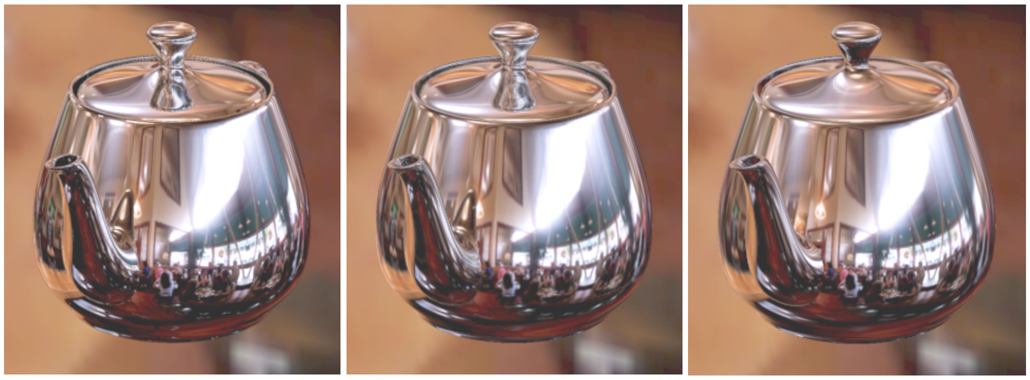
Parameterized Environment Maps

Ziyad S. Hakura
Stanford University
zsh@stanford.edu

John M. Snyder
Microsoft Research
johnsny@microsoft.com

Jerome E. Lengyel
Microsoft Research
jedl@microsoft.com



(a) Ray-traced (b) PEM (c) Static EM

Figure 1: Simulating Reflections. Note the missing self-reflections of the knob and spout in (c) and the fidelity of the PEM (b) matched to the ray-traced image (a).

Abstract

Static environment maps fail to capture local reflections including effects like self-reflections and parallax in the reflected imagery. We instead propose parameterized environment maps (PEMs), a set of per-view environment maps which accurately reproduce local reflections at each viewpoint as computed by an offline ray tracer. Even with a small set of viewpoint samples, PEMs support plausible movement away from and between the pre-rendered viewpoint samples while maintaining local reflections. They also make use of environment maps supported in graphics hardware to provide real-time exploration of the pre-rendered space. In addition to parameterization by viewpoint, our notion of PEM extends to general, multidimensional parameterizations of the scene, including relative motions of objects and lighting changes.

Our contributions include a technique for inferring environment maps providing a close match to ray-traced imagery. We also explicitly infer and encode all MIPMAP levels of the PEMs to achieve higher accuracy. We propose layered environment maps that separate local and distant reflected geometry. We explore several types of environment maps including finite spheres, ellipsoids, and boxes that better approximate the environmental geometry. We demonstrate results showing faithful local reflections in an interactive viewer.

Additional Keywords: ray tracing, reflections, Fresnel modulation, parameterized texture maps, surface light fields, IBR.

1. Introduction

Accurate, real-time rendering of shiny objects has long been a goal of computer graphics. Environment maps (EMs) [2][8][9][15][20], which store a sphere of radiance incident at a point, achieve a reasonable approximation of reflections and are easily supported in hardware. Unfortunately, because EMs are constructed from a single point like the reflective object's center, they fail to accurately reproduce *local* reflections (Figure 1c). Self-reflections are lost since the object itself is omitted during EM construction and geometric accuracy suffers when the surface point is not exactly at the object center or if the reflected object is not very distant.

Our solution is to record multiple EMs at a set of viewpoints in a pre-rendered space. These EMs are not spherical images of the environment at a point. Instead, they are inferred as a least-squares best match to a ray-traced image taken at each viewpoint, *when applied as an EM in a rendering by the target graphics system*. The result is a sequence of EMs, which we call a parameterized environment map, or PEM (Figure 2). More generally, PEMs can be parameterized with any number of dimensions, which can control positions of objects and lights as well as view, and any number of samples per dimension.

Using PEMs, we are able to closely match local reflection effects like self-reflections evident in each ray-traced sample (compare Figure 1a and b). Furthermore, we can move the viewpoint away from the ray-traced samples, plausibly maintaining these local reflection effects. In fact, by inferring PEMs from ray traced images along only a 1D subspace of views, we achieve convincing local reflections from an entire 3D viewspace quite far from the original samples. Capturing a 2D viewspace provides even better accuracy but uses many more maps. PEMs also capture much of the coherence in view-dependent shading, and thus compress very well.

With simple EM models, this approach does not necessarily match the ray traced images exactly, since the mapping from the surface of the reflector to the EM is not necessarily one-to-one. In other words, two or more reflected positions in the world can map to the same EM point.[1] We reduce these conflicts by using *layered* EMs which separate local and distant parts of the environment. The final result is superior to images generated using a static EM.

[1] Another difficulty is that the same point in the world can be multiply imaged by the reflector with different shading if its surface is non-Lambertian.

2. Previous Work

Accurate reflections between arbitrary objects can be produced by ray-tracing [21], but at significantly higher cost than traditional texture-mapped polygon rendering. This has led to efforts that exploit fast graphics hardware to produce realistic reflections. Reflections on planar surfaces, explored by Diefenbach [6], can be achieved using a rendering that mirrors the viewpoint about the reflection plane. Reflections on curved objects, studied by Ofek and Rappoport [17], can be performed by transforming each vertex in the reflected image with respect to the reflector's geometry. This scheme handles smooth reflecting objects that are either concave or convex; objects with mixed convexity or saddle regions require careful decomposition.

Image-based rendering (IBR) methods such as the Light Field [13] and Lumigraph [7] reduce the plenoptic function to a tabulated 4D field. Surface light fields [16][22] are an alternative that parameterize the radiance field over surfaces rather than views to better capture spatial coherence, especially when surfaces are mostly diffuse. For mirror-like reflectors, the surface light field is essentially identical to a sphere-at-infinity EM per surface point[2]. We obtain a better prediction of how reflections change with view by using more accurate geometric approximations of the environment (Section 4) including simple finite ellipsoids and boxes, and by separating reflections into multiple layered maps for local and distant elements. We also associate a single EM per object, but parameterized by view, rather than multiple, view-independent ones over a dense set of its surface points.

When applying existing IBR methods to highly reflective surfaces, it is not clear whether the required sampling density of views (for view-based IBR) or of emitted radiance per surface point (for surface-based IBR) is practical. Current results demonstrate only fairly blurry highlights from light sources. In [22] for example, this is not surprising given that the 258 lumisphere samples used per point represent 2-3 orders of magnitude fewer samples than typical EMs contain. Furthermore, to reconstruct an image from a particular view requires visiting an irregular scattering of samples over the entire 4D light field. With PEMs, all the information needed to reconstruct a particular view is spatially coherent in the form of a single EM image (two are needed for smoother interpolation in a 1D viewspace) whose access is already supported by hardware. PEMs also handle arbitrary view subspaces (e.g., 1D ones) and other scene parameterizations.

Heidrich et al. [11] decouple geometry from illumination by using a light field to map incoming view rays into outgoing reflected or refracted rays, thus handling self-reflections. These outgoing rays then index either a static environment map, which ignores local effects further from the reflector, or another light field representing the environment, which is more accurate but also more costly. The result allows independent change to the reflecting object geometry and the environmental radiance, but suffers from the limitations of other IBR methods mentioned above.

Cabral et al. [3] also decouple the reflecting object from the illumination. They store a collection of view-dependent EMs where each EM pre-integrates a specific BRDF with a lighting environment. The lighting environments for these EMs are generated using standard techniques, such as taking photographs of a physical sphere in a desired environment or rendering the six faces of a cube from the reflecting object center using a ray-tracer. As a result these EMs suffer from the same problems as traditional EMs in handling local reflections.

[2] Though surface light fields store emitted radiance rather than incident, the required BRDF integration is trivial for mirror-like reflections.

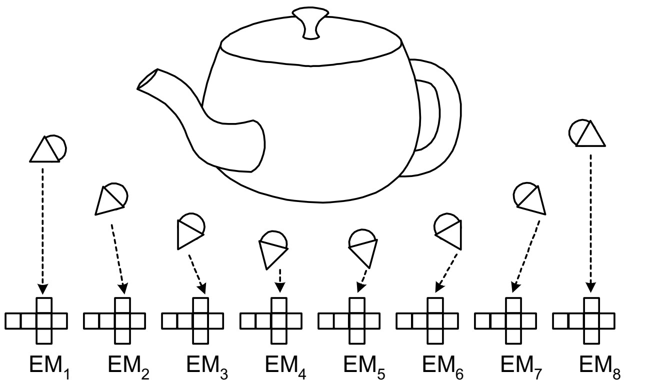


Figure 2: PEM. A PEM is a sequence of environment maps (EMs) recorded over a set of viewpoints (or other parameters). The diagram suggests a cube map parameterization for each EM, but other choices are possible.

Lischinski and Rappoport [14] propose two ideas: layered light fields, which are a collection of view-dependent LDIs for glossy objects with fuzzy reflections, and image-based ray-tracing for sharp reflections, where rays are traced through three view-independent LDIs that accurately represent the scene geometry. Bastos et al. [1] reproject LDIs into a reflected view for rendering primarily planar glossy surfaces in architectural walkthroughs. Our approach succeeds with much simpler and hardware-supported EMs rather than LDIs.

Hakura, et. al., [10] capture realistic, pre-rendered shading effects including reflections as parameterized texture maps (PTMs) on surfaces. This method of capturing view-dependent shading makes it hard to move "off the manifold" or away from the sampled views – the shading looks (and indeed is) pasted on. We apply their method of texture inference, but instead of texture maps we compute EMs in which a reflection ray is actually bounced off the surface and intersected with a simple approximation of the environment. Our result provides much better quality off the manifold.

Our factorization of the Fresnel modulation layer from the incident specular radiance and use of graphics hardware for its evaluation is based on the work of Heidrich and Seidel [12].

3. System Overview

We begin with a ray-traced image at each viewpoint as in Figure 2, or more generally, each point in the parameter space which is to be interactively explored later. The ray tracer segments the imagery by separating the images of individual objects (Figure 4). It also segments various shading terms on each object, including a diffuse layer, Fresnel reflection modulation layer, and an incident specular layer. We use a modified version of Eon, a Monte Carlo distribution ray tracer [19].

Given the parameterized, segmented image layers for each object, we compute, or *infer,* parameterized texture and environment maps that accurately reproduce the ray-traced images when applied to the original geometry by graphics hardware. PTMs are inferred using the technique of [10]. PEMs are handled by applying that technique to hardware rendering with environment maps, as discussed in detail in the next section. Because of the segmentation, the PTMs and PEMs exhibit much coherence and can be greatly compressed, using schemes like MPEG for 1D parameter spaces, or the multidimensional Laplacian pyramid of [10].

At run-time, as a user explores the space, the system decodes a per-object texture and environment map sample closest to the user's current parameter location (e.g., viewpoint). The resulting maps are then loaded into the graphics hardware. Using appropriate texture blending modes, we recombine the diffuse layer, if present, with the product of the Fresnel-modulation layer and environment-mapped result. This produces a rendering that accurately matches the ray traced imagery at the sampled parameter locations, and successfully interpolates images away from those samples. We also blend between neighboring parameter samples rather than choosing the closest to provide a smoother result.

4. Parameterized Environment Maps

The incident specular layer for a shiny object is the incident light that is then attenuated by the surface reflectance and reflected towards the viewer. Separating out the view-dependent Fresnel modulation by recording incident rather than emitted radiance makes this layer more view independent and thus more coherent. After generation by the ray tracer, the layer is captured as an EM. The next subsections present our approach for representing and inferring EMs. By running the inference method at each point in parameter space, we obtain a PEM which can then be compressed.

EM Representations. It is important to distinguish the geometry of an EM from its parameterization. An EM's *geometry* refers to how it approximates the reflecting environment. For example, the environment can be approximated by a sphere at infinity, by a finite cube, or by a finite hemisphere with planar bottom. By picking an EM geometry that closely matches the actual environment's, we obtain better predictions of how reflections move as the view changes. Of course, as we move to even better approximations of the environment, such as light fields, LDIs, or even ray traces of the original environment, the cost of computing samples becomes impractically high.

The *parameterization* of an EM refers to how the geometry is represented in a 2D map. For example, the infinite sphere can be represented using a latitude/longitude parameterization, the gazing ball (or OpenGL) parameterization, or by six faces of a cube. The choice of EM parameterization is less crucial than the choice of geometry in determining final accuracy, but does impact how much resolution the maps will require. Note that confusion arises because parameterizations tend to have names related to shape; for example, a finite sphere geometry can be parameterized using a cube parameterization and vice versa.

We have tried several types of EM geometry achieving best results with finite rectangular parallelepipeds, called *box maps*, and finite spheres and ellipsoids. Box maps are useful for room environments. For objects resting on flat surfaces, it is effective to align the box bottom with this flat surface and extend the other sides to match the average distance to environmental geometry. Finite ellipsoid maps have proved useful for local reflected geometry (see next section).

To index such maps, we resort to a combination of software and hardware texture coordinate generation. Graphics systems, such as the Nvidia GeForce running under DirectX, currently support only the infinite sphere geometry with cube or gazing ball parameterization. In software, for each polygonal mesh vertex, we bounce the view ray off the vertex using its associated normal to determine a reflecting ray. The resulting ray is intersected with the EM geometry, such as a finite box, sphere, or ellipsoid. The

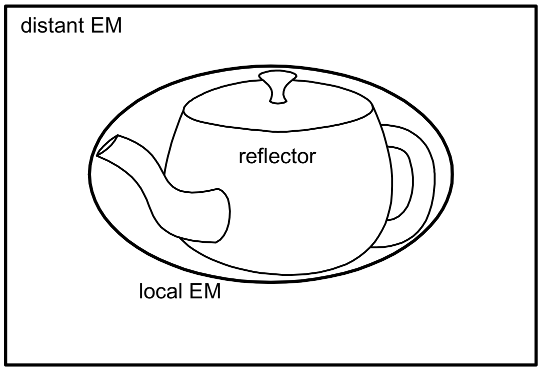


Figure 3: Layered EM. Here a box geometry is used for the distant EM, modeling more distant reflected parts of the environment, while an ellipsoidal geometry is used for the local EM, modeling parts of the reflector itself present in self-reflections.

simplicity of these models makes the ray intersection calculation practical for real-time applications. Taking this point of intersection and subtracting the object's EM origin point yields a vector that is used as the hardware EM index (normalization isn't required for the cube parameterization). The EM origin is chosen as an area-weighted average of the object's triangle centroids.

The hardware-supported cube map parameterization can thus be used for any of these simple geometries.[3] Alternatively, direct use of the 6 cube face texture maps would require expensive texture state switching and explicit handling of triangles whose vertex indices straddle the cube edges or corners. The singularity in the gazing ball parameterization makes it inappropriate for generating views off the ray-traced samples, as observed in [12].

Layered EMs. Segmenting the environment into separate maps for local and distant elements better approximates how each elements' reflections behave (Figure 3). The separation allows different EM geometries to be used to approximate the environmental geometry in each layer. It also supports parallax between imagery in each layer.

To perform this separation, the ray tracer records separate layers for rays which bounce off a reflective object and immediately reach the distant environment and rays which bounce one or more additional times off the object (Figure 4). EMs are then inferred for each layer separately (Figure 5). We have found that a finite ellipsoidal EM geometry works well for the local layer. An ellipsoid is selected to tightly bound the reflecting object. We currently use an axis-aligned bounding ellipsoid centered at the object centroid with axis scales that minimize resulting volume, determined using brute force optimization.

To handle occlusion effects, the local EM is computed as a 4 channel image with transparency representing fraction of coverage in the local layer from an antialiased rendering. To compute the distant layer without local occlusions, the ray tracer propagates rays through the reflecting object if they intersect after the first bounce, thus defining all distant layer samples without need for an alpha channel. At run-time, we use the "over" blending mode to composite the local layer (subscript *L*) over the distant (subscript *D*) before modulating by the Fresnel term, *F*, via

$$\left(\alpha_L\, rgb_L + (1-\alpha_L)\, rgb_D\right)F \ .$$

[3] Note that the cube map parameterization (6 equal-sized square faces) uses texture area inefficiently for ellipsoids, but is a limitation of current hardware.

Note that this method easily generalizes to more than two environmental shells.

Because the local EM geometry only approximates the local geometry being reflected, erroneous local reflections can be generated. We minimize such problems by separating polygons on the reflector that lie on its convex hull from those lying inside it as a view-independent pre-process. At run-time, polygons lying on the convex hull are rendered only with the distant EM, since they can never exhibit self-reflections. The rest of the polygons are textured using the local/distant EM combination.

EM Inference. Our inference approach is based on the observation from [10] that a texel, whether from a texture map or an EM, contributes to zero or more display pixels. Neglecting quantization effects in the hardware, a texel that is twice as bright contributes twice as much to these display pixels. We therefore model hardware rendering as a linear system, called the *rendering matrix*, which maps texels to display pixels. To find the rendering matrix, we perform test renderings that isolate the contribution of each texel to the display. Given the rendering matrix, A, we then find the least-squares best EM, x, which when applied matches the ray tracer's segmented incident specular layer, b. This results in the linear system $Ax=b$, which we solve using conjugate gradient. The only difference in computing an EM instead of a texture as in [10] is that the rendering matrix is created using test renderings with environment mapping instead of texture mapping.

The method of [10] includes *regularization* terms which ensure all texels are in range and non-contributing ones are filled in smoothly from defined neighbors. It also provides an efficient method of computing the rendering matrix.

EM MIPMAPs. Our inference method solves for all MIPMAP levels of the EM simultaneously. Interestingly, we find that these levels are not simple filtered versions of each other, even when performing weighted filtering that accounts for the variation of solid angle over the EM parameterization, as in [12]. This is because in computing the best matching EM at a particular viewpoint, our inference method implicitly accounts for the reflector geometry, which can magnify and distort the reflection in a spatially varying way. Therefore, unlike [10], we explicitly encode all levels of the EM MIPMAP rather than creating them on-the-fly as decimated versions of the finest level. The result is improved sharpness in the reflections. To increase compression, encoded levels are stored as residuals from corresponding decimated versions of the finest level.

EM Resolution. Choosing the proper EM resolution is important to preserve frequency content in the reflections. A very conservative approach is to use 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 somewhat lower resolutions.

5. Run-Time Implementation

The Fresnel modulation layer is generated on-the-fly using a per-vertex software shader that copies the ray tracer's computation of the Schlick model [18]. We make use of a 1D texture map to better interpolate the fifth order polynomial involved in that model. Such per-vertex computation requires adequate tessellation of the reflecting object's geometry.

We use multi-pass rendering to assemble the shading layers. Purely reflective objects in a 1D viewspace require 5 texture map accesses: 2 EMs for the local/distant dual at each of 2 viewpoints for smooth interpolations and the 1D Fresnel map. Addition of a diffuse layer requires one more texture access. Current PC graphics hardware performs 2 texture accesses per pass, so 3 passes are needed.

Factoring surface reflectance from incident radiance is problematic on current hardware with fixed point 8-bit texture arithmetic; it is difficult to fit the dynamic range needed by the incident specular layer [5]. We clip samples that are too bright, sometimes resulting in artificially dimmed highlights. Solving this problem will require more dynamic range in texture processing, perhaps using a floating point representation.

6. Results

We tested our approach on a simple scene of a reflective teapot in a room environment. The teapot contains ~40k triangles and was ray traced from a 1D viewspace partially circling the teapot at 1° per view sample to generate a PEM containing 100 EMs. We used the local/distant dual EM with finite ellipsoid for the local map and box map for the distant at 256×256×6 resolution. To improve accuracy, we decomposed the teapot into two parts: the lid in one and the spout, body, and handle in the other. Each part used a separate layered EM model. A more efficient approach would be to decompose the teapot only for the local EM layer while using a single distant EM for the combined teapot.

Our viewer performs at about 17.5 frames per second with blending between adjacent viewpoints on 733MHz PC with Nvidia GeForce graphics accelerator. Downloading textures into hardware memory is a performance bottleneck; this will be alleviated by higher bandwidth memory and hardware-supported decoding of compressed textures. The following table details run-time performance, assuming the EM textures are already resident in system (but not video) memory. "Texgen time" below is the time to compute EM coordinates using ray intersection with simple box and ellipsoid primitives, performed on the CPU. Use of programmable vertex shaders supported in the graphics system, such as those available in DirectX DX8, may speed up this computation.

	On the Manifold (unblended)	Off the Manifold (blended)
#geometry passes	2	3
texgen time	35ms	35ms
frame time	45ms	57ms
FPS	22	17.5

For each viewpoint sample, ray tracing required about 15 minutes while EM inference took 5.5 minutes.

Figure 1 compares results "on the manifold"; i.e., at the ray traced samples. Our PEMs achieve good fidelity to the ray traced images matched, while static EMs eliminate local effects.

To test off the manifold quality, we tried several alternatives, including PTMs and non-layered PEMs using a single sphere-at-infinity EM geometry. Figure 6 compares these choices for EM geometry at a viewpoint between the original samples. A ray tracing at the exact viewpoint is included for comparison. It can be seen that the layered EM model produces greater accuracy than the sphere-at-infinity EM (notice especially the reflection of the knob on the lid). The results are even more obvious interactively than in a static image, where the dual layered PEM model eliminates "wobble" exhibited by the simpler model as the user moves off the manifold.

Figure 7 compares results between PEMs and PTMs above the plane of viewpoint samples. Note especially the lack of fidelity of the PTM image in the teapot lid reflections. These images and, more dramatically, the video results, show the pasted-on effect obtained by PTMs off the manifold, resulting in a popping artifact when switching between textures inferred at adjacent viewpoint samples. PTMs also suffer from disocclusions where non-contributing texture area is revealed in a different view. While disocclusions can also occur with PEMs since they too are solved at a given view, they are less frequent and less visible as long as the object reflects most of its environment.

Figures 4 through 7 are reproduced in the color plate section.

7. Conclusions and Future Work

PEMs provide a faithful approximation to ray-traced images at pre-rendered viewpoint samples and the ability to plausibly move away from those samples using real-time graphics hardware. In future work, we are interested in further exploring the use of multiple environment map shells to better approximate the environment, including finding the optimal placement for such shells. Applying the analog of the optimal plane placement in planar-projection IBR [4] to spherical imagery may prove useful. More generally, we seek automatic selection of EM geometry.

Another area of future work is handling cases where the reflector does not image parts of its environment that can nevertheless be seen in nearby views "off the manifold". For example, a reflector can be partially occluded or occlude itself (like the teapot spout obscuring its body), thus potentially eliminating from its EM the part of the environment reflected in this occluded portion. This problem can be solved by inferring EMs for all front-facing parts of the reflector, even if they are occluded in the particular view sample. Incomplete imaging of the environment also occurs when the reflector's set of normals incompletely cover the sphere such as with non-closed objects (like a small portion of a sphere) or objects with zero curvature (like a plane or cylinder). One solution may be to infer EMs across multiple neighboring viewpoints rather than at a single viewpoint sample to provide more complete knowledge of the reflected environment.

We are also interested in applying these methods to refractive as well as reflective objects. Finally, handling glossy surfaces as well as mirror-like ones should be possible with our techniques but remains to be investigated.

Acknowledgements

We acknowledge our supervisors Anoop Gupta and Turner Whitted for advice and assistance. Ravi Ramamoorthi and Li-Yi Wei provided much-needed help in producing the video.

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. SIGGRAPH 99, pp.165-170.

[4] Chai, J., Tong, X., Chan, S.C., and Shum, H., Plenoptic Sampling, SIGGRAPH 2000, pp.307-318.

[5] Debevec, P., Rendering Synthetic Objects into Real Scenes: Bridging Traditional and Image-Based Graphics with Global Illumination and High Dynamic Range Photography, SIGGRAPH 98, pp.189-198.

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

[7] Gortler, S., Grzeszczuk, R., Szeliski, R., and Cohen, M. The Lumigraph. SIGGRAPH 96, pp.43-54.

[8] Greene, N. Environment Mapping and Other Applications of World Projections. IEEE CG&A, 6(11), Nov. 1986.

[9] Haeberli, P., Segal, M. Texture Mapping as a Fundamental Drawing Primitive. Eurographics Rendering Workshop 1993, pp.259-266.

[10] Hakura, Z., Lengyel, J., and Snyder, J. Parameterized Animation Compression. Eurographics Rendering Workshop 2000, pp.101-112.

[11] Heidrich, W., Lensch, H., Cohen, M. F., and Seidel, H. Light Field Techniques for Reflections and Refractions. Eurographics Rendering Workshop 1999, pp.195-375.

[12] Heidrich, W., Seidel, H. Realistic, Hardware-Accelerated Shading and Lighting. SIGGRAPH 99, pp.171-178.

[13] Levoy, M., Hanrahan, P. Light Field Rendering. SIGGRAPH 96, pp.31-41.

[14] Lischinski, D., Rappoport, A. Image-Based Rendering for Non-Diffuse Synthetic Scenes. Eurographics Rendering Workshop 1998, pp.301-314.

[15] Miller, G. S., Hoffman, C. R. Illumination and Reflection Maps: Simulated Objects in Simulated and Real Environments. SIGGRAPH 84: Advanced Computer Graphics Animation Seminar Notes, July 1984.

[16] Miller, G., Rubin, S., and Ponceleon, D. Lazy Decompression of Surface Light Fields for Precomputed Global Illumination. Eurographics Rendering Workshop 1998, pp.281-292.

[17] Ofek, E., Rappoport, A. Interactive Reflections on Curved Objects. SIGGRAPH 98, pp.333-341.

[18] Schlick, C., A Customizable Reflectance Model for Everyday Rendering, Fourth Eurographics Workshop on Rendering (Paris, France), June 1993, pp.73-83.

[19] Shirley, P., Wang, C., and Zimmerman, K. Monte Carlo Methods for Direct Lighting Calculations, ACM Transactions on Graphics, January 1996, pp.1-36.

[20] Voorhies, D., Foran, J. Reflection Vector Shading Hardware. SIGGRAPH 94, pp.163-166.

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

[22] Wood, D. N., Azuma, D. I., Aldinger, K. et al. Surface Light Fields for 3D Photography. SIGGRAPH 2000, pp.287-296.

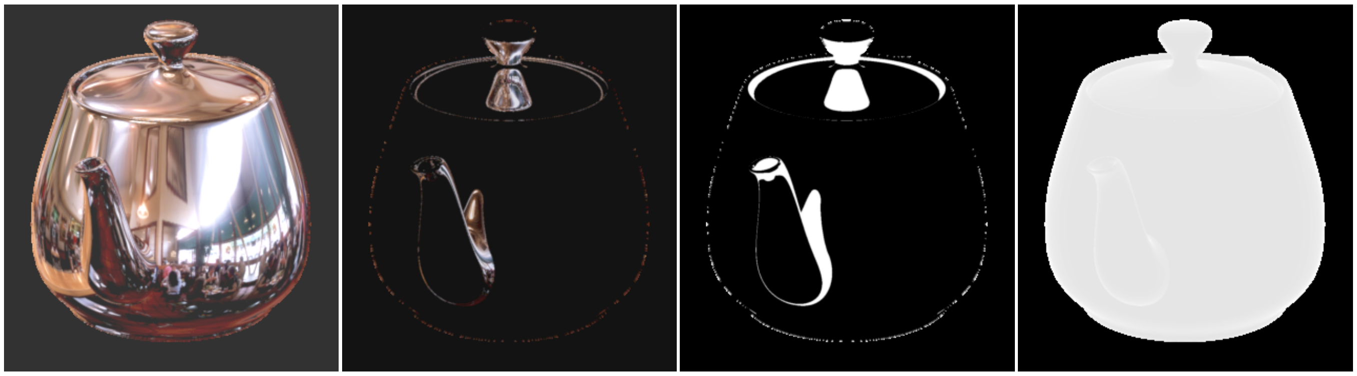


Figure 4: Segmented Layers Produced by the Ray Tracer for one Viewpoint.

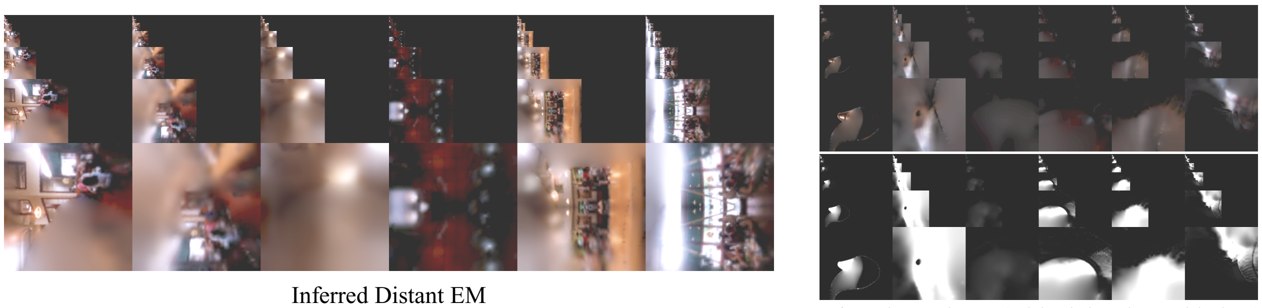


Figure 5: Inferred Layered EMs with MIPMAPs using cube map parameterization (6 faces) for teapot body part.

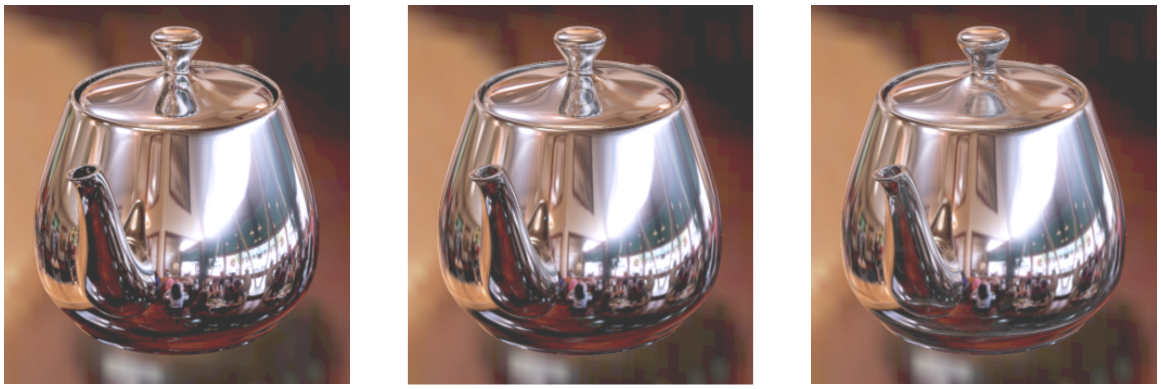


Figure 6: Results Between Viewpoint Samples

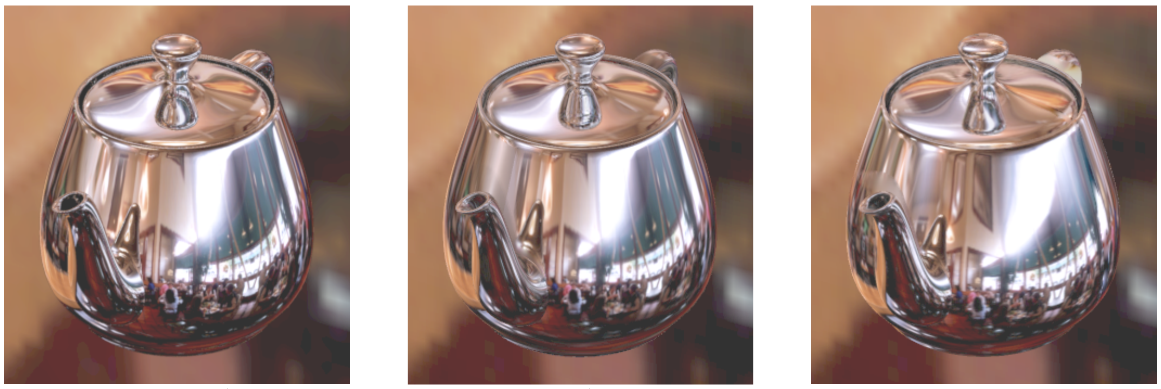


Figure 7: Results Above Viewpoint Samples