Forward Mapped Planar Mirror Reflections

Rui Bastos, Wolfgang Stürzlinger
Department of Computer Science
University of North Carolina at Chapel Hill
Bastos@cs.unc.edu

Abstract: This paper presents a new technique, which we call *depth-preserving reflection mapping*, to render mirror-like reflections on planar surfaces in constant time. It is a hybrid solution which combines geometry-based rendering and image-based rendering into a two-pass mirror reflection rendering approach. The technique extends the traditional reflection mapping to preserve depth per texel and uses forward warping to approximate the mirror-like reflections on planar surfaces. For clarity, to distinguish these texels from the ones of traditional reflection mapping, we call them *zexels*.

1 Introduction

Architectural walkthrough applications enable the user to interact in real-time with computer-simulated environments [5]. An important goal is to represent the rich visual complexity of the scenes. Such complexity is usually achieved with realistic illumination and shading models to generate images of the scene.

Unfortunately, interactive walkthrough applications trade photo-realism for frame rate. Local illumination or coarse global illumination solutions are preferred because they are "cheaper" to render. The usually-adopted global illumination approximations only consider ideally diffuse environments (radiosity) providing only view-independent effects [3]. Specular (mirror-like) effects considerably enhance the realism of a scene but require costly computation of view-dependent information on a per frame basis. However, they add non-static shading and visual cues to the scenes, which make them more interesting for walkthroughs. A few approaches exist to render mirror-like reflections but they are either too slow or inaccurate for practical use in interactive walkthroughs [12][8][4][9] [13][14]. Ray tracing is the most accurate technique to compute mirror reflections, but it is also the slowest. Reflection mapping provides quick mirror reflections, but it is inaccurate. For highly curved surfaces and for surfaces with highly varying curvature, the inaccuracies of reflection mapping are almost unnoticeable, but for planar and almost-planar surfaces the artifacts on the reflected images can be very severe.

This paper describes a new technique, called *depth-preserving reflection mapping*, for rendering mirror reflections on planar surfaces. It is a hybrid solution combining traditional geometry-based rendering with image-based rendering on a two-pass (multi-pass) reflection rendering approach. Range images of the scene (*depth-preserving reflection maps*) are captured *a priori*, and warped at rendering time to compute mirror-like reflections on planar specular surfaces.

The next sections describe how our method compares with traditional reflection mapping. We show how the new method overcomes problems of traditional reflection mapping to rapidly generate reflections with correct perspective foreshortening and correct visibility (motion parallax) without requiring the rendering of the entire scene for each mirror every frame.

2 Background and Related Work

Several approaches have been proposed to render mirror-like reflections, but usually they are either slow or inaccurate for practical use in interactive walkthroughs. We will now review background material for the special case of planar reflections and describe some approaches to their simulation.

Planar reflections can be rendered by a multi-pass rendering approach [7]. If there is a single planar mirror in the scene, a two-pass approach suffices. In the first pass, the reflected image of the scene is rendered. In the second pass, the rest of the scene (non-mirror surfaces) is rendered [12].

The first pass takes the eye-point, $\mathbf{E}_p$, and the eye-direction, $\mathbf{E}_d$, reflects them according to the planar mirror, and renders the scene from that reflected location and reflected view-direction ($\mathbf{M}_p$ and $\mathbf{R}_d$ in Figure 1). The following formulae are used to compute the mirrored point, $\mathbf{M}_p$, and the reflected direction, $\mathbf{R}_d$:

(1) $$\mathbf{M}_p = \mathbf{E}_p + 2d(-\mathbf{N})$$

(2) $$\mathbf{R}_d = 2(\mathbf{N} \cdot \mathbf{E}_d)\mathbf{N} - \mathbf{E}_d$$

where d is the shortest (orthogonal) distance from the eye-point to the planar surface and $\mathbf{N}$ is the normal vector to the planar mirror.

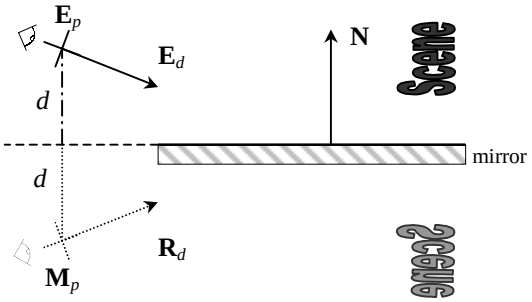

Figure 1 Reflection of the eye-point and the eye-direction with respect to the normal vector of a planar mirror.

In the second pass, the scene is re-rendered for the original eye-point and eye-direction ($\mathbf{E}_p$ and $\mathbf{E}_d$) without rendering the mirror surface. Alternatively, the second pass can use a texture mapping technique. The reflected image computed in the first pass is applied as a texture onto the surface of the mirror in the second pass.

This interpretation of planar reflections leads directly to the algorithms described in [12][13][7]. Although this technique is simple and sounds attractive for polygonal models, it is intensive in terms of computational time. Every mirror polygon requires the re-rendering of the whole scene from its mirrored viewpoint at each frame. This means dealing with the entire complexity of the scene several times every frame, which is unpractical for large scenes.

Ray tracing relaxes the limitation of planar mirror reflections. Instead of mirroring the viewpoint and view-direction, and computing an entire image per mirror, ray-tracing tracks rays from the eye to the scene and reflects their direction whenever they hit a mirror-like surface [8]. Although ray tracing can be more efficient in terms of the number of sampled directions, compared to scene re-rendering, it cannot exploit current graphics hardware. This limitation makes ray tracing a slow technique for rendering mirror-like reflections and not applicable for interactive applications.

Reflection maps [4][9][13][14] have long been used for rapid rendering of mirror reflections. The scene surrounding specular objects is projected onto the surface of a sphere or cube. The captured scene is then used as a texture map on the specular objects. Even though reflection mapping is much faster than ray tracing and scene re-rendering, it computes only quick approximations of the mirror reflections. The artifacts can be severe, due to assumptions about distance from the specular object to the rest of the scene, as described in Section 3.

3 Traditional Reflection Mapping

Reflection mapping [4][9] is the most efficient shading technique to simulate specular reflections for interactive rendering. Images of the environment around a particular viewpoint (e.g., the center of the scene) are precomputed and then mapped onto the surface of specular objects by stretching and compressing the map, depending on the viewing direction and the surface normal at each pixel. The limitation of this approach is that the mirror reflection is geometrically correct only for a surface point at the exact same location the images (textures) were generated from. Perspective distortions and incorrect occlusion (motion parallax) are the most noticeable artifacts.

Although reflection mapping generates inaccurate reflections, it does a great job for indicating the curvature of the mapped surfaces. On surfaces with high curvature or with highly varying curvature, the inaccuracies of the reflections are almost imperceptible. The reflected images are so complex for such surfaces that our brain can not discern their structure. However, on planar and near-planar surfaces the inaccuracies are obvious and can lead to incorrect perception of an environment.

Reflection mapping provides quick approximations of mirror-like reflections by trading correctness for computational time. As a preprocessing step, a convex cell (a sphere or a cube) is created in the center of the scene or surrounding the reflective object. Then, also as preprocessing, images of the scene are rendered from the center of the cell and stored as textures on its surface (Figure 2). At rendering time, a reflection vector is computed per pixel ($\mathbf{r}_1$, $\mathbf{r}_2$, and $\mathbf{r}_3$ in Figure 3) using equation (2). Texture coordinates are derived from the reflection vector, as explained in [14], and index the texture maps captured in

the previous phase to give the color of the specular reflection at each pixel (Figure 3). The reflection on the specular surface reconstructs the color of the surrounding environment but only approximates the geometry of the reflected scene.

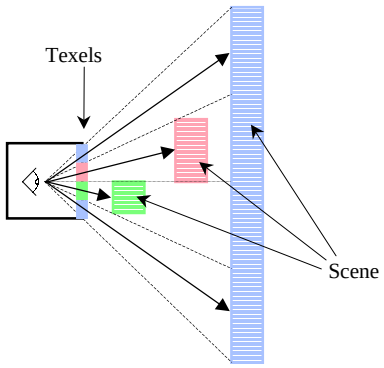

Figure 2 *Capturing a texture for traditional reflection mapping*: the black square is the reflection cell. The dashed lines represent the solid angles associated with each texel of the cell. The filled colored rectangles represent objects in the scene. The objects are projected onto the surface of the reflection cell and a color is stored at each texel.

The main limitation of traditional reflection mapping is due to an assumption about distance from the specular surface to the rest of the scene. The environment surrounding the specular reflective surface is assumed to be remote (distant). A single image of the scene is used to map the reflected environment onto the specular surface. This assumption eliminates motion parallax on the reflections. No matter from where the reflection is seen, the reflected image is always obtained by simply stretching and compressing the environment map. This two-dimensional stretch and compress does not resolve changes in perspective distortion and motion parallax caused by variations in the distance from viewpoint to viewpoint. When the remote environment assumption is satisfied, i.e, when the reflected environment is really far away from the specular object, the artifacts are almost unnoticeable. However, in general, this assumption is not respected and artifacts are quite perturbing.

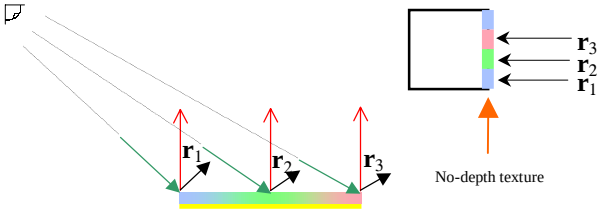

Figure 3 *Using a traditional reflection map*: the yellow surface is the mirror where the reflection is mapped. The red vectors represent the normal vector to the mirror. The green vectors depict viewing directions. The reflection vectors, in black, index, indirectly, the texture in the reflection map. Notice the color reconstruction of the environment map on the mirror.

4 Depth-Preserving Reflection Mapping

We introduce a new reflection mapping technique that generates mirror-like reflections with correct perspective foreshortening and correct occlusion (motion parallax) for planar surfaces. We call it *depth-preserving reflection mapping*.

As an alternative for computing mirror-like reflections for planar surfaces at interactive frame rates, we propose a modified version of reflection mapping that exploits 3D image warping [15]. Instead of projecting the scene onto the surface of the reflection cell, we preserve depth (disparity [10]) at each texel of the reflection map (Figure 4). Both color and depth are stored at each texel. This modification makes texels 3D entities and allows their 3D warping during the rendering stage. For clarity, we call these entities *zexels*, in order to distinguish them from texels (no depth) of traditional reflection mapping.

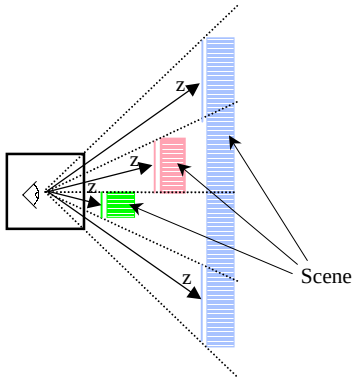

Figure 4 *Capturing a texture for depth-preserving reflection mapping*: the filled colored rectangles represent objects in the scene. The black square is the reflection cell. The dashed lines represent solid angles associated with zexels of one side of the cell. Each zexel stores the color and depth (z) of the object that is visible from its corresponding solid angle. Notice that zexels are not attached to (projected onto) the reflection cell. Each zexel is floating in space in front of the corresponding visible object. *Z* is the depth (distance) from the visible object to the viewing (sampling) point.

Similarly to reflection mapping, we create a cell around the specular object and render images of the scene taken from the center of that cell. In fact, there is no explicit need for the cell, but we keep it for clarity. As we store the depth of each zexel, measured relatively to the sampling point (viewpoint), the zexels are not actually attached to (projected onto) the surface of the cell. They are floating in space right in front of the object hit by the corresponding solid angles (see Figure 4).

At rendering time, zexels are warped from the reference viewpoint space into the reflected viewpoint space. This operation is performed by forward warping [11][10] and by taking into account the geometry of the mirror and the current viewpoint (Figure 5). After warping all zexels of all the mirrors in the scene, the geometry (except the mirror polygons) is rendered on the same buffer. The net effect is the mapping of the reflected image on the screen region corresponding to the mirror(s).

As the warping operations do not depend on the number of polygons of the scene, note that our reflection mapping technique runs in constant time with respect to the complexity of the scene. The algorithm depends linearly on the number of zexels.

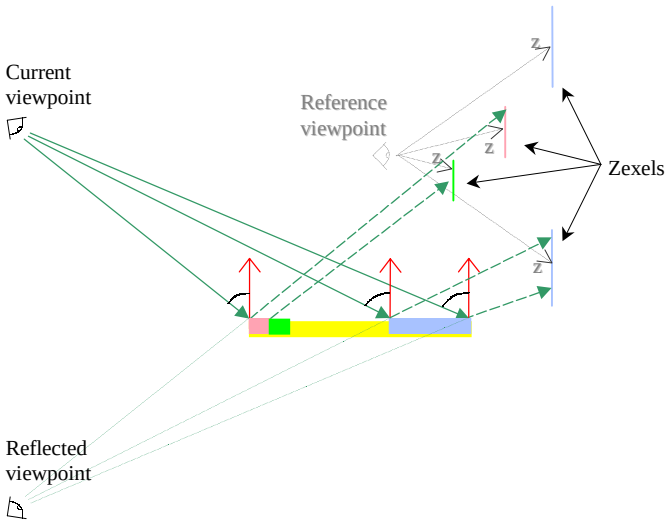

Figure 5 *Using a depth-preserving reflection map*: the yellow surface is the mirror where the reflection is mapped. The red vectors represent the normal vector to the mirror. The solid green vectors define viewing directions. The dashed green vectors define reflected directions. For clarity, the captured depth-preserving reflection map from Figure 4 is shown here in gray. Zexels in the reference 3D space (the depth-preserving reflection map) are warped into the reflected space and give location and color to the pixels of the specular (yellow) surface. Warping is performed taking into account the viewing direction (or, analogously, the reflected direction) and the normal vector at each point on the specular surface. Notice that the internal region of the mirror surface (in yellow) does not have any corresponding zexel in the depth-preserving reflection map, for the particular viewpoint of this figure. This characterizes an artifact, called exposure, in the reflected image.

4.1 Warping Reflection Images

Our technique follows the principle of planar mirrors presented in section 2. Given the planar mirror geometry and the eye-point and eye-direction, it uses equations (1) and (2) to compute the reflected eye-point and eye-direction. These reflected parameters, together with the depth-preserving reflection maps, feed the forward image warper, which computes the scene reflection for the planar mirror. The forward warper traverses all the zexels in the reflection maps and warps each one of them to the current viewing space.

Each zexel is warped from the depth-preserving reflection map onto the mirror-reflected image by using the following equation:

(3) $$\mathbf{x}_o = \delta(\mathbf{x}_i) P_o^{-1}(c_i - c_o) + P_o^{-1} P_i \mathbf{x}_i$$

where

- $\mathbf{x}_i$ represents the coordinates of a zexel in a depth-preserving reflection map,
- $\mathbf{x}_o$ represents the coordinates of the corresponding point in the reflected image,
- $\mathbf{c}_i$ and $\mathbf{c}_o$ are the centers of projection of the reference image (depth-preserving reflection map) and of the desired image (reflected image),
- $\boldsymbol{P}_i$ and $\boldsymbol{P}_o$ are the transformation matrices of the reference image (depth-preserving reflection map) and of the desired image (reflected image),
- and $\delta(\mathbf{x}_i)$ is the disparity of zexel $\mathbf{x}_i$.

The disparity [10][11] of a zexel is given by:

(4) $$\delta(\mathbf{x}_i) = 1 - z(\mathbf{x}_i)\frac{far - near}{far}$$

where $z(\mathbf{x}_i)$ is the depth for zexel $\mathbf{x}_i$ (the OpenGL z-buffer value for a pixel at $\mathbf{x}_i$), *far* is the distance from the reference viewpoint to the far clipping plane, and *near* is the distance from the reference viewpoint to the near clipping plane.

This warping operation characterises a mapping of a reference image (depth-preserving reflection map) into a reflected image. Zexels from the reference image are mapped into pixels in the reflected image. As we simply map discrete points from a grid into another grid without performing any reconstruction, there may be regions in the reflected image which do not receive any value. This means that we do not have a one-to-one mapping and results in uncovered regions in the reflected image (see Figure 5). This defines the image-based rendering artifact known as *hole* or *exposure* [6][11]. A similar problem can occur if a reference image has fewer zexels than the number of pixels required to fill in the projected area of the mirror on the screen.

We apply three techniques to minimize exposures in our reflected images. First, we use an ad hoc approach of not clearing the color buffer between frames. As we work with walkthroughs of the scene, there is color coherence from frame to frame (assuming slow/small changes in eye-point and eye-direction). Secondly, instead of warping a zexel into a single pixel, we fill in a region of 3x3 pixels in the reflected image (splatting). The third technique is to warp more than one depth-preserving reflection map for the same mirror. By computing more than one depth-preserving reflection map around a mirror we capture more visibility information which is then used to reduce exposure problems. Multiple maps are computed by regular sampling (placing reference image views) around a hemisphere behind the mirror.

As we can have more than one zexel mapping onto the same pixel in the reflected image, we need to break the ambiguity and take the one closer to the viewpoint. This could be done using z-buffering, but the back-to-front traversing (occlusion preserving) proposed by McMillan [11] has been shown to be faster. This approach also allows paralellization of the warping algorithm. It subdivides the reference image into a number of regions, depending on the projection of the center of projection of the desired image onto the

reference image. Each one of these regions can then be processed (warped) independently still preserving occlusion. This allows the parallelization of the forward warper on machines with multiple CPUs.

5 Results

We have implemented our system in C++ and OpenGL on a Silicon Graphics Onyx2 with 4 CPUs and InfiniteReality^2 graphics.

Our technique was tested with several different models. Here we present the results for a simple model, to simplify the understanding of the advantages of our technique compared to traditional reflection mapping, and the results for a more complicated model, to illustrate the use and performance of our technique on a real-world environment.

Figure 6 compares the results of traditional reflection mapping (on the left) with results of our technique (on the right). The scene contains a teapot on top of an inversed pyramid inside a box. Five sides of the box have a checkerboard pattern and the sixth wall has a planar mirror partially covering it. The mirror uses a single reflection (reference) map with 512x512 zexels. Final images have 640x480 pixels. The model contains 1850 polygons and renders, with mirrors, at 10 frames per second on the workstation described above. The same model renders, with mirrors, at 15 frames per second by using the parallel implementation described in the previous section. For comparison, the same model without the mirror renders at 60 frames per second on the same machine using a single CPU.

The checkerboard pattern aims to facilitate the identification of artifacts caused by the traditional reflection mapping which are correctly handled by our depth-preserving reflection mapping. Observe the continuity along straight lines of the checkerboard between the real pattern and on the reflected image. There are clear discontinuities created by the traditional reflection mapping which are correctly approximated by our technique. Notice also the incorrect motion parallax of the results from traditional reflection mapping, which is correctly captured by our technique. For example, notice the partial occlusion of the teapot and the inversed pyramid, due to the change in viewpoint with respect to the mirror and the scene, on the last row of images. Finally, observe that the traditional reflection mapping only stretches and compresses the reflection texture among different viewpoints. Conversely, the depth-preserving reflection mapping actually uses three-dimensional information about the reflected scene and creates mirror reflections as if the scene were rendered from the reflected viewpoint using the mirror as a window to a virtual (mirrored) scene.

Figure 7 shows a similar comparison of results for a larger model. The scene contains 189,982 polygons and two large planar mirrors (only one mirror can be seen in the images – on the (right) back wall with a gray frame around it). Each mirror uses up to 64 depth-preserving reflection maps of 480x480 zexels each (at most 2 out of these 64 maps are used per frame). Final images have 640x480 pixels. This model renders at 8.6 frames per second in a single CPU of the machine described above. The same model runs at 12.9 frames per second using 4 CPUs and the parallel implementation described in the previous section. For comparison, the model with no mirrors renders at 15.9 frames per

second on a single CPU. Notice that performance is bound by geometry and not by our reflections warping. Similar artifacts described for Figure 6 can be observed in the images of Figure 7. Observe, for instance, the slope discontinuities between straight lines in the scene and the corresponding ones in the reflected image (the pianos, the edge between ceiling and left wall, etc). Motion parallax artifacts can also be observed on the images. Note, on both Figure 6 and Figure 7, the remapping artifacts (exposures) described in section 4.1.

The simple model of Figure 6 is intended only to illustrate the visual differences between the traditional and the depth-preserving reflection mapping techniques. It is not intended for direct frame rate comparison between the two techniques. The more complex model of Figure 7 serves both purposes, though. It should be clear that, compared to the scene re-rendering technique of section 2, our technique becomes worthwhile when scene rendering time is greater than warping time for a single mirror. This observation is clearly valid for non-trivial scenes such as the model of Figure 7, where performance is bound by time to render the geometry of the scene.

6 Conclusions and Future Work

We have presented a hybrid rendering technique that combines image-based warping with geometry-based rendering for interactive handling planar mirror reflections. Our depth-preserving reflection mapping extends traditional reflection mapping by preserving depth in the texels, which we call zexels, and by forward warping these 3D entities to approximate the planar mirror-reflected images. This provides correct perspective foreshortening and proper motion parallax on the planar mirror reflections, effects that are not possible to achieve with traditional reflection mapping. To make this technique as interactive as traditional reflection mapping we need a hardware implementation of forward warping as presented in section 4.1.

The use of forward warping has the disadvantage of requiring the warping of all the zexels of the reflection map to create the mirror-reflected image. Even if the projected area of the mirror on screen is small, we currently still warp all the corresponding zexels to that small area. In order to reduce the number of zexels to warp for each mirror, a hierarchical representation of the reflection maps, similar to mip mapping, is under investigation. Under such a representation, the level of detail in the hierarchy is selected based on the distance from the viewer to the specular object.

Alternatively, the application of McMillan's [11] inverse-warping to mirror reflections is also under investigation. Instead of warping all the zexels from the reflection map into the specular surface, inverse-warping indexes the reflection map similarly to the traditional reflection mapping. Pixels on the specular surface are mapped into zexels in the depth-preserving reflection map. The use of inverse-warping would allow handling curved surfaces and could lead to a great reduction in the number of warping operations for each specular surface. However, it is known that inverse warpers are slower than and less implementable in hardware than forward warpers.

7 Acknowledgments

We would like to thank Daniel Aliaga for the portals and cells code and the warping code from which we began our development. We would also like to thank Anselmo Lastra, Chris Wynn, and Filippo Tampieri for reviewing an earlier version of this paper.

The music room model is part of the Brooks house model developed by many generations of members of the Walkthrough Team at UNC Chapel Hill.

This work was partially supported by the following grants: NIH/National Center for Research Resources P41RR02170-13, fellowship 201142/93-7 CNPq – Brasília/BRAZIL, and fellowship FWF-J01317-TEC.

Bibliography

1. D. Aliaga and A. Lastra. Architectural Walkthroughs using Portal Textures. In *IEEE Visualization*, Tempe, AZ, 1997.
2. D. Aliaga. Visualization of Complex Models Using Dynamic Texture-based Simplification. In *IEEE Visualization '96*: IEEE, October 1996.
3. R. Bastos, M. Goslin, and H. Zhang. Efficient Rendering of Radiosity using Texture and Bicubic Interpolation. In *ACM Symposium on Interactive 3D Graphics*, Providence, RI, 1997.
4. Jim Blinn and Martin Newell. Textures and Reflection in Computer Generated Images. *Communication of the ACM*. Vol. 19, no. 10 (1976), pp. 542—547.
5. F. Brooks. Walkthrough: A dynamic graphics system for simulating virtual buildings. In *ACM Symposium on Interactive 3D Graphics*, Chapel Hill, NC, 1986.
6. Shenchang Eric Chen and Lance Williams. View Interpolation for Image Synthesis. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, vol. 27, J. T. Kajiya, Ed., August 1993, pp. 279--288.
7. Paul Diefenbach. Pipeline Rendering: Interaction and Realism Through Hardware-Based Multi-Pass Rendering. *University of Pennsylvania, Department of Computer Science*, Ph.D. dissertation, 1996.
8. Andrew Glassner. **Principles of Digital Image Synthesis**. Morgan Kaufmann. San Francisco, 1995.
9. Ned Greene. Environment mapping and other applications of world projections. In *IEEE CG&A* 6(11), Nov 1986, pp. 21-29.
10. Leonard McMillan and Gary Bishop. Plenoptic Modeling: An Image-Based Rendering System. In *SIGGRAPH 95 Conference Proceedings*, *Annual Conference Series*, R. Cook, Ed.: ACM SIGGRAPH, August 1995, pp. 39--46.
11. Leonard McMillan. An Image-Based Approach To Three-Dimensional Computer Graphics. *University of North Carolina at Chapel Hill, Department of Computer Science*, Ph.D. dissertation, 1997.
12. Tom McReynolds *et. al*. Programming with OpenGL: Advanced Techniques. In *SIGGRAPH 97 Course Notes*, Ed.: ACM SIGGRAPH, August 1997.
13. SGI OpenGL Optimizer. http://www.sgi.com/Technology/openGL/optimizer_wp.html, 1997.
14. Douglas Voorhies and Jim Foran. Reflection Vector Shading Hardware. In *Proceedings of ACM SIGGRAPH 1994*, pp. 163—166.
15. Wolberg, George. **Digital Image Warping**. IEEE Computer Science Press, Los Alamitos, CA, 1990.

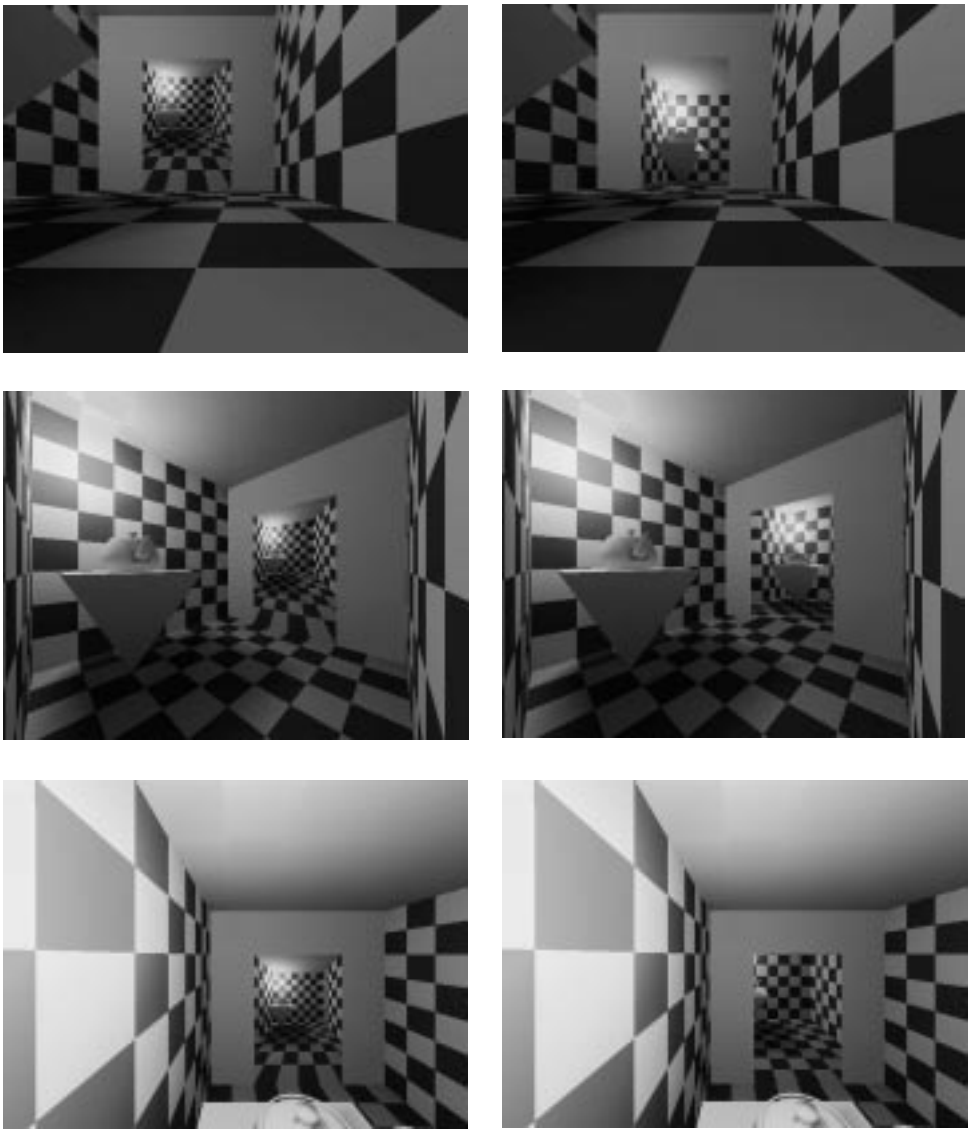

Figure 6 Comparison of the *traditional reflection mapping* (left) and our *depth-preserving reflection mapping* (right) for a simple model.

Figure 7 Comparison of the *traditional reflection mapping* (left) and our *depth-preserving reflection mapping* (right) for a real-world model.