Ray Tracing Algorithm for Object Visualization

MARIANA ÁVALOS ARCE, Universidad Panamericana

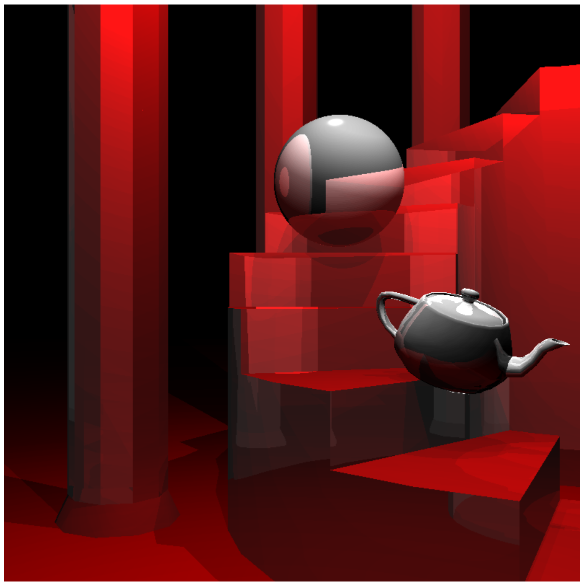

Fig. 1. Scene render using ray tracing

The following document focuses primarily on an object visualization technique known as **Ray Tracing**, including secondary theorems and concepts that were used in order to make computer generated images capable of describing phenomena such as lightning, reflection and refraction. Elements that took part in making the rendering complete are also included: material interpretation (shading models), intersection analysis, among others.

Additional Key Words and Phrases: ray tracing, shading, refraction, reflection

ACM Reference format:
Mariana Ávalos Arce. 2019. Ray Tracing Algorithm for Object Visualization. *ACM Trans. Graph.* 1, 1, Article 1 (June 2019), 3 pages.
https://doi.org/http://dx.doi.org/10.1145/8888888.7777777

1 INTRODUCTION

Ray Tracing is an algorithm for representing a three dimensional scene in a two dimensional image. The algorithm works by casting rays or, mathematically speaking, lines. The ray caster casts as many rays/lines as there are pixels in the final image. It is a highly precise technique, with time performance being its only disadvantage.

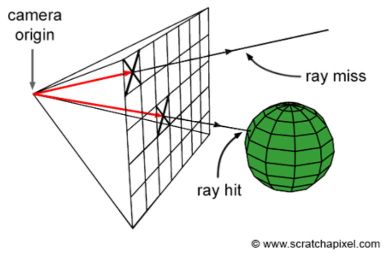

Fig. 2. Ray tracing algorithm. Image by Scratch a pixel, 2019. [(http://www.scratchapixel.com/lessons/3d-basic-rendering/)]

2 RAYS

A ray, as mentioned before, is represented by a mathematical line:

$$\vec{r} = \vec{o} + \vec{d} \tag{1}$$

With o being the line's origin as a position vector and d being the line's director vector. The origin coordinates of the vector are given by the pixel position inside the image, while the line direction is given by substracting the camera position to the pixel position.

Therefore, if we have an image with dimensions m x n, we cast m x n rays.

3 INTERSECTIONS

Once the rays are defined, the algorithm needs to determine if the ray hits an object inside the scene, which mathematically means finding the solution to the equation that involves the line and each of the objects. In this ray tracer, an object can be either a sphere or a polygon.

In case of a sphere-line intersection:

$$t = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \tag{2}$$

Where

$$a = \vec{d} \cdot \vec{d} \tag{3}$$

$$b = 2\vec{d} \cdot (\vec{o} - \vec{center}) \tag{4}$$

$$c = (\vec{o} - \vec{center}) \cdot (\vec{o} - \vec{center}) - radius^2 \tag{5}$$

And then, when a t value is negative, it means there is no intersection. This makes the algorithm return the background color. Otherwise, it returns the object's color.

If the algorithm encounters a polygon, it needs to read the OBJ file to interpret its shape. This is done by reading its vertices first, storing them in Triangle objects and then finding out whether the ray hits each triangle by following a similar procedure, but with line-plane intersection equation and then applying **barycentric coordinates** to determine if the ray hits that section of the polygon.

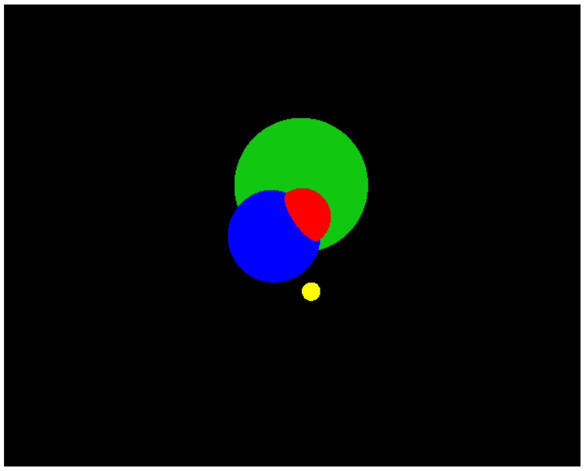

Fig. 3. Basic intersection analysis

4 LIGHTS

This algorithm includes only two lights: **Directional lights** and **Point lights**. A light in ray tracing is simply information about the *source* of illumination. A **directional light** is a vector with no position, describing the direction of such light. A **point light** consists of a vector, which represents its direction, and a position vector. This information, together with the normal of each intersection point, is crucial for **shading**.

5 SHADING

When it comes to realistic rendering, returning simply the object's color when an intersection is found results unpleasant. After the intersection proves to be existent, what we do with the color in order to make it look real is called shading.

In this algorithm, **Blinn-Phong Shading Model** and **Lambertian Shading Model** were implemented to shade an object's color. The Lambertian model was coded by following this next concept:

$$Projection = \vec{N} \cdot \vec{L} \tag{6}$$

Where $\vec{N}$ is the triangle normal and $\vec{L}$ is the light's inverted director vector.

For Blinn-Phong, the following was used:

$$\vec{h} = \vec{v} + \vec{L} \tag{7}$$

Where $\vec{v}$ represents view direction, calculated by $\vec{camera} - \vec{hit}$. It is important to note that $\vec{h}$ needs to be normalized. After that, $\vec{h} \cdot \vec{N}$ is computed to generate the specular component, which will be added to the diffuse or lambertian component after it is multiplied by the object's color. [Marschner and Shirley 2016]

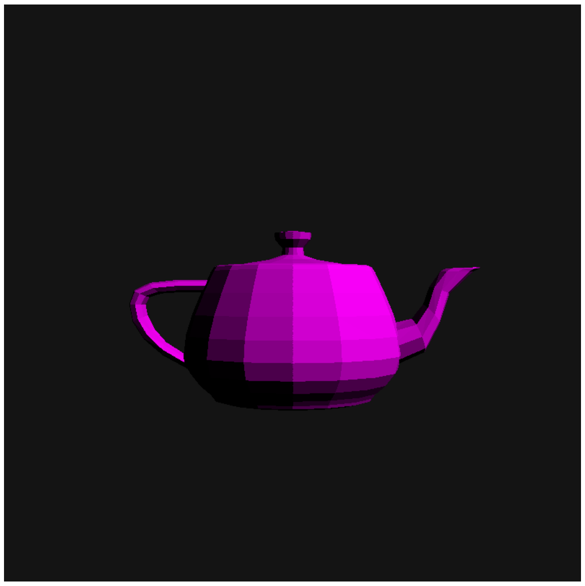

Fig. 4. Lambertian flat shading

6 REFLECTION AND REFRACTION

This two features were very challenging, but interesting. They both are material characteristics that cannot create a material by themselves. Therefore, they now are added to either the Lambert or Blinn-Phong materials.

With **reflection**, what is done is bouncing the incoming ray, producing a secondary ray called the reflective ray. This ray has a specific direction, given by:

$$\vec{r} = 2(\vec{N} \cdot \vec{v}) * \vec{N} - \vec{v} \tag{8}$$

And once we have this, we find the intersections with the secondary ray just as with the primary ray. As with **refraction**, the algorithm gets longer. But essentially, we need another secondary ray, which is called the refractive ray. Its direction is given by:

$$t = \frac{n1}{n2}(\vec{r} + (\vec{N} \cdot \vec{r}) * \vec{N}) - \sqrt{1 - \frac{n1^2}{n2^2}(1 - (\vec{N} \cdot \vec{r})^2) * \vec{N}} \tag{9}$$

After we have both of these directions, the only thing left to create either of those secondary rays is its origin, which is the hit position. It is important to add to the resulting rays an epsilon to avoid **acne** or noise in the rendered image.[University 2017]

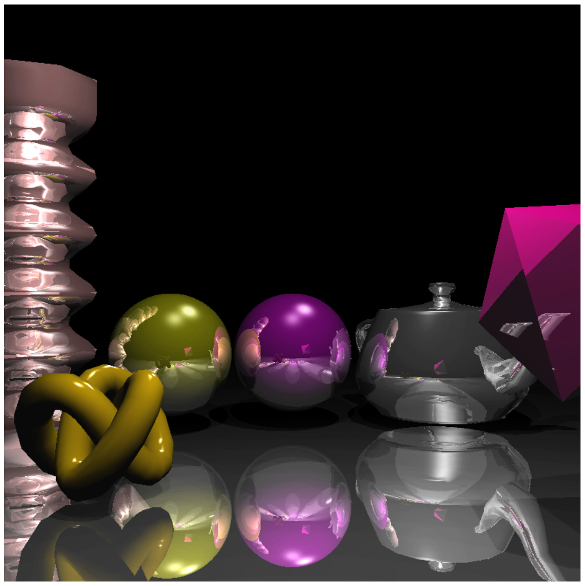

Fig. 5. Scene with reflection and refraction.

7 SMOOTH SHADING

The ray tracer in question includes a feature called **smooth shading**, which is an implementation of the Phong Interpolation Method. This algorithm affects the polygon visualization, for it makes the object look softer and not faceted. This is done without changing the OBJ file, and instead it defines a **normal vector** for each intersection point. This is accomplished by getting each triangle's vertex normals. A vertex normal is the linear combination of all the face normals that use that vertex.

Once we have the vertex normals, we compute the areal proportion of that point of intersection inside the triangle, so that we get the **smoothed normal** of that hit point.

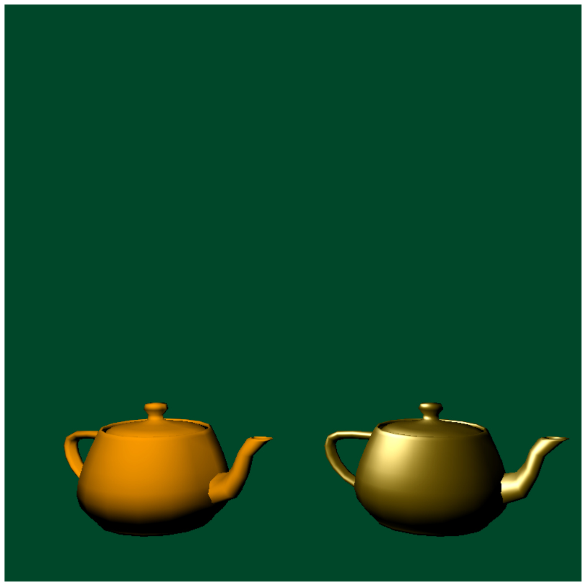

Fig. 6. Phong Interpolation Method.

8 CONCLUSIONS AND FUTURE WORK

The main conclusion from this project is that Mathematics rule not only the functioning of our world, but also the way we see it. Everything was just a matter of dot and cross products, and the outcome was a representation of space that looked almost real. More techincally speaking, a conclusion would be how valuable are the little optimizations such as variable-stored information and object oriented programming. Both of these were also essential for the understanding of the models described previously.

I would like to continue this project with the implementation of Depth of Field and surface scattering, especially because of the shadowing issue, so that refractive objects and their shadows look more realistic.

ACKNOWLEDGMENTS

I would like to thank our professor Jafet Rodríguez for all the lectures given and the help provided while developing this project. His classes are always very memorable. I would also like to thank campus professor Raquel Ruiz de Eguino for helping me understand the way in which a transformation matrix works. Rotation and scaling were crucial for testing the algorithm.

REFERENCES

Steve Marschner and Peter Shirley. 2016. *Fundamentals of Computer Graphics*. CRC Press.

Cornell University. 2017. Ray Tracing Reections and Refractions. http://www.cs.otago.ac.nz/cosc342/2017-notes/342-2017lect18.pdf. (2017).