Nonpinhole Approximations for Interactive Rendering

Paul Rosen ■ *University of Utah*

Voicu Popescu ■ *Purdue University*

Kyle Hayward ■ *Human Head Studios*

Chris Wyman ■ *University of Iowa*

Orthographic or perspective images are often used to approximate geometry for rendering high-order effects. These approximations capture only what's visible from one direction or viewpoint. Nonpinhole cameras can improve approximation quality at little additional cost. These proposed pinhole cameras demonstrate these advantages by way of reflections, refractions, relief texture mapping, and ambient occlusion.

In the quest for higher-quality and higher-performance rendering, researchers have resorted to approximating scene geometry with more efficient representations. Such representations should have three main properties. First, they should sufficiently capture the geometry they replace such that the resulting images are virtually indistinguishable from the images obtained when rendering with the original geometry. Second, to support dynamic scenes, the representations must be created on the fly, which requires fast construction. Finally, the alternative representation must deliver the desired performance boost to the application.

We distinguish between two types of applications:

- applications in which the representation can be rendered directly with the conventional feed-forward approach of projection followed by rasterization, such as when a distant tree is rendered using a billboard (for more on billboards, see the "Image-Based Geometry Approximation" sidebar), and
- applications in which the representation must be rendered by intersection with one ray at a time, such as for dealing with reflections, refractions, relief texture mapping, and ambient occlusion.

This article focuses on the second type of application. For such applications, fast computation of the intersection between a ray and the alternative geometry representation is a central concern.

Images enhanced with per-pixel depth have two of the desired properties. Efficient construction of a depth image employs graphics hardware to render the geometry that the image replaces. The projection of a ray onto the depth image is a segment. This reduces the dimensionality of the search space for the intersection of the ray and depth image from two to one, enabling fast intersection. Modern graphics hardware allows stepping along the ray projection, per pixel, at interactive rates. However, depth images are acquired either from a single viewpoint (with a planar pinhole camera) or along a single view direction (with an orthographic camera), which limits their geometry-modeling power. Such depth images miss surfaces that become visible when the application renders the geometry approximation; this lowers the results' quality.

We propose constructing depth images using nonpinhole cameras. Such depth images offer a high-fidelity approximation of scene geometry while keeping construction and rendering costs low. Because a nonpinhole camera's rays don't have to pass through one point, we can design them to sample all surfaces exposed by the application. To ensure construction and rendering efficiency, we design

 Published by the IEEE Computer Society

Image-Based Geometry Approximation

Paolo Maciel and Peter Shirley coined the term *impostor*, which now widely denotes an image-based simplified representation of geometry to make rendering more efficient.[1] The simplest impostor is a *billboard*, a quad texture mapped with the image of the original geometry, with transparent background pixels. Billboard construction is efficient, intersecting a billboard with a ray is trivial, and billboards provide good approximations of geometry seen orthogonally from a distance. However, when the billboard is near the viewer, the drastic approximation of geometry is unacceptable.

Billboard clouds use several quads to improve modeling quality.[2] The quads and the assignment to the original geometry are optimized for modeling fidelity. The number of quads is small, and a billboard cloud can be intersected with a ray one quad at a time. However, the optimization makes constructing a billboard cloud a lengthy process that precludes dynamic scenes. Moreover, the approximation quality is still insufficient for close-up viewing.

Depth images greatly improve billboards' modeling power.[3] Constructing a depth image is just as inexpensive as constructing a billboard, but the cost of intersection with a ray is no longer constant but linear in the depth image width. To avoid searching for the intersection in the entire image, depth image construction leverages epipolar-like constraints: the intersection is known to belong to the image plane projection of the ray. Because the depth image is constructed with an orthographic or a perspective projection, it captures only samples visible along the reference direction or from the reference viewpoint. When the application exposes surfaces the depth image didn't capture, objectionable disocclusion artifacts occur.

The simplest method for alleviating disocclusion errors is to use additional depth images,[4] which is expensive and only palliative. A breakthrough came with the introduction of layered representations such as the multilayered z-buffer[5] and the layered depth image,[6] which allow for more than one sample along a ray and control disocclusion errors effectively. However, expensive construction restricts layered representations to static scenes. Moreover, the lack of a connected representation makes ray intersection difficult, precluding rendering effects such as those we address in the main article.

As graphics hardware evolved, intersecting depth images with individual rays became possible, and research shifted to rendering effects involving higher-order rays.

References

1. P. Maciel and P. Shirley, "Visual Navigation of Large Environments Using Textured Clusters," *Proc. 1995 Symp. Interactive 3D Graphics* (I3D 95), ACM Press, 1995, pp. 95–102.
2. X. Decoret et al., "Billboard Clouds for Extreme Model Simplification, *ACM Trans. Graphics*, vol. 22, no. 3, 2003, pp. 689–696.
3. L. McMillan and G. Bishop, "Plenoptic Modeling: An Image-Based Rendering System," *Proc. Siggraph*, ACM Press, 1995, pp. 39–46.
4. W. Mark, L. McMillan, and G. Bishop, "Post-rendering 3D Warping," *Proc. 1997 Symp. Interactive 3D Graphics* (I3D 97), ACM Press, 1997, pp. 7–16.
5. N. Max and K. Ohsaki, "Rendering Trees from Precomputed *Z*-buffer Views," In *Rendering Techniques '95: Proc. Eurographics Rendering Workshop*, Eurographics Assoc., 1995, pp. 45–54.
6. J. Shade et al., "Layered Depth Images," *Proc. Siggraph*, ACM Press, 1998, pp. 231–242.

the nonpinhole camera to provide fast projection. This enables constructing the nonpinhole depth image in feed-forward fashion by projection followed by rasterization. We leverage the closed-form, unambiguous projection of the nonpinhole camera a second time, during rendering, to compute the ray's projection onto the nonpinhole image.

Why Use Nonpinhole Cameras?

Nonpinhole depth images provide advantages regarding reflection, refraction, relief texture mapping, and ambient occlusion (also see the accompanying video at http://doi.ieeecomputersociety.org/10.1109/MCG.2011.32).

Reflection and Refraction

In Figure 1, the nonpinhole depth image captures all the teapot surfaces exposed in the reflections, whereas a conventional depth image produces incomplete reflections. We find the intersection between a reflected ray and the depth image by searching along the curved projection of the ray (see Figure 2). This differs from planar pinhole cameras, where the ray projection is a straight line.

We similarly support refractions by intersecting the depth image with refracted rays (see Figure 3).

Relief Texture Mapping

Nonpinhole cameras greatly enhance the modeling power of relief texture mapping. A single-layer nonpinhole relief texture captures the entire top and sides of a barrel or an entire car to produce quality frames with rich detail (see Figure 4a). In contrast, a single-layer conventional relief texture misses a considerable part of the barrel (see Figure 4b). A multilayered relief texture is impractical in this case because a proper sampling of the barrel's side would require a prohibitively large number of layers.

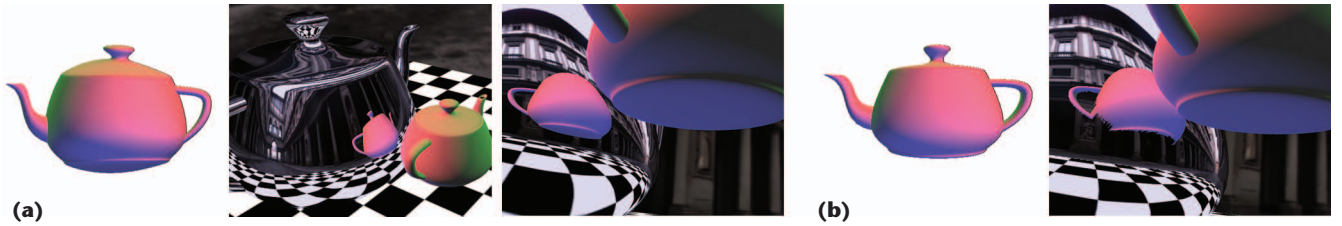


Figure 1. Reflections calculated using nonpinhole versus conventional images. (a) A nonpinhole depth image of a teapot and the reflections rendered with it. (b) A conventional depth image and reflection rendered with it. The nonpinhole depth image captures the teapot's lid and bottom and produces quality reflections; the conventional depth image produces incomplete reflections.

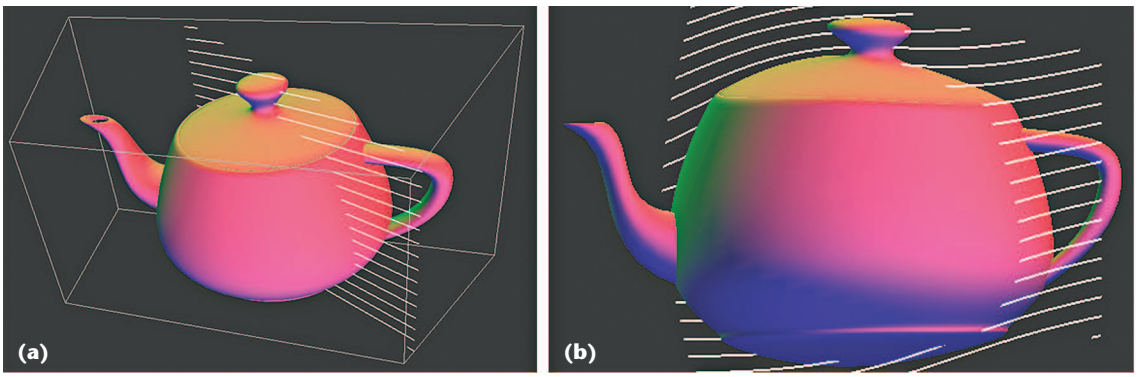


Figure 2. Visualization of (a) rays in 3D and (b) their curved projection on the nonpinhole depth image in Figure 1. The curved rays of the nonpinhole allow access to regions not visible from a perspective or orthographic view.

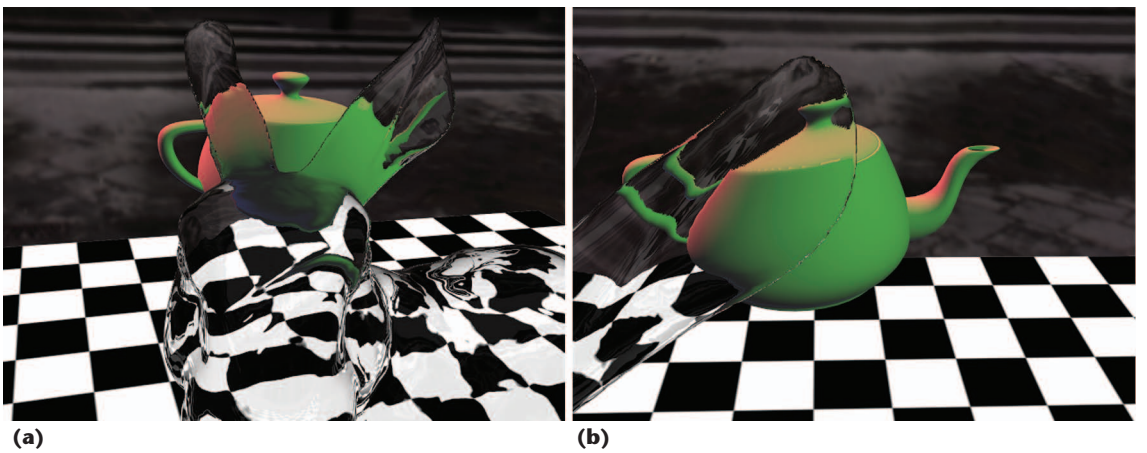


Figure 3. Refractions calculated using a nonpinhole depth image of a teapot similar to the one in Figure 1. Whereas accurate reflections might require many conventional views to calculate, nonpinholes can be used to circumvent this limitation.

Figure 4. Nonpinhole versus conventional relief texture mapping. (a) Nonpinhole relief texture maps and the frames rendered with them. (b) Conventional relief texture mapping. The nonpinhole relief texture captures the top and sides of the barrel and car; the conventional relief texture misses a considerable part of the barrel.

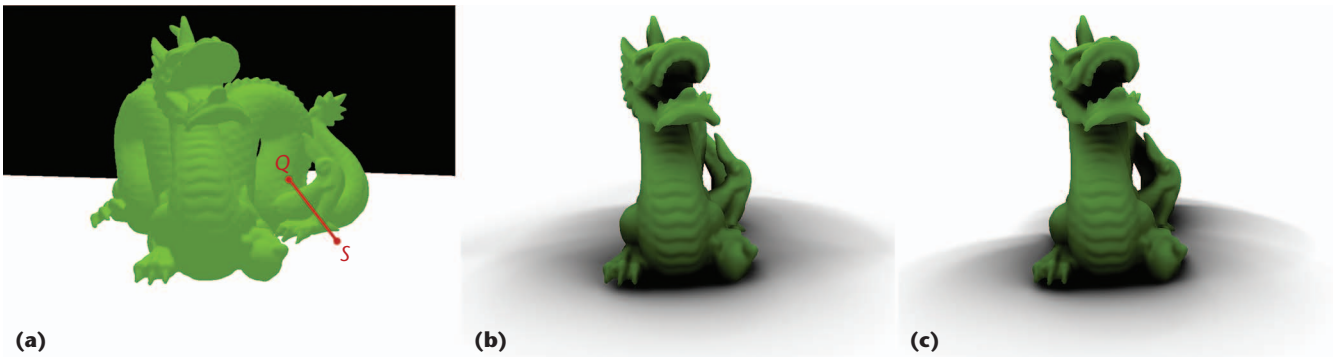


Figure 5. Screen-space ambient occlusion calculated using nonpinhole versus conventional images. (a) A nonpinhole *z*-buffer illustrated with color (*Q* and *S* indicate two samples). (b) The ambient occlusion rendered using it. (c) The ambient occlusion rendered using the output image *z*-buffer. The nonpinhole *z*-buffer captures hidden parts of the dragon for a more complete and more stable ground shadow.

Ambient Occlusion

A promising technique for rendering with ambient occlusion at interactive rates is to use the output image *z*-buffer to approximate the exposure of output image samples to the environment.[1] However, the amount of occlusion for visible samples also depends on samples not visible in the output image. This makes the ambient occlusion unstable, with dark regions appearing and disappearing as geometry subsets appear and disappear in the output image. We propose using a nonpinhole *z*-buffer that captures most samples needed to estimate the occlusion of output image samples, producing more complete and more stable ambient occlusion (see Figure 5).

Nonpinhole-Camera Depth Images

Abandoning the pinhole constraint lets us design the camera model to obtain the depth image best suited for the application and dataset at hand. Before we go into detail into our method, let's look at construction and ray intersection for nonpinhole depth images in general.

Construction

Assume we have a nonpinhole camera with fast projection that maps a 3D point (x, y, z) to (u, v, z), where (u, v) indicates the image coordinates and z is a measure of depth linear in the image space. We can efficiently construct a nonpinhole depth image by projecting the vertices of the geometry that it will approximate and rasterizing the projected triangles conventionally. The unconventional projection occurs in a vertex program that implements the nonpinhole-camera model. Because lines don't necessarily project to lines anymore and because rasterization parameters don't vary linearly (before the perspective divide) anymore, the triangles must be sufficiently small to provide an adequate approximation. Complex objects are frequently modeled with small triangles, so additional tessellation usually is unnecessary. We can subdivide meshes with large triangles on the fly by exploiting primitive-level GPU programmability.

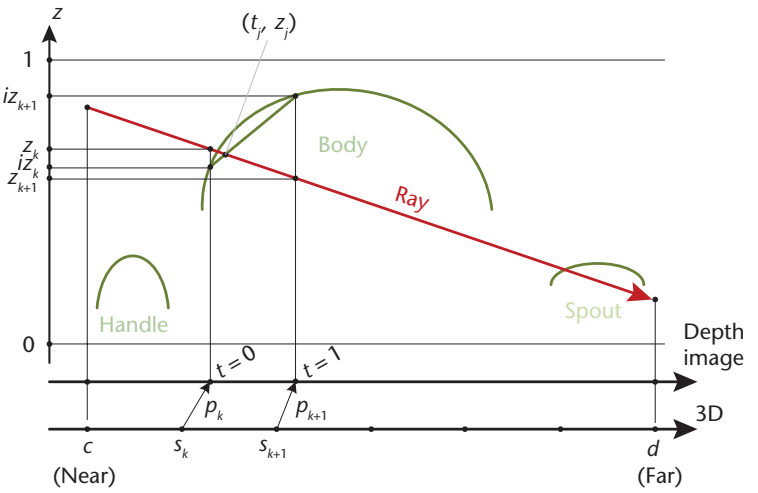


Figure 6. The intersection of a ray and a nonpinhole depth image, for a depiction of a teapot. i_z indicates the image depth, (c, d) indicates a segment, (s_k, s_{k+1}) indicates a subsegment, p indicates a pixel, t is the parameter locating the intersection along a segment, and u and v are the image coordinates.

Ray Intersection

Like a regular depth image, a nonpinhole depth image is defined by an image with color and depth per pixel and by a camera model that allows projection. We compute the intersection of a ray (a, b) with a nonpinhole depth image *NPI* in two main steps.

First, we clip segment (a, b) with the bounding volume of *NPI* to obtain segment (c, d) (see Figure 6).

Second, we interpolate (c, d) in 3D from near to far to create n subsegments. For each subsegment (s_k, s_{k+1}), we project s_k and s_{k+1} to the depth image at pixels $p_k = (u_k, v_k, z_k)$ and $p_{k+1} = (u_{k+1}, v_{k+1}, z_{k+1})$. Next, we perform a lookup of image depths iz_k, iz_{k+1} at (u_k, v_k), (u_{k+1}, v_{k+1}). We then intersect $[(0, z_k), (1, z_{k+1})]$ and $[(0, iz_k), (1, iz_{k+1})]$ in 2D segments to obtain intersection (t_j, z_j), where t is the parameter

(a) (b)

Figure 7. Visualizations of samples stored by (a) a conventional depth image and (b) a single-pole occlusion camera (SPOC) depth image. The SPOC depth image captures about half of the teapot, which is sufficient to intersect most of the reflected rays that will be needed to produce complete reflections.

locating the intersection along segment (p_k, p_{k+1}). If (t_j, z_j) is valid, we return the depth image color ic_j at lerp((u_k, v_k), (u_{k+1}, v_{k+1}), t_j). Otherwise, we continue with the next subsegment.

We interpolate the ray in 3D because its projection isn't a straight line and we can't simply rasterize the segment connecting the endpoint projections. We project each intermediate point with the nonpinhole camera to trace the curved projection correctly. Because the depth z stored by the depth image varies linearly in the image, we can efficiently compute the intersection in a 2D space (t, z).

Specializing the Depth Images

Here, we show how we construct depth images by adapting two recent nonpinhole camera models: the *single-pole occlusion camera* (SPOC)[2] and *graph camera*.[3]

The SPOC

The SPOC reaches around an object's silhouette to gather samples not visible from the reference viewpoint but near the silhouette. Such "barely" occluded samples are needed when the depth image is sampled from nearby viewpoints. The SPOC is well suited for approximating single objects; we used it for Figures 1 to 4.

The SPOC projection uses a conventional projection followed by a distortion that moves the projected sample away from a pole.[2] The pole is the projection of the object's center. The distortion's magnitude increases with depth, so deeper samples move more, escaping the occluding front surface. For the SPOC depth images in Figures 1 and 4, the distortion pushes the object silhouettes back, revealing the teapot's lid and bottom, the barrel's entire side, and the car's side and wheels. Figure 7 shows that the SPOC depth image captures about half of the teapot, which is sufficient to intercept all reflected rays that would intersect the original teapot geometry.

SPOC construction and intersection closely follow the algorithms we described in the previous section. We choose the number of subsegments n as the Euclidian distance between the projection of the endpoints of the clipped ray. This provides a good approximation of the actual number of pixels covered by the curved projection of the ray. Figure 2 visualizes the curved projections of a set of coplanar rays.

The Graph Camera

To construct a graph camera, we take a planar pinhole camera and perform a series of frustum bending, splitting, and merging operations. The result is a graph of planar pinhole cameras. The graph camera circumvents occluders to sample an entire 3D scene in a single-layer image. In Figure 8, the graph camera depth image models the entire reflected scene.

For the graph camera, we generalize the concept of a camera ray to the set of points projecting at the same given image location, which allows for rays that aren't straight lines but are piecewise linear. A ray changes direction as it crosses the shared face separating two frustums, but it remains continuous, which makes the graph camera image continuous. Figure 9 shows the graph camera we constructed for the maze in Figure 8. The construction used a breadth-first traversal starting at the entrance.

Projecting a point with the graph camera takes two steps. First, we find the frustum containing the given 3D point. Then, we project the point directly to the output image with a 4D matrix that concatenates the projections of all the cameras on the path from the current frustum to the root. To find the frustum containing the point, we can use an octree or another hierarchical space subdivision.[3] Using the projection, construction of the graph camera depth image proceeds as in the section "Construc-

tion," except that a triangle must be processed with each frustum it intersects.

To intersect a graph camera depth image with a ray, we can follow the generic algorithm: interpolate the ray uniformly in 3D space, and project each new point onto the graph camera image. However, because each frustum is a pinhole, a ray's projection is piecewise linear (see Figure 10), which enables the following optimization. Given a ray r, for each graph camera frustum F_i, we follow these steps:

1. Intersect ray r with F_i to produce 3D subsegment (s_i, e_i).
2. Project segment (s_i, e_i) to graph camera image segment (p_i, q_i).
3. Walk on (p_i, q_i) to find the intersection.

The algorithm computes the ray's linear pieces directly by intersecting the ray with all the frustums, producing a set of subsegments (s_i, e_i). This is more efficient than the generic algorithm, which requires small 3D steps just to model the breaking points of the piecewise-linear projection with high fidelity. Each frustum is a planar pinhole camera, so each subsegment projection remains a straight line segment (p_i, q_i) in the output graph camera image. The optimized algorithm interpolates the subsegment to search for the intersection step by step, similarly to the generic algorithm.

Applications to Interactive Rendering

Nonpinhole depth images accelerate reflection, refraction, relief texture mapping, and ambient occlusion as follows.

Reflection

To render a frame of a scene with specular reflections, we first update the depth images that approximate the reflected geometry that's dynamic. Then, we render each reflector by computing a reflected ray per pixel, a step similar to environment mapping, and by intersecting the reflected ray with the reflected geometry's depth images. (For more on environment mapping, see the "Using Depth Images to Accelerate Rendering" sidebar.)

As with environment-mapped reflections, the process produces multiple reflections of the same object at no extra cost. Consider Figure 11a, in which the concave bunny reflects the teapot multiple times. We compute the reflection one ray at the time; the fact that two or more rays reflect the same 3D point has no bearing on the method's cost.

Our method supports fully dynamic scenes because it doesn't require precomputation involving the reflector and it efficiently computes the nonpinhole depth images. It supports higher-order reflections by storing per-pixel normals. We use the normal at a first intersection point to create a second-order reflected ray that intersects depth images again (see Figure 11b).

When deciding how to approximate reflected geometry, we want to devise the simplest approximation that captures all samples visible in reflections. For example, a billboard perfectly captures the black-and-white ground plane in the reflections in Figure 11. The billboard captures the SPOC

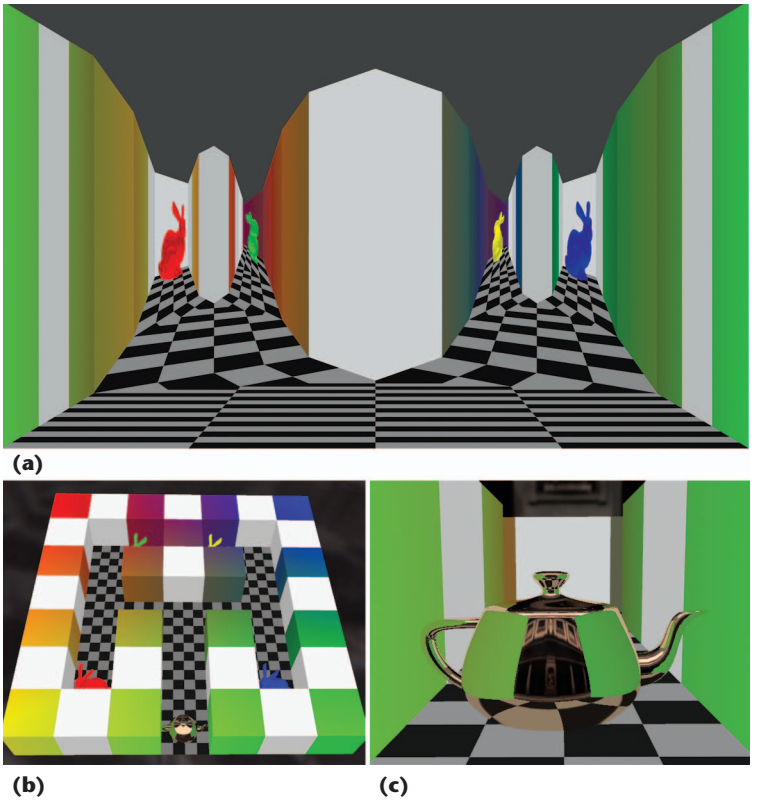


Figure 8. Using a graph camera for reflections. (a) A graph camera depth image capturing an entire 3D maze. (b) An overhead view of the maze. (c) The reflection rendered using the depth image.

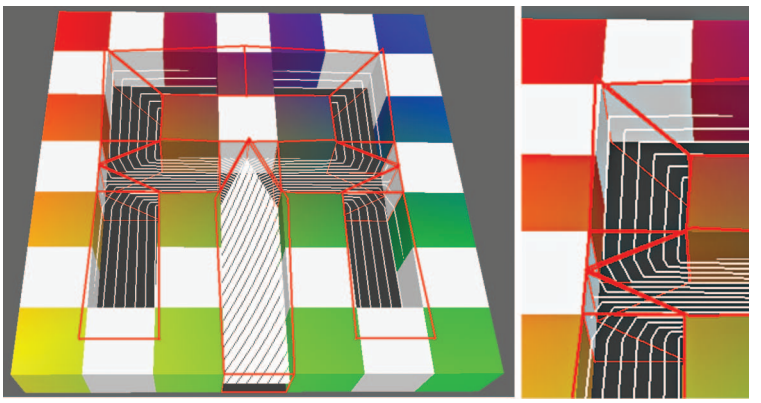


Figure 9. A graph camera model for the maze in Figure 8. The frustums are red; a few rays are shown in white. The construction used a breadth-first traversal starting at the entrance.

Using Depth Images to Accelerate Rendering

Here, we look at previous research applying depth images to reflection, refraction, relief texture mapping, and ambient occlusion.

Reflection and Refraction

Although interactive-rendering research has extensively studied reflection and refraction, no complete solution exists. We assign reflection and refraction rendering techniques to four groups: ray tracing,[1] image-based rendering,[2] projection,[3] and reflected- or refracted-scene approximation. Here, we discuss only the last group, which is the most relevant to our research.

Environment mapping approximates reflected scenes with a cube map.[4] It's the preferred approach for interactive applications owing to its efficiency, robustness, and good results when the scene geometry is far from the reflector or refractor. However, it performs poorly when the scene geometry is near the reflector or refractor. Approximating the scene with a sphere improves the results,[5] but few environments are spherical, so the fidelity is still quite limited. Using depth images can improve scene approximation.[6,7] Environment mapping produces quality reflections for simple objects or select viewpoints, but its insufficient coverage is a limitation for nontrivial scenes or wide viewpoint translations.

Compared to reflection, refraction rays require additional work because most rays interact with the refractor at least twice—once when entering the object and once when leaving it. Researchers have developed several techniques for computing the second refraction at interactive rates, including precomputed distance fields,[8] GPU ray tracing,[9] and image-space approximations.[10]

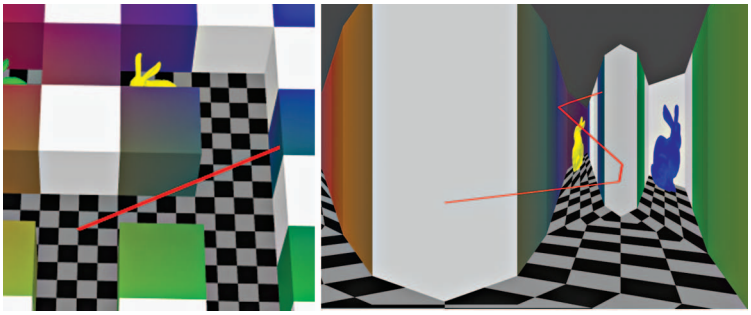
Relief Texture Mapping

Another rendering effect that requires intersecting depth images with individual rays is relief texture mapping.[11] This approach adds true geometric detail to a coarse model by texturing each triangle with a height map. A conventional relief texture samples surface detail orthographically, along the direction of the normal of the underlying coarse model, which limits the technique to height-field surfaces. Even so, sampling degrades when the geometric detail becomes aligned with the normal of the underlying surface.

Researchers have extended relief texture mapping to non-height-field surface detail by resorting to a relief texture with multiple layers, each sampled orthographically.[12] The extension works well when complex detail can be captured in a few layers, as with a chain link fence, for example. This extension can capture double-sided detail. However, capturing geometric detail perpendicular to the underlying surface remains challenging because it requires many layers. Our research extends relief texture mapping in an orthogonal direction; multilayered nonpinhole relief textures could be developed to exploit both techniques' advantages.

Ambient Occlusion

Ambient-occlusion techniques add realism to local illumination models by approximating the amount of light a surface point receives on the basis of how much of the environment is hidden from the point by nearby geometry. The computational cost is high because a ray must be cast from each point in all directions. The first implementations precomputed ambient occlusion in model space offline,[13]



(a) (b)

Figure 10. Visualization of (a) a 3D ray and (b) its piecewise linear projection in a graph camera image. This projection lets us optimize the ray intersection algorithm.

depth image of the teapot along the direction that connects the centers of the bunny and teapot. We set the SPOC's field of view to the smallest value that encompasses the teapot. The SPOC allows for some flexibility in tuning the disocclusion. For the teapot, more disocclusion means a better sampling of the lid and bottom and pushing back the body's silhouette. However, a single SPOC can't disocclude the entire teapot because the handle and spout will start occluding the body. If the teapot were spinning or if multiple reflectors surrounded the teapot, the best solution would be to use two depth images capturing complementary halves of the teapot. SPOCs are suitable for capturing individual objects, and graph cameras are suitable capturing an entire environment.

We want to select the minimum depth image resolution that captures the reflected geometry well. This way, we obtain the best reflection regardless of the rate at which the depth image is sampled by reflected rays in any given frame. Consider a divergent pattern of rays, as obtained off a convex reflective surface or a concave reflective surface beyond the convergence point. Such a pattern will minify the reflected object, which is handled straightforwardly through mip-mapping. Like conventional depth images, nonpinhole depth images have the great advantage

which precluded dynamic scenes. Initial attempts to use GPUs to accelerate ambient occlusion employed many (128 to 1,024) spherical shadow maps of the scene.[14]

Our nonpinhole *z*-buffer ambient-occlusion method builds on an image-space technique that Louis Bavoil and his colleagues introduced.[15] Their technique approximates the amount of ambient occlusion at an output pixel using the output image *z*-buffer. They noticed that to sample occlusion at a pixel for an entire half plane, traversing one *z*-buffer segment is sufficient. The result is fast ambient occlusion that supports dynamic scenes. However, the technique computes ambient occlusion as if the geometry seen by the output image were the only geometry in the scene, which can cause missing and unstable ambient-occlusion artifacts.

References

1. T. Whitted, "An Improved Illumination Model for Shaded Display," *Comm. ACM*, vol. 23, no. 6, 1980, pp. 343–349.
2. P. Debevec, Y. Yu, and G. Borshukov, "Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping," *Proc. 1998 Eurographics Workshop Rendering*, Eurographics Assoc., 1998, pp. 105–116.
3. E. Ofek and A. Rappoport, "Interactive Reflections on Curved Objects," *Proc. Siggraph*, ACM Press, 1998, pp. 333–342.
4. J.F. Blinn and M.E. Newell, "Texture and Reflection in Computer Generated Images," *Comm. ACM*, vol. 19, no. 10, 1976, pp. 542–547.
5. K. Bjorke, "Image-Based Lighting," *GPU Gems*, R. Fernando, ed., Addison-Wesley, 2004, pp. 307–322.
6. L. Szirmay-Kalos et al., "Approximate Ray-Tracing on the GPU with Distance Impostors," *Computer Graphics Forum*, vol. 24, no. 3, 2005, pp. 171–176.
7. V. Popescu et al., "Reflected-Scene Impostors for Realistic Reflections at Interactive Rates," *Computer Graphics Forum*, vol. 25, no. 3, 2006, pp. 313–322.
8. B. Chan and W. Wang, "Geocube—GPU Accelerated Real-Time Rendering of Transparency and Translucency," *The Visual Computer*, vol. 21, nos. 8–10, 2005, pp. 579–590.
9. D. Roger, U. Assarsson, and N. Holzschuch, "Whitted Ray-Tracing for Dynamic Scenes Using a Ray-Space Hierarchy on the GPU," *Proc. 2007 Eurographics Symp. Rendering*, Eurographics Assoc., 2007, pp. 99–110.
10. C. Wyman, "An Approximate Image-Space Approach for Interactive Refraction," *ACM Trans. Graphics*, vol. 24, no 3, pp. 1050–1053.
11. F. Policarpo, M. Oliveira, and J. Comba, "Real-Time Relief Mapping on Arbitrary Polygonal Surfaces," *Proc. 2005 Symp. Interactive 3D Graphics and Games* (I3D 05), ACM Press, 2005, pp. 155–162.
12. F. Policarpo and M. Oliveira, "Relief Mapping of Non-Height-Field Surface Details," *Proc. 2006 Symp. Interactive 3D Graphics and Games* (I3D 06), ACM Press, 2006, pp. 55–62.
13. S. Zhukov, A. Iones, and G. Kronin, "An Ambient Light Illumination Model," *Proc. 1998 Eurographics Rendering Workshop*, Eurographics Assoc., 1998, pp. 45–56.
14. M. Pharr and S. Green, "Ambient Occlusion," *GPU Gems*, R. Fernando, ed., Addison-Wesley, 2004, pp. 279–292.
15. L. Bavoil, M. Sainz, and R. Dimitrov, "Image-Space Horizon-Based Ambient Occlusion," *Siggraph 2008 Talks*, ACM Press, 2008, article 22.

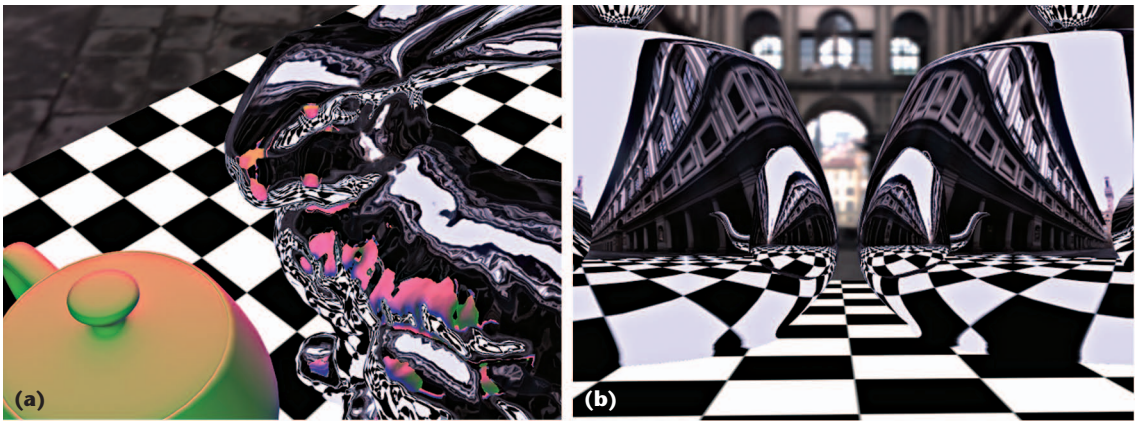
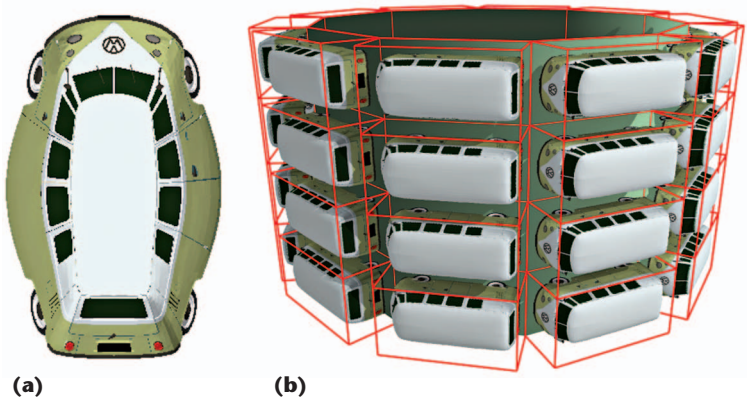


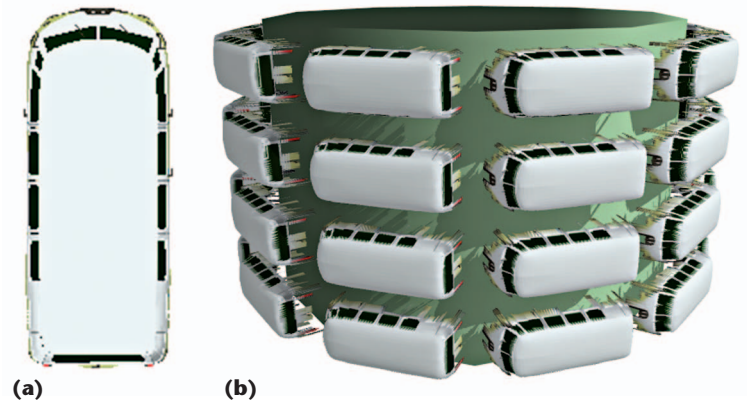
Figure 11. Reflections: (a) Multiple. (b) Second-order. Using a nonpinhole depth image, multiple reflections can be calculated at no additional cost. Second- and higher-order reflections can also be calculated at the cost of an additional ray–depth-image intersection for each ray.

of reducing the problem of geometry minification to the much simpler problem of image minification. A convergent pattern of rays, on the other hand, magnifies the reflected object, and the reflection will become undersampled once the sampling rate exceeds the depth image's resolution. Because this is also the original geometry's resolution, the problem can't be imputed to the depth image.



(a) (b)

Figure 12. Relief texture mapping. (a) The SPOC relief texture. (b) The output image with a relief bounding-box visualization. Rendering the bounding boxes lets us obtain correct silhouettes.



(a) (b)

Figure 13. Relief texture mapping, continued. (a) A conventional relief texture. (b) The output image. The conventional relief texture misses the wheels and severely undersamples the car's sides.

Refraction

As we mentioned before, we render refractions by intersecting the emerging ray with the depth images that approximate the geometry. The algorithm for computing emerging refracted rays is orthogonal to the research we report here. To perform this computation, we use an image-space approximation.[4] Basically, the approximation uses a first rendering pass to store depth and surface normals for back-facing surfaces, which a second pass then uses to compute the ray emerging after a second refraction.

Relief Texture Mapping

We trigger relief-texture-map rendering by rendering the primitives of the coarse underlying model. To obtain correct silhouettes, we render each relief tile's bounding box (see Figure 12). For every pixel, we transform the eye ray to the current relief tile's coordinate system, and intersection proceeds as before. We compute the world space z at the intersection for correct z-buffering with the rest of the scene and for casting and receiving correct shadows. We could compute shadows by shooting a second ray from the intersection to the light source and intersecting it with the relief texture. Instead, we use a conventional shadow map such that the relief surface casts and receives shadows from other objects and other relief tiles.

Nonpinhole relief textures capture complex objects in a single layer. Figure 13 shows that a conventional relief texture misses the wheels and severely undersamples the car's sides. When tuning the nonpinhole-camera parameters, our only concern is to capture the relief adequately, independently of the underlying base geometric model. With the increased complexity of the geometry modeled with the relief texture comes the desire to modulate the appearance of individual instances of that texture. We obtained the different-colored cars in Figure 4 with a single relief texture by simply modifying the color of the intersection if it corresponds to the car body, identified through its yellow color.

Ambient Occlusion

The horizon-based screen-space algorithm[1] produces plausible ambient occlusion at little cost and therefore has great appeal for interactive rendering applications. It measures ambient occlusion solely on the basis of the output image z-buffer. In other words, it approximates the entire scene's geometry with the point samples visible from the current viewpoint. This approximation is justified by the heuristic that for simple scenes, the geometry occluding an output image pixel will likely be visible from the current viewpoint and thus sampled by the output image z-buffer.

However, this heuristic breaks down for more complicated geometry. Some output image pixels are occluded by geometry that isn't visible from the current viewpoint. The occluding geometry isn't represented in the output image z-buffer, and the algorithm fails to assign the appropriate ambient occlusion to these pixels. The output image z-buffer underestimates the geometry responsible for occluding the output image samples; consequently, the algorithm underestimates the ambient occlusion.

Moreover, as the output view changes, occluding geometry can appear and disappear in the output image z-buffer, which causes the ambient occlusion to appear and disappear. When only the view changes, the ambient occlusion should be stable. However, the instability is quite noticeable, incorrectly suggesting that the lighting changes (see the accompanying video).

Nonpinhole cameras can overcome this fundamental limitation of the horizon-based screen-space algorithm. The main idea is to compute the

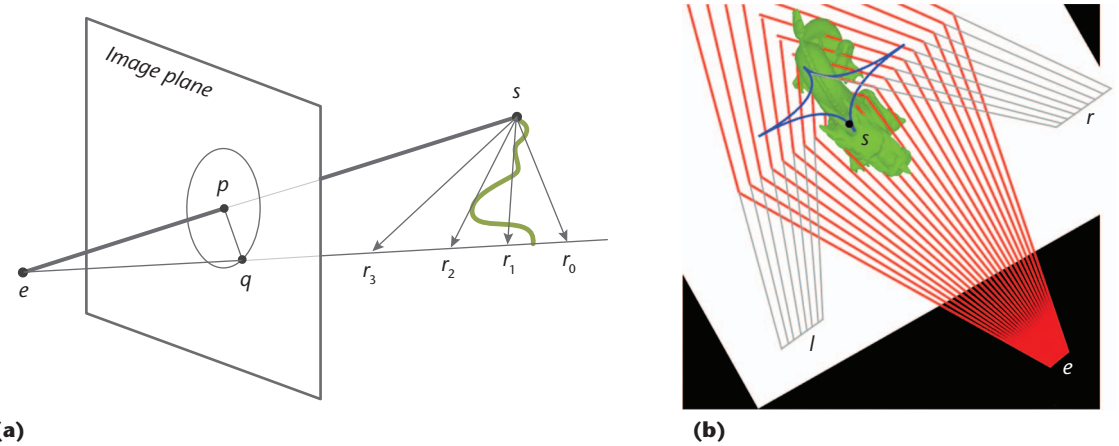


Figure 14. Ambient occlusion. (a) The screen-space ambient-occlusion algorithm. (b) Visualization of the graph camera model used in Figure 5. *e* is the output image frustum, *p* is an arbitrary output image pixel, *q* is the extent of the search space, *s* is the splitting point, r_i indicates various rays, and *l* and *r* are the left and right frustums.

ambient occlusion using a nonpinhole *z*-buffer that samples more of the geometry that occludes the samples in the output image. This leads to a better approximation of the ambient occlusion in each output image and to more stability from image to image. To realize this potential of nonpinhole *z*-buffers, we must overcome two challenges:

- specifying a nonpinhole camera that captures the geometry needed to compute the ambient occlusion for the current view, and
- preserving the performance advantage of the horizon-based ambient-occlusion algorithm.

Unlike specular reflection and relief texture mapping, ambient occlusion requires probing visibility along a 2D set of rays for each output image pixel. The visibility half space at a pixel is sampled with half planes; then each half plane is sampled with rays. The horizon-based screen-space algorithm is fast because it estimates occlusion in an entire half plane by traversing a single output image *z*-buffer segment. In Figure 14a, *p* is an arbitrary output image pixel, *e* is the output image frustum, and *q* is the extent of the search space. All rays r_i project to *pq*, and we estimate the occlusion on the *q* side of *ep* by simply traversing *pq*. We don't need to cast rays r_i. The dimensionality of the space of rays that must be cast to probe visibility decreases from 2 to 1.

We must maintain this property when porting the algorithm to nonpinhole *z*-buffers. The SPOC doesn't have this property. For any pixel other than the pole, no set of planes spans space with each plane projecting to a curve. The planes have a projection with a nonzero area. So, even though the SPOC might sample more of the geometry needed for decent ambient occlusion, using it in this context is prohibitively slow.

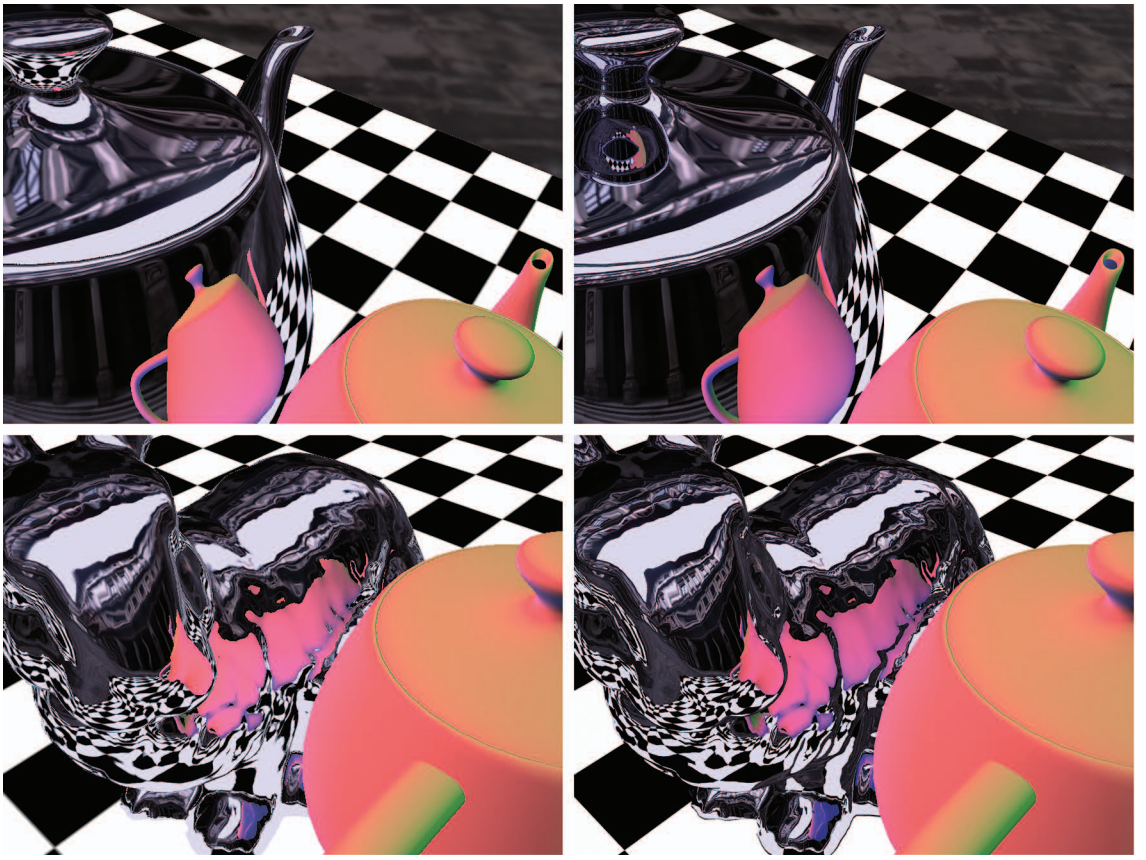
Our graph camera enhances the output image *z*-buffer with samples visible from two nearby viewpoints and exhibits the desired property. The camera topology is a simple binary tree consisting of a root and two children. Figure 14b visualizes the rays of the graph camera used to render the nonpinhole *z*-buffer in Figure 5. The graph camera has three subfrustums: the output image frustum *e* up to a vertical plane through the splitting point *s*, and the left and right frustums *l* and *r* beyond.

Recall that the graph camera projection is equivalent to a series of conventional projections. Once the leaf projection collapses a plane to a line, subsequent projections map the line to lines. Consequently, just as with a conventional planar pinhole camera, a line segment in the graph camera image corresponds to a plane in 3D space. Consider sample *S* in the graph camera *z*-buffer in Figure 5 and the line segment connecting it to sample *Q*. The plane defined by the viewpoint of the right frustum and 3D points *S* and *Q* projects to graph camera image line *SQ*. As before, we examine visibility in the entire plane by tracing *SQ*.

Our enhanced horizon-based screen-space algorithm proceeds as follows:

1. Render the output image without ambient occlusion.
2. Construct a graph camera for the output view.
3. Render the *z*-buffer with the graph camera.
4. For each pixel in the output image, add ambient occlusion using the graph camera *z*-buffer.

Like the conventional algorithm, this algorithm delays the calculation of ambient occlusion until the final step, in order to calculate only the effect for the visible samples. Steps 2 and 3 are the new steps added to the conventional algorithm.



(a) (b)

Figure 15. Comparing (a) our method to (b) ray tracing. Our method achieves comparable results.

At step 2, we construct the graph camera to achieve the desired disocclusion. The graph camera depends on the current view. The splitting plane where the root frustum ends and the left and the right frustums begin is perpendicular to the current view direction. One option is to keep the splitting point at the object's centroid. For the example in this article, the splitting point moves on the blue curve in Figure 14b. We designed the curve offline to move the splitting point smoothly behind the dragon. We did this because the dragon is seen sideways, so a conventional *z*-buffer suffices (see the accompanying video).

At step 3, the rendering of the graph camera z-buffer efficiently leverages the camera's closed form and low-cost projection. At step 4, we use the graph camera z-buffer like a conventional *z*-buffer: we integrate the current pixel's ambient occlusion along line segments that emanate radially from the pixel. Like a conventional camera, the graph camera lets us recreate a 3D point from a pixel and a depth value by unprojection.

Limitations

Our method achieves results comparable to ray tracing (see Figure 15). However, it has several limitations.

Absent Self-Reflections

Our method could, in principle, support self-reflections by also intersecting the reflected rays with a depth image of the reflector. However, the additional intersection is probably a price interactive applications aren't willing to pay.

Coarse Silhouettes

An SPOC depth image doesn't sample the entire object it replaces. The sampled area ends with a jagged edge when the SPOC rays are tangential to the replaced geometry (see Figure 7). When the jagged edge is exposed, the reflection's silhouette becomes coarse. One possible solution is to smooth the edge as a preprocess, which would preclude dynamic scenes. Instead, we alleviate the problem at runtime by alpha-blending the intersection sample with greater transparency when the SPOC ray becomes tangential to the sampled surface.

Undersampling

As with all sample-based methods, the quality of the results from nonpinhole depth images is contingent upon adequate sampling. The SPOC sampling rate is uniform and controllable. The graph camera sampling rate isn't uniform: it's higher closer to the initial frustum and lower for the dis-

tant frustums. We constructed the graph camera depth image to capture the entrance at a higher resolution, where reflections are of the highest quality (see Figure 8). Deeper into the maze the resolution decreases, leading to aliasing artifacts due to the large output-image projection of depth image pixels, a problem similar to inadequate shadow map resolution. In Figure 16, the teapot is at the maze's upper left (see Figure 8) and thus deepest in the graph camera depth image, where sampling insufficiency is noticeable.

However, this case is particularly challenging: a smooth, highly specular surface reflects a contrasting checker pattern. We use a graph camera depth image resolution of 1,920 × 1,175. A brute-force solution is to increase the resolution further. Another possibility is to divide the maze into several parts, each with its own smaller graph camera depth image. Finally, we could also use a hybrid sample-based and geometry-based technique that incorporates "infinite frequency" edges into textures.

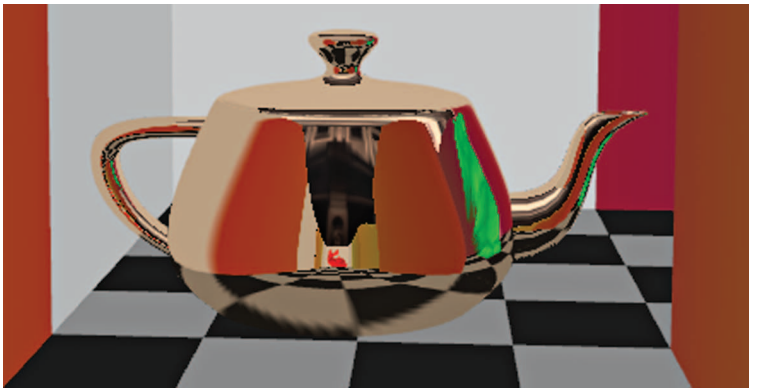


Figure 16. Undersampling artifacts on floor reflection. The teapot is at the maze's upper left (see Figure 8) and thus deepest in the graph camera depth image, where sampling insufficiency is noticeable.

Missing Samples

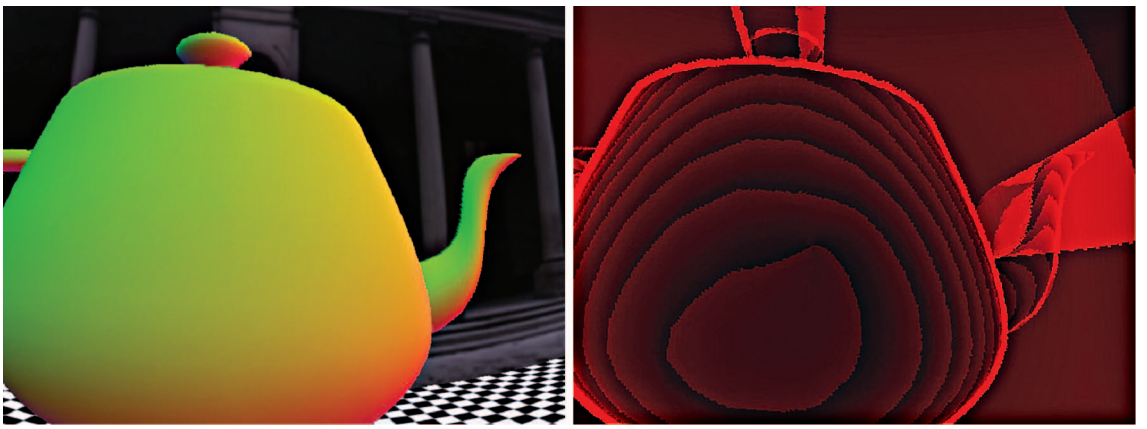
The most visible artifacts in nonpinhole relief texture mapping are due to samples still missing from the relief texture owing to residual occlusions. The rear bumper of the car in Figure 4 occludes some of the car body in the relief texture, which causes the shimmering "rubber band" surface seen in the video. One solution is to modify the car model to reduce the distance between the bumper and the car's body by pushing the bumper in or thickening it. Another solution is to encode the bumper in a second relief texture layer.

Performance

We collected the timing information on a 3.4-GHz, 2-Gbyte Intel Xeon PC with a 768-Mbyte Nvidia 8800 Ultra card. We used Nvidia's Cg 2.0 shading language with gp4 profiles. An important performance factor was the number of steps along the ray's projection, which we analyzed for reflections.

We took coarse steps first and performed a fuzzy intersection of the coarse ray segment with the nonpinhole depth map. If the two endpoints projected at unoccupied locations or the coarse ray segment clearly didn't intersect the impostor depth map, the coarse segment was trivially rejected. We refined coarse segments by performing fine steps of 1, 1/2, or 1/4 depth image pixels.

Figure 17 illustrates the number of steps for a 512 × 512 SPOC depth image, a six-pixel coarse step, and a 1/4-pixel fine step. More steps were needed when the reflected ray narrowly missed the teapot, which caused the fuzzy test to return a false positive. The average number of steps was 48 per output pixel, including both coarse and fine steps. For fine steps of 1 and 1/2 pixel, the average number of steps was 22 and 31, respectively. These



(a) **(b)**

Figure 17. Visualization of intersection operations needed for each reflected ray. (a) A diffuse teapot reflected in the body of a large teapot. (b) Visualization of the number of intersection steps per pixel. More intense red color indicates regions that required more intersection steps.

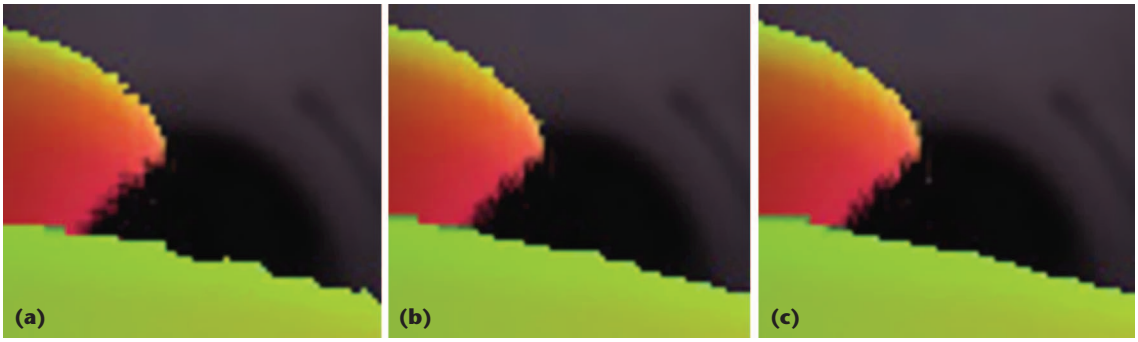


Figure 18. A silhouette detail with fine steps of (a) 1, (b) 1/2, and (c) 1/4 pixel. Smaller step sizes produce better antialiasing effects at object silhouettes, at the cost of increased intersection calculations.

Table 1. Frame rates for the scene in Figure 1.

	Output resolution								
	640 × 480			800 × 600			1,024 × 768		
Fine-step size (pixels)	1	1/2	1/4	1	1/2	1/4	1	1/2	1/4
Avg. fps	26.5	20.7	15.0	20.9	16.5	11.6	15.2	11.8	8.0
Min. fps	18	14	8	14	10	4	10	8	6
Max. fps	54	46	36	48	15	34	40	34	28

numbers don't account for pixel processor idling due to SIMD (single instruction, multiple data) processing constraints. Figure 18 shows the reflection silhouette quality for various fine-step sizes.

Performance depended on the output image resolution and fine-step size (see Table 1). We measured performance on a typical path (see the video) for the scene in Figure 1. We used eight-sample multisampling antialiasing (8 × MSAA), a 512 × 512 SPOC depth image, and a coarse step of six pixels. For an output resolution of 640 ×480, with 8 × MSAA, the average frame rates for SPOC depth images of resolution 128 × 128, 256 × 256, 512 × 512, and 1,024 × 1,024 were 55.8, 36.6, 26.5, and 16.1 fps, respectively. For coarse steps of 3, 6, 9, and 12 texels, the average frame rates were 18.2, 31.0, 37.2, and 39.8 fps, respectively. The only feature thin enough to be affected by the coarser steps was the spout's tip. For a sequence in which the SPOC depth image was recomputed on the fly (see the video), the average frame rates were 22 fps for no antialiasing and 17.3 fps for 16 × MSAA.

Coarse stepping reduced the number of steps for the graph camera depth image as well. This is evident in Figure 19, in which the average number of steps decreased from over 155 to 27. The reflection of the main entrance, where the graph camera impostor had the highest resolution, remained the area with the most activity on the teapot. However, only a few pixels had large step numbers. The graph camera impostor was constructed at over 100 fps. The average, minimum, and maximum frame rates for the path that followed the teapot through the maze (see the video) were 45.5, 30.0, and 105.0 fps without antialiasing, and 26.8, 20.0, and 42.0 fps with 8 × MSAA, respectively.

We plan to accelerate the ray-and-depth-image intersection computation further by leveraging ray coherence. We envision a two-pass approach that first renders the reflection at lower resolution and then upsamples by interpolation in coherent regions. The second pass would compute intersections only at regions where the lower-resolution results weren't sufficient to reconstruct a quality intersection, such as at edge regions.

We constructed nonpinhole relief texture maps with an SPOC; our previous discussion of the performance of ray intersection in the context of reflections still applies. For Figure 4, the overall performance, including shadow mapping, was 14 fps for the 40 cars and 18 fps for the 60 barrels. The output resolution was 640 × 480, and the relief texture resolution was 512 × 512. For 20 cars, 10 cars, and 1 car, the performance was 26, 51, and 219 fps, respectively. All the examples we've shown used tall relief, which implies long ray projections. For scenes with short relief, performance was even higher—for example, 46 fps for 160 cars half the size of the previous examples (see Figure 20).

We investigated ambient-occlusion performance for two quality occlusion sampling settings: regular (six sampling directions and six steps per direction) and fine (32 and 20). In both cases, the blur kernel width was 21 pixels and the output image resolution was 1,024 × 1,024.

The average performance for the dragon scene (see Figure 5) was 35 and 16 fps for the regular and fine settings, respectively. Figure 5 and the

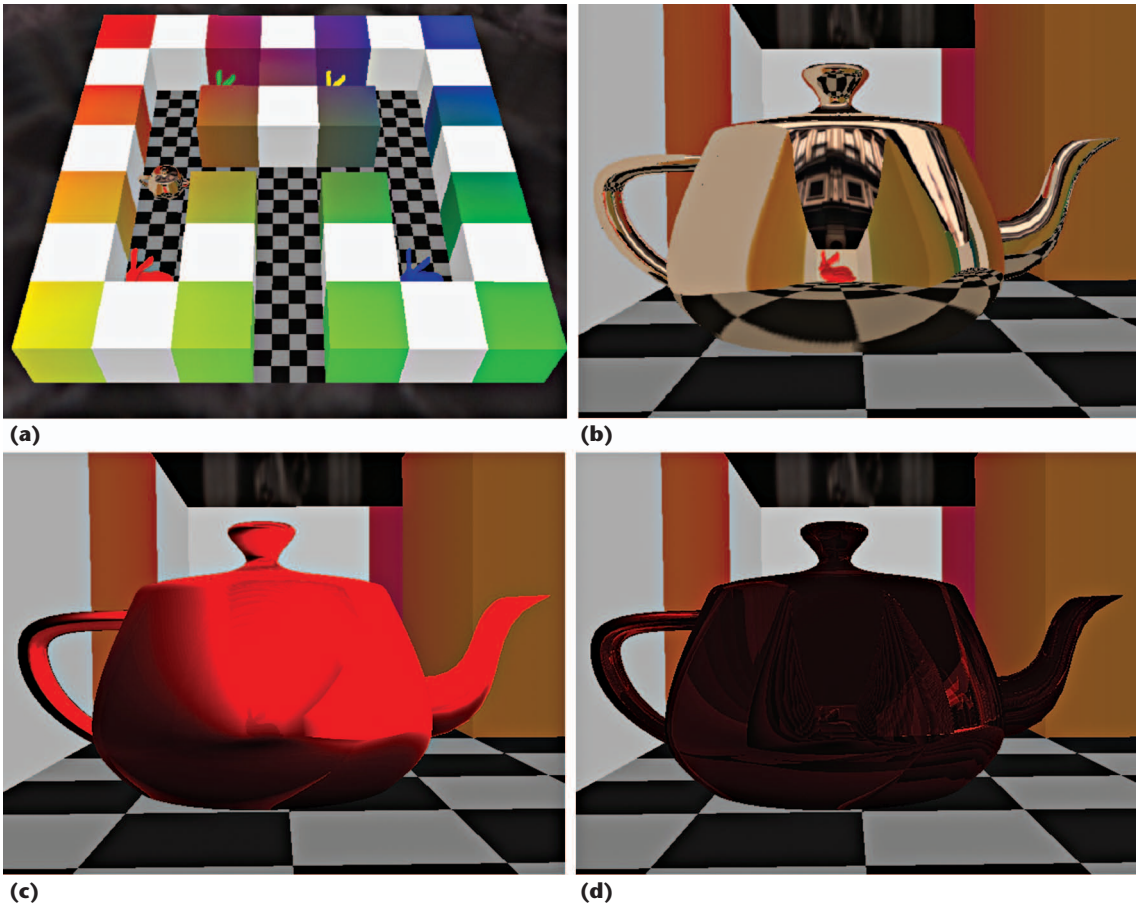


Figure 19. Teapot (a) location and (b) reflection, and the number of steps for visualization (c) without and (d) with coarse stepping. Brighter red indicates more steps. Initially taking coarser steps can significantly reduce the total number of steps. However, this approach might miss fine details.

video used the fine setting. The regular setting produced noisier ambient occlusion; this issue was orthogonal to the use of the nonpinhole *z*-buffer. Rendering the locally illuminated output image, rendering the graph camera *z*-buffer, and adding ambient occlusion took 5.9, 16.0, and 6.0 ms for the regular setting and 5.9, 16.0, and 40.0 ms for the fine setting, respectively. For the conventional algorithm, rendering the locally illuminated output image and adding the ambient occlusion took 5.9 and 5.9 ms for the regular setting and 5.9 and 32.0 ms for the fine setting, respectively. Sampling occlusion in the graph camera *z*-buffer as opposed to the conventional *z*-buffer incurred a 25 percent penalty. Most performance loss was due to having to render the graph camera *z*-buffer for the 890K-triangle scene. However, in some scenes, updating the graph camera *z*-buffer for every frame was unnecessary.

The nonpinhole *z*-buffer saw more than what was visible in the output image, which greatly extended its resiliency to output view changes. For example, we could reuse the *z*-buffer in Figure 5 if we viewed the dragon only from the front. This increased performance to 21 fps for the fine setting, approaching the conventional algorithm's 26 fps. We could also use a simplified version of the model when computing the nonpinhole *z*-buffer.

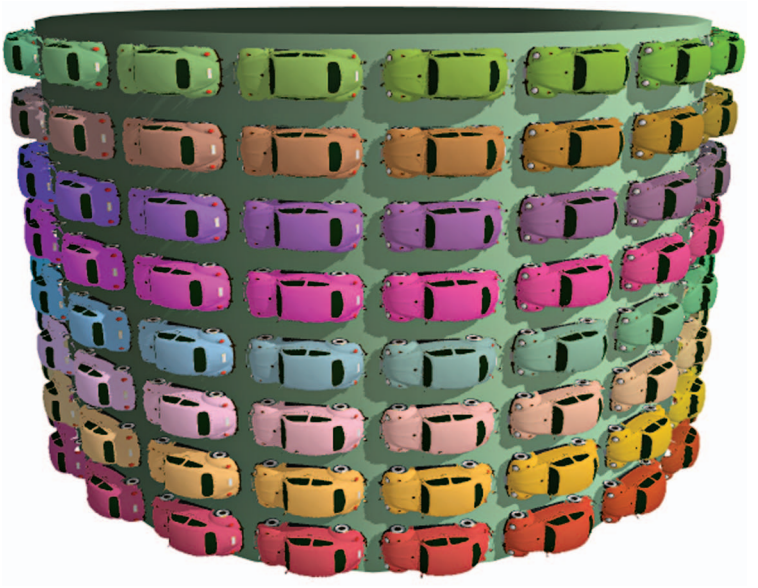


Figure 20. A short-relief example. Here, performance was higher than for tall relief.

Nonpinhole Cameras

The overwhelming majority of the images used in computer graphics, visualization, and computer vision are rendered or acquired with a pinhole camera: all rays pass through a common point, the pinhole. This leads to efficient computational and physical camera implementations that compute and acquire images similar to those captured by the human visual system. However, the pinhole requirement is quite restrictive.

Nonpinholes are powerful cameras that can capture rays originating at different points in space. Computer vision researchers have used them to model complex lens and catadioptric systems, including the pushbroom camera,[1] the two-slit camera,[2] and their generalization, the general linear camera.[3] General linear cameras enable a powerful framework for designing multiperspective images, but rendering is done by ray tracing.[4]

Research in artistic rendering has used nonpinhole cameras to simulate deviations from the conventional perspective that artists adopt for aesthetic reasons.[5] In scene sampling, the most relevant context here, one prior nonpinhole camera is the multiple-center-of-projection camera, which samples the scene with a vertical slit along a user-chosen path, thus avoiding redundancy and offering sampling flexibility.[6] However, construction requires rendering the scene for each position along the path. In cel animation, multiperspective panoramas capture all 3D scene samples seen along a camera path. This lets animators simulate camera motion by sliding a frame over the panorama, but the view is confined to the path, and the scene must be static.[7]

Occlusion cameras aim to address disocclusion errors. Given a reference view and a 3D scene, occlusion cameras build a single-layer image that stores samples visible from the reference viewpoint and from nearby points. They include the single-pole occlusion camera we discuss in the main article, the depth-discontinuity occlusion camera (DDOC),[8] and the epipolar occlusion camera (EOC).[9] The DDOC specifies the distortion through a map; the added flexibility comes at the cost of increased construction time. The EOC captures all the samples visible as the viewpoint translates between two given points. It effectively generalizes a planar pinhole camera's viewpoint to a *viewsegment*. However, it supports translation only along a single direction.

References

1. R. Gupta and R. Hartley, "Linear Pushbroom Cameras," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 19, no. 9, 1997, pp. 963–975.
2. T. Pajdla, *Geometry of Two-Slit Camera*, tech. report CTU–CMP–2002–02, Czech Technical Univ., 2002.
3. J. Yu and L. McMillan, "General Linear Cameras," *Proc. 8th European Conf. Computer Vision* (ECCV 04), vol. 2, LNCS 3022, Springer, 2004, pp. 14–27.
4. J. Yu and L. McMillan, "A Framework for Multiperspective Rendering," *Proc. 2004 Eurographics Symp. Rendering*, Eurographics Assoc., 2004, pp. 61–68.
5. M. Agrawala, D. Zorin, and T. Munzner, "Artistic Multiprojection Rendering," *Proc. 2000 Eurographics Workshop Rendering Techniques*, Eurographics Assoc., 2000, pp. 125–136.
6. P. Rademacher and G. Bishop, "Multiple-Center-of-Projection Images," *Proc. Siggraph*, ACM Press, 1998, pp. 199–206.
7. D. Wood et al., "Multiperspective Panoramas for Cel Animation," *Proc. Siggraph*, ACM Press, 1997, pp. 243–250.
8. V. Popescu and D. Aliaga, "The Depth Discontinuity Occlusion Camera," *Proc. 2006 Symp. Interactive 3D Graphics* (I3D 06), ACM Press, 2006, pp. 139–143.
9. P. Rosen and V. Popescu, "The Epipolar Occlusion Camera," *Proc. 2008 ACM Symp. Interactive 3D Graphics* (I3D 08), ACM Press, 2008, pp. 115–122.

We've shown that nonpinhole depth images provide efficient ray intersection at the small additional cost of traversing a curved—as opposed to a straight—ray projection. As we mentioned before, we leveraged this property to render specular reflections, refractions, relief texture mapping, and ambient occlusion. In prior research, we also used depth-discontinuity occlusion cameras to render soft shadows at interactive rates.[5] (For more on these and other nonpinhole cameras, see the "Nonpinhole Cameras" sidebar.) The suitability of nonpinholes to so many rendering problems argues for the technique's generality.

Compared to reflection-rendering techniques, such as explosion maps,[6] that approximate the projection of reflected points, our method produces multiple projections of the same object at no extra cost and handles complex reflectors. Compared to color-caching image-based techniques, our method supports dynamic scenes and has reduced memory requirements. Color-caching techniques excel at capturing the appearance of complex real-world materials that are glossy but not specular. Our method produces better results than environment mapping, but at a higher per-pixel cost. Compared to ray tracing, our method more easily minifies and magnifies reflections by working in the color map at different resolution levels, and it achieves fast ray-and-geometry intersection. Ray tracing has a quality advantage because it doesn't approximate the reflected geometry. Compared to conventional relief texture mapping, the nonpinhole relief maps bring greater modeling power at the cost of a more expensive ray-and-relief-texture intersection.

Besides the directions for future research that we already sketched, we'll investigate integrating our nonpinhole rendering framework into popular digital 3D content creation tools. Our research argues for the benefits and practicality of camera models that abandon the pinhole constraint. Nonpinhole cameras can provide powerful yet inexpensive approximations for many applications in graphics and beyond.

References

1. L. Bavoil and M. Sainz, "Image-Space Horizon-Based Ambient Occlusion," *Siggraph 2008 Talks*, ACM Press, 2008, article 22.
2. C. Mei, V. Popescu, and E. Sacks, "The Occlusion Camera," *Computer Graphics Forum*, vol. 24, no. 3, 2005, pp. 335–342.
3. V. Popescu, P. Rosen, and N. Adamo-Villani, "The Graph Camera," *ACM Trans. Graphics*, vol. 28, no. 5, 2009, article 158.
4. C. Wyman, "An Approximate Image-Space Approach for Interactive Refraction," *ACM Trans. Graphics*, vol. 24, no 3, pp. 1050–1053.
5. Q. Mo, V. Popescu, and C. Wyman, "The Soft Shadow Occlusion Camera," *Proc. 15th Pacific Conf. Computer Graphics and Applications* (PG 07), IEEE Press, 2007, pp. 189–198.
6. E. Ofek and A. Rappoport, "Interactive Reflections on Curved Objects," *Proc. Siggraph*, ACM Press, 1998, pp. 333–342.

Paul Rosen *is a research assistant professor in the University of Utah's Scientific Computing and Imaging Institute. His research interests are camera model design, multicore and parallel desktop computing, visualization of large-scale datasets, and uncertainty visualization. Rosen has a PhD in computer science from Purdue University. Contact him at prosen@sci.utah.edu.*

Voicu Popescu *is an associate professor in Purdue University's Computer Science Department. His research interests are computer graphics, computer vision, and visualization. Popescu has a PhD in computer science from the University of North Carolina at Chapel Hill. Contact him at popescu@cs.purdue.edu.*

Kyle Hayward *is a graphics programmer for Human Head Studios. His research interest is real-time ray tracing on GPUs. Hayward has a BS in computer science from Purdue University. Contact him at khayward@purdue.edu.*

Chris Wyman *is an associate professor of computer science at the University of Iowa. His research interests are interactive global illumination and other interactive and realistic rendering problems, visualization, and perceptual issues in rendering. Wyman has a PhD in computer science from the University of Utah. Contact him at cwyman@cs.uiowa.edu.*