REAL-TIME REFLECTION AND REFRACTION ON A PER-VERTEX BASIS

Masanori Kakimoto[1)] *Tomoaki Tatsukawa*[1)] *Geng Chun*[1)] *Tomoyuki Nishita*[2)]

[1)] SGI Japan, Ltd. [2)] The University of Tokyo

ABSTRACT

This paper proposes a novel method for real-time rendering of polygon mesh surfaces with reflection or refraction. The basic process is similar to dynamic environment mapping or cube mapping. Our proposed method is superior to those in that the accurate ray direction is reflected in the resulted image at every vertex on the mesh. Existing real-time techniques suffer from the differences between the viewpoint for the environment map and each reflection point. The proposed method minimizes this by finding an optimal viewpoint for the reflective or refractive mesh. With a sufficient number of vertices and map image resolutions, the users can render reflected images as accurate as ray tracing for all practical purposes, except for reflected objects around ray converging points of reflection on concave surfaces or refraction through convex lenses. The method can be applied to areas which require accuracy such as industrial design. Experiments with a CAD model of a car rear-view mirror and spectacle lenses exhibited results of sufficient quality for design verification.

1. INTRODUCTION

Reflection and refraction are essential elements of realistic image generation. Ray tracing [1] is a common rendering technique to simulate reflection or refraction. Although environment mapping and its variations are used for fast rendering of these effects, these techniques are effective for games and some concept design applications in which the users require only plausible reflection or refraction. In contrast, for simulation oriented applications such as mirror or lens design, ray tracing has been mandatory because of its accuracy. While acceleration techniques of ray tracing have been investigated from various standpoints, real-time rendering is still difficult to achieve for scenes of particularly large data sets.

Our method provides a solution to real-time, high quality reflection/refraction by limiting the ray-object intersection calculation to per-vertex basis on the given surfaces. At surface points other than vertices, the result is interpolated in the image space. The proposed method covers simple and moderately curved surfaces but it can be extended for large and complex surfaces.

This paper contributes to the application fields that require some reflection or refraction accuracy, rather than games and entertainment in which environment or cube mapping suffice. For example, automobile designers need to check the reflecting image on the rear-view mirror under design and are forced to use ray tracing. Our method enables them to verify the reflection/refraction result in real-time.

2. PREVIOUS WORK

A common technique to simulate accurate reflection is ray tracing [1], which has a drawback in slow processing speed. A number of methods to accelerate ray tracing have been proposed in the past [2][3][4]. Even with the current CPU performance, however, real-time rendering is difficult to achieve for data sizes of practical use.

Environment mapping [5][6] and cube mapping [7] are popular techniques to mimic reflection or refraction. Since they can be implemented in graphics hardware, real-time processing is possible even when the maps are dynamically rendered.

A major problem of these real-time techniques is their accuracy for reflection of local objects. As a solution to this issue, Hakura et al. [8][9] presented parameterized environment mapping (PEM). It pre-renders with ray tracing multiple environment maps or cube maps from different sample viewpoints around the reflector object, creating layered maps for both distant and local environment to be synthesized using the alpha channel. They showed that the results of PEM are very similar to those of ray tracing. While PEM is effective when viewing a static or rotating object [10] or when the viewpoint rotates around a target object, it requires vast amount of pre-computation when the object rapidly moves through the environment.

To compensate the local reflection errors more generally, distance impostors have been proposed recently [11][12][13]. They use the depth information of the environment map and calculate more accurate reflection/refraction by iterating and converging toward the true ray-scene intersections. The accuracy of their methods highly depends on the structure of the reflected objects in the environment. For large planar objects distance imposters easily find the accurate ray-object intersection. However, if an object size is small or its depth largely fluctuates, i.e. its surface is irregular, the iteration process can fail to find a more accurate intersection.

Our proposed method computes accurate ray-scene

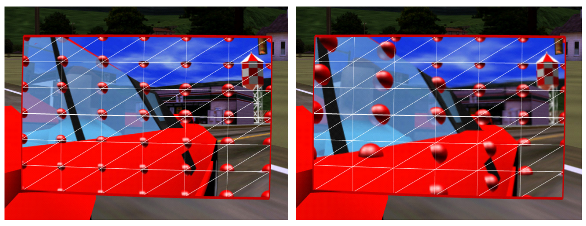


Figure 1. Left: A result of per-vertex reflection. Right: A cube mapping result as a reference. The reflected red balls are placed at the exact intersections between the reflection rays from the mirror vertices (cross points of the white wireframe mesh) and the scene. Note that in per-vertex reflection every mirror vertex accurately reflects the intersection between its reflection ray and the scene.

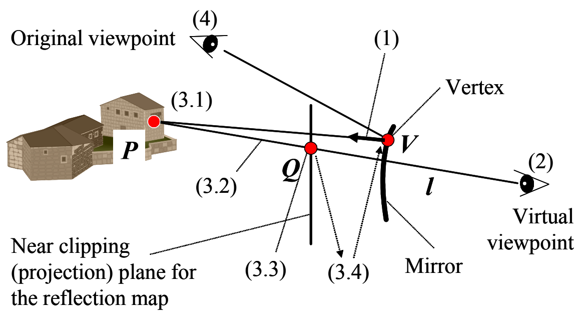


Figure 2. The processing steps of per-vertex reflection.

intersection for each vertex of the reflectors/refractors. While there is a performance penalty compared to distance impostors, our method computes accurate reflection at each vertex. The performance issue could potentially be improved by combining our method with recent acceleration methods for ray-scene interaction [14][15][16]. By using sufficient number of vertices the result becomes practically as accurate as ray tracing while still keeping interactive frame rates.

A different approach for real-time reflection was introduced by Ofek et al. [17]. Their system deforms the reflected 3D objects according to the shape of the mirror surface. Since the computational complexity of their method is greater than the dynamic environment mapping techniques, it is not suitable for scenes with large data sets and even less suitable for more accurate reflection.

For refraction, Ohbuchi [18] realized a real-time method using refraction vectors but his method fails to use optimal viewpoint for the refraction map and cannot be applied to reflection. A couple of other real-time refraction techniques were introduced [19][20] but they are limited to refraction for distant objects. Our method handles refraction of both local and distant objects, maintaining the accuracy of the result at each vertex on the refracting mesh.

3. PER-VERTEX REFLECTION

This section describes the algorithm of the proposed method, per-vertex reflection and refraction (PVRR), focusing on the reflection on a single, convex surface. Our proposed PVRR method assumes that the reflective or refractive objects are polygonal mesh surfaces. This section focuses on a moderately curved surface as a target. Section **4** will describe how to cope with complex surfaces. *Figure 1* presents a sample result of PVRR compared with a standard cube mapping method.

3.1. Overview of the process

The process of PVRR is similar to that of cube mapping. In the first pass, the system renders an image which will be mapped to the surface of the reflector. The second pass renders the surface using the first pass result as a texture, which we call a reflection map in this context.

The basic idea of PVRR is that it warps the reflection map so that each vertex reflects the accurate scene intersection point of its reflection ray.

An overview of the algorithm carried out within a frame is shown in *Figure 2* and described as follows using the same annotation number for each step. In the current implementation, no customized shader programs were used.

(1) From a given original viewpoint and the normal vectors of the mirror (reflective object), find a reflection ray from each vertex of the mirror.
(2) Find a virtual viewpoint, define a view volume, and render an intermediate image to be used as a reflection map (1st pass).
(3) For each vertex ***V*** of the reflective objects, generate texture coordinates as follows.
  (3.1) Trace the extension of the reflection ray from ***V*** to find the nearest 3D intersection ***P*** with the scene.
  (3.2) Find the line ***l*** that connects ***P*** and the virtual viewpoint.
  (3.3) Find the point ***Q*** where the line ***l*** intersects with the projection plane for the reflection map.
  (3.4) Copy the local image coordinates of ***Q*** in the reflection map to the texture coordinates of the relevant vertex ***V***.
(4) Render the scene and the mirror from the original viewpoint (2nd pass).

3.2. Finding virtual viewpoints

In ray tracing, intersection tests and rendering are carried out on a per-pixel basis. In our proposed method, an intersection calculation occurs for each vertex of the reflector and the rendering is carried out only once for a reflector object on the 1st pass. To minimize the image discrepancy, the 1st pass viewpoint which we call the virtual viewpoint should be close to every reflection ray from the reflector.

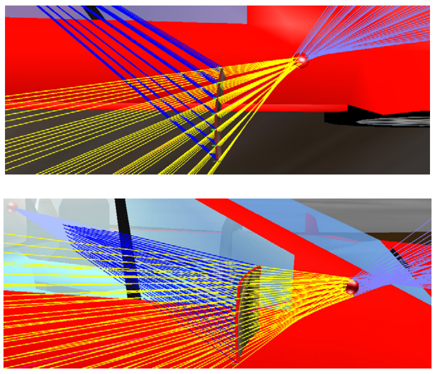


Figure 3. A top view and a side view of a curved mirror (center), its extended reflection lines (yellow and light-blue), rays (blue) from the original viewpoint (back in the side window), and the virtual viewpoint (red ball) found by the least-squares method.

We used a least-square method to find the virtual viewpoint which should be as close to all extended lines of the reflection rays as possible. The square distance from a point to each line is summed up as an evaluation function. Finding the point which minimizes the square distances is a similar problem to finding the straight line that best represents the correlated distribution of given points.

Figure 3 is an example of a virtual viewpoint found from a given original viewpoint and a meshed mirror.

3.3. Defining a view volume

Once the virtual viewpoint is found, it is a simple process to determine a view volume, or a view frustum, for rendering the 1st pass reflection map image. The frustum should contain all reflection lines between the near and far clipping planes. *Figure 4* is a 2D illustration of such a view frustum.

First, a viewing direction is defined by averaging the reflection ray directions. Second, the near clipping plane is placed just in front of the mirror object. This will exclude the mirror itself from the view frustum and will make the objects before the mirror to be contained in the view frustum. Then the far clipping plane is set at a sufficient distance from the virtual viewpoint, to contain the whole scene in the viewing direction. Then the other four boundaries, upper, lower, left, and right, are determined so that the frustum contains all reflection line segments between the near and far clipping planes.

Using the above view frustum, the system renders a reflection map and transfers the image to the texture memory.

3.4. Assignment of texture coordinates

Since the algorithm uses a single image (1st pass result) viewed from the best viewpoint for the reflection texture, mapping an accurate reflected image is equivalent to

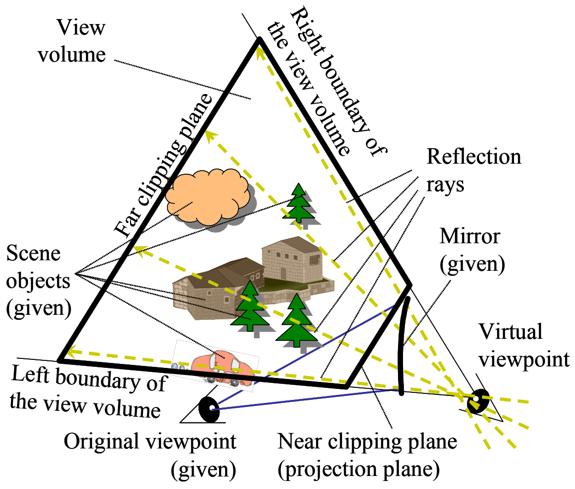

View volume
Right boundary of the view volume
Far clipping plane
Reflection rays
Mirror (given)
Scene objects (given)
Virtual viewpoint
Left boundary of the view volume
Original viewpoint (given)
Near clipping plane (projection plane)

Figure 4. The view volume for the 1st pass rendering.

warping the image appropriately. The texture mapping hardware warps the image and the problem is equivalent to computing appropriate texture coordinates for each vertex of the reflector.

The process of texture coordinates computation has been described in step (3) in Section **3.1** and in *Figure 2*. The heart of the algorithm is to find the 3D intersection point ***P*** between the reflection ray and the scene (step 3.1). The rest of the process is to warp the 1st pass image so that the relevant vertex ***V*** accurately corresponds to ***P*** in the image.

The 1st pass image can be regarded as has been projected onto the near clipping plane in *Figure 2*. In this context ***P*** should have obviously been rendered at the point ***Q*** on the image. ***Q*** can be texture-mapped onto the relevant vertex ***V*** by simply assigning ***Q***'s normalized local image coordinates (*s*, *t*) as the texture coordinates of ***V***, where $s,t \in [0,1]$.

With the above process this algorithm guarantees that the mirror reflects the accurate reflection point ***P*** at its vertex ***V***.

With the texture coordinate generation step, the most time-consuming sub-step is the ray-scene intersection (step 3.1). Our method should be able to combine most ray-scene intersection acceleration techniques including recent work such as kd trees [14], Bounding Volume Hierarchy [15], and vertex culling [16]. Many other classical acceleration methods are surveyed in a book [21].

In our implementation, we used a scene graph toolkit OpenGL Performer [22]. The toolkit uses an optimized ray-scene intersection routine with a bounding sphere for each node in the hierarchical tree-structured scene graph.

3.5. NURBS models as reflectors

We applied the method to a tessellated NURBS surface model for a prototype mirror exported from a CAD system. *Figure 5* shows its reflection results.

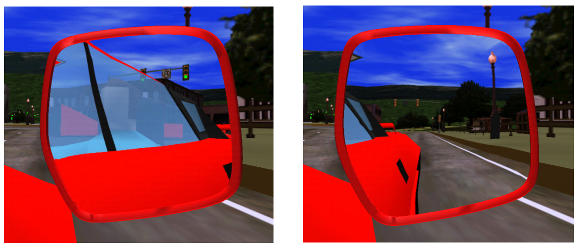


Figure 5. Results using a NURBS model of a prototype door mirror with two different mirror angles.

4. VARIATIONS OF PER-VERTEX REFLECTION AND REFRACTION

This section describes the extensibility of the PVRR method and indicates, by showing sample results, that the method can cover many of the general features of standard ray tracing.

4.1. Reflection image continuity between two surfaces

Our proposed per-vertex reflection handles moderately curved surfaces in order to avoid resolution degrading or distorted images. This will be a limitation when we apply this technique to a large, extended reflector or refractor. Such a shape contains vertices of greater distributions of positions and normal vectors. Since some reflection rays may point in significantly different directions from others in these cases, using a single common virtual viewpoint would easily cause a badly formed view volume and the reflection image may be severely distorted.

To handle such extended reflective/refractive surfaces, one may need to subdivide the surface into a set of moderately curved surfaces, as illustrated in *Figure 6*, until the field-of-view angle of each view volume falls below a certain threshold. This paper leaves the problem of adaptively subdividing large surfaces for future work.

Here, we focus on a necessary condition for the correct subdivision result: the continuity of reflected images between two adjacent mesh surfaces which have separate virtual viewpoints and view volumes. If the two meshes share vertices on their boundary, the two adjacent reflected images will be somehow continuous at the vertices since the reflected result at each vertex should have been shared between the two images by the nature of per-vertex reflection. A thorough analysis of the image continuity is a future subject.

Figure 7 demonstrates a simple example of two subdivided mirrors, expressing little discontinuity along the boundary of the reflected images.

4.2. Reflection on concave surfaces

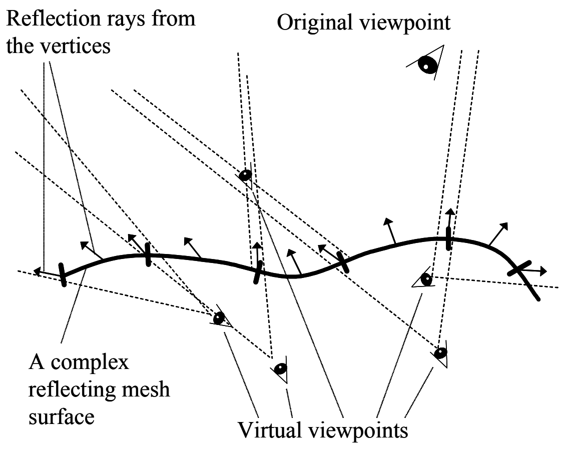
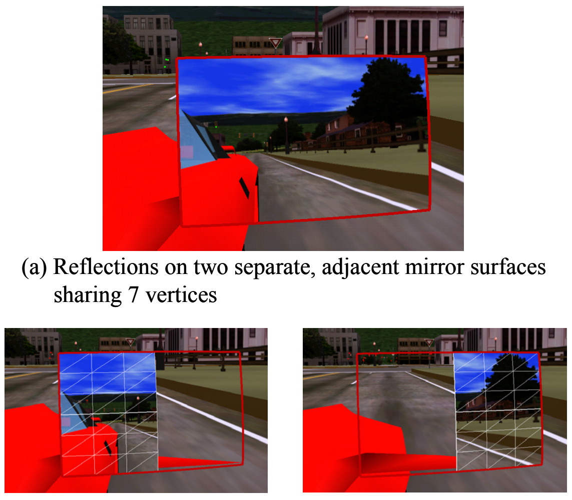

Reflection rays from the vertices
Original viewpoint
A complex reflecting mesh surface
Virtual viewpoints

Figure 6. Adaptive subdivision of a complex surface, which may be needed for future extension of PVRR.



(a) Reflections on two separate, adjacent mirror surfaces sharing 7 vertices

(b) The left mirror surface (c) The right mirror surface

Figure 7. Reflection image continuity between two subdivided polygonal surfaces. Each surface has its own virtual viewpoint, rendering pass, and reflection texture.

Per-vertex reflection can be naturally extended to reflection with concave mirrors. *Figure 8* shows how a virtual viewpoint is defined in such cases.

With the current implementation, we simply ignore the “zooming region” where local objects are very much expanded when reflected. *Figure 9* presents a series of mirror reflection results arranged from convex mirror surfaces, through almost flat, then to concave surfaces (using the reversed region reflection map described in *Figure 8*(b)).

4.3. Reflection on a surface with complex curvatures

Saddle shape surfaces are the most difficult type of object as reflectors for our method. It cannot be

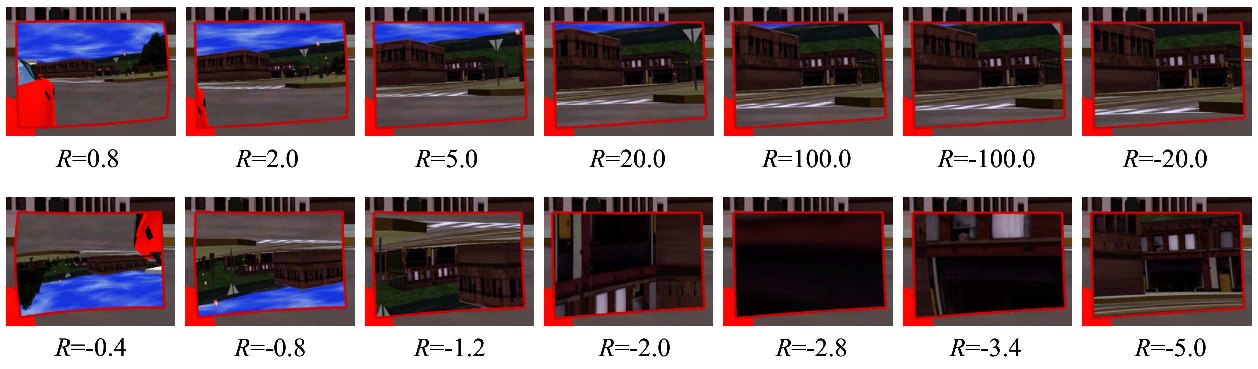


Figure 9. Examples of a transition from convex reflection to concave (from top-left to top-right and then from bottom-right to bottom-left) using a polygon-meshed spherical surface. R is the radius of the curvature. It is negative when the curve is concave.

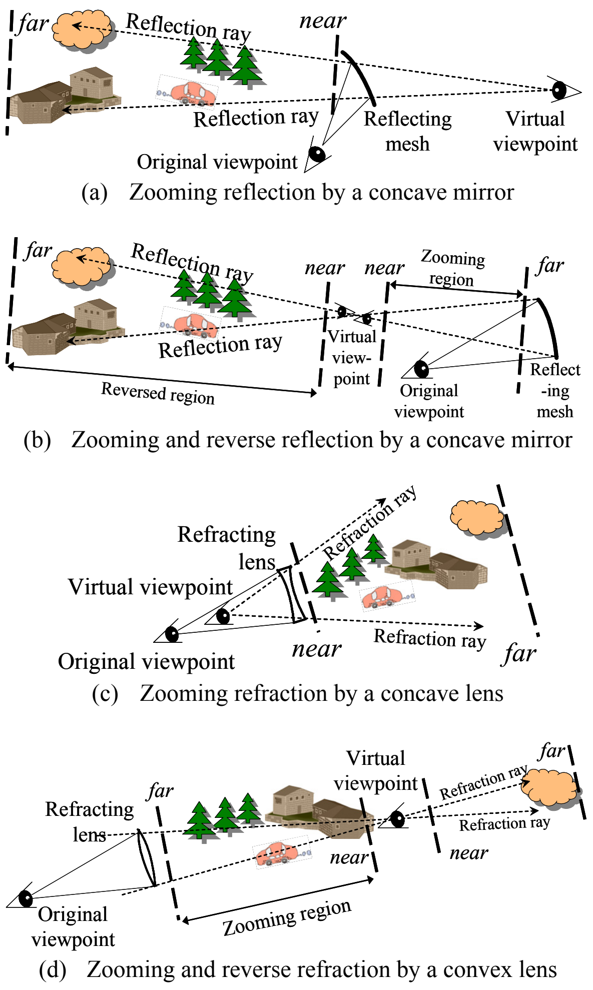


Figure 8. Reflection rays and the virtual viewpoints for a concave mirror, a concave lens, and a convex lens.

decomposed into a set of moderately curved surfaces. In those cases our algorithm would not cause a fatal error but the concave reflection image might be expanded so much that the quality of the reflection would be

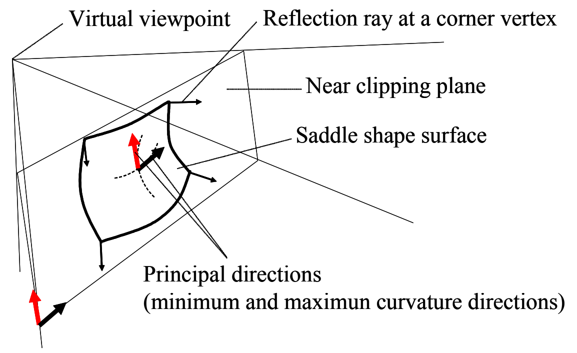


Figure 10. Defining a view volume for a shape with complex curvatures such as saddle and cylindrical shapes. Instead of showing the whole view volume, we show only its near clipping plane here.

degraded. This is because the view volume is conservatively built to contain all reflection rays including the convex reflection rays, and the field-of-view angle tends to be too wide for the concave reflection.

A similar problem should be expected for cylindrical surfaces although the quality loss may be more moderate.

A solution to this problem would be to align the up vector of the 1st pass view volume to one of the two principal directions of the reflective saddle or cylindrical surface. Since the principal directions are perpendicular to each other, the direction along which rays exhibit the widest angular distribution, tends to be perpendicular to the direction with the narrowest reflection ray angular distribution. By aligning the up vector to a principal direction, the aspect ratio of the view volume could be optimized for the reflection ray distribution. As a result, the distortion of the reflection map should be minimized.

Figure 10 illustrates a case that the up vector (a red arrow at the bottom-left corner) is aligned to the minimum (most concave) curvature direction (another red arrow at the center of the surface).

In this case the reflection rays along the most

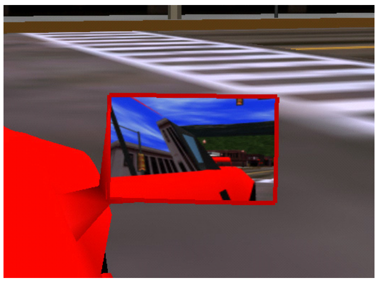


Figure 11. A multiple reflection example.

convex direction should tend to have a wide range of angular distribution and they are contained by defining a wide field-of-view (FOV) of horizontal direction. The rays along the most concave direction tend to have a narrow distribution of their angles, and are contained by a small FOV in the vertical direction of the view volume.

4.4. Multiple reflection

Multiple reflection and/or refraction can be carried out by simply placing two or more such objects at appropriate positions and directions. Its rendering process is recursive, which is similar to ray tracing. A ray tracing recursive process targets microscopic granularity; it manages ray hit points and brightness on a per-pixel basis. On the other hand, multiple PVRR should maintain surfaces and texture maps in a stack data structure for the recursion.

A multiple reflection example is shown in *Figure 11*. A background scene is first reflected in the side windows of the car model and then in the mirror.

4.5. Refraction

The per-vertex reflection method can be naturally extended to refraction. We developed a lens simulator using this method. *Figure 12* presents sample results of refraction for simple spectacle lenses. For each ray from the original viewpoint to a vertex on the lens front face, the refracted rays are calculated twice; first at the front face vertex and then at a point on the back surface.

Figure 13 shows another convex-lens refraction example with a much greater distance between the viewpoint and the lens than the previous example. As with real magnifying glasses, the image is reversed when the lens is far from the eye.

Figure 14 demonstrates the double refraction accuracy at each vertex on the lens surface. In the wireframe portion of the enlarged image (right), the refracted red small balls at the accurate ray-scene intersections match the vertices of the lens.

Figure 15 is a color aberration simulation for a convex lens. The lens surface is rendered three times while recalculating all the texture coordinates (step 3 in Section **3.1**) with different refraction indices for red, green, and blue, and also controlling color masks for red,

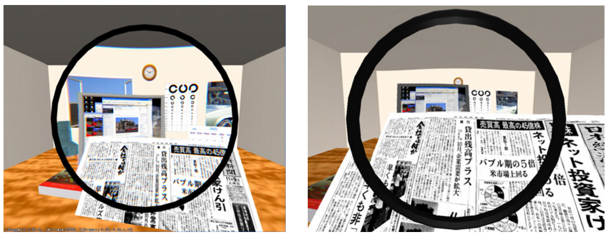


Figure 12. Examples of per-vertex refraction using a convex lens model (left) and a concave lens model (right).

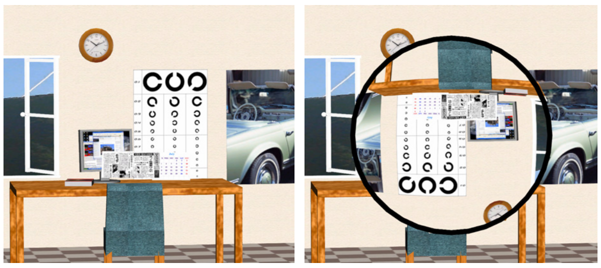


Figure 13. A reversed image example with convex lens refraction (right). The distance from the viewpoint to the lens is set to be 76cm in contrast to 7cm and 6cm for Figure 12 examples.

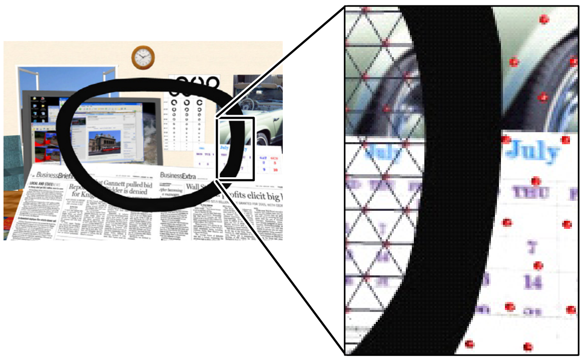


Figure 14. Refraction by a concave lens. The red tiny balls in the enlarged image had been placed at the 3D intersections between double refracted rays and the scene. Note that refracted balls match the vertices of the lens. This means that the refraction image is accurate at each vertex of the lens.

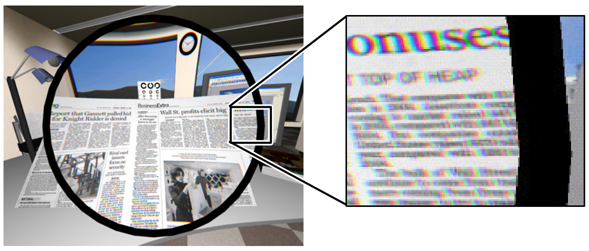


Figure 15. Simulation of color aberration of a convex lens. Three refraction indices were used for red (1.60), green (1.61), and blue (1.62).

green and blue. The 1st pass image for the lens texture is shared by the three lens-rendering tasks.

5. PERFORMANCE

Most scenes introduced in this paper ran in real-time or interactive frame rates. Most of the reflection examples using the car and the town models (total 10,000 polygons) ran in 140-170fps with NVIDIA GeForce 8800 Ultra and 2.66GHz Intel Core 2 Quad. A multiple reflection example (*Figure 11*) was 70-90fps.

The refraction examples in *Figures 12, 13* and *14* ran at 8-10fps. With an assumption that the lens position relative to the eye (the original viewpoint) is fixed, we can skip updating the refraction vectors and the virtual viewpoint. In this case the frame rate goes up to 70-80fps without color aberration. With color aberration it ran at 30-35fps (*Figure 15*). The lens model (both front and back surfaces) has 2,000 triangles and the rest of the scene has 5,000 triangles.

The most time-consuming process is searching ray-scene intersection for each reflector/refractor vertex (step (3.1) in Section **3.1**). Thus the number of ray-issuing vertices affects the performance. In the car mirror examples, the mirror was tessellated into 49 vertices to obtain desired quality (see *Figure 1* left). On the other hand, in our lens refraction examples, we used lens models with 500-1000 ray-issuing vertices for sufficient quality. This is why our lens refraction is slower than the mirror reflection. However, the assumption described in the previous paragraph is acceptable for most lens simulation and real-time results are easily achievable.

The snapshots in this paper have been taken with hardware antialiasing. In the meantime however, in measuring frame rates, hardware antialiasing was turned off because current hardware antialiasing leads to performance loss of 30% to 60%. A common solution is to switch off antialiasing in interactive operations and to switch it on when the scene is still.

6. DISCUSSION

A limitation of the proposed method is that the quality is affected by the layout of the vertices of the reflecting or refracting objects. For example, it is difficult to apply this method to reflective models defined by sparse vertices such as a large quadrilateral. The users may need to preprocess such geometry by tessellating into a polygonal mesh with a sufficient number of vertices.

Another limitation is a potential inconsistency of scene occlusion. This is the case that some object which is not hit by the reflection ray, resides on a line connecting the virtual viewpoint and the intersection of the reflection line with the 3D scene. *Figure 16* illustrates such a case. In this example, the car model will be reflected at a vertex in the mirror, which is incorrect because the reflection ray from the vertex does not intersect with the car. The reason for this is that the

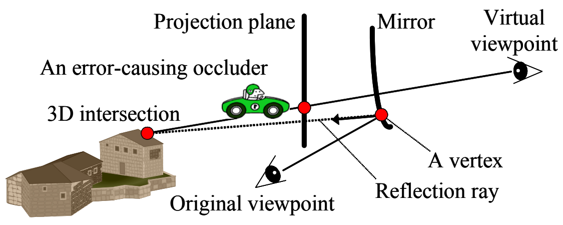


Figure 16. A case in which an occlusion inconsistency occurs in a reflection example.

virtual viewpoint does not reside on the extended lines of the reflection rays in general (it does for planar reflection). In other words, the viewing direction for the virtual viewpoint is different from those of the reflection rays.

This occlusion inconsistency problem is impossible to avoid unless the mirror is planar. However, its possibility should be minimized by selecting the virtual viewpoint as close as each reflection ray using the least-squares method described in Section **3.2**.

7. CONCLUSION AND FUTURE WORK

This paper proposes a per-vertex reflection and refraction method. It has the following advantages.
Performance: Runs in real-time, or in interactive rates.
Scalable accuracy: Guarantees the accurate direction to be reflected at each vertex of the reflector. More precisely tessellated surfaces would result in image quality closer to that of ray tracing.
Quality: The reflection map can be high in resolution and low in distortion in handling a moderately curved surface.
Continuity: The reflected images are continuous at the shared vertices between adjacent mesh surfaces.
Extensibility: Concave reflection, multiple reflection, and refraction are possible in the same framework.

Because of its scalable accuracy, our method is suitable for engineering simulation fields rather than games and entertainment, in which dynamic cube mapping or distance impostors should be sufficient because they produce plausible results.

Our future work includes adaptive surface subdivision as already mentioned in Section **4.1**, and concave reflection or convex refraction of zooming region as shown in *Figures 8* (b)(d). In order to render the zooming region, where the viewing direction goes back toward the mirror or the lens, the system needs to reverse the evaluation function of the depth buffer [23]. This will make the farther polygons from the virtual viewpoint occlude closer polygons, resulting in a correct depth order from the mirror or the lens.

Since the most time-consuming task is to find ray-scene intersections, it is worthwhile to implement recent acceleration techniques [14][15][16] from the ray tracing community as well as to adopt multi-core CPUs. A GPU implementation of a ray-scene intersection calculation or other sub-tasks may be a valuable subject to pursue as we currently do not utilize any customized shader programs.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their valuable comments. We also thank several people from SGI Japan; Naoya Hirai and Masaki Sawai for modeling interior and writing a lens model generator, Tomofumi Teratani for making IEVC2007 demo movies, and Gordon Jolley for proofreading the draft of this paper.

REFERENCES

[1] T. Whitted, "An Improved Illumination Model for Shaded Display," *Communications of the ACM 23*(6), pp. 343–349, 1980.

[2] A.S. Glassner, "Space subdivision for fast ray tracing," *IEEE CG&A*, 4(10), pp. 15-22, 1984.

[3] P.S. Heckbert, and P. Hanrahan, "Beam tracing polygonal objects," *Proc. SIGGRAPH '84*, pp. 119–127, 1984.

[4] J. Erickson, "Plücker Coordinates," *Ray Tracing News*, 10(3), http://www.acm.org/tog/resources/RTNews/html/rtnv10n3.html#art11, 1997.

[5] J. F. Blinn, and M.E. Newell, "Texture and reflection in computer generated images," *Communications of the ACM 19*(10), pp. 542-547, 1976.

[6] P. Haeberli, and M. Segal, "Texture Mapping as a Fundamental Drawing Primitive," *Proc. Fourth Eurographics Workshop on Rendering*, pp. 259-266, 1993.

[7] D. Voorhies, and J. Foran, "Reflection vector shading hardware," *Proc. SIGGRAPH '94*, pp. 163-166, 1994.

[8] Z.S. Hakura, and J. Snyder, "Realistic reflections and refractions on graphics hardware with hybrid rendering and layered environment maps," *Proc. Eurographics Workshop on Rendering*, 2001.

[9] Z.S. Hakura, J. Snyder, and J.E. Lengyel, "Parameterized environment maps," *SI3D '01: Proc. 2001 symposium on Interactive 3D graphics*, pp. 203-208, 2001.

[10] M.M. Stark, and R.F. Riesenfeld, "Reflected and transmitted irradiance from area sources using vertex tracing," *Proc. Eurographics workshop on Rendering*, 2001.

[11] L. Szirmay-Kalos, B. Aszodi, I. Lazanyi, M. Premecz, "Approximate Ray-Tracing on the GPU with Distance Impostors," *Computer Graphics Forum (Proc. Eurographics 2005) 24*(3), pp. 685-704, 2005.

[12] V. Popescu, C. Mei, J. Dauble, and E. Sacks, "Reflected-Scene Impostors for Realistic Reflections at Interactive Rates," *Computer Graphics Forum (Proc. Eurographics 2006) 25*(3), pp. 313-322, 2006.

[13] D. Roger, and N. Holzschuch, "Accurate Specular Reflections in Real-Time," *Computer Graphics Forum (Proc. Eurographics 2006) 25*(3), pp. 293-302, 2006.

[14] I. Wald and V. Havran, "On building fast kd-trees for ray tracing, and on doing that in O(N log N)," *Proc 2006 IEEE Symposium on Interactive Ray Tracing*, pp. 61–69, 2006.

[15] I. Wald, S. Boulos, and P. Shirley, "Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies," *ACM Transactions on Graphics.* 26(1), pp. 1-18, 2007.

[16] A. Reshetov, "Faster Ray Packets - Triangle Intersection through Vertex Culling," *Proc. Synposium on Interactive Ray Tracing*, 2007.

[17] E. Ofek, and A. Rappoport, "Interactive reflections on curved objects," *Proc. SIGGRAPH '98*, pp. 333-342, 1998.

[18] E. Ohbuchi, "A real-time refraction renderer for volume objects using a polygon-rendering scheme," *Proc. Computer Graphics International 2003 (CGI'03)*, pp. 190-195, 2003.

[19] E. Lindholm, M. Kilgard, and H. Moreton, "A user programmable vertex engine," *Proc. SIGGRAPH 2001*, pp. 149-158, 2001.

[20] C. Wyman, "An approximate image-space approach for interactive refraction," *Proc. SIGGRAPH 2005*, pp. 1050-1053, 2005.

[21] A. Glassner, "*An Introduction to Raytracing*," Academic Press, 1989.

[22] J. Rohlf, J. Helman, "IRIS performer: a high performance multiprocessing toolkit for real-time 3D graphics," *Proc. SIGGRAPH '94*, pp. 381-394, 1994.

[23] S. Vallance, P. Calder, "Inward looking projections," *Proc. GRAPHITE '03*, pp. 219-222, 2003.