rendering shading model and its combination of dynamic ray tracing with hardware-supported EMs. We improve on the parameterized EMs (PEMs) described in [8] by generalizing to more than two layers and providing tools for determining their placement. In addition, we handle self-reflections by ray tracing the local geometry rather than representing it as an EM, and so achieve good results with as much as ten times sparser sampling of EMs compared with [8]. The method of [8] also has the problem that its inferred EMs represent only the part of the environment imaged at one viewpoint, resulting in disocclusions from nearby viewpoints. We ensure completeness in our layered EMs by matching to a layered image that includes occluded surfaces, as well as simultaneously over multiple nearby viewpoints or direct images of the environment. Results show our method supports rendering of realistic reflective and refractive objects on current graphics hardware.

2 Previous Work

2.1 Reflections

Several efforts have been made to exploit fast graphics hardware to produce realistic reflections. Diefenbach [5] simulates planar reflections by mirroring the viewpoint about the reflection plane. Ofek and Rappoport [20] extend the idea to curved objects, requiring careful decomposition of objects with mixed convexity or saddle regions.

Image-based rendering (IBR) methods [6][14] can tabulate view-dependent shading effects like reflections. Surface light fields [19][23] are a variant that parameterize the radiance field over surfaces rather than views. These methods visit an irregular scattering of samples over the entire 4D light field to reconstruct a particular view, and lack hardware acceleration. They also require very high sampling densities to reconstruct specular objects, a problem we address by ray tracing and tabulating EMs over simpler 1D viewpoint subspaces.

Cabral et al. [3] store a collection of view-dependent EMs where each EM pre-integrates a BRDF with a lighting environment. The lighting environments are generated using standard techniques and thus ignore local reflections and refractions.

Lischinski and Rappoport [16] ray trace through a collection of view-dependent LDIs for glossy objects with fuzzy reflections, and three view-independent LDIs representing the diffuse environment. Bastos et al. [1] reproject LDIs into a reflected view for rendering primarily planar glossy surfaces in architectural walkthroughs. Our approach succeeds with simpler and hardware-supported EMs rather than LDIs, resorting to ray tracing only for the local "lens" geometry where it is most necessary.

Hakura et al. [8] introduce parameterized and layered environment maps to simulate local reflections including self-reflections. We make use of these ideas for the more distant geometry, but use local ray tracing to more accurately simulate self-reflections and extend to refractive objects. We also improve their EM inference to handle disocclusions and produce more accurate parallax.

2.2   Refractions

Heidrich et al. [10] attempt to handle refractive as well as reflective objects using a light field to map incoming view rays into outgoing reflected or refracted rays. These outgoing rays then index either a static environment map, which ignores local effects further from the object, or another light field, which is more accurate but also more costly. Though our hybrid rendering similarly partitions local and distant geometry, we obtain sharper, more accurate reflections and refractions using local ray tracing rather than light field remapping. We also exploit the texture-mapping capability of graphics hardware using layered EMs for the more distant environment rather than light fields.

Chuang, et. al. [4], and Zongker et. al. [24] capture the remapping of incident rays for real and synthetic reflective/refractive objects, but only for a fixed view. Kay and Greenberg [12] simulate refractive objects with a simple, local model restricted to surfaces of uniform thickness.

Adaptive sampling has been used in ray tracing since it was first described [22]. To decouple local and distant geometry, our adaptation is based on ray path, not color or radiance, differences. Kajiya's idea of ray paths rather than trees [13] forms the basis of our local model. Finally, the caching and ray intersection reordering of Pharr et. al. [21] is another, completely software-based approach for memory-coherent access to the scene database.

3   Shading Model And Overview

Ray tracing simulates refractive objects by generating child reflective and refractive rays whenever a ray strikes a surface interface. The relative contribution of these children is governed by the Fresnel coefficients [9], denoted $\hat{F}_R$ and $\hat{F}_T$ for reflected and transmitted (refracted) rays, respectively. These coefficients depend on the ray's angle of incidence with respect to the surface normal and the indices of refraction of the two media at the interface. Purely reflective objects are simpler, generating a single child reflected ray modulated by $\hat{F}_R$ but can be considered a special case with $\hat{F}_T = 0$.

Rather than generating a full binary tree for refractive objects which can easily extend to thousands of ray queries, we use a two-term model with greedy ray path propagation. When a ray from the viewpoint first strikes the refractive object, we consider two paths: one beginning with an initial reflection and the other with an initial refraction. These paths are then propagated until they exit the local object by selecting the child ray having the greatest Fresnel coefficient. The result is two terms whose sum approximates the full binary tree. Reflective objects require
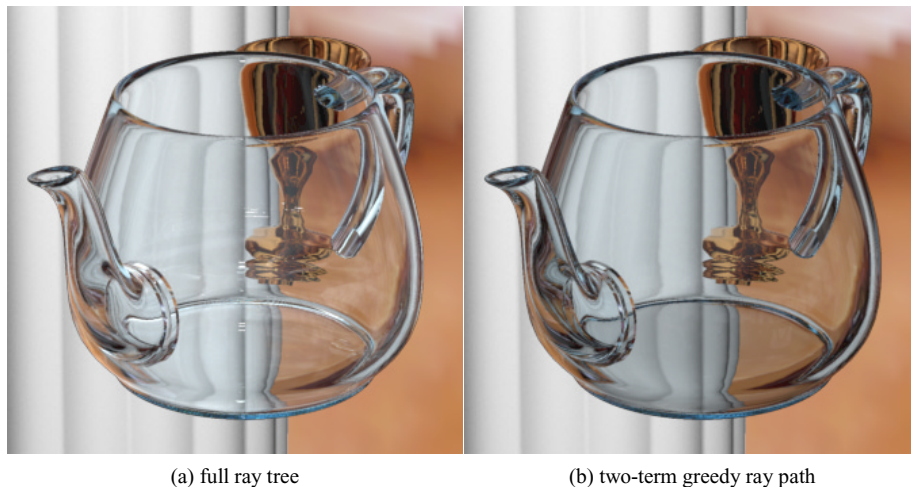
(a) full ray tree   (b) two-term greedy ray path

Figure 2: Shading Models. The full ray tree (a) requires 5 times more ray queries than our greedy ray path model (b).

only a single term but use the same ray propagation strategy in case the local system contains other refractive objects. Figure 2 compares the quality of this approach with a full ray tree simulation.

Our model also includes a simple transparency attenuation factor, G, which modulates the ray color by a constant for each color channel raised to a power depending on the thickness of glass traversed between interfaces [12]. The resulting model is

$$T\,F_T\,G_T + R\,F_R\,G_R$$

where respectively for the refracted and reflected ray paths: T and R are radiances along exit ray, F_T and F_R multiply the Fresnel coefficients along the path, and G_T and G_R multiply the transparency attenuation along the path (see Figure 3).

As a preprocess, for each viewpoint sample, we use a modified ray tracer to compute the two terms of this model as separate images. We then infer an EM that produces the best least-squares match to both terms simultaneously. The result is a viewpoint-dependent sequence of inferred EMs. Each viewpoint's EM is layered by segmenting the environmental geometry into a series of spherical shells.

At run-time, we dynamically trace rays through each vertex of the local geometry according to our ray path model to see where they exit the local system and intersect the EM shells. We select the EM corresponding to the viewpoint closest to the current view or blend between the
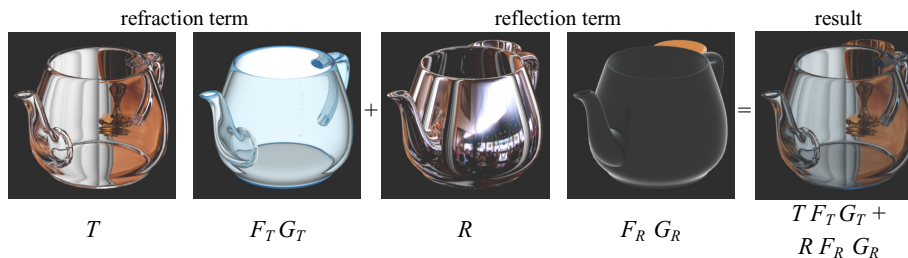
refraction term   reflection term   result

T $F_T\,G_T$ R $F_R\,G_R$ $T\,F_T\,G_T + R\,F_R\,G_R$

Figure 3: Two-Term Modulated Shading.

two closest. A separate pass is used for each term and then summed in the framebuffer. The Fresnel and transparency attenuation factors are accumulated on-the-fly as the path is traced, and produce per-vertex terms that are interpolated over each triangle to modulate the value retrieved from the EM. A better result can be achieved using 1D textures that tabulate highly nonlinear functions such as exponentiation [11].

While our current system neglects it, a diffuse component can be handled as an additional view-independent diffuse texture per object that is summed into the result. Such textures can be inferred to match ray traced imagery using the technique of [7].

4 Layered EMs

A local lens object is associated with a layered EM per viewpoint in which each layer consists of a simple, textured surface. We use a nested series of spherical shells sharing a common origin for the geometry because spheres are easy to index and visibility sort. Other kinds of simple geometry, such as finite cubes, cylinders, and ellipsoids, may be substituted in cases where they more accurately match the actual geometry of the environment. A layer's texture is a 4-channel image with transparency, so that we can see through inner shells to outer ones where the inner geometry is absent. At run-time, we perform hybrid rendering to compute outgoing rays and where they intersect the layered EMs. We use the "over" blending mode to composite the layers L_i in order of increasing distance before modulating by the Fresnel and transparency attenuation terms, F and G, via

$$(L_1 \text{ over } L_2 \text{ over } \ldots \text{ over } L_n)\ F\ G$$

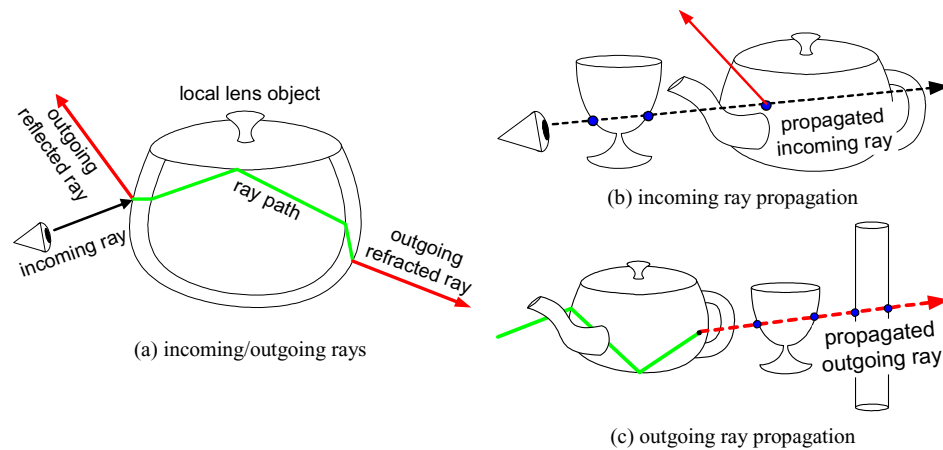for each term in the two-term shading model.

Figure 4: Incoming/Outgoing Rays and Ray Propagation.

4.1 Compiling the Outgoing Rays

To build layered EMs, the ray tracer compiles a list of intersections which record the eventual *outgoing rays* exiting the local lens object and where they hit the more distant environment. These intersections are generated from *incoming rays* originating from a supersampled image at a particular viewpoint including both terms of the shading model (reflection and refraction), each of which generates different outgoing rays (Figure 4a). Between incoming and outgoing rays, the ray paths are propagated using the model of Section 3. The intersection record also

stores the image position of the incoming ray and color of the environment at the outgoing ray's intersection.

To avoid disocclusions in the environment as the view changes, we modified the ray tracer to continue rays through objects to reach occluded geometry. For each outgoing ray, we record all front-facing intersections with environmental geometry along the ray, not just the first (Figure 4c). Once the layer partitions are computed (Section 4.2), we then discard all but the first intersection of each outgoing ray with that layer. This allows reconstruction of parts of the environment in a distant layer that are occluded by a closer one. We also continue incoming rays in a similar fashion (Figure 4b) so that occluded parts of the lens object still generate intersection records. For example, we continue incoming rays through a cup to reach a glass teapot it occludes.

4.2   Building Layered EM Geometry

To speed run-time performance, we seek a minimum number of layers. But to approximate the environmental geometry well, we must use enough shells and put them in the right place.

We use the LBG algorithm [15] developed for compression to build vector quantization codebooks. The desired number of layers is given as input and the cluster origin is computed as the centroid of the local object. The LBG algorithm is run over the list of intersections, clustering based on distance to this origin. This algorithm begins with an initial, random set of cluster distances and assigns each intersection to its closest cluster. It then recomputes the average distance in each cluster, and iterates the assignment of intersection to closest cluster. Iteration terminates when no intersections are reassigned.

When partitioning geometry into layers, parts of coherent objects should not be assigned to different layers. This can cause incorrect tears in the object's reflected or refracted image. Our solution assigns whole objects only to the single cluster having the most intersection records with that object. The clustering algorithm should also be coherent across the parameterized viewpoints. This is accomplished by clustering with respect to all viewpoints simultaneously. Figure 7 shows clustering results on an example scene in which our algorithm automatically segments a glass teapot's environment into three layers.

Layer shells are placed at the average distance of intersections in the cluster, where "continued ray" intersections are represented only by their frontmost cluster member.

Layer Quads Often a spherical shell is only sparsely occupied. In that case, to conserve texture memory we use a simple quadrilateral impostor for this geometry rather than a spherical one. To define the impostor quadrilateral, we can find the least-squares fit of a plane to the layer's intersection points. A simpler, but less optimal, scheme is to find the centroid of the layer's intersection points and define the normal of the plane as the direction from the lens object center to the centroid. The plane's extent is determined from a rectangular bounding box around points computed by intersecting the outgoing rays associated with the layer's intersection records with the impostor plane. One complication is that during run-time, outgoing rays may travel away from the quad's plane, failing to intersect it. This results in an undefined texture access location. We enforce intersection for such rays by subtracting the component of the ray direction normal to the impostor plane, keeping the tangential component but scaling it to be very far from the impostor center.

Texture maps for spherical shells or quads are computed using the same method, described below.

4.3 Inferring Layered EM Texture Maps

As in [7][8], we base our EM inference on the observation that a texel contributes to zero or more display pixels. Neglecting hardware quantization effects, a texel that is twice as bright contributes twice as much. Hardware rendering can thus be modeled as a linear system, called the *rendering matrix*, mapping texels to display pixels. To find the rendering matrix, we perform test renderings that isolate each texel's display contribution. Given the rendering matrix, A, we then find the least-squares best EM, x, which when applied matches the ray tracer's segmented incident radiance layer, b. This results in the linear system $Ax=b$, which we solve using conjugate gradient. Details about the rendering matrix inference and linear system solution are found in [7].

There are two advantages of this inference method over a simple projection of the environment onto a series of shells. By matching a view-dependent ray traced image, it reproduces view-dependent shading on reflected or refracted objects, like a reflective cup seen through a glass teapot. It also adjusts the EM to account for the geometric error of approximating environmental objects as simpler geometry, such as a spherical shell. The result is better fidelity at the ray traced viewpoint samples.

We infer each layer of the layered EM independently, but simultaneously over both terms of the shading model (Figure 7). After performing the layer cluster algorithm on samples from a supersampled image, each layer's samples are recombined into a single image and filtered to display resolution to form two images b_R and b_T, corresponding to the two terms of the shading model. Only a single image is needed for reflective objects. These images have four channels – the alpha channel encodes the fraction of supersampled rays through a given pixel whose outgoing ray hit environmental geometry in that layer, while the rgb channels store the color of the environment intersected by those outgoing rays. We infer the two rendering matrices, A_R and A_T, corresponding respectively to hybrid rendering (Section 5) with an initial reflection or refraction. We then find the least-squares solution to

$$A_R\, x = b_R$$
$$A_T\, x = b_T \qquad\qquad (1)$$

to produce a single EM for the layer, x, matching both terms. Figure 7 shows an example of these b terms and resulting inferred EM, x, for each of three layers.

It is possible for the ray tracer to generate widely diverging rays when sampling a single output pixel, causing noise in the environment map solution. We therefore modified the ray tracer to generate a confidence image. The per-pixel confidence value is computed as a function of the maximum angle between the directions of all ray pairs contributing to the particular pixel. More precisely, we use the formula

$$1 - \min(\theta_m^2, \theta_c^2)\big/\theta_c^2$$

where θ_m is the maximum angle between ray pairs and $\theta_c = 5°$ is an angular threshold. We multiply the confidence image with both sides of equation (1) prior to solving for x.

To conserve texture memory, it is beneficial to share more distant EM layers between local lens objects. To do this, we can add more equations to the linear system (1) corresponding to multiple lens objects and simultaneously solving for a single EM x.

As observed in [8], choosing the proper EM resolution is important to preserve frequency content in the imaged environment. A very conservative approach generates test renderings to determine the most detailed EM MIPMAP level actually accessed by the graphics system. Texture memory bandwidth and capacity limitations may dictate the use of lower resolutions.

Simultaneous Inference Over Multiple Viewpoints A difficulty with this inference technique is that the lens objects can fail to image all of its environment. For example, a flat mirror does not image geometry behind it and a specular fragment images only a small part of its environment from a particular viewpoint. These missing portions can appear in a view near but not exactly at the pre-rendered viewpoint. We solve this problem by inferring EMs that simultaneously match ray traced images from multiple views. The views can be selected as a uniform sampling of a desired viewspace centered at the viewpoint sample.

To compute simultaneous viewpoint inference, outgoing rays are compiled for each of these multiple viewpoints. A single EM layer, x, is then inferred as a least-squares simultaneous match at all viewpoints, using the system of equations (1) for each viewpoint. Although this blurs view-dependent shading in the environment, good results are achieved if the set of viewpoints matched are sufficiently close.

An alternative method to fill in the missing portions in the environment map is to infer it using extra rays in addition to the ones that exit the lens object. These additional rays can be taken from the object center as in traditional environment maps. They can also be taken from the viewpoint, but looking directly at the environment (i.e., without the lens object), to approximate view-dependent shading in the environment. Direct images from the lens object center tend to work better for reflective objects while direct images from the viewpoint are better suited for refractive (transparent) objects.

We use the confidence-weighted, least-squares solution method in (1), but solve simultaneously across images of the lens object from the viewpoint as before (*lens object images*), combined with direct images of the environment without the lens object (*direct images*). In these direct images, per-pixel confidence is computed as a function of the likelihood that the pixel represents a missing portion of the environment. We compute this via distance of the direct image ray's intersection with its closest point in the intersection records of the lens object images. The advantage of this scheme is that it fills in the missing portions of the environment with a fixed, small number of extra images, regardless of the size of the viewspace around a viewpoint sample.

5   Hybrid Rendering

Adaptive Tessellation Performing ray tracing only at lens object vertices can miss important ray path changes occurring between samples, producing very different outgoing rays even at different vertices of the same triangle. Our solution is to perform adaptive tessellation on the lens object based on two criteria: the ray path "topology" and a threshold distance between outgoing rays. Using topology, ray paths are considered different if their path lengths are different, or the maximum coefficient at an interface changes between reflection and refraction. Using outgoing ray distance, they are different if the angle between their directions is more than 3°. Figure 5 illustrates adaptive tessellation using an example

Figure 5: Adaptive Tessellation.

image shaded darker with increasing subdivision level. Places where rays go through more interfaces or where the surface is highly curved require more sampling.
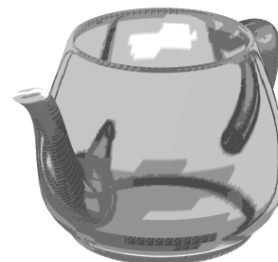
When the ray paths at a triangle's vertices are too different, the triangle is subdivided at the midpoints of each of its edges in a 1-to-4 subdivision, and the metric is recursively applied. The process is continued until the three ray paths are no longer different or the triangle's screen-projected edge lengths are less than a threshold, τ. We also allow 1-to-3 and 1-to-2

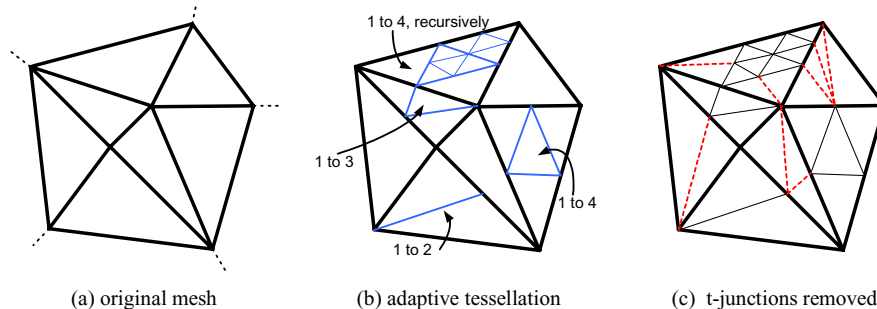|		
(a) original mesh	(b) adaptive tessellation	(c) t-junctions removed

Figure 6: The original mesh, (a), is adaptively subdivided when ray paths at the vertices are sufficiently different, (b). The resulting t-junctions are removed by additional tessellation of adjacent triangles, (c), illustrated with dotted lines, to form a triangular tessellation.

subdivision in cases where some of the triangle edges are already small enough (see Figure 6). We adapt the tessellation simultaneously for both terms of the shading model, subdividing a triangle if either ray path is considered different. We ignore differences in subdivision between neighboring triangles, fixing the resulting "t-junction" tessellation as a postprocess. A hash table on edges (vertex pairs) returns the edge midpoint if it has already been computed from a more highly subdivided neighboring triangle. Recursive querying yields all vertices along the edge.

Given all the boundary vertices, it is simple to compute a triangular tessellation that avoids t-junctions. The hash table is also used to quickly determine whether a triangle vertex ray has already been computed, thus avoiding redundant ray queries.

To avoid unnecessary ray tracing and tessellation in occluded regions, we compute whether each vertex is directly visible from the viewpoint using a ray query. If all three vertices of a triangle are occluded, we do not subdivide the triangle further, but still compute correct texture coordinates for its vertices via ray tracing in case some part of its interior is visible. Another optimization is to avoid ray tracing at vertices whose triangles are all backfacing. Using a pass through the faces, we mark each triangle's vertices as "to be ray traced" if the triangle is front-facing; unmarked vertices are not ray traced.

Multipass Rendering with Layered EM Indexing Assuming we have a refractive object with n layers in its EM, we perform n passes for its refractive term, and n passes for the its reflective term. We multiply each term by the $F\,G$ function interpolated by the graphics hardware across each triangle. The texture index for each term/layer pass is generated by intersecting the outgoing ray with the layer's geometric impostor, such as a sphere. Taking this point of intersection and subtracting the EM origin point yields a vector that forms the hardware EM index, recorded with the vertex. As in [8], we use a hardware-supported cube map spherical parameterization which doesn't require normalization of this vector. Note that the texture indices change per layer since the distances to the EM spheres are different and the outgoing rays do not emanate from the EM origin.

When rendering from a viewpoint away from a pre-rendered sample, a smoother result is obtained by interpolating between the two closest viewpoints. Thus, we perform $4n$ passes, $2n$ for each viewpoint, blended by the relative distance to each viewpoint. Ray tracing, adaptive tessellation, and texture coordinate computation are performed just once per frame. Ray tracing is performed with respect to the actual viewpoint, not from the adjacent viewpoint samples.

Each layer is computed in a separate pass because of texture pipeline limitations in the current graphics system (Microsoft Direct3D 7.0 running on an Nvidia GeForce graphics accelerator).

To begin the series of compositing passes for the second of the two summed shading terms, the framebuffer's alpha channel must be cleared. This is accomplished by rendering a polygon that multiplies the framebuffer's rgb values by 1 and its alpha by 0. We then render its layers from front to back, which sums its contribution to the result of the first term's passes. With the advent of programmable shading in inexpensive PC graphics hardware and the ability to do four simultaneous texture accesses in each pass, it will be possible to reduce those $4n$ passes to n and avoid the alpha clear step [17][18].

6   Results and Discussion

We tested a scene containing a glass teapot, a reflective cup, a ring of columns, and distant walls. EMs were parameterized by viewpoints circling around the teapot in 8° increments. The scene contains two lens objects, a teapot and cup; we used our clustering algorithm to select 3 EM layers for the teapot and 2 for the cup. A quadrilateral impostor was used for the sparsely-occupied cup environmental layer of the teapot (Figure 7, top), a cylindrical shell for the columns environmental layer of the teapot (Figure 7, middle), and spherical shells for all other layers. We also tried a solution that was constrained to a single EM layer for each lens object, still using the clustering algorithm to determine placement of the single shell.

Figure 8 compares the quality of our results for two novel views: one in the plane of the circle of viewpoints (a), and one above this plane (b). Using multiple EM layers, we achieve quality comparable to the ray tracer. Reconstruction using a single layer is noticeably blurry because of conflicts where different points in the environment map to identical ones in the spherical shell approximation. Moreover, the video results for the single layer solution show significant "popping" when switching between viewpoint samples. The multi-layer solution better approximates the environment, providing smooth transitions between viewpoints.

Together, we call the method of ray continuation to reach occluded geometry from Section 4.1, and the simultaneous solution across multiple viewpoints from Section 4.3, *EM disocclusion prevention*. Figure 9 and our video results show the effectiveness of these methods in eliminating environmental disocclusions which would be obvious otherwise.

To measure performance, we tried two different adaptive subdivision thresholds of $\tau=3$ and $\tau=5$ (measured in subpixels) in a 3×3 subsampled 640×480 resolution rendering.   Performance was measured in seconds on a 1080MHz AMD Athlon with Nvidia GeForce graphics card; reduction factors are with respect to ray tracing without hybrid rendering.   For comparison, the ray tracer required 480 seconds to render each frame, using 6,153,735 rays and 168,313,768 triangle intersection tests. The

	$\tau=3$	$\tau=5$
Ray tracing at vertices	13.48	7.82
Texture coord. generation	0.71	0.38
Tessellation	2.11	0.83
Other (inc. rendering)	2.57	1.45
Total frame time	18.87	10.48
Time reduction factor	25.4	45.8
Ray count	1,023,876	570,481
Ray reduction factor	6	10.8
Triangle intersection tests	11,878,133	6,543,993
Intersection red. factor	14.2	25.7

version with $\tau=3$ is shown in Figure 8; the two versions are compared in Figure 10. The faster $\tau=5$ achieves good quality but suffers from some artifacts when animated. The difference is discernible in the still image in Figure 10 as slightly increased noise along edges such as the bottom of the teapot and where the spout joins the body.

Hybrid rendering was 25-45 times faster than a uniformly-sampled ray tracing, though both used identical ray casting code and the greedy ray path shading model. (Using a full tree shading model would incur an additional factor of 5.) This is not entirely accounted for by the reduction of roughly a factor of 6-11 in ray queries or 14-25 in triangle intersection tests obtained by hybrid rendering. The reason for our increased performance is the increased locality we achieve by ray tracing only through the lens object's geometry, and the hardware acceleration of texture map access. Although adaptive sampling reduces triangle intersection tests and ray queries by roughly the same factor, triangle intersection tests (which include ray misses as well as actual intersections) are reduced by an additional factor because the environment is approximated with simple spherical shells.

Though our performance falls short of real-time, significant opportunity remains both to optimize the software and parameters (like the initial lens object's tessellation), and to tradeoff greater approximation error for higher speed. We note that more complicated environmental geometry will increase the benefit of our use of approximating shells. To speed up ray tracing, it may be advantageous to exploit spatial and temporal coherence, possibly combined with the use of higher-order surfaces rather than memory-inefficient polygonal tessellations. In any case, we believe that future improvement to CPU speeds and especially support for ray tracing in graphics hardware will make this approach ideal for real-time rendering of realistic shiny and glass objects.

7   Conclusion

Hybrid rendering combines ray tracing, which simulates complicated ray bouncing off local geometry, with environment maps which capture the more distant geometry. This exploits the hardware's ability to access and resample texture maps to reduce the number of ray casts and consider them in a memory-coherent order. By inferring layered EMs parameterized by viewpoint, we preserve view-dependent shading and parallax effects in the environment without performing unaccelerated ray casts through its complicated geometry. With these techniques, we obtain a realistic simulation of highly specular reflective and refractive objects that would be impractical with light field based methods.

A major benefit of this work is to make the cost of ray tracing *low* and *predictable* without sacrificing quality. Lower cost, but probably not higher predictability, results from our adaptive ray tracing algorithm. Two of our other techniques enhance both. We avoid large variations in the ray tree of refractive objects from one pixel to the next by substituting two ray paths. We also substitute a fixed set of simple shells for arbitrarily complex environmental geometry.

One area of future work is to study the effect of hybrid rendering on compression of parameterized image spaces. We expect that PEMs should better capture the coherence in image spaces compared with parameterized texture maps that are statically mapped on objects [7]. Another possible application of this work is in speeding up the rendering of realistic animations. Hybrid rendering could be used to interpolate between ray traced key frames at which view-dependent layered environment maps are inferred.

References

[1] BASTOS, R., HOFF, K., WYNN, W., AND LASTRA, A. Increased Photorealism for Interactive Architectural Walkthroughs. Interactive 3D Graphics 1999, pp.183-190.

[2] BLINN, J. F., NEWELL, M. E. Texture and Reflection in Computer Generated Images. Comm. ACM, 19(10), Oct. 1976, pp.542-547.

[3] CABRAL, B., OLANO, M., AND NEMEC, P. Reflection Space Image Based Rendering. SIG-GRAPH 99, pp.165-170.

[4] CHUANG, Y., ZONGKER, D., HINDORFF, J., CURLESS, B., SALESIN, D., AND SZELISKI, R., Environment Matting Extensions: Towards Higher Accuracy and Real-Time Capture, SIG-GRAPH 2000, pp.121-130.

[5] DIEFENBACH, P. J. Pipeline Rendering: Interaction and Realism through Hardware-based Multi-Pass Rendering. PhD thesis, University of Pennsylvania, June 1996.

[6] GORTLER, S., GRZESZCZUK, R., SZELISKI, R., AND COHEN, M. The Lumigraph. SIG-GRAPH 96, pp.43-54.

[7] HAKURA, Z., LENGYEL, J., AND SNYDER, J. Parameterized Animation Compression. Euro-graphics Rendering Workshop 2000, pp.101-112.

[8] HAKURA, Z., SNYDER, J, AND LENGYEL, J. Parameterized Environment Maps. Interactive 3D Symposium 2001, March 2001, pp. 203-208.

[9] HECHT, E., Optics, Second Edition, Addison-Wesley, 1987.

[10] HEIDRICH, W., LENSCH, H., COHEN, M. F., AND SEIDEL, H. Light Field Techniques for Reflections and Refractions. Eurographics Rendering Workshop 1999, pp.195-375.

[11] HEIDRICH, W., SEIDEL, H. REALISTIC, Hardware-Accelerated Shading and Lighting. SIG-GRAPH 99, pp.171-178.

[12] KAY, D., AND GREENBERG, D., Transparency for Computer Synthesized Images, Siggraph 1979.

[13] KAJIYA, J., The Rendering Equation, SIGGRAPH '86, Aug. 1986, pp.143-150.

[14] LEVOY, M., HANRAHAN, P. Light Field Rendering. SIGGRAPH 96, pp.31-41.

[15] LINDE, Y., BUZO, A., AND GRAY, R. M., An algorithm for Vector Quantizer Design, IEEE Transactions on Communication COM-28, 1980, pp. 84-95.

[16] LISCHINSKI, D., RAPPOPORT, A., Image-Based Rendering for Non-Diffuse Synthetic Scenes. Eurographics Rendering Workshop 1998, pp.301-314.

[17] MICROSOFT DIRECTX8.0, http://www.microsoft.com/directx/.

[18] MICROSOFT XBOX, http://www.xbox.com/xbox/flash/specs.asp.

[19] MILLER, G., RUBIN, S., AND PONCELEON, D. Lazy Decompression of Surface Light Fields for Precomputed Global Illumination. Eurographics Rendering Workshop 1998, pp.281-292.

[20] OFEK, E., RAPPOPORT, A. Interactive Reflections on Curved Objects. SIGGRAPH 98, pp.333-341.

[21] PHARR, M., KOLB, C., GERSHBEIN, R., AND HANRAHAN, P., Rendering Complex Scenes with Memory-Coherence Ray Tracing, SIGGRAPH 97, pp.101-108.

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

[23] WOOD, D. N., AZUMA, D. I., ALDINGER, K. ET AL. Surface Light Fields for 3D Photogra-phy. SIGGRAPH 2000, pp.287-296.

[24] ZONGKER, D., WERNER, D., CURLESS, B., AND SALESIN, D., Environment Matting and Composition, SIGGRAPH 99, pp.205-214.

Layer	Reflection term b_R	Refraction term b_T	Inferred EM x from both terms
L_1 (cup)			
L_2 (cols)			
L_3 (walls)			

Figure 7: Layered EMs inferred at a viewpoint sample for a glass teapot (three layers). A quadrilateral was used for the L_1 layer, a cylindrical shell for L_2, and a spherical shell for L_3. Shells are parameterized by a six-faced cube map. Entire MIPMAPs are inferred; only the finest level is shown.

(a) between viewpoint samples (in plane)

(b) above viewpoint samples

| Ray Traced | Hybrid (multi-layer) | Hybrid (single layer) |

Figure 8: Hybrid rendering results. The right two columns were generated by a PC graphics card.

(a) full ray tree   (b) two-term greedy ray path

Figure 2: Shading Models. The full ray tree (a) requires 5 times more ray queries than our greedy ray path model (b).

(a) with prevention

(b) without prevention

Figure 9: EM Disocclusion

Ray Traced (480s/frame)

Hybrid Slower, $\tau = 3$ (19s/frame)

Hybrid Faster, $\tau = 5$ (10s/frame)

Figure 10: Quality Comparison.