EUROGRAPHICS 2006 / E. Gröller and L. Szirmay-Kalos
(Guest Editors)

Volume 25 (2006), Number 3

Reflected-Scene Impostors for Realistic Reflections at Interactive Rates

Voicu Popescu, Chunhui Mei, Jordan Dauble, and Elisha Sacks
Purdue Univsersity

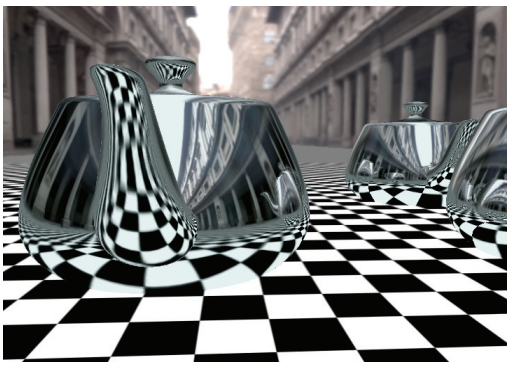


Figure 1: *Four inter-reflecting teapots, with 12 second order reflections. 720x512 pixels (all images), 40 fps.*

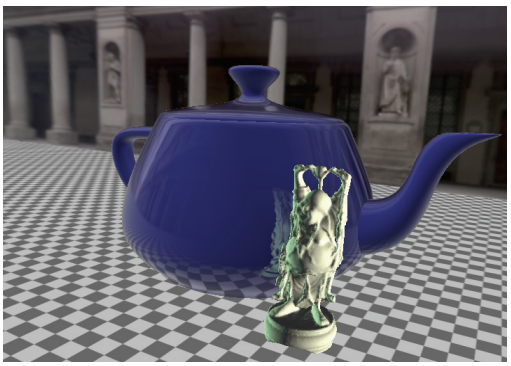


Figure 2: *Reflection attenuation with distance and Fresnel effects, 66 fps.*

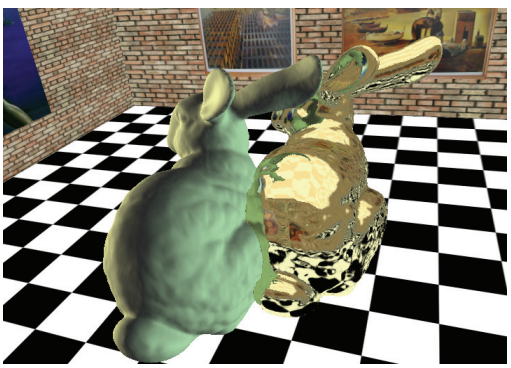


Figure 3: *Intersecting diffuse and reflective bunnies; reflection rendered with depth map impostor, 16 fps.*

Abstract

We present a technique for rendering reflections on complex reflectors at interactive rates based on approximating the geometry of the reflected scene with impostors. The reflections correctly convey the distance to the reflector surface and provide motion parallax. Two types of impostors are adapted to the reflections framework: billboards and depth maps. Billboards remove most of the problems of environment mapped reflections at only a small additional cost. Second order reflections are supported by introducing reflective billboards. Higher quality reflections that provide motion parallax within a reflected object are obtained by approximating the reflective geometry with depth maps. The computation of the intersection between a reflected ray and a depth map is accelerated by leveraging epipolar constraints. Like environment mapping, our technique does not pose any restriction on the geometry of the reflector, supports dynamic scenes, and runs at interactive rates with the help of graphics hardware.

Categories and Subject Descriptors (ACM CCS): I.3.3. [Computer Graphics]—Three-Dimensional Graphics and Realism.

1. Introduction

Reflections are an important effect in computer graphics not only because of their intrinsic esthetic value, but also because of tangible contributions such as revealing surface properties and conveying the relative position of objects. The problem of rendering reflections at interactive rates has been approached from many directions, but no complete solution exists. Rendering reflections implies solving the following problems at each pixel:

1. Computing the intersection between the pixel ray and the reflector.
2. Computing the reflected rays according to reflector surface properties.
3. Tracing each reflected ray into the scene, and proceeding recursively if a reflector is hit again.
4. Antialiasing to account for the non-zero pixel size.

The method of choice for rendering reflections in interactive graphics applications is environment mapping, which is simple, versatile, and produces quality reflections when the reflected objects are far away from the reflector [BN76, GRE86, HS93, VF94]. The success of environment mapping can be attributed to employing good approximate solutions to problems 1, 2, and 4 enumerated above.

Problem 1 is solved using the feed-forward graphics pipeline: the triangle mesh modeling the reflector surface is rasterized, which provides the intersection point at each pixel efficiently. Problem 2 is solved by approximating the bidirectional reflectance distribution function (BRDF) at the intersection point with a single normal. The normal is computed by interpolation of vertex normals, or by looking it up in a map. The single normal generates a single reflected ray. Problem 4 is handled by reducing it from a general antialiasing problem of geometry to the tractable problem of finding the appropriate level of detail in a texture, solved by mip-mapping.

When the reflected objects are close to the reflector, the reflection fails to convey the proximity to the reflector surface. When an object is touching the reflector, its reflection remains trapped deep inside the reflector. The artifact is disturbing, comparable to a shadow disconnected from a character that *does* touch the ground. Moreover, environment mapping does not provide motion parallax, and does not support higher order reflections. These problems are due to the drastic approximation employed to solve problem 3: the intersection between a reflected ray and the scene is looked up in a cube map using only its direction. The reflected scene is assumed to be infinitely far away, which allows replacing it with an image.

We describe a method that alleviates most of the disadvantages of environment mapped reflections, but maintains its advantages of efficiency and applicability to a wide range of reflectors. Our method is based on improving the approximation of the reflected scene. A successful such approximation has to satisfy two conditions. First, one has to be able to compute the approximation quickly in order to support dynamic scenes. Second, one has to be able to intersect the approximate geometry with an individual ray efficiently. This is needed since the coherence of the desired view rays is perturbed by complex reflectors. The lack of reflected ray coherence prevents one from amortizing the cost of the intersection over a neighborhood of similar rays, and each ray has to be treated individually.

We approximate the reflected scene with impostors, term coined by Maciel [MS96], and now broadly used to designate image-based rendering (IBR) geometry approximations for rendering acceleration. We adapt two types of impostors to the reflection rendering framework: billboards and depth maps. Both types of impostor can be constructed efficiently by rendering the geometry they replace with the help of graphics hardware. While the intersection between a ray and a billboard is inexpensive, intersecting a depth map with a ray is more challenging.

We have devised an algorithm that takes advantage of the fact that the depth map was acquired with a planar pinhole camera to greatly simplify the general problem of ray tracing a triangle mesh. We search for the intersection in 1D, along the projection of the ray onto the image plane of the depth map. We further accelerate the intersection computation by pre-simplifying several rotated depth maps. This way the projection of the reflected ray always aligns with a row in a rotated map, and the number of samples considered along a row is bounded. This allows implementing the intersection on the GPU with a pixel program. For a 256x256 depth map the total construction time of the simplified rotated depth map is ~200ms, and their aggregate memory cost is ~1MB.

Our method produces realistic specular reflections at interactive rates: the reflections convey proximity to the reflector and exhibit motion parallax. We support complex reflectors, second order reflections, dynamic scenes, and complex reflective surface properties (see Figures 1-3 and accompanying video). The current implementation does not support self-reflections. Our technique can be readily used in most interactive computer graphics applications.

The remainder of the paper is organized as follows. The next section reviews prior work. Sections 3 and 4 describe approximating reflected objects with billboards and depth maps, respectively. Results and conclusions are given in sections 5 and 6.

2. Prior work

Image-based rendering

The IBR approach is to pre-acquire reference reflections with a camera or by ray tracing, and then to use them in novel views. Light fields [LH96, GGS*96] naturally support reflections at no extra cost. Surface light fields [Mil98, WAA*00] reduce the size of the ray database using the geometry of the reflective surfaces. View dependent

texture mapping [DYB98] samples surfaces of limited reflectivity from a few viewpoints. Heidrich [Hei99] and Cabral [CON99] describe systems that decouple reflector properties from illumination. This is achieved with a light field that maps incident to reflected rays, and a set of radiance environment maps, respectively. Yu describes a method for replacing the environment map with a light field [YM04c]; the reflected ray is looked up using not only its direction but also its origin, for more accurate results. Hakura [HAK01] proposes using several reference reflection images, each parameterized to best match in least-mean-squares sense the true reflection when used as an environment map. The method provides good reflections nearby the reference viewpoints, but quality degrades for large viewpoint translations. A disadvantage common to all these techniques is the lack of support for dynamic scenes: if the reflecting or reflected objects move, or if the lighting conditions change, the reference rays become obsolete. A second disadvantage is limited support for highly reflective, mirror-like surfaces, which require a high sampling rate and generate impractically large ray databases.

Projection methods

An elegant approach to rendering reflections is to solve the problem of projecting reflected points. Computing the image plane projection of a reflected vertex enables rendering reflections with the hardware feed-forward graphics pipeline. The approach works well for planar reflectors, since the projection is trivial [Die96, MB97, Bas99]. For curved reflectors there is no closed form solution to reflected vertex projection. Hanrahan and Mitchell [HM92] describe a search procedure applicable to reflective surfaces given by an implicit equation. Ofek and Rappoport [OR98] handle triangle mesh reflectors using a reflection subdivision accelerated by an explosion map. The major disadvantage is inefficiency. The explosion map depends on the desired view and thus has to be computed for every frame. Complex reflectors generate highly complex and expensive explosion maps.

Ray tracing

Ray tracing [Whi80, Gla89] is a general technique which produces stunning images with complex reflections. The challenge is to avoid considering ray-primitive pairs that do not yield an intersection. A wide range of acceleration schemes have been proposed, and ray tracing has been shown to run at interactive rates on shared memory parallel computers [Par99], on special hardware [Hal01], on a single CPU [Wal01, WSB01, RSH05], and on GPUs [PBM*02, CHH02, WSE04]. We compare our method to GPU ray tracers in Section 6. In spite of these efforts the efficiency advantage of the feed-forward pipeline persists.

Reflected-scene approximation methods

The idea of simplifying the reflected scene in order to facilitate the intersection with the reflected rays is not new. As stated earlier, environment mapping does just that, except that the approximation employed is too drastic. Lischinski [LR98] proposes a scene representation based on layered depth images (*LDIs*) [SGH98]. Mirror-like reflections are supported by ray-tracing the LDIs, but this impacts performance considerably.

The first steps in the direction of our solution are taken by Bjorke [Bjo04], who describes rendering reflections in a room like environment by replacing the reflected scene with a sphere of size comparable to that of the room. A pixel shader intersects the reflected ray with the sphere, and the intersection point is used as the tip of the direction vector used in the environment map lookup. This improves the reflection accuracy since the environment is placed at an approximately correct distance from the reflector. Few environments are spherical, so a box-like proxy should be preferred, even at the cost of a slightly less compact intersection code. Bjorke's method is a special case of reflected billboard impostors, which offer greater modeling flexibility and support second order reflections.

An improved reflected scene approximation is achieved by using *distance impostors* [SALP05], which are cube maps enhanced with per-texel depth. Reflected or refracted rays emanating from the eye or from a point light source are intersected with the distance impostor to compute localized reflections, refractions, or caustics. The technique is efficient and implemented in hardware. The major drawback of the technique is the approximate nature of the intersection algorithm. An initial guess is constructed and then refined iteratively by making *severely* simplifying assumptions. As the authors note, the algorithm converges only for scenes that consist of large planar surfaces. For such scenes, ray tracing each of the few planar surfaces should be preferred not only for improved quality and reliability, but also for efficiency. The use of a depth map is not warranted when most texels correspond to coplanar points. Compared to the distance impostors, our technique works with complex depth maps (Figure 3), and our algorithm robustly finds a quality approximation for the intersection between the reflected ray and the depth map.

3. Reflected billboards

The simplest type of impostor is the billboard, a texture mapped quad, tailored to the contour of the foreground object using the alpha channel. Since Maciel's 1995 paper, researchers have described extensions that include using billboards in novel graphics architectures [TK96], caching and improving visibility computation for billboards [Sch97], transitioning from geometry to impostors [AL98], and placing impostors [AL99, DDSD03]. Billboards are particularly well suited for rendering reflections. First, the alternative of ray tracing the original geometry is very costly. This is not the case for impostors that replace directly seen geometry, which compete against the ever-increasing power of the fixed graphics pipeline. Second, the "cardboard cutout" artifact is less noticeable in the case of reflected billboards. Although the billboard is close to the

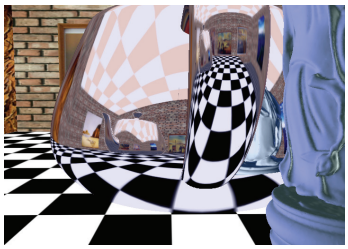
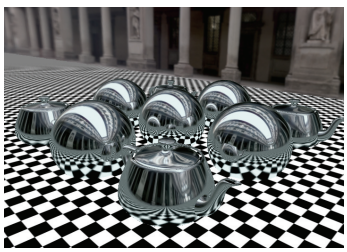


Figure 4: *Two inter-reflecting teapots and a statue placed on a table. The reflection is rendered using 4 billboards (statue and table for each teapot), 2 reflective billboards (teapots), and an environment map of the room. 33 fps*



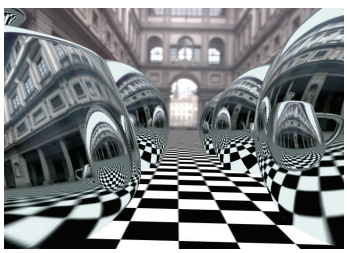


Figure 5: *Scene with 9 reflectors that generate 72 second order reflections. The spheres are modeled with triangle meshes. 11 and 6 fps, respectively.*

reflector, it is not necessarily close to the eye. Moreover, the distortion produced by the curved reflector hides the approximation from the user who has little intuition for the exact shape of such reflections.

When the billboard tightly approximates the geometry it replaces (e.g. flat floor, table top, walls, ceiling), the placement of the billboard is dictated by the original geometry. In this case the same billboard can be used with all reflectors in the scene, and it does not need to be re-rendered when the reflectors or the replaced object move. When the billboard replaces an object with non-negligible volume, we place the billboard plane through the centroid B of the object, and perpendicularly to the line connecting B to the centroid of the reflector. The billboard is constructed using a camera with a narrow field of view (10^o) to avoid perspective effects. A different billboard is used for each reflector. View changes alone do not require re-rendering the billboards, since the billboard is independent of the view. A billboard is re-rendered when the object or the reflector move or deform, or when the lighting changes. This avoids the large memory consumption, the preprocessing cost, and the transition problems of pre-computing a range of billboards in each dimension in which the scene changes.

We support higher order reflections using reflective billboards, which are billboards enhanced with per-texel normals. The normal at a texel is stored in texture memory. When a reflected ray intersects a reflective billboard, a tertiary ray is computed using the normal at the intersection point, which generates second order reflections (Figure 4). Although it is possible to continue the reflection process recursively, we find that reflections of order 3 and higher are not sufficiently useful to warrant the additional cost. A second order reflection is the reflection of an object that happens to be a reflector, and is therefore important for conveying the relative position of two close-by reflectors, the same way first order reflections are important in the case of a reflector and a diffuse object. Third and higher order reflections are too rare and too complex for users of 3D graphics applications to derive information from them. In real life, the only circumstance when we recall encountering such high order reflections is that of parallel planar mirrors, which could be supported as a special case.

3.1. Rendering algorithm

Given a scene with D diffuse billboards, R reflective billboards, and an environment map EM, the pixel shader implements a straight algorithm:

1. Compute reflected ray r at current pixel.
2. Intersect r with all other (D+R-1) billboards.
3. If no intersection return $EM(r)$.
4. If closest intersection is with diffuse billboard, return intersection texel.
5. If closest intersection is with reflective billboard
    a. Compute tertiary ray rr.
    b. Intersect rr with the D diffuse billboards.
    c. If no intersection return $EM(rr)$
    d. Else return intersection texel.

The billboards' geometry is passed in as a uniform parameter. The environment map is looked up whenever a ray does not intersect any billboard. The billboard intersection takes into account the alpha channel. The cost

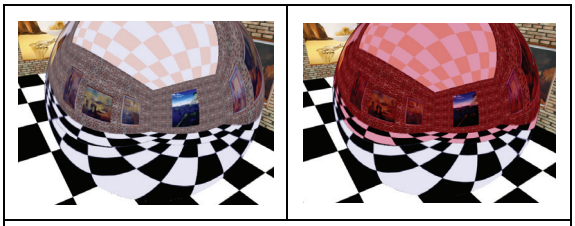


Figure 6: *Reflection discontinuity from floor billboard to environment mapping. Red highlight shows environment mapping in right image.*

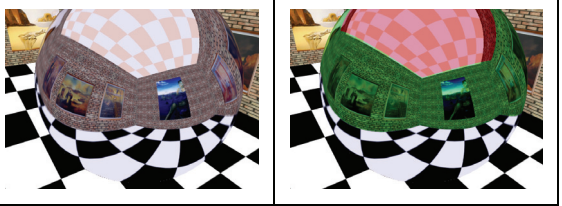


Figure 7: *The morph corrects the problem. Right image shows transition area in green.*

of the algorithm is $D+R-1+D=2D+R-1$ intersections. For Figure 4, Figure 1, and Figure 5 the costs are 2*2+2-1=5, 1*2+4-1=5 and 1*2+9-1=10 intersections. The number of reflections is $O(R^2)$, but shader complexity is linear.

3.2. Billboard to environment mapping morphing

So far we have discussed the case of an object that is not connected to the environment, such as the table top in Figure 4. When a billboard is connected to the distant scene rendered by environment mapping, maintaining continuity is important. In Figure 6 the floor appears twice, once correctly as the reflection of its billboard and once incorrectly in the environment map. One solution is to employ as many billboards as needed to cover all geometry connected to billboards. We prefer the better solution of morphing between a billboard and the surrounding environment map for a smooth transition (Figure 7).

In Figure 8, reflector R generates reflected rays r_0, r_1, *and* r_e. The center of the environment map is at E. The ground is modeled with billboard q_0q_1. Ray r_0 hits the billboard quad and it is set to the billboard texel color as before. Ray r_e clears the transition walls of height H and is set by environment mapping. Ray r_1 hits the transition wall at point a and it is replaced with r_m, a linear blend between rays r_1^d and r_a. Ray r_1^d originates at E and has the direction of r_1. Ray r_a also originates at E but points at intersection point a. When the height h of a is 0, the ray is set exclusively from the billboard, which ensures continuity.

When h equals H, the color is computed by environment mapping only. Objects A and B are sufficiently far away to be environment mapped. In Figure 9 a small ground

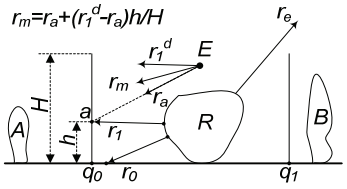


Figure 8: *Billboard to environment mapping morphing.*

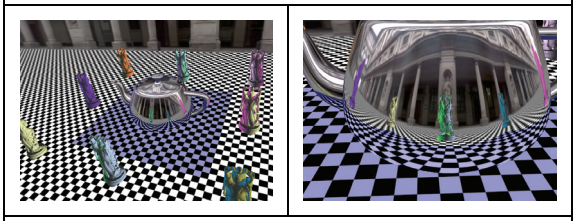


Figure 9: *Illustration of morphing from billboard to environment mapping. The quad used for the ground is highlighted with blue. The statues are not modeled with billboards and are reflected by environment mapping.*

billboard provides the correct reflection close to the teapot, and the morph produces a continuous reflection.

Whenever possible, the color texture of the billboard is stored in the environment map. In Figure 9 the ground does not occlude any part of the environment and can be stored in the environment map. Referring to Figure 8 again, the color of ray r_0 is fetched from the environment map along the ray from E to the tip of r_0. The billboard and the transition walls form a box, which is intersected with a ray more efficiently than 5 individual quads; the aggregate cost is comparable to that of 3 individual quads.

4. Reflected depth maps

Billboards provide great reflections in many situations. There are however circumstances when higher modeling power is needed. One example is when it is important for the reflection to show the exact floor or ground footprint of the reflected object, which billboards cannot provide. Another example is the case of a complex reflected object that gets close to the reflector surface in several points (see Figure 3): this case over constrains the billboard placement problem. A third example is the case of a diffuse object intersecting a reflector: the oversimplified intersection curve would reveal the flatness of the billboard, and a depth map is needed (see right side of Figure 3).

Modeling and rendering from depth maps [MB95] has the advantages of motion parallax within the depth map and invariance under rigid body transformation. The first step of using depth maps in the context of reflections is trivial: depth map impostors are constructed the same way and at

the same cost as billboards, the only difference is that the texture is extended with a z channel. Reflecting the depth map impostor into a complex reflector is more challenging. Extending the 3D warping operation to handle reflected depth samples cannot be easily done. This would be equivalent to the complex problem of projecting reflected vertices, problem pursued by the projection reflection methods discussed earlier. Instead we have devised a fast algorithm for intersecting a depth map with a ray. An acceleration data structure is constructed first and is then used to find the intersection between a reflected ray and the depth map efficiently.

4.1. Simplified rotated depth maps

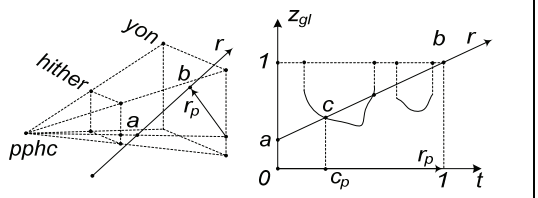


Figure 10: *Intersection computed with a 1D search in the depth map. The depth stores normalized 1/z.*

Given a reflected ray *r* and a depth map *DM*, the goal is to find whether and where *r* intersects *DM*. In Figure 10 *DM* was acquired with a planar pinhole camera *pphc*, using hither and yon planes that encompass the bounding box of the original geometry. The image plane was chosen to coincide with the yon plane for clarity. The intersection of *r* with *DM*, if any, has to project on r_p, which is the *pphc* projection of *r*. This constraint is similar to classic epipolar geometry constraints from multi-camera systems. The reflector and its rays can be seen as a complex camera whose rays are projected onto the image plane of *pphc*.

One approach for finding the intersection is to walk along r_p until an intersection is found. In Figure 10, *right,* the first intersection point *c* is found at c_p. This approach is similar to inverse warping, a technique for rendering from depth images by searching for the color of a desired image pixel along an epipolar segment in the reference image [McM97]. Although it is just a 1D search, the approach is inefficient. For the 256x256 depth map used for the images in Figure 3, the average length of r_p is 91 texels, and 63 texels have to be examined on average before an intersection is found.

It is our goal to reduce and bound the number of steps along r_p. One obvious approach is to simplify the depth map. However, it is difficult to bound the number of triangles that *any* r_p segment crosses. Moreover, intersecting with irregular triangles along r_p is expensive. Instead, we simplify the depth map on individual rows in several rotated domains. A simplified rotated depth map (SRDM) is produced for each rotated domain. The angles

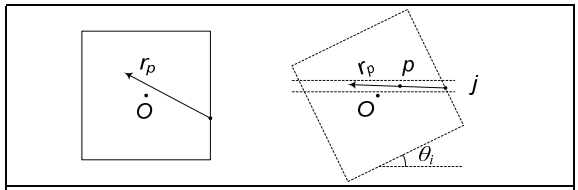


Figure 11: *The ray projection r_p is contained in row j in the rotated domain θ_i.*

that define the rotated domains evenly and densely span the 1D space of slopes in 2D. This way for any r_p there is an SRDM where r_p is almost horizontal (Figure 11). For a given ray *r,* the search is limited to the SRDM *i* and to its row *j* that contains the projection r_p. Given a depth map *DM* of resolution *w* x *w*, the desired number *n* of SRDMs, and the maximum number *m* of depth samples on each simplified row, the SRDMs are generated with the following algorithm:

For $i = 0$ to $n-1$
  $\theta_i = \pi i/n$
  For $j = 1$ to w
    $SRDM_{ij}$ = SimplifyRow(j, *DM*, θ_i, m)

Each row of each SRDM is obtained by simplifying the corresponding rotated depth map segment. No rotated depth maps are actually computed.

4.2. Row simplification algorithm

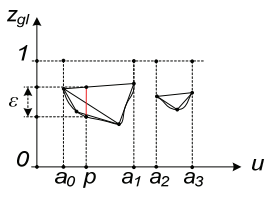
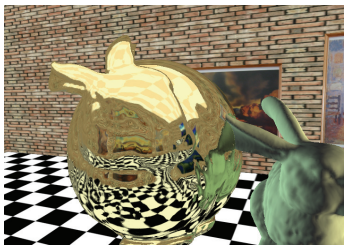


Figure 12: *Row simplification. This row has two scene spans. The initial approximation is given by points with u coordinates a_i. The final approximation is a poly-line with 7 points. The initial error at depth sample p is ε.*

Row *j* of rotated domain θ_i is simplified to *m* points using the depth map *DM* as follows:

1. Set the depth values of row *j* from *DM.*
2. Compute an initial poly-line approximation of the row by approximating each non-background scene span with two points (Figure 12).
3. Refine the poly-line in greedy fashion by splitting the worst segment in two, until the number of points reaches *m*.

The worst segment is the segment with the largest error. The error of a segment is the maximum error over the depth samples it covers. The refinement is implemented efficiently using a priority queue that stores each segment



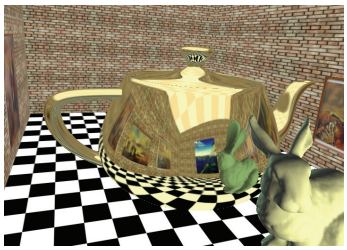


Figure 13: *Diffuse bunny reflections rendered with a single depth map impostor. The dynamic intersection line (top, 16fps) and motion parallax between (bottom, 20fps) are handled correctly (also see video).*

together with the row coordinate u where the error is largest. The queue is sorted in decreasing segment error order. The worst segment is available in constant time and it is split at the largest error point. The two new segments are inserted into the queue in logarithmic time. The SRDMs are constructed on the CPU and are stored on the GPU in texture memory as an array of $n*w*m$ 2D points. A point is defined by a pair of floats (u, z_{gl}).

4.3. Intersection algorithm

The role of the SRDMs is to provide a pre-simplified version of any depth map segment ever needed. Given a depth map acquired by *pphc* and simplified to an array $SRDM_i$, the reflection color is computed by a pixel program with the following steps:

1. Compute the reflected ray r at the current pixel.
2. Clip r with the 6 planes of *pphc*'s frustum, and if the entire ray is discarded go to 7.
3. Compute r_p by projecting the clipped ray onto *pphc*'s image plane.
4. Find $SRDM_i$ where r_p is closest to being horizontal, and let j be the row that contains r_p.
5. Iterate over the $m-1$ segments of the poly-line at row $SRDM_{ij}$ to find the intersection between r and DM.
6. If an intersection is found, lookup the impostor texture at the intersection point.
7. Else environment map using the direction of r.

Most of the work is performed by step 5, which requires reading $2m$ floats from texture memory and performs $m-1$ 2D segment intersections. The SRDMs truthfully approximate the depth map even for small m values (8-16), so the algorithm produces a quality intersection approximation efficiently Figure 13). The SRDMs do not need to be recomputed when the reflecting or reflected objects move. The DM camera is simply transformed to reflect the current position and orientation. Of course, more than one DM is needed to capture a reflected object. For the scene in Figure 13 a DM of the front of the bunny is needed when the two bunnies swap places.

5. Results

We have tested our approach on several scenes. The scene in Figure 1 uses the *Uffizi Gallery* environment map [Debwww], in which we replaced the original ground texels with a black and white checker board to better illustrate the quality of the rendered reflections. The ground is rendered with a billboard continuous with the environment, as described in Section 3.2. Most geometric models are courtesy of the Stanford 3D Scanning Repository [SSR06]. Our method uses several approximations which make it efficient without compromising the quality of the reflections.

5.1. Performance

Performance was measured on a 3GB 3.4GHz Pentium 4 Xeon PC with a Quadro FX 3400 Nvidia graphics card.

Reflected billboards

The reflected billboard pixel program can handle up to 17 intersections in one pass, so $D+(R!=0)*(D+R-1) < 18$. We measured the frame rate dependence on the number of reflectors using the scene and view shown in Figure 5, *top*. The ground is modeled with a diffuse billboard connected to the environment, and each reflector is modeled with a reflective billboard which produces second order reflections. Enabling the 9 reflectors one by one produces the following frame rates [fps].

1	2	3	4	5	6	7	8	9
166	125	77	66	46	37	25	17	11

Six inter-reflecting teapots are handled comfortably. The limited pixel program length does not become a factor since the frame rate decreases before the maximum number of instructions is reached. The performance dependence on the number of pixels covered by the reflectors was measured by varying the resolution of the output image. For 4 reflectors (Figure 1), frame rates [fps] are as follows.

640x480	800x600	960x720	1120x840	1280x960
46	36	31	25	11

The frame rate for 4 reflectors in the second table is lower because the reflectors cover a larger screen area in Figure 1 than in Figure 5, *top*. The shader is complex so it

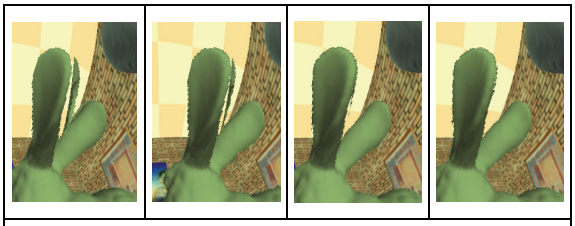


Figure 14: *Study of edge quality as a function of the angular increment. From left to right: 20°, 10°, 3°, and 2°.*

would pay off to shade only visible reflector fragments. This could be done by producing a reflected ray map in a first pass, and then shading it in a second pass.

Reflected depth maps

For the 256x256 bunny depth map used in Figure 3 and Figure 13 we construct *n=60* SRDMs (angular increment $180^o/60=3^o$). The total construction times [ms] for various numbers of poly-line vertices *m* is given below.

m	8	16	32	64
c.time	210	300	480	980

Even if 25% of the original depth samples are kept, the pre-processing takes less than a second. In practice we use *m* values between 8 and 16. The 210-300ms construction times open the door to building the SRDMs on the fly. This is particularly important for applications where several viewpoints are needed simultaneously, as for example in stereo. The aggregate memory footprint of the SRDMs is 960KB (60SRDMs x 256rows x 8vertices x 2floats x 4bytes), about four times the 256KB needed for the color or for the depth. The run time performance is also controlled by parameter *m*, which gives the number of iterations in the intersection search loop. The pixel program length limit is reached for *m=16*, which is more than enough for our test scenes. The SRDMs are stored as textures with 4 float channels, which allows fetching two poly-line vertices at a time. The frame rate [fps] dependence on *m* measured for the bottom image of Figure 13 is given below.

m	6	8	12	16
frame rate	25	20	17	16

5.2. Quality

Quality reflections are obtained with both types of impostors. Like environment mapping, our method is independent of reflector complexity (Figure 17).

For depth map impostors, quality is controlled by parameters *n* and *m*. A finer angular increment reduces the approximation error when mapping a ray projection to an SRDM row. Good results are obtained for *n=60* (3^o in Figure 14). Parameter *n* does not affect the run time performance, but an *n* value of 18 (i.e. 10^o) would reduce the SRDMs construction time to 70ms. We will investigate alleviating the edge artifacts for such large angular increments. One option is to lookup the two best rows and interpolate. A smaller price could be paid for the second row if the rows store additional information to correlate a row with its immediate neighbors. Parameter *m* controls the quality of the depth map approximation. Small *m* values (e.g. 8) produce remarkably good results. This is because the simplification occurs along each reflected ray, because the simplified depth map is *not* used for shading, and last but not least because the simplification error is small. The average relative z error [%] is given below for several *m* values and the same bunny depth map. The average is computed over all depth map texels and all SRDMs.

m	8	10	12	14	16
relative z error	1.43	0.9	0.7	0.5	0.4

Our method does not require reflected ray coherence and works equally well if the reflected rays are generated by vertex normal interpolation, bump mapping, or normal mapping. Therefore our method readily supports any material developed for environment mapped reflections. Moreover, the intersection code can easily provide the distance to the impostor, needed to simulate advanced effects inaccessible to environment mapping (Figure 15).

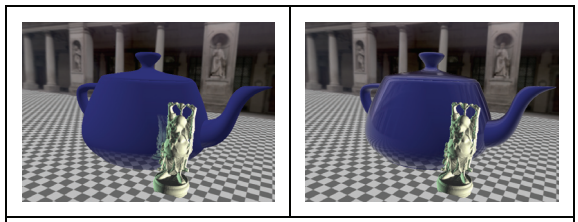


Figure 15: *Attenuation with distance (left) and Fresnel effects (right) used together in Figure 2.*

Curved reflectors complicate antialising since their convex and concave reflective surfaces introduce significant local perturbations of the spatial sampling rate. Sophisticated ray tracers handle the problem accurately, but at great expense. Both types of impostors used by our method are texture mapped, which enables antialiasing by mip-mapping. As in environment mapping, this approximate solution is effective and efficient (Figure 16).

6. Conclusions and future work

Rendering reflections by approximating the reflected scene works well. We have adapted depth maps, a powerful modeling primitive, to the context of reflections. Reflection rendering methods can be classified on a continuum based on the complexity of the geometry used to approximate the reflected scene. Environment mapping is at one extreme, with no geometry, reflected billboards are next, with a few quads, followed by reflected depth maps. Ray tracing is at the other extreme, since the reflected scene is used at its full complexity. We will continue to explore this continuum, to improve the reflected scene approximation while maintaining versatility and efficiency.

Our method for rendering reflected depth map impostors can be described as a GPU ray tracing method that combines specialized space partitioning—depth map projection of reflected ray—with specialized simplification—poly-line approximation along ray-aligned depth map rows. An accurate performance comparison with prior GPU ray tracing techniques is difficult. We are rendering the left image in Figure 3 at 16Hz. Since the reflective bunny covers a screen region of about 250x350pixels, we process approximately 1.4 million reflected rays per second. This figure includes the time needed to generate the reflected rays and to render the diffuse bunny. The ray-triangle intersection rate reported for the ray engine is 100Mtris/s [CCH02]. If the ray engine does not do anything else except for computing ray-triangle intersections, it achieves the same performance if 100/1.4=71 triangles are intersected for each reflected ray. Factoring in the time needed to identify which triangles to intersect, to render the diffuse part of the scene, and to compute the reflected rays reduces this figure even more. The ray engine computes intersections accurately, whereas our method requires lighter pre-processing.

Reflected depth map impostors reduce the problem of rendering reflections to the lesser problem of modeling and rendering with depth maps. Inevitably the problems of IBR by 3D warping are inherited. One such problem are disocclusion errors. Even if 6 depth maps could be processed efficiently, disocclusion errors would currently prevent us from using a depth cube map. The bunny would either be connected by unsightly rubber bands to the background, or would leave a background-colored shadow behind it when the view translates. LDIs are unappealing in our context since they lose the regularity of single layer depth images and the connectivity information, which makes the intersection expensive. Occlusion camera reference images [PA06] are a promising new solution which we will adapt to the context of reflections. The major hurdle seems to be the curved reflected ray projections, which complicate the creation and use of SRDMs.

We will work towards supporting self-reflections. Billboards do not support self-reflections since a reflected ray does not hit the billboard plane again. However, there is no fundamental obstacle preventing our depth map impostors to support self-reflections. Self-reflections can be produced by intersecting the reflected ray with a depth map of the reflector. The challenge is to provide an appropriate depth map for each point on the surface of the reflector. Clearly, more than a single depth map is needed. For example for the ears of the bunny to correctly reflect each other, at least two depth maps are needed. We will investigate optimizing the placement of reflector depth maps to minimize their number.

We will extend our method to handle view dependent lighting and shading. A specular highlight for example does not occur at the same locations on a surface and its reflection. We will explore using the true eye vector given by the reflected ray at each pixel. It is also our goal to improve the implementation. The reflected ray is a geometric primitive currently projected, clipped and rasterized in a pixel program. A more efficient approach is to handle the ray like all other primitives, with a vertex program followed by a pixel program. A difficulty we foresee is assembling the output image, which requires extensions for writing into texture memory within a pixel program. Finally, we will investigate constructing the SRDMs on the GPU, achieving full hardware support.

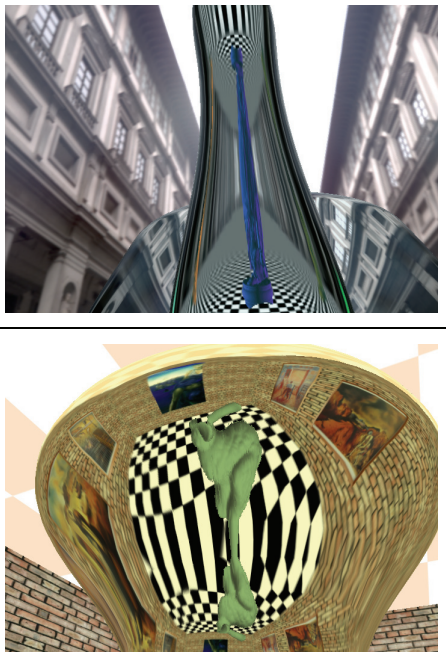


Figure 16: *Cases of extreme magnification and minification, handled well by reflected billboard (top) and depth map impostors (bottom).*

7. Acknowledgments

We thank the anonymous reviewers for their contribution towards improving this paper. This work was supported by the United States National Science Foundation through grants SCI-0417458 and CCR-0306214, and by the Computer Science Department at Purdue University. Equipment was donated by Intel and IBM.

References

[AL98] ALIAGA D., LASTRA A.: Smooth Transitions in Texture-Based Simplific. *Comp. & Graphics 22:1*, pp. 71-81, 1998

[AL99] ALIAGA D., LASTRA A.: Automatic image placement to provide a guaranteed frame rate. *SIGG '99*.

[Bas99] BASTOS R.: Increased Photorealism for Interactive Architectural Walkthroughs. *ACM Symposium on Interactive 3D Graphics* (1999), pp.183-190.

[Bjo04] BJORKE K.: Image-based lighting. *GPU Gems*, Fernando R., (Ed.). NVidia, (2004), pp. 307–322.

[BN76] BLINN J.F., NEWELL M.E.: Texture and Reflection in Computer Generated Images. *CACM 19:10*, 542-547, 1976.

[CHH02] CARR N., HALL J.D., HART J.C. The Ray Engine, Graphics Hardware (2002), pp. 1-10

[CON99] CABRAL B.: OLANO M., NEMEC P.: Reflection Space Image Based Rendering. In *Proc. of SIGG '99*, pp.165-170.

[DDSD03] DECORET X., DURAND F., SILLION F., DORSEY J.: Billboard clouds for extreme model simplification. *ACM Transactions on Graphics, volume 22, Issue 3, pp.689-696 2003*.

[Debwww] DEBEVEC P. Light Probe Image Gallery. http://www.debevec.org/Probes/

[Die96] DIEFENBACH P. J.: Pipeline Rendering: Interaction and Realism Through Hardware-Based Multi-Pass Rendering. PhD thesis, University of Pennsylvania, (June 1996).

[DYB98] DEBEVEC P., YU Y., BORSHUKOV G.: 1998. Efficient view-dependent image-based rendering with projective texture-mapping. In *EG Workshop on Rendering*, 105–116.

[GGS*96] GORTLER S., GRZESZCZUK R., SZELISKI R., COHEN M.: The Lumigraph. *In Proc. of SIGGRAPH '96*, 43-54.

[Gla89] GLASSNER, A. *An introduction to ray tracing*. Academic Press, 1989.

[GRE86] GREENE, N. Environment mapping and other applications of world projections. *IEEE CG&A*, 6:11, (1986).

[HAK01] HAKURA Z.: Parameterized Environment Maps. *In Proc. of ACM Symposium on Interactive 3D Graphics 2001* (2001), pp 203-208.

[Hal01] HALL, D., The ar350: Today's ray trace rendering processor. *In ACM SIGGRAPH / Eurographics conference on Graphics hardware - Hot 3D Presentations.*

[Hes99] HEIDRICH W.: Light Field Techniques for Reflections and Refractions. *EG Workshop on Rendering* (1999), pp.195-375.

[HM92] HANRAHAN P., MITCHELL D.: Illumination from curved reflectors. *In Proc. of SIGG. '92*, ACM Press, pp. 283–291.

[HS93] HAEBERLI P., SEGAL M.: Texture mapping as a fundamental drawing primitive. *In Eurographics Workshop on Rendering* (1993), 259-266.

[LH96] LEVOY M., HANRAHAN P.: Light Field Rendering. *Proc. of SIGGRAPH 96* (1996), 31-42.

[LR98] LISCHINSKI D., RAPPOPORT A.: Image-Based Rendering for Non-Diffuse Synthetic Scenes. *Eurographics Workshop on Rendering 1998* (1998), pp.301-314.

[MB95] McMILLAN L., and BISHOP G.: Plenoptic modeling: An image-based rendering system. *In Proc. SIGG. '95*, pages 39-46.

[MB97] McREYNOLDS T., BLYTHE D.: *Programming with OpenGL: Advanced Rendering. SIGGRAPH '97 course*.

[McM97] McMILLAN, L. An Image-Based Approach to Three-Dimensional Computer Graphics (1997). PhD thesis, Univ. of North Carolina at Chapel Hill,.

[Mil98] MILLER G.: Lazy Decompression of Surface Light Fields for Precomputed Global Illumination, *EGW on Rendering* (1998).

[MS96] MACIEL P., AND SHIRLEY P. Visual Navigation of Large Environments Using Textured Clusters, *Symposium on Interactive 3D Graphics* (1995) pp 95-102

[OR98] OFEK E., RAPPOPORT A.: Interactive reflections on curved objects. *In Proc. of SIGGRAPH '98*, ACM Press, 333-342.

[PA06] POPESCU, V. and D. ALIAGA: Depth Discontinuity Occlusion Camera, In *Proc. of ACM Symposium on Interactive 3D Graphics and Gaming,* 2006.

[Paj02] PAJDLA T.: Geometry of Two-Slit Camera. (2002) *Research Report CTU–CMP–2002–02*.

[Par99] PARKER, S.: Interactive ray tracing. *ACM Symposium on Interactive 3D Graphics* (1999), 119–126.

[PBM*02] PURCELL T.J., BUCK I., MARK W. Ray Tracing on Programmable Graphics Hardware, *ACM Transactions on Graphics. 21 (3), pp. 703-712, 2002.*

[RSH05] RESHETOV A., SOUPIKOV R., HURLEY J.: Multi-Level Ray Tracing Algorithm, *ACM Transactions on Graphics. volume 24, Issue 3, pp. 1176-1185, 2005.*

[SALP05] SZIRMAY-KALOS L., et al.: Approximate Ray-Tracing on the GPU with Distance Impostors. *Computer graphics forum, 24(3), 2005. pp. 171-176.*

[Sch97] SCHAUFLER G.: Nailboards: A Rendering Primitive for Image Caching in Dyn. Scenes. *EG Workshop on Render.* (1997).

[SGH98] SHADE J., GORTLER S., HE L., SZELISKI R.: Layered Depth Images, *In Proc. of SIGG 98* (1998), 231-242.

[SSR06] The Stanford 3D Scanning Repository, *http://graphics.stanford.edu/data/3Dscanrep/*

[TK96] TORBORG J., KAJIYA J.: Talisman: Commodity realtime 3d graphics for the pc. *In Proc. of SIGGRAPH '96* (1996).

[VF94] VOORHIES D., FORAN J.: Reflection vector shading hardware. *In Proc. SIGGRAPH '94* (1994), pp. 163-166.

[WAA*00] WOOD D.N., AZUMA D. I., ALDINGER K.: Surface light fields for 3D photography. *In Proc. of SIGG '00,* pp. 287-296.

[Wal01] WALD I.: Interactive rendering with coherent ray tracing. *Computer Graphics Forum 20, 3* (2001), 153–164.

[Whi80] WHITTED T.: An improved illumination model for shaded display. *Comm. Of the ACM* (1980), 23, 6, pp. 343-349.

[WSB01] WALD I., SLUSSALEK P., BENTHIN C.: Interactive distributed ray tracing of highly complex models. *Eurographics Workshop on Rendering 2001*, pp. 277–288.

[WSE04] WEISKOPF D., SCHAFHIZEL T., ERTL T. GPU-Based Nonlinear Ray Tracing, *Computer graphics forum volume 23,Issue 3, 2004, pp. 625-633.*

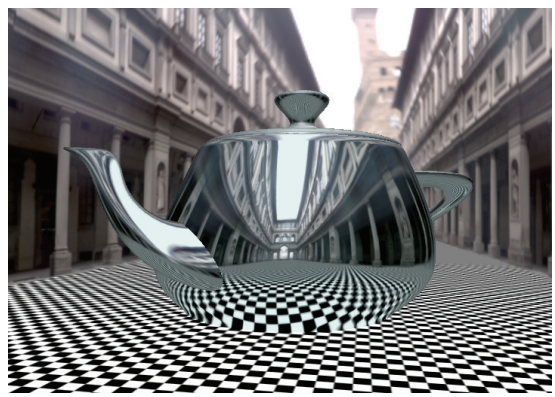


Figure 17: *Reflector distorted in real-time.*