Ray Tracing Renderer Engine

Jia-Liu Wang, Kayla Carr, Hsiao-Lun Wang, Tuo Piao
CSCI 580: 3-D Graphics and Rendering

Abstract

This paper present our work of final project for CSCI580. For the final project, we build a ray trace engine without any external library. Our ray trace engine can show reflection, refraction effect, and shadow. To make the scene more nature, we also do soft-shadow, area light through Monte-Carlo sampling. Normal mapping is also supported in our engine.

I. Introduction

For our project, we explored an alternative method of rendering to the scan-line and LEE rasterization used in our homework projects and implemented a ray tracing renderer. Using our ray tracing renderer, we were able to produce global illumination effects such as reflection, refraction, and shadows, as well as area lighting, soft shadows, and normal mapping. The API we built for our renderer allows the user to easily make use of our engine to produce hyper realistic scenes by inputting their own scenes, lighting, camera, and materials.

Ray tracing is a technique for generating images by tracing the path of light through pixels and simulating the effects with virtual objects. It has a high degree of visual realism due to its physical nature, simulating the way light reacts with real objects and material properties. However, due to its great computational cost it is poorly suited for real-time applications such as video games. The algorithm was invented by Arthur Appel in 1968 [1]. In 1979, an important breakthrough on recursive ray tracing was achieved by Turner Whitted. Features such as basic reflection, refraction and shadow could be implemented with Whitted's algorithm [2]. Unlike rasterizing rendering, techniques such as shadow, reflection, and refraction (in other words, global lighting features) could be directly implemented in ray trace rendering. However, Whitted's recursive ray tracing algorithm does not have a good simulation on soft shadow and recursive diffuse reflection. Using the Monte Carlo sampling method along with an area light source would be one way to simulate soft shadow. Monte Carlo sampling would also be an acceptable solution for diffuse reflection. In addition, normal mapping based on tangent and bi-tangent vector reconstructing is also achievable.

Our engine is built off of our homework projects, keeping the Microsoft Foundation Classes (MFC) framework and display functions, but rewriting all rendering functions to use ray-tracing algorithms. We created our own geometry classes to represent objects programmatically, rather than through triangles and uv coordinates from a file. Lights, geometry, materials, textures, cameras, vectors, colors and scenes are all represented by objects. We did not use any outside libraries. Our code is written in C++ using

object-orientation and inheritance.

II. Project Structure

Our project is open source and can be downloaded for other feature extensions [6]. The main ray tracing process is built in GzRender class. GzRender emit ray from the camera to the scene. Gather the color value of each pixel with refractive, reflective, diffuse part. GzMaterial is the material class that stores the refraction index(n), refraction(f), reflection(r) and diffuse coefficient (Kd,Ks), i.e. Kd, Ks. GzGeometry is the class that construct objects in the scene. We have built infinite plane, sphere, ellipsoid, rectangle in our class. GzVector and GzColor are constructed for arithmetic operation of vector, and color. GzLight class supports three kinds of light, direction light, point light and area light. Point light is the light that has an origin and direction. Direction light has no origin. Instead, any place that is visible to the direction light have same light direction. The area light is sample through Monte Carlo method. GzTexture supports normal mapping, checkered board, or loading external texture.

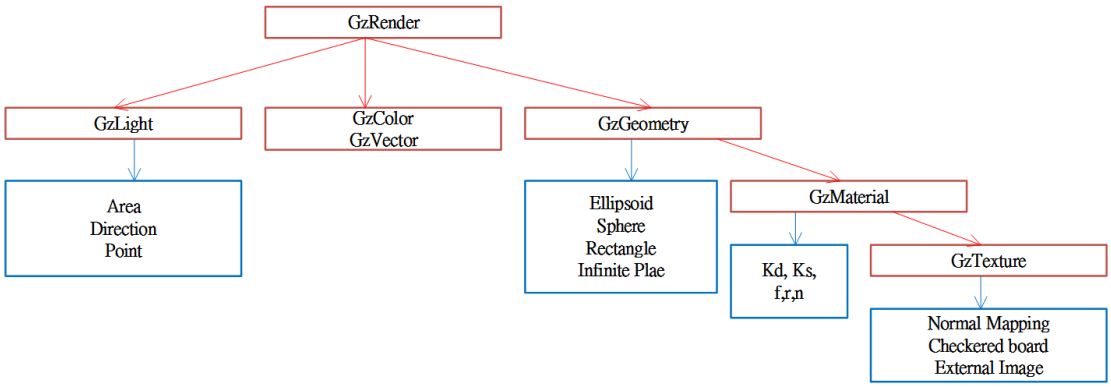


Fig. 1 Project structure.

III. Implementation Details

The renderer receives a scene consisting of one or more geometries each with a material attribute, a camera, and one or more light sources. The light sources can be point, directional, or area light sources. Point and directional lights are represented as vectors for the position or direction, and area light sources are represented as rectangular areas. Geometries can currently be spheres, infinite planes, rectangles, or ellipsoids. Materials store information about the diffuse color, specular index, refractive index, texture, normal map, reflectiveness and transparency of the object. This is needed to create an accurate physical representation.

For each pixel of the display, a ray represented as a vector is cast from the camera into the scene. When a ray hits a geometry, information about the point of intersection is recorded and returned as an IntersectResult. This includes the point of intersection, the

geometry that was hit, its distance from the camera, its normal, and its texture coordinates. The logic for intersection is calculated inside the geometries. This is so each geometry can have a specific collision, texture mapping, and normal calculation. For example, the normal of a sphere is the normalized vector of the sphere's center to the point of intersection and its texture coordinates are calculated with spherical mapping algorithm. The tangent of a sphere is calculated from the perpendicular to its normal along the direction of the u coordinate. If normal mapping is included in the material property, the tangent, bi-tangent, and normal are used to transform the normal map into tangent space and the newly calculated normal is sent instead.

Using the IntersectResult, shading is performed in the renderer with blinn-phong with additional recursive logic for global illumination effects. For refraction, an additional refraction vector is calculated and cast into the scene. The color information is shaded and stored. For reflection, the ray is reflected off of the hit geometry along its normal and cast into the scene again, stored as color information from the shader. If no further intersections occur, there is no need to keep recursively casting rays. For each light in the scene, blinn-phong shading is performed. For area lights, further sampling must be done along the area of the light source. Shadow rays are calculated here to detect if an object lies in the path between the light and the intersection point. For semi-transparent objects, the shadow is diluted from the light coefficient. Finally, the final color for the pixel is calculated from the diffuse, reflect, and refract colors. For intensive scenes with lots of reflection and refraction, the renderer can take an extensive amount of time to draw the scene due to recursion in the shading. In the future, we can limit the amount of recursive calls performed. This will sacrifice accuracy but improve speed.

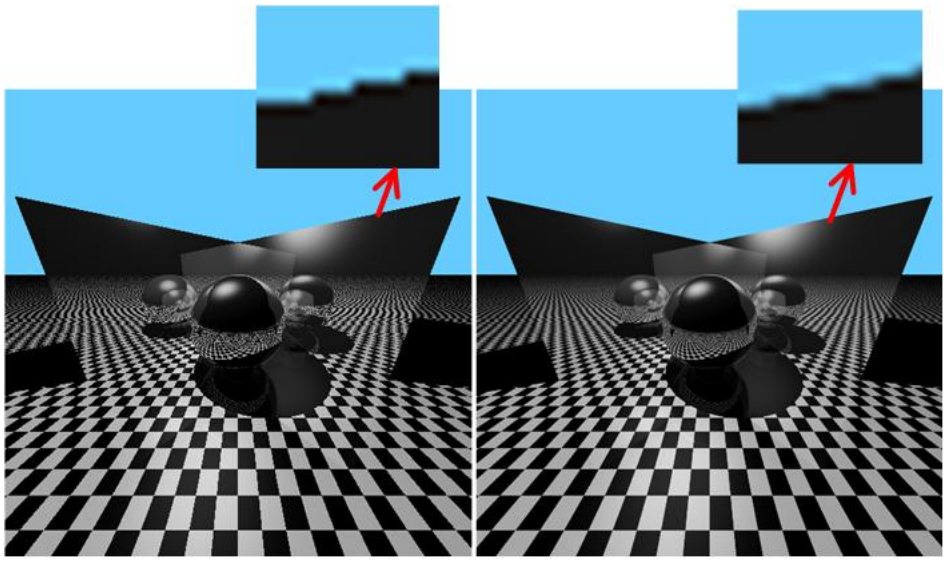
Fig. 2 shows the abstract of ray tracing algorithm. The inputs of RayTrace are the objects in the scene (Objects), the ray to trace in this iteration, the number of times to trace recursively. The return value of RayTrace is the color that the ray hits. IR means IntersectResult. The IntersecResult class includes the distance to hit the targeting object O, the normal information of the hitting point. And the reflective(IR.r), refractive(IR.f) constant for refractive and reflective ray. IR.d means the intersect distance. The FOREACH loop from line 5 to line 8 is to find the nearest points the ray intersect with. GenerateRay function in line 9 is to generate a new ray from for reflection and refraction with the normal information from IntersectResult (IR). The return color is the combination of the color from diffusion, and color from refraction, reflection times the coefficient from IR.

```
Algorithm RayTrace
1. Input:   Objects, Ray, Times
2. Output: Color
3. IR.d = infinite
4. IF(Times>0)
5.         FOREACH O in Objects
6.                 IF IR.d < Intersect(Ray, O).d
7.                  IR = Intersect(Ray,O)
8.         END
9.         {ReflectRay, RefratRay}=GenerateRay(IR,Ray)
10.        Color= IR.Kd* (1-IR.r-IR.f)*diffuseColor
11.              +IR.r*RayTrace( Objects, ReflectRay,Times-1)
12.                +IR.f*RayTrace( Objects, RefractRay,Times-1)
13. RETURN Color
```

Fig. 2 Algorithm of RayTrace. IR is Intersect Result. IR.d means the distance between the origin of ray and the intersection point. DiffuseColor in line 10 means the the diffuse color of hitting point.

IV. Result

A ray-tracing renderer was built so that all the key features are implemented. These features include reflection, refraction, shadows, area lighting with soft shadows, texture mapping, normal mapping, and anti-aliasing. Several output rendered scenes were generated, and each of them represents a particular key feature mentioned above. In a ray-tracing scene, a ray is usually shot from the camera to the surface of objects, so shadow and reflection features are naturally built in the rendering process. It is straightforward to understand that if an object is in the path from a light source to the intersection point, then that particular point will be in shadow. Another simple feature reflection is achieved by calculating the reflection ray, which is very similar to the Phong lighting model taught in the class. Then the ray is eventually shaded with color at its intersection point. These two features are demonstrated by Fig. 3. There are two rectangles mirrors behind the sphere. The shadow of the sphere and mirrors are shown. At the same time, the mirror reflections images of the sphere are displayed. The checkboard expands perfectly into the mirror is a coincidence of how the two mirrors are placed and the angle between them. The reflection was achieved in a recursive way by having light rays bounce from surface to surface, as in Whitted's model [2]. Since the material with perfect reflection ratio does not exist, the reflected ray becomes dark as the recursion continues.



(a)without anti-aliasing (b)anti-aliasing

Fig. 3 Two Mirrors and a Sphere

Another feature demonstrated by Fig. 3 is the anti-aliasing effect. The image on the left is produced when multi-sampling anti-aliasing feature is turned off. The sphere, infinite checkerboard plane, and rectangular mirrors are all part of the GzGeometry class. The program is designed as object-oriented as possible.

The next successfully implemented feature is refraction as shown in Fig. 4. A refraction light ray is calculated according to the physical refractive index of that object.

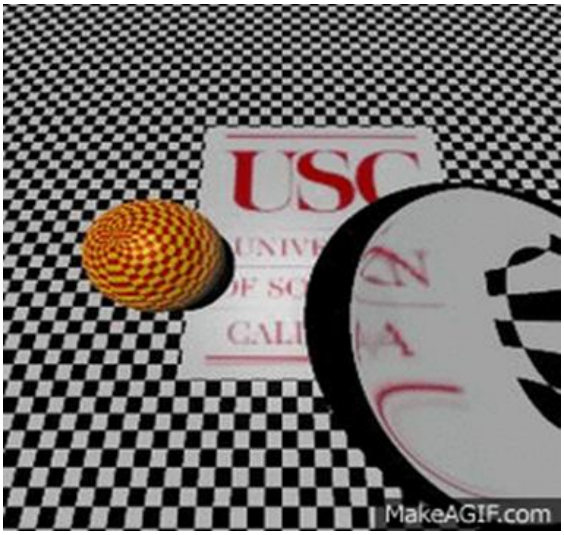
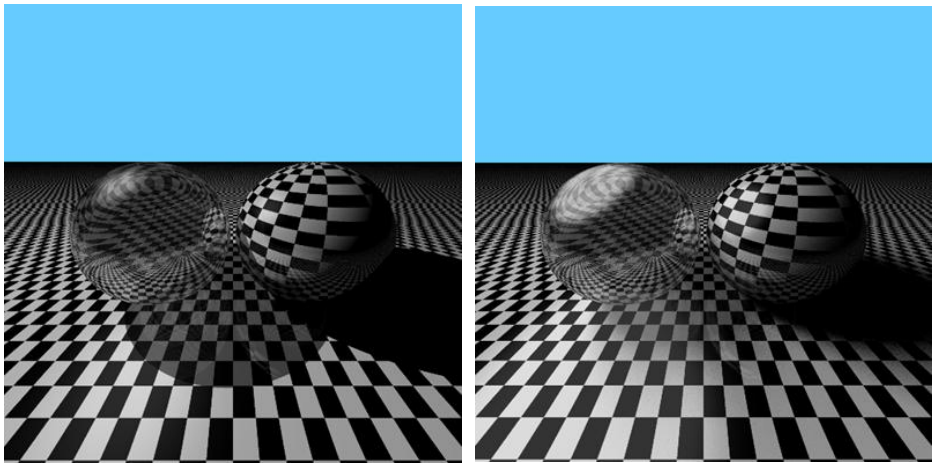


Fig. 4 Refraction produced by a magnifying glass.

The image in Fig. 4 shows a magnifying-glass effect which was achieved with refraction feature. The magnifying glass is essentially an ellipsoid with no diffuse color and reflection. Its refractive index is close to the real glass material, and these properties make the ellipsoid transparent. A video with magnifying glass moving around can be watched from [6].

Area light is another key feature implemented by the ray-tracing engine as shown in Fig. 5.

Area light is the third kind of light source, the first and second being directional and point lights, respectively. The area light has its own geometry, and with the aid of area light, soft shadow can be displayed naturally. The area light is done with Monte Carlo sampling in the region of area light source. The image below on the right shows the soften shadow for the spheres.



(a)without area light **(b)with area light**

Fig. 5 Different kinds of light.

In Fig. 5, the left sphere in both figures is partially transparent. The shadow for a partially transparent object will be partially black. This feature was implemented with another recursive function, and it will be valid if there are multiple partially transparent objects in the path. The color of the shadow behind is determined by accumulated transparency.

Lastly, normal mapping feature on the geometries was achieved. Normal mapping on a sphere is generally more challenging than a flat surface object such as a triangle. However, the basic principle stays the same. Reconstruction of tangent and bi-tangent vectors along the surface is required. Then the normal vector at the intersection point will be altered accordingly by calculating it based off the normal map and tangent space. The next pair of figures demonstrated the normal mapping results acquired.

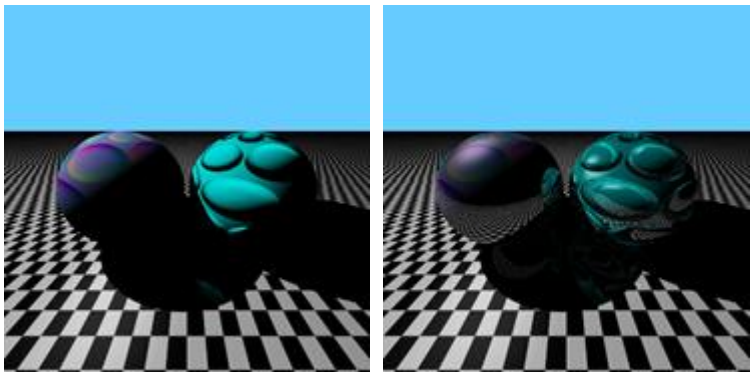


Fig. 6 The sphere on the left of each figure does not have altered normal vector, but the right sphere has.

V. Conclusion

Using the algorithms and techniques put forth by Appel [1] and Whitted [2], we were able to

build a ray trace renderer with global illumination effects of reflection, refraction, and shadows. To improve this engine, Monte Carlo sampling technique and area light geometry was added to reproduce soft shadowing effects. Finally, basic normal mapping with tangents and normal maps were added to create more interesting and dynamic geometries. The net result is a renderer that can produce realistic and convincing global illumination effects.

VI. Reference

[1] Arthur Appel. 1968. Some techniques for shading machine renderings of solids. In *Proceedings of the April 30--May 2, 1968, spring joint computer conference* (AFIPS '68 (Spring)). ACM, New York, NY, USA, 37-45.

[2] Turner Whitted. 1979. An improved illumination model for shaded display. In *Proceedings of the 6th annual conference on Computer graphics and interactive techniques* (SIGGRAPH '79). ACM, New York, NY, USA, 14-.

[3] Robert L. Cook, Thomas Porter, and Loren Carpenter. 1984. Distributed ray tracing. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques* (SIGGRAPH '84), Hank Christiansen (Ed.). ACM, New York, NY, USA, 137-145.

[4] Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. 1988. A ray tracing solution for diffuse interreflection. SIGGRAPH Comput. Graph. 22, 4 (June 1988), 85-92.

[5] Venkat Krishnamurthy and Marc Levoy. 1996. Fitting smooth surfaces to dense polygon meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*(SIGGRAPH '96). ACM, New York, NY, USA, 313-324.

[6] https://github.com/hsiaoluw/RayTracingEngine

[7] http://makeagif.com/lM_F3w