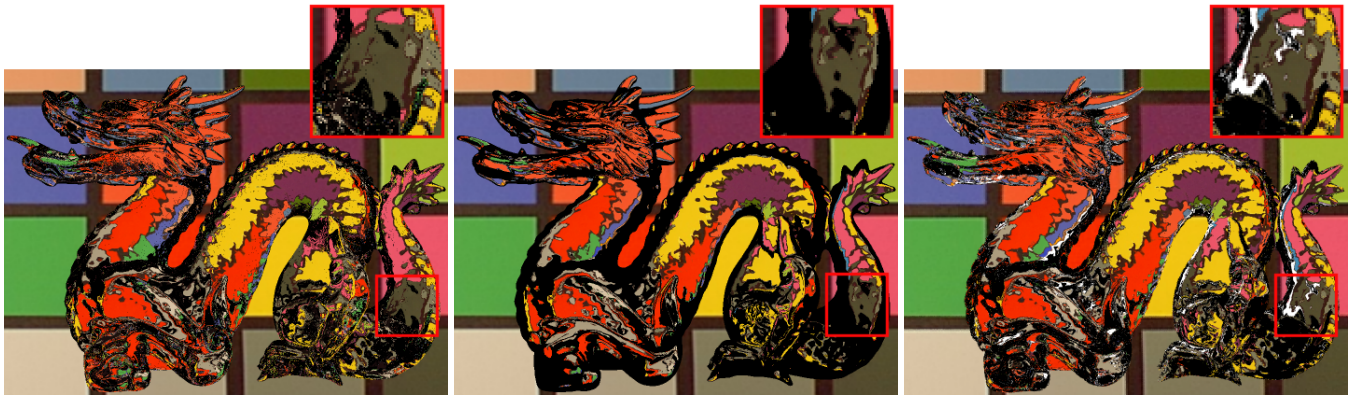
Interactive Refractions with Total Internal Reflection

Scott T Davis*
University of Iowa

Chris Wyman†
University of Iowa



(a) (b) (c)

Figure 1: Comparison with (a) our approach with 3 internal bounces, (b) image-space refraction [20] and (c) ray tracing with 8 samples per pixel.

ABSTRACT

A requirement for rendering realistic images interactively is efficiently simulating material properties. Recent techniques have improved the quality for interactively rendering dielectric materials, but have mostly neglected a phenomenon associated with refraction, namely, total internal reflection. We present an algorithm to approximate total internal reflection on commodity graphics hardware using a ray-depth map intersection technique that is interactive and requires no precomputation. Our results compare favorably with ray traced images and improve upon approaches that avoid total internal reflection.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

Keywords: interactive refractions, total internal reflections, binary search, graphics hardware

1 INTRODUCTION

The computational complexity of photorealistic rendering currently is too demanding for an interactive application. Such applications are designed to either forgo realism or approximate the relevant realistic parts. Given enough computing time, photorealistic image generation is possible and typically involves the use of an offline renderer using some variant of the ray tracing paradigm. Conversely, interactive rendering conventionally runs on a graphics processing unit (GPU) using a rasterization based algorithm.

A photorealistic rendering requires many cooperating components, but, most importantly, must capture the behavior of light in a scene. Material properties can alter the direction of light rays and thus place more burden on a renderer where coherency is generally more efficient such as is the case with the projection matrix in rasterization. A dielectric is a common real world material that bends light and thus offers realistic renderers a challenging material to simulate efficiently. Dielectrics mostly refract incident light although some is reflected and the rest is absorbed. The phenomenon of total internal reflection (TIR) takes place when the dielectric refracts no light, but rather mostly reflects and to a small extent absorbs. TIR occurs frequently in dielectric objects and thus is important for realism.

Previous approaches to interactive refraction have placed assumptions on the light propagation, the 3D model, or require offline preprocessing. Typically these approaches avoid handling TIR. We propose a technique for generating interactive refractions specifically designed to approximate TIR. Our technique involves an iterative search in 2D for ray intersections using depth maps to represent objects. See Figure 1 for a comparison of our results with image-space refraction [20] and ray tracing. Notice how our approach more closely resembles the ray traced image, especially in areas of TIR.

In the following, we discuss work in interactive refraction and iterative techniques followed by our approach. We conclude with our results, a discussion, and our ideas for future work.

2 PREVIOUS WORK

Reflections and refractions in computer graphics have long been accomplished through ray tracing [19]. Ray tracing offers an elegant method to handle TIR by recursively tracing rays through a dielectric until an intersection with a diffuse material. The drawback to ray tracing is the computational requirement that prohibits interactivity on commodity hardware.

Commodity graphics cards use rasterization to generate images. With rasterization, dielectrics largely are approximated using only the front side of the object [11], [13] or using the front and back sides of an object [20]. Dividing an object into two sides, a front and a back, gives much more plausible results than using the front side only, although it still cannot faithfully represent all objects from all viewpoints.

Krüger et al. [10] present a GPU algorithm for reflection and refrac-

*e-mail: scott-davis-1@uiowa.edu
†e-mail:cwyman@cs.uiowa.edu

tion using a multi-pass approach that rasterizes photon paths into a texture. This paper alleviates the two sided refraction restriction by using depth peeling. Szirmay-Kalos et al. [18] present an approximate ray tracing algorithm for the GPU that handles refraction by creating a distance impostor around a refractive object. Then they use an iterative technique to determine the distance a ray travels inside an object. Chan and Wang [2] use a similar data structure called a geocube that, with some extra preprocessing, allows for efficient and more accurate distance estimation. Although Estalella et al. [4] do not handle refractions and therefore is not directly related, they do present a technique for interactive reflections using an iterative approach.

An early approach to refraction by Kay and Greenberg is the transparency algorithm [9]. Although it can handle an arbitrary number of refractive interfaces, it requires processing geometry strictly back to front, the assumption that light travels parallel to the z axis when not in a refractive medium, and also that an object has constant thickness.

Diefenbach and Badler [3] propose a multi-pass technique to handle planar refractors where light is assumed to refract exactly twice thereby shifting incident light to a parallel transmission direction. Guy and Soler [6] discuss rendering gemstones with complex lighting behavior, but restrict the geometry to convex and faceted objects.

Hakura and Snyder [7] present a hybrid algorithm of rasterization and ray tracing that capitalizes on the GPU's efficient environment mapping for distant reflections and refractions and ray tracing for quality required for close-ups. Using ray tracing in a preprocess to determine where light exits an object, Ohbuchi [12] then uses an iterative technique to approximate a final intersection. After sampling a sufficient number of light paths through a refractive object offline, Génevaux et al. [5] compress this data using spherical harmonics and then are able to decompress online at interactive rates. These algorithms require ray tracing in their precomputation and are therefore currently infeasible for realtime applications.

Recently Hu and Qin [8] have developed a similar algorithm to ours that uses a binary search to determine ray-object intersections with a depth map. Their work differs from ours in three main ways: 1) the heuristic they have chosen to determine front versus back facing intersections, 2) their choice of endpoint for the binary search does not involve clipping, and 3) they do not discuss how to handle multiple internal bounces.

3 Algorithm

Image space refraction utilizes textures to represent geometry where the advantage is that complex 3D geometry is reduced to a discrete 2D structure very suitable for current GPU processing. The textures typically used are depth maps for positions and normal maps for surface normals. The assumption from [20] is an object has two sides: a front and a back. While this is a coarse approximation for some concave geometry from some viewpoints, it still proves to generate plausible effects. We continue in this vein using 2 passes to generate 4 textures: front facing normals and depths and back facing normals and depths (Figure 2). To facilitate nearby scene geometry, we render color and depth maps for nearby scene geometry in an additional pass. The fourth and final pass is wherein we compute refractions and total internal reflections using the previous passes as input.

To compute refractions, we use Snell's Law

$$\eta_i \sin\theta_i = \eta_t \sin\theta_t$$

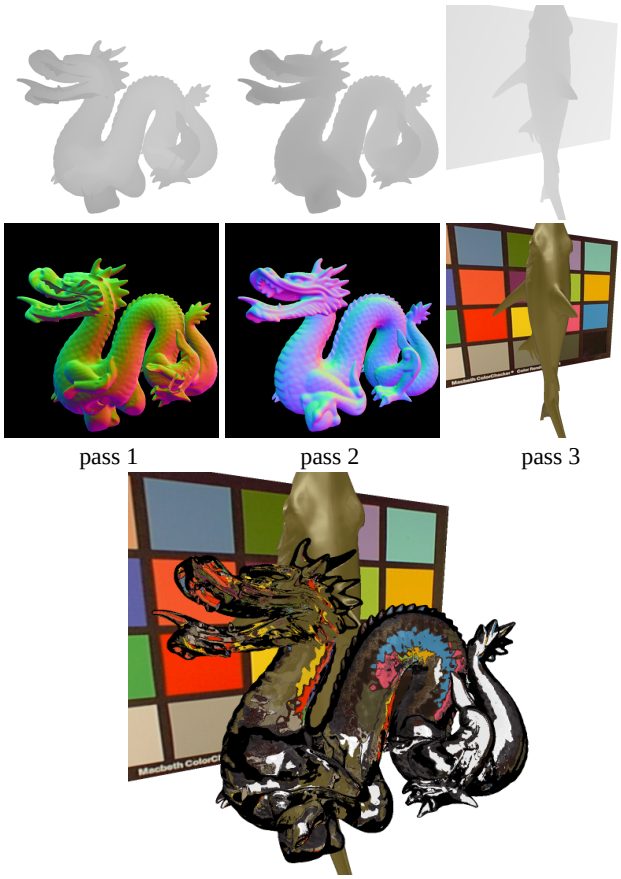


Figure 2: Depth and normal maps are generated in a first pass for the back facing geometry. In the next pass, we generate depth and normal maps for front facing geometry. In the third pass, we generate a nearby geometry depth map and color map and in the fourth and final pass, we render refractions with our technique.

to determine a transmitted angle (θ_t) when given an incident angle (θ_i) and the indices of refraction of the two participating media (η_i, η_t). From a front facing refractive surface point p, a simple fragment shader computes a transmitted direction T. The ray $p+tT$ intersects the scene at some $t = d$. Approximating this distance d (see Figure 3(d)) is the basis for most image space refraction techniques. Along with finding this distance, one must determine when TIR occurs.

The critical angle for a dielectric is the maximum angle incident light can impinge a surface without undergoing TIR (see Figure 4). Light incident at the critical angle transmits at 90°. It is possible, then, to determine the critical angle with Snell's Law for a dielectric within a certain media. Note that TIR is impossible when $\eta_i < \eta_t$ because $\sin^{-1} x$ is defined only for $x \in [-1,1]$. This means that TIR can only occur when light is moving from a more refractive to a less refractive medium. The formula for the critical angle is

$$\theta_{critical} = \sin^{-1} \frac{\eta_t}{\eta_i}.$$

Image-space refraction [20] approximates the distance d by linearly interpolating between two distances, namely the value in the back facing depth map (d_v) at the texel where the first intersection point projects and the distance in the negative normal direction to an in-

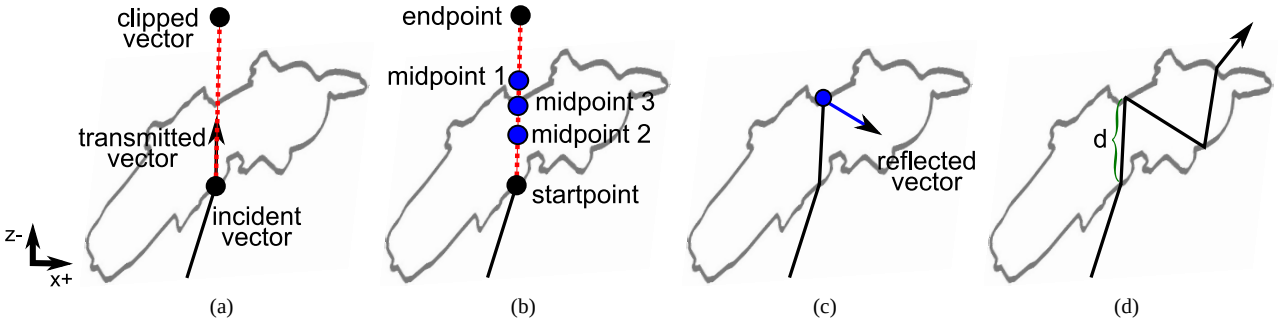


Figure 3: Compute the transmitted direction using the incident direction, normal, and indices of refraction and clip this ray against the view frustum (a). Use a binary search (b) to approximate the intersection between the transmitted ray and the object. Next this search is run on the front facing depth map (not shown) and the intersection point most aligned with the transmitted vector is chosen as the intersection. With this intersection point, lookup the normal stored in a texture and compute (in this case) a reflected vector. Now clip the vector and repeat the steps from (b) and (c) to find the next intersection. The actual intersections and ray directions are shown in (d).

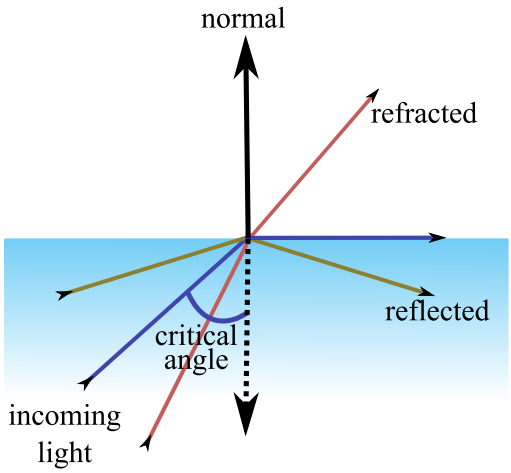


Figure 4: Light approaches the interface between two refractive media. When light is incident at an angle less than the critical angle (red), most light is transmitted through. Light incident at the critical angle (blue) transmits perpendicular to the surface normal and light incident greater than the critical angle totally reflects back into the surface.

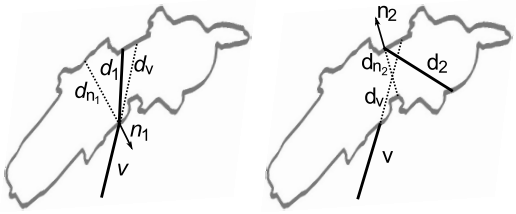


Figure 5: Image-space refraction uses d_n (the distance in the negative normal direction to an intersection) and d_v (the distance between the depth of the current pixel to the depth of the pixel on the backfacing geometry) to approximate the true refracted distance d. The first refracted ray is shown in a) along with the first normal n_1, approximate distances d_{n_1} and d_v, and the true distance d_1. The next (reflected) ray is shown b) along with the new normal n_2 and distance d_{n_2}. Notice the d_v value is the same as in a) as there is no analog for the distance from the viewer for the second ray.

tersection (d_n) (see Figure 5). While the first can be computed interactively per fragment in two rendering passes, the second value must be precomputed offline. The problem with this technique is the approximate d is too coarse and there is no guarantee that d follows the interpolation of the two distances. This leads to inaccurate refractions and ultimately the inability to recursively bounce for TIR. Further, d_v makes little sense to use for almost any of the internal light bounces. Our algorithm strives for more accurate refractions at fast enough speeds as to allow for multiple light bounces and retain interactivity.

A solution to find d is to rasterize in 2D along the ray comparing the z value on the ray to the z value in the depth texture at the current projected point. When these two values are within a certain threshold, an intersection is found. The speed of the solution is heavily dominated by the number of texture lookups, where there is one per iteration (rasterization step) making this solution too slow for interactive applications.

We seek to reduce the number of iterations to find an intersection by using a binary search similar to that described in [15]. The binary search converges on an intersection between the ray and a depth map value in no more than $\log_2 dim$ steps, where dim is the maximum of the two texture dimensions. To use the binary search, we need a startpoint and an endpoint. The starting point is the ray origin and the endpoint should be a point further along the ray than an intersection point. To find an endpoint, we choose to clip the ray against the viewing frustum and use the clipping point as the endpoint (see Figure 3a). If the correct intersection point is outside the viewing frustum, our endpoint is not far enough along the ray. Without depth information for the unseen geometry, though, our method fails to find an intersection in any case. Next is the binary search loop that will take at most $\log_2 dim$ iterations. To avoid the ray tracing "acne" problem, we offset the startpoint slightly in the direction of the ray. Find the midpoint m between the current two points, which at the beginning is the startpoint and the endpoint (see Figure 3b). Project m into texture space to yield m^{proj} with components $(m_x^{proj}, m_y^{proj}, m_z^{proj})$. Use (m_x^{proj}, m_y^{proj}) to index into the back facing depth texture to determine d_{lookup}, the value in the depth map.

There are five possibilities at this point and there is one choice to make: is it the startpoint or the endpoint that moves to the midpoint. To discuss the possibilities, consider a ray r that has a direction vector (r_x, r_y, r_z) and assume depth increases with $-Z$.

- When $r_z < 0$ then move the endpoint to the midpoint (when $m_z^{proj} < d_{lookup}$) or move the startpoint to the midpoint (when $m_z^{proj} > d_{lookup}$)

- When $r_z > 0$ then move the endpoint to the midpoint (when $m_z^{proj} > d_{lookup}$) or move the startpoint to the midpoint (when $m_z^{proj} < d_{lookup}$)
- d_{lookup}=1 (the farplane) then move the endpoint to the midpoint

This algorithm is not guaranteed to converge on the correct intersection. To ensure that the correct intersection is found, one could use a linear search before the binary search as in [15] or the more robust and efficient technique described in [1]. This work entails a preprocess to determine step sizes along a ray that ensure intersections are not missed, not falsely identified, and quickly determined. Our algorithm yields plausible results and so we prefer to not use the linear search and to skip the preprocessing. After we have completed a sufficient number of iterations, we take the last projected midpoint value from the depth map as our intersection point candidate.

After we have run the binary search on the back facing depth map, we run the search again on the front facing depth map. At this point we have two candidates for the intersection point. To determine which is more valid, we compute which point is more closely aligned with the ray using dot products (see Figure 6). If both intersection points are far enough from the transmitted direction, we consider this ray to have missed the object and we are done bouncing. With an intersection point i, we project i into the appropriate (front or back facing) normal map to retrieve the normal at this surface point. Now we determine if TIR has occurred and compute the next refracted or reflected direction accordingly. With this new direction and point, we can repeat this algorithm to simulate another light bounce. After we have bounced a sufficient number of times, we intersect this ray with the nearby geometry and or an environment map and quit. To intersect nearby geometry we can use the same binary search from above on the nearby geometry depth texture. However, since the ray that is intersecting the nearby geometry has a different view than the one used to create the nearby geometry textures, there could be parallax, disocclusion, and aliasing artifacts [21].

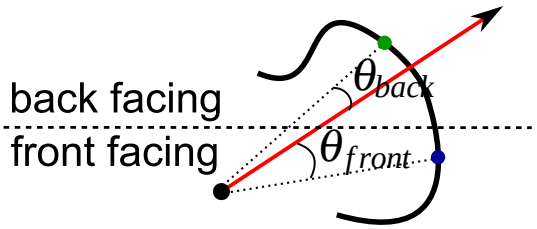


Figure 6: This figure shows how we choose the intersection point between the front and back candidates. The ray starting at the black point intersects the front and back depth maps according to our technique and results in the back intersection candidate (green) and front intersection candidate (blue). We choose the point that is more closely aligned with the ray (the smaller of θ_{back} and θ_{front} by means of two dot products.

4 Results and Discussion

We have implemented our approach using Cg shaders on a single processor Pentium 4 machine with an nVidia 8800 GTX. The ray traced image in Figure 1 has a ray depth of 10 and 8 samples per pixel while the rasterization-based images are 4x supersampled images at 512x512. The background in Figure 1 is a quad that is intersected analytically with outgoing rays from the refractive object. Other options are available for non-analytic and complex background geometry such as using a depth map [21] [17] or distance impostors [18] and an iterative root finding approach.

The intermediate rendering passes described in Figure 2 were created in offscreen framebuffer objects in 16-bit floating point textures. We experimented with using lower resolution intermediate textures and the results are in Figure 7. With only one bounce, it may be possible to get away with very low resolution $\frac{1}{16}$ or even $\frac{1}{64}$ the size of the final rendering. With adding one more bounce, intermediate buffers less than 256^2 produce too much noise in the final image.

To see the effect of adding more bounces as compared to ray tracing see Figure 8. Note in the ray traced image that reflected rays are not traced besides totally internally reflected rays so as to make an easier comparison with the other approaches. Figure 9 shows a refractive Buddha model with 250K vertices in front of the complex Venus model and inside the Uffizi environment map. Rays exiting the Buddha are intersected with the Venus using the binary search described in this paper with the nearby geometry depth map. Figure 10 is another example scene where complex geometry can be seen through a refractor.

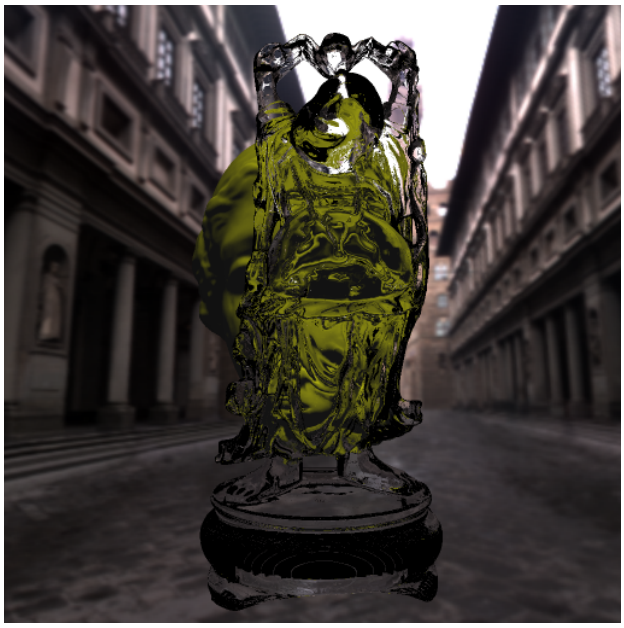


Figure 9: A Buddha refracting light in front of Venus.

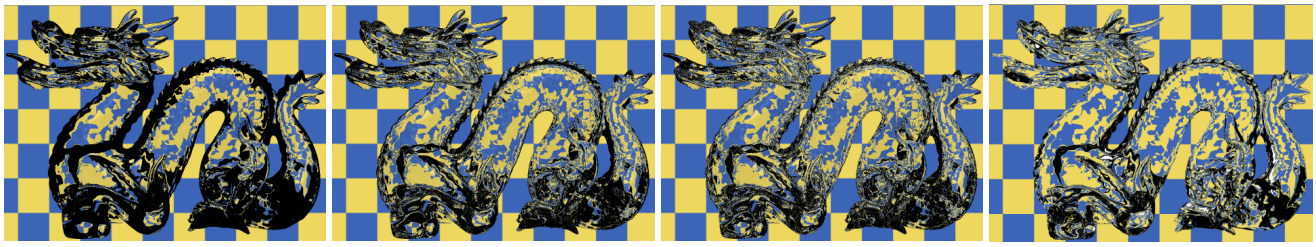
After a ray transmits through the front facing geometry, it typically intersects the back facing geometry. This is especially true for objects with low indices of refraction as the eye rays are not bending very much. If front facing geometry can effectively be ignored, one less binary search per bounce per pixel can be avoided. Some scenes will still require using both front and back facing surfaces such as the one in Figure 11.

Another place for speed up is in the number of binary search iterations. The more bounces a ray takes, the less impact each bounce has on the final image. In this spirit, we have chosen to make fewer iterations on bounces beyond the first. We found that using 8 iterations on the first transmitted ray and 5 iterations on the following bounces gives visual results similar to using more iterations.

Table 1 shows our frame rates for the sample scenes shown in this paper. Our approach is very much fragment bound and so the number of refractive fragments is most highly related to frame rate. For example, when the refractive dragon is viewed head on compared

64x64 128x128 256x256 512x512

Figure 7: This figure shows the effects of adding additional bounces using varied resolution intermediate buffers. With only one bounce, very low resolution buffers can be used with little artifact. The first row shows results for zero bounces, the second with one bounce, and the third with two bounces.



(a) (b) (c) (d)

Figure 8: The results of adding more light bounces with 1 bounce in (a), 2 in (b), and 3 in (c). (d) is the ray traced reference image.

to the side view (Figure 1a) there is a speed up by at least a factor of two because there are many fewer refractive pixels. We have deliberately chosen scenes that show the refractive object covering many pixels in the final image showing off the visual quality of the scene at the cost of slower, yet still interactive, image generation.

5 Conclusion and Future Work

We have presented an algorithm for modeling refractions with total internal reflection that requires no precomputation and runs interactively on commodity graphics hardware. We achieve this through a binary search for the distance a ray travels inside an object until an intersection. With this approximation technique for the distance, we recursively trace rays until we have exited the object or we have reached a defined recursion depth.

A limitation of this approach occurs when the object is only partially within the viewing frustum. The part that exists outside the frustum will not render to the depth textures that represent the object and therefore no intersections are made with that part of the object. A workaround is to render a slightly larger view frustum to try and capture more of the refractor as is presented in [4]. Another similar problem occurs when the nearby geometry partially or wholly lies outside the view frustum. Fragments outside the frustum never make it to the depth maps and thus can never be considered for intersection.

Our approach could be improved by adding more depth information about the object. It is possible with a concave object to have more than two interfaces project to the same texel. Such surfaces are disregarded in our technique. We could consider somehow generating a multichannel depth map, as in [14], so that we could intersect more surfaces and do so in parallel.

References

[1] Lionel Baboud and Xavier Décoret. Rendering geometry with relief textures. In *Graphics Interface '06*, 2006.

	512^2				1024^2			
bounces	1	2	3	4	1	2	3	4
kitchen	138	110	92	79	75	60	51	44
Buddha and Venus	43	34	29	25	26	21	17	15
dragon (250K) and Macbeth	38	31	26	22	23	18	15	13

Table 1: Frame rates in frames per second for test scenes shown in the paper with variable framebuffer resolutions and number of bounces. All timings are made using only the back facing geometry for multiple bounces and 8 binary search iterations on the first ray followed by 5 iterations on further rays.

Figure 10: A refractive Stanford Bunny in a kitchen with the Utah Teapot.

(a) (b)

Figure 11: In some scenes it is vital to use both the front and the back facing surfaces as shown in this example where in (a) only the back facing surfaces are used and in (b) both front and back are used.

[2] Bin Chan and Wenping Wang. Geocube - gpu accelerated real-time rendering of transparency and translucency. *The Visual Computer*, 21(8-10):579–590, 2005.

[3] Paul J. Diefenbach and Norman I. Badler. Multi-pass pipeline rendering: realism for dynamic environments. In *SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 59–ff., New York, NY, USA, 1997. ACM Press.

[4] Pau Estalella, Ignacio Martin, George Drettakis, and Dani Tost. A gpu-driven algorithm for accurate interactive reflections on curved objects. In Tomas Akenine Möller and Wolfgang Heidrich, editors, *Rendering Techniques'06 (Proc. of the Eurographics Symposium on Rendering)*. Eurographics/ACM SIGGRAPH, June 2006.

[5] Olivier Génevaux, Frédéric Larue, and Jean-Michel Dischler. Interactive refraction on complex static geometry using spherical harmonics. In *SI3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 145–152, New York, NY, USA, 2006. ACM Press.

[6] Stephane Guy and Cyril Soler. Graphics gems revisited: fast and physically-based rendering of gemstones. *ACM Trans. Graph.*, 23(3):231–238, 2004.

[7] Ziyad S. Hakura and John M. Snyder. Realistic reflections and refractions on graphics hardware with hybrid rendering and layered environment maps. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 289–300, London, UK, 2001. Springer-Verlag.

[8] Wei Hu and Kaihuai Qin. Interactive approximate rendering of reflections, refractions, and caustics. *IEEE Transactions on Visualization and Computer Graphics*, 13(1):46–57, 2007.

[9] Douglas Scott Kay and Donald Greenberg. Transparency for computer synthesized images. In *SIGGRAPH '79: Proceedings of the 6th annual conference on Computer graphics and interactive techniques*, pages 158–164, New York, NY, USA, 1979. ACM Press.

[10] Jens Krüger, Kai Bürger, and Rüdiger Westermann. Interactive screen-space accurate photon tracing on GPUs. In *Rendering Techniques (Eurographics Symposium on Rendering - EGSR)*, pages 319–329, June 2006.

[11] Erik Lindholm, Mark J. Kligard, and Henry Moreton. A user-programmable vertex engine. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 149–158, New York, NY, USA, 2001. ACM Press.

[12] Eisaku Ohbuchi. A real-time refraction renderer for volume objects using a polygon-rendering scheme. In *Computer Graphics International*, pages 190–195, 2003.

[13] Gustovo Oliveira. Refractive texture mapping, part two. Gamasutra, November 2000. http://www.gamasutra/features/20001117/oliveira_01.htm.

[14] Fábio Policarpo and Manuel M. Oliveira. Relief mapping of non-height-field surface details. In *SI3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 55–62, New York, NY, USA, 2006. ACM Press.

[15] Fábio Policarpo, Manuel M. Oliveira, and João L. D. Comba. Real-time relief mapping on arbitrary polygonal surfaces. In *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 155–162, New York, NY, USA, 2005. ACM Press.

[16] C. M. Schmidt. Simulating refraction using geometric transforms. Master's thesis, Computer Science Department, University of Utah, 2003.

[17] Musawir A. Shah and Sumanta Pattanaik. Caustics mapping: An image-space technique for real-time caustics. Technical Report CS TR 50-07.

[18] Laszlo Szirmay-Kalos, Barnabas Aszodi, Istvan Lazanyi, and Matyas Premecz. Approximate Ray-Tracing on the GPU with Distance Impostors. *Computer Graphics Forum*, 24(3):695–704, 2005.

[19] Turner Whitted. An improved illumination model for shaded display. *Commun. ACM*, 23(6):343–349, 1980.

[20] Chris Wyman. An approximate image-space approach for interactive refraction. *ACM Trans. Graph.*, 24(3):1050–1053, 2005.

[21] Chris Wyman. Interactive image-space refraction of nearby geometry. In *GRAPHITE '05: Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 205–211, New York, NY, USA, 2005. ACM Press.