Implementation of a Rendering Pipeline Based on Microfacets Models with Reflection and Refraction

Hongyang "Lyon" Lin
linh8@rpi.edu

Jacob "Jay" Idema
idemaj@rpi.edu

ABSTRACT

In this paper, we explore our processes and struggles implementing efficient, interactive-performance reflection, refraction and other physically-based lighting effects using environmental mapping in a rasterized, game engine context. We compared our rasterized results to results generated via a much slower, less efficient ray-tracing method and found that the behavior of our implementation closely matched the ray-tracing results. We also discuss some pitfalls of our implementation and some of our successes.

1. INTRODUCTION

Rasterization is a commonly used method to render 3D scenes. Its capability to utilize GPU power allows it to excel in runtime efficiency, which is ideal for real-time rendering. Therefore rasterization is commonly adopted in graphics fields when speed is crucial, such as in video games, where 30 ~ 60 frames need to be rendered within a second. Our implementation aims to function as a part of a game engine.

Game engines play an important role in modern game development. By abstract complex tasks and computations, game engines are used to aid design, help designers and artists, and provide an easy workflow with as little artistic restriction as possible. Our project implements rasterization models for physically-based materials in a game engine, allowing users of the engine to intuitively apply diffuse and non-lambertian materials to arbitrary meshes and to manipulate the properties of these materials using texture maps and exposed parameters (of roughness, metalness, IOR).

Save for the ray tracing application that we used to verify our baseline reflection and refraction work, we implemented our solutions in Lynx Engine, Hongyang Lin's game engine, which had a number of features already implemented. This decision allowed for more structured and intuitive debugging.

2. RELATED WORK

Cook and Torrance first introduced microfacets models into the field of computer graphics. His BRDF provides a flexible framework for later scholars to extend and modify [Cook & Torrance, 1981]. The proposed BRDF and its variations are widely used in graphics [Ngan et al., 2004].

Walter et al developed a method to render refraction through rough surfaces. The paper also proposed a modified combination of the normal distribution and geometry functions [Walter et al., 2007].

Brian Kevis, a graphics engineer who worked on the rendering pipeline Unreal Engine 4, did a comprehensive study on various components in BRDF. His work has provided an invaluable source for us to decide on BRDF components. Brian also discussed the approach for implementing the rendering pipeline in Unreal Engine 4 [Karis, 2013a][Karis, 2013b].

Blinn and Newell proposed a then novel technique, called environmental mapping, for mapping an environment, represented by a 2D texture, onto the surfaces of objects in a rendering pipeline [Blinn et al., 1976]. Blinn and Newell sampled this texture by converting vectors reflected from the surface of objects into polar vectors and querying uniquely warped textures using the azimuth angle of the vector as the abicass coordinate and the polar angle as the ordinate coordinate. Hoffman and Gene described an alternative environmental mapping method, called cube mapping or cube projection mapping, which utilizes six 2D textures representing a cube formed around a scene [Greene, 1986]. Hoffman and Gene's cube mapping is the method of environmental mapping we've made use of for reflection and refraction.

Chris Wyman introduced an approach to modeling refraction using two rendering passes, which nets results that are much closer to raytraced results than for single normal refraction used to sample an environmental map [Wyman, 2005]. We've directly adapted Wyman's method for our implementation of refraction.

3. THEORY

v	view vector
l	light vector
h	halfway vector (between light and view)
ω_o	outgoing direction
ω_i	incoming direction
N	the normal of a surface
I	normalized directional vector from camera to surface
R	directional vector of refracted light

3.1 Roughness, Microfacets

Radiance is a physics quantity originated in radiometry, in the field of optics. Radiometry measures electromagnetic radiation and the distribution of radiation's power in space. Radiance is a compound quantity that measures the radiant flux of a surface over a solid angle. The unit of radiance is watt per steradian per square metre. Radiance is useful not only because it allows us to measure the outgoing light from a surface, but also because it can be conveniently transcribed into shaders. If the surface area and solid angle are set to infinitely small, radiance effectively models a single light ray hitting one fragment. Irradiance measures the sum of radiance over all directions. The unit of irradiance is watt per square metre.

Microfacets theory states that every surface is composed by countless small mirrors, which are referred to as microfacets. When a light hits a microfacet, part of the energy enters the surface, while the rest part of the energy is reflected. The roughness of a surface is determined by the alignment of

microfacets. On rough surfaces, microfacets tend to face different directions, causing the reflected image to appear blurry. While smooth surfaces tend to have microfacets facing roughly the same direction, resulting in clearer reflections and more concentrated specular light. Metals and dielectrics react differently to light. Metals absorb all refracted light, showing no diffused color. Therefore, metals only leaves reflected light.

Instead of modeling individual microfacets, the surface is simplified as a macrosurface. We use the BRDF framework first introduced by Cook & Torrance to model the macrosurface with three components: a normal distribution function, a geometry function, and fresnel's equation. The normal distribution function **D** calculates the approximated ratio of microfacets that reflects the light in the viewer's direction. The geometry function **G** approximates the self-masking or shadowing ratio of the microfacets. Fresnel's equation **F** calculates the reflection ratio of the surface according to viewing angle.

$$f_{cook-torrance} = \frac{DFG}{4(\omega_o \cdot n)(\omega_i \cdot n)}$$

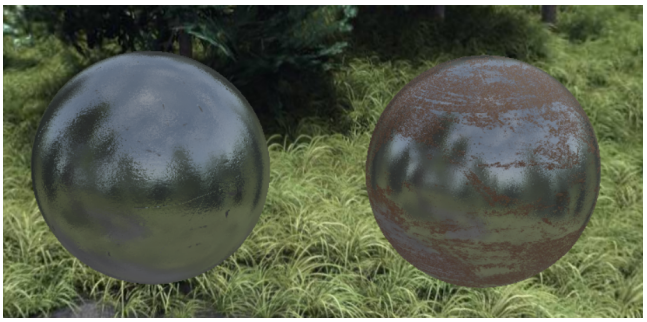

Figure 1: *Roughness Results on Reflective Surfaces, Steel (left) and Rusted Iron (right), both with a roughness value of 0.7.*

3.2 Refraction

Refractive objects bend light according to Snell's law due to the change in a light ray's speed when crossing between mediums:

$$n_1 * sin(\theta_1) = n_2 * sin(\theta_2)$$

n_1 is the index of refraction of the entered object. n_2 is the index of refraction of the exitted medium. θ_1 is the angle between the incoming ray's directional vector and the directional vector of the normal of the surface; if N is the normalized directional vector of the normal at the surface and I is the normalized directional vector from the camera to the surface, then:

$$\theta_1 = acos(dot(-I, N))$$

θ_2 is the angle between the inverted directional vector of the surface's normal and the refracted ray's directional vector. If R is the normalized directional vector of the ray after it has refracted:

$$\theta_2 = \text{acos}(\text{dot}(R, -N))$$

A vector-based equation can be derived from Snell's law and is as follows:

$$ratio = \frac{n_2}{n_1}$$

$$K = 1.0 - ratio * ratio * (1.0 - dot(N, I)^2)$$

$$R = ratio * I - (ratio * dot(N, I) + \sqrt{K}) * N$$

4. IMPLEMENTATION

4.1 Roughness

We implemented the glossy effect of microfacet roughness by forming mipmaps of our cube map textures, downscaling a texture's resolution by a factor of two for each sequential mipmap level of detail (LOD).

In our implementation, we use eight total images of decreasing resolution for our mipmaps. The material roughness of an object is clamped between zero (representing no glossiness) and one (representing the maximum glossiness). Given the material roughness of an object, we query the desired level of detail from the cubemap mipmap by multiplying the roughness by the max LOD value, which in the case of eight images would be a value of seven. At intermediate LOD, such as 1.5, GLSL can interpolate between the closest two mipmap textures, allowing for a gradual effect without the need for additional mipmap images.

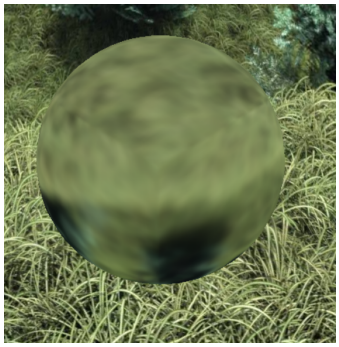

Figure 2: *Visible seams from poor convolution results on the edges of images of our cubemaps.*

For smaller values of roughness, this simple technique works well for a glossy effect, however at larger roughness values, the diminishing resolution of the mipmap images begins to become apparent. We attempted to perform blurring convolutions on the cubemap images before downscaling them in a naive attempt to correct some of these resolution artifacts. We tested convolutions using a three by three gaussian blur kernel and a five by five gaussian blur kernel. These convolutions did reduce visible artifacts although they did not completely remedy them. We ultimately chose to use a three by three gaussian kernel. While the three by three kernel does not hide resolution artifacts as effectively as the five by five kernel, it produces results more visually similar to our original implementation and produces less visible seams in the cubemap edges, see Figure 2. These seams are caused by convolving individual images of the cubemap with no knowledge on what is on the other side of edges. These seam issues could be remedied by convolving across images: sampling from the appropriate edge of a different image on the boundaries of an image when the kernel is partially outside the bounds of the image, but we did not implement this extension.

The roughness value also influences the normal distribution function, **N**, and geometry function, **G**. We adopted the combination of **N, G, F** recommended by Brian Karis [Karis, 2013b]. Normal distribution is calculated using Trowbridge-Reitz GGX method [Walter et al., 2007]:

$$D_{GGX}(m) = \frac{\alpha^2}{\pi((n \cdot m)^2(\alpha^2 - 1) + 1)^2}$$

The input *m*, in our case, is the halfway vector that sits between the light and view vector. α is defined as the square of roughness. The symbol α will be used across the following formula.

The Schlick-GGX method is selected for geometry masking and shadowing, combined with Smith's method. [Walter et al., 2007]

$$G_{Schlick}(v) = \frac{n \cdot v}{(n \cdot v)(1 - k) + k}$$

k is a remapping of the roughness term α.

$$k = \frac{\alpha}{\pi}$$

The Schlick-GGX method is used with Smith's method, which breaks **G** into two components, light and view. Same equation is used for both components[Smith, 1967].

$$G(l, v, h) = G_l(l)G_l(v)$$

Schlick's approximation is used for Fresnel's equation. **F0** is the reflectance at normal incidence [Schlick, 1994].

$$F_{Schlick}(v, h) = F_0 + (1 - F_0)(1 - (v \cdot h))^5$$

The BRDF is separated into three parts, light reflected, refracted, and diffused. Metalness is used to calculate the ratio reflected, and diffuse rate is used to divide the diffused light and refracted light.

$$f_r = k_d * f_{lambert} + k_r * f_{refraction} + k_s * f_{cook}$$

$$k_d + k_r + k_s = 1$$

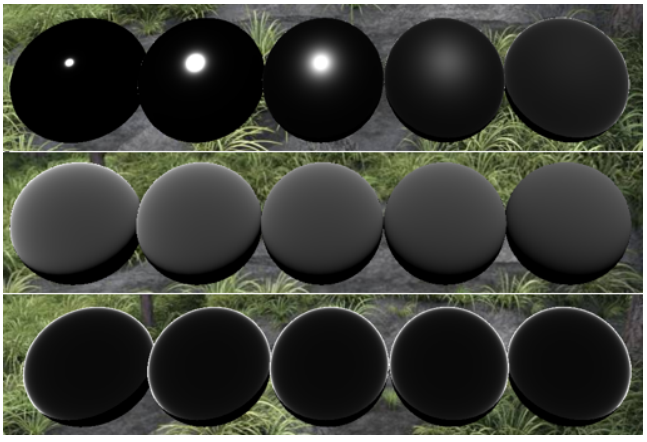

Figure 3: *Intermediate results of each component in the BRDF, from low roughness (left) to high roughness (right). Top: normal distribution function. Middle: geometry function. Bottom: Fresnel equation (not affected by roughness)*

4.2 Refraction

Figure 4: *A sphere modeled with a single refraction (left) and a raytraced sphere modeled with two refractions (right). The single refraction result imitates the inversion effect seen in the correct image (right) but fails to perfectly model its behavior.*

4.2.1 Motivation

While rays in a ray-tracing context are scene-aware and capable of refracting or reflecting an arbitrary number of times, pixel fragments in a rasterization context are much less scene-aware and can only easily be refracted a single time. For this reason, it is common to only model the first entrance refraction when modeling transparent objects. This can produce reasonable results, especially for simple objects with low indices of refraction, but failing to model the exit refraction can result in noticeable inaccuracies when an object is more refractive, see Figure 4, or when an object has surface normals which vary rapidly, making it obvious that the exiting refraction is not modeled.

4.2.2 Description

A rasterization refraction method was proposed by Chris Wyman [Wyman, 2005] which models the first entrance and first exit refraction of light through transparent objects. Our implementation of refraction is a direct adaptation of his method, in a hopeful attempt to capture the effects of exiting refractions in our materials.

Our implementation first makes a preliminary pass which renders the back faces of refractive objects, writing their world normals and camera depth to textures. In our case, we also need to write the roughness of the back faces.

The front faces of refractive objects are then rendered. As we know the direction from the camera to the surface, the normal of the surface, and location of the surface for a given fragment, we can compute the exact behavior of the entrance refraction using the vector equation derived from Snell's law. The second exiting refraction, however, cannot be as exactly computed, as we do not know the exact location nor normal of the surface where the "light" vector will exit the object. Wyman proposes estimating the point of exit by estimating the length the refracted vector must travel by linearly interpolating between the distance, D_d, a vector would travel if it had not refracted at all and the distance, D_l, a vector would travel if it had completely refracted, such that the refracted "light" vector is pointed in the exact opposite direction of the surface normal. The distance, D_d can be determined by:

$$D_d = Depth_{back} - Depth_{front}$$

We query $Depth_{back}$ by performing screen-space projection sampling to retrieve the depth value that was stored in the preliminary rendering pass.

D_I is precomputed using ray-tracing at the vertices of refractive objects. For each vertex, a ray, which can detect collisions with all the triangles of the mesh, is cast in the opposite direction of a vertex normal. The distance traveled by the ray until it strikes a triangle of the object is stored into the original vertex. Under the assumption that refractive objects are rigid-bodies and will not under-goes deformations, we only need to calculate these D_I distances once.

We interpolate between these two distances, D_d and D_I, using the ratio of the angles, θ_I and θ_r. θ_I is the angle between the negation of the directional vector of the incoming "eye ray" and the surface normal:

$$\theta_I = acos(dot(-I, N))$$

θ_r is the angle between the negation of the surface normal and the direction of the first refraction:

$$\theta_r = acos(dot(R, -N))$$

Finally, using the ratio of the angles to interpolate we get:

$$D_F = \theta_r/\theta_I * D_d + (1.0 - \theta_r/\theta_I) * D_I$$

With this distance, we estimate the point of exit, P_{exit}, which we then use as the world position for another screen-space projection to query the normal to refract by for the exiting refraction. We use this twice refracted directional vector R_2 to sample our cubemaps.

$$P_{exit} = Surface\ Position + normalized(R) * D_F$$

$$R_2 = refract(R, textureProj(P_{exit}), \frac{n_1}{n_2})$$

4.2.3 Refraction Difficulties

4.2.3.1 Preliminary Texture

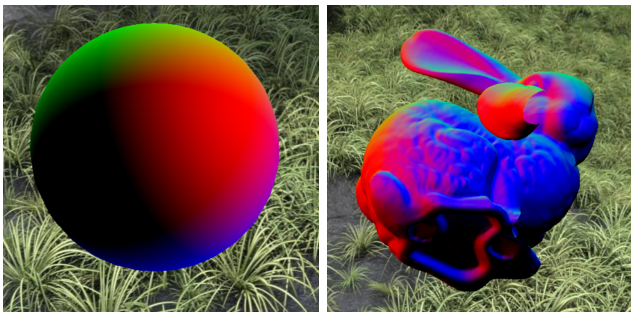

Figure 5: *Back face normals from the preliminary rendering pass, rendered onto the front faces using screen-space texture projection.*

A considerable amount of time was spent debugging construction and use of the preliminary projection textures during the preliminary rendering pass. The images in Figure 5 are some intermediate debug renders which output the normal values in the preliminary texture onto the front faces of the object.

4.2.3.2 Scaling

While simply storing the scalar distance from a vertex to the other side of an object was sufficient for Chris Wyman's implementation, in a game engine, objects can be scaled at runtime and can be scaled in non-uniform ways. To achieve interactive performance, it is not possible to recompute the internal distances by ray-tracing again whenever a transparent object is rescaled. Instead, we store a vector equal to the position struck by a vertex's casted ray subtracted from a vertex's position. While this does increase the amount of data we must store in each vertex, we can transform the "distance vector" by the world space matrix of the object (and thereby its scaling). Computing the distance from the transformed vector to an origin vector (0,0,0) transformed by the same matrix gives us an internal distance which adjusts with scale in an efficient manner.

4.2.3.3 Total Internal Reflection (TIR)

Figure 6: *Refractive sphere with IOR 1.5 with fragments for which TIR occurs highlighted in red (left) and refractive sphere with IOR 1.5 (right).*

When a ray exits a more refractive medium into a less refractive medium, there is a critical angle threshold (between the normal of the surface and the direction of the ray) where the ray refracts so drastically that it does not exit the object. This phenomenon, called Total Internal Refraction, is represented in the vector refraction equation when K in the refraction equation is less than zero:

$$ratio = \frac{n_2}{n_1}$$

$$K = 1.0 - (ratio * ratio * (1.0 - dot(N, I)^2)) < 0$$

Since we model "light" vectors "backwards" (from the camera into the scene), this is a difficult issue to resolve as in cases where TIR occurs we cannot model the exiting refraction. We compensate for this by only using the first entering refraction. This first refraction will never contribute to Total Internal Refraction, as it is a refraction caused by entering a less refractive medium (air) into a more refractive medium. This is not an ideal solution and causes very noticeable artifacts, see Figure 6. It is also noteworthy that our implementation of refraction seems to exacerbate this issue of Total Internal Refraction; while ray-tracing can also fall victim to TIR, it occurs more noticeably and at lower indices of refraction for our rasterization method than it does in our ray-tracing implementation, likely due to errors, minor and major, in our estimation of the exit location of our "light" vectors.

5. RESULTS

5.1 Main Results

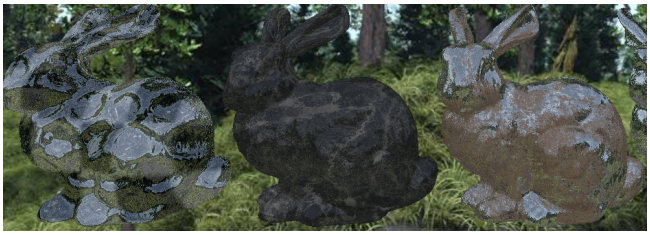

Figure 7: *Bunnies with custom material (left), marble material (middle), rusted iron material (right).*

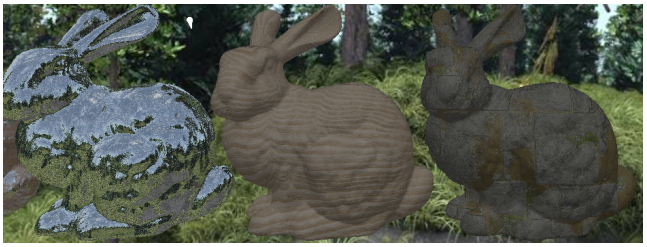

Figure 8: *Bunnies with steel material (left), wood material (middle), and stone material (right)*

With our work, we are able to intuitively model arguably convincing real world materials such as the marble and rusted iron materials seen in Figure 7, the stone and steel materials seen in Figure 8, and the frosted glass materials seen in Figure 9, by utilizing different texture maps and material parameters.

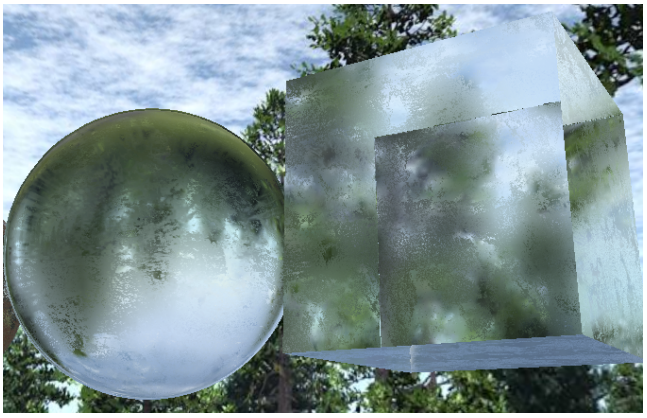

Figure 9: *Frosted glass material*

Our reflection and refraction implementations model many of the properties of real reflective and refractive objects, such as the inversion of light captured by a sphere and the increasing intensity of refraction when a surface is viewed from an angle, illustrated by Figure 10.

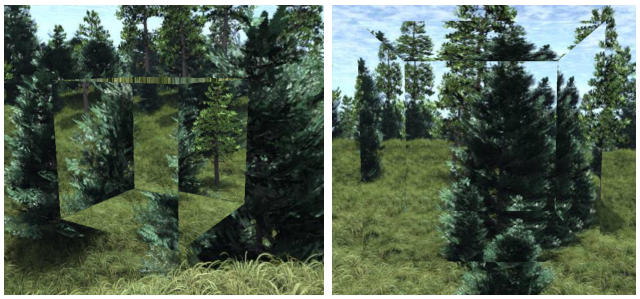

Figure 10: *Refraction cubes of IOR 1.15. Cube viewed from the side (left), and cube seen dead-on (right), resulting in minimal visible refraction.*

While some of these results are very appealing, there are some meshes for which we've failed to produce realistic materials. For example, due to total internal reflection and the likelihood that more than two refractions will occur for objects, we've failed to produce convincing results for refractive toruses, see Figure 11. A key goal of this project was to enable use of these materials on arbitrary meshes, but as it stands meshes of surface topology of genus greater than zero are likely to produce poor results for refractive materials.

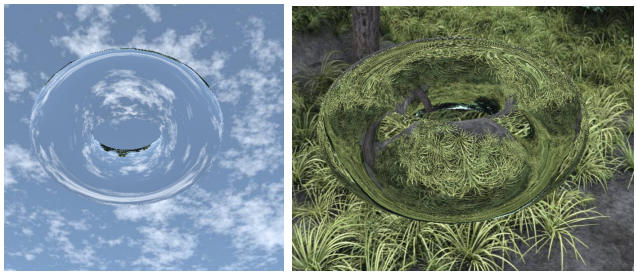

Figure 11: *Refractive Torus at IOR 1.2. Right shows very noticeable refraction artifacts near its inner hole.*

5.2 Ray-Tracing Results

We implemented reflection, refraction, and cube map environmental mapping in a ray tracing context in order to verify our results with a more physically realistic rendering method. As it was not a priority of our project, our ray tracing implementation is not able to model roughness, metalness, use normal maps, nor does it support arbitrary meshes as our rasterization pipeline can. Comparing our rasterized results to ray tracing results, we found that for the very simple case of a single sphere without roughness or normal maps, our implementations for reflection and refraction performed very similarly in terms of visual results, but the rasterized image was generated considerably more quickly. Each of the two ray traced images shown in Figure 12 required seven minutes and thirty-two seconds to render at an image size of 500 by 500 pixels on a single Intel Core I7 CPU without threading. The two rasterized images in Figure 13 were both rendered at a capped framerate speed of one sixtieth of a second (~17 milliseconds).

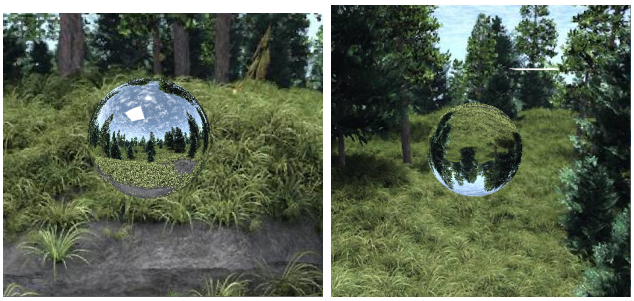

Figure 12: *Raytraced reflection (left) and raytraced refraction (right) for an object with IOR of 1.3. Each rendered in 7 minutes and 32 seconds.*

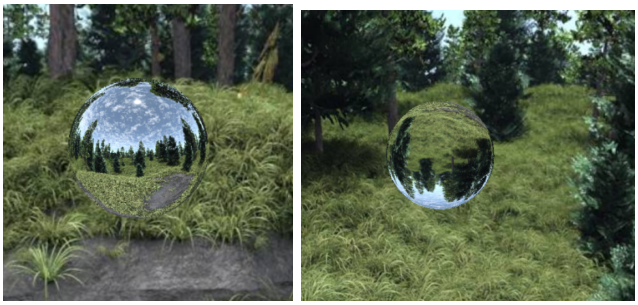

Figure 13: *rasterized reflection (left) and rasterized refraction (right) for an object with IOR of 1.3. Each rendered in 17 milliseconds.*

6. ACKNOWLEDGMENTS

Our thanks to ACM SIGCHI for allowing us to modify templates they had developed. Our thanks to Joey de Vries for composing the wonderful OpenGL reference that is learnopengl.com. Our thanks to Kathleen Ruiz, Eric Ameres, W. Randolph Franklin for overseeing Hongyang's graduate project. Our thanks to Barbara Cutler, for all her knowledge and support. Our thanks to FarmPeeps (http://www.farmpeeps.com/fp_skyboxes.html) for supplying our environmental cubemaps. Our thanks to FreePBR (https://freepbr.com) for supplying our texture sets.

7. CONTRIBUTIONS

7.1 Hongyang Lin

Hongyang set up the development environment of this project, including light sources, skybox, model generating / loading. During this project, Hongyang wrote GLSL code to implement the environmental map for reflection, the reflectance equation and BRDF, and combining reflection with refraction. He also maintained the engine to support new features and UI. He has spent approximately fifty hours on the project.

7.2 Jay Idema

Jay implemented the OpenGL and GLSL shader code for refraction, the mipmapping and glossiness effect for roughness, and the ray-tracing environment for ray-traced reflection, refraction, and ray-traced cube mapping, and took some of the images used in our paper. His work on this project, outside of helping compose this paper and this project's presentation slides, amount to over twenty hours.

8. REFERENCES

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

Cook, R. L., & Torrance, K. E. (1982). A reflectance model for computer graphics. *ACM Transactions on Graphics* (ToG), 1(1), 7-24

Greene, N. (1986). Environment mapping and other applications of world projections. *IEEE computer graphics and Applications*, 6(11), 21-29.

Karis, B. (2013). Real shading in unreal engine 4. Proc. Physically Based Shading Theory Practice, 4, 3.

Karis, B. (2013). Specular BRDF Reference. http://graphicrants.blogspot.com/2013/08/specular-brdf-reference.html.

Miller, G. S., & Hoffman, C. D. (1984). Illumination and reflection maps. *Course Notes for Advanced Computer Graphics Animation*, SIGGRAPH 84.Ngan, A., Durand, F., & Matusik, W. (2004). Experimental validation of analytical BRDF models. *ACM SIGGRAPH 2004 Sketches on - SIGGRAPH '04*. https://doi.org/10.1145/1186223.1186336

Schlick, C. (1994). An Inexpensive BRDF Model for Physically-based Rendering. Computer Graphics Forum, 13(3), 233–246. https://doi.org/10.1111/1467-8659.1330233

Smith, B. (1967). Geometrical shadowing of a random rough surface. IEEE Transactions on Antennas and Propagation, 15(5), 668–671. https://doi.org/10.1109/tap.1967.1138991

Walter, B., Marschner, S. R., Li, H., & Torrance, K. E. (2007). Microfacet Models for Refraction through Rough Surfaces. *Rendering techniques*, 2007, 18th.

Wyman, C. (2005). An approximate image-space approach for interactive refraction. ACM transactions on *graphics (TOG)*, 24(3), 1050-1053.