International Journal of Innovation Scientific Research and Review
Vol. 06, Issue, 05, pp.6313-6316, May 2024
Available online at http://www.journalijisr.com
SJIF Impact Factor 2023: 6.599

Research Article

ISSN: 2582-6131

REAL TIME RAY TRACING WITH GEOMETRIC PRIMITIVES USING OPENGL

*** Charisma Tubagus Setyobudhi**

Departemen Teknik Komputer Fakultas Teknik, Universitas Diponegoro, JL. Prof. H. Soedarto, SH Tembalang Semarang Indonesia.

Received 02th March 2024; Accepted 03th April 2024; Published online 25th May 2024

ABSTRACT

Computer graphics is a branch of computer science that is widely used in the real world. One application of computer graphics is in the field of real time rendering. Real time rendering allows a 3D scene to be rendered realistically. One of the real time rendering techniques that is currently most widely used is ray tracing. By using ray tracing the shadow and lighting effects will become more realistic. Ray tracing has a side effect, namely the calculation process is quite long. In this paper, we will discuss how the implementation of ray tracing techniques is used on several primitive geometries using OpenGL. We hope that by reading this paper readers can better understand the advantages and benefits of ray tracing techniques in the field of computer graphics.

Keywords: Surface Intersection, Quadratic Polynomial, OpenGL and GLM libraries, GL Wrapper, Scene Manager

INTRODUCTION

Computer graphics has two algorithm methods in the process of rendering 3D scenes into a 2D screen. The dominant algorithm used in the rendering process is rasterization. The rasterization process is a process in which a primitive geometry has its pixels colored clearly by a renderer. And after the coloring process is complete the pixels are displayed on layer[1]. Because the hardware does not know how the scene is formed, it is the programmer's responsibility to determine the shading or coloring of the primitive geometry. This is done by creating vertex shader and fragment shader programs.

On the other hand, the second rendering process uses ray tracing techniques. Ray tracing begins by shooting a ray beam from the camera towards the primitive geometry then tracing the direction of the beam towards pixels located in the geometry then reflecting towards the light source. This allows the 3D scene to be more realistic than the rasterization process even though this process takes longer.

THEORETICAL BACKGROUND

Introduction to Ray Tracing

Ray tracing is an algorithm in rendering whose method is to follow the journey of light rays emitted from the source. The applications of ray tracing are light map generation, visibility determination and collision detection. In this chapter we will discuss how to calculate the intersection between objects and light rays (intersection points).

Finding Root Point

To find the intersection point using the straight line method, we can use the following formula.

$$\mathbf{P}(t) = \mathbf{S} + t\mathbf{V}$$

For planar surfaces, the degree of polynomial for finding roots is 1, while for quadratic surfaces the degree of polynomial for finding roots is 2.

Quadratic Polynomial

For a quadratic polynomial equation that looks like the one below:

$$at^2 + bt + c = 0.$$

So the formula for finding roots is as follows:

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Surface Intersection

A straight line ray is defined using the equation below.

$$\mathbf{P}(t) = \mathbf{S} + t\mathbf{V},$$

Where S is the initial ray position, t is the time parameter and V is the direction vector of the ray.

Intersection Between Ray and Triangle

A triangular plane has three Points P_0, P_1 P_2. And having a Normal N equation is as follows:

$$\mathbf{N} = (\mathbf{P}_1 - \mathbf{P}_0) \times (\mathbf{P}_2 - \mathbf{P}_0)$$

Meanwhile, the vector L is defined as follows:

$$\mathbf{L} = \langle \mathbf{N}, -\mathbf{N} \cdot \mathbf{P}_0 \rangle$$

And the intersection between the ray and the triangular plane is at time t with the following equation.

***Corresponding Author: Charisma Tubagus Setyobudhi,**
Departemen Teknik Komputer Fakultas Teknik, Universitas Diponegoro, JL. Prof. H. Soedarto, SH Tembalang Semarang Indonesia.

$$t = -\frac{\mathbf{L}\cdot\mathbf{S}}{\mathbf{L}\cdot\mathbf{V}}.$$

Intersection Between Ray and Box

A box is defined as follows:

$$\begin{array}{ll} x=0 & x=r_x \\ y=0 & y=r_y \\ z=0 & z=r_z, \end{array}$$

Where r_x, r_y and r_z are the dimensions of the box. At most 3 of the 6 equations above must be considered in the intersection with the ray. We can determine which edges will intersect by estimating the vector V of the ray. If $V_x = 0$, then the ray will not intersect x=0 or x=r_x. If V_x> 0, then there is no need to test the intersection with x=r_x. If V_x< 0, then there is no need to test the intersection with x=0. The same principle applies to the y and z axes. For the intersection between ray and box, t is calculated

$$t = \frac{r_x - S_x}{V_x}.$$

And we also need to test whether the constraints below are met.

$$0 \le [\mathbf{P}(t)]_y \le r_y$$
$$0 \le [\mathbf{P}(t)]_z \le r_z.$$

Intersection Between Ray and Sphere

The sphere equation can be written as follows:

$$x^2 + y^2 + z^2 = r^2.$$

Substituting Ray's equation, we get the following equation:

$$(S_x + tV_x)^2 + (S_y + tV_y)^2 + (S_z + tV_z)^2 = r^2.$$

Then we expand the equation into the equation below:

$$(V_x^2 + V_y^2 + V_z^2)t^2 + 2(S_xV_x + S_yV_y + S_zV_z)t + S_x^2 + S_y^2 + S_z^2 - r^2 = 0$$

This is a quadratic equation with coefficients a, b, and c as follows:

$$a = V^2$$
$$b = 2(\mathbf{S}\cdot\mathbf{V})$$
$$c = S^2 - r^2$$

Then we use the abc formula to find the roots of the ray equation:

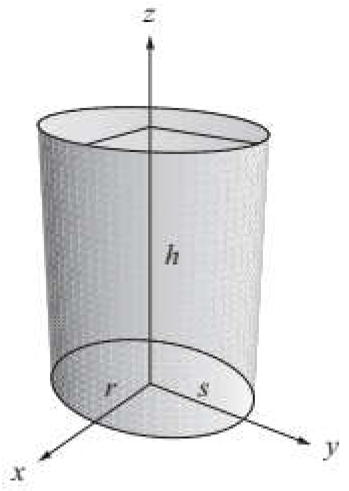
$$t = \frac{-b - \sqrt{D}}{2a}$$

Intersection Between Ray and Cylinder

The equation of a cylinder can be formulated as follows:

$$x^2 + m^2y^2 = r^2$$
$$0 \le z \le h,$$

Where m = r/s. R is the radius on the x axis and s is the radius on the y axis.



Substituting the original ray equation, we get the following equation:

$$(S_x + tV_x)^2 + m^2(S_y + tV_y)^2 = r^2.$$

Then we expand the equation and we get the equation

$$(V_x^2 + m^2V_y^2)t^2 + 2(S_xV_x + m^2S_yV_y)t + S_x^2 + m^2S_y^2 - r^2 = 0$$

Then we get components a, b, and c from above and we can also calculate the root of t using the abc formula.

RESEARCH METHODS

As a research method, we created this ray tracing implementation using the OpenGL and GLM libraries. As a first step, there are several definitions used in the implementation of ray tracing, namely:

1. RT Defines
   These RT Defines are defined as follows:

```
structrt_defines
{
        int sphere_size;
        int plane_size;
        int surface_size;
        int box_size;
        int torus_size;
        int ring_size;
        int light_point_size;
        int light_direct_size;
        int iterations;
        glm::vec3 ambient_color;
        glm::vec3 shadow_ambient;
};
```

2. RT Material
   This RT Material is defined as follows:

```
typedefstruct {
        glm::vec3 color; float __p1;

        glm::vec3 absorb;
        float diffuse;
        float reflect;
        float refract;
        int specular;
        float kd;

        float ks;
        float __padding[3];
} rt_material;
```

3. RT Sphere
   This RT Sphere is defined as follows:

```
typedefstruct {
        rt_material material;
        glm::vec4 obj; // pos + radius
        glm::quat quat_rotation = glm::quat(1, 0, 0, 0);
        int textureNum;
        bool hollow;
        float __padding[2];
} rt_sphere;
```

4. RT Plane
   This RT Plane is defined as follows:

```
typedefstruct {
        rt_material material;
        glm::vec3 pos; float __p1;
        glm::vec3 normal; float __p2;
} rt_plane;
```

5. RT Box
   This RT Box is defined as follows:

```
typedefstruct {
        rt_material mat;
        glm::quat quat_rotation = glm::quat(1, 0, 0, 0);
        glm::vec3 pos; float __p1;
        glm::vec3 form;
        int textureNum;
} rt_box;
```

6. RT Torus
   This RT Torus is defined as follows:

```
typedefstruct {
        rt_material mat;
        glm::quat quat_rotation = glm::quat(1, 0, 0, 0);
        glm::vec3 pos; float __p1;
        glm::vec2 form; float __p2[2];
} rt_torus;
```

7. RT Ring
   This RT Ring is defined as follows:

```
typedefstruct {
        rt_material mat;
        glm::quat quat_rotation = glm::quat(1, 0, 0, 0);
        glm::vec3 pos; int textureNum;
        float r1, r2;
        float __p2[2];
} rt_ring;
```

8. RT Surface
   This RT Surface Is defined as follows:

```
typedefstruct {
        rt_material mat;
        glm::quat quat_rotation = glm::quat(1, 0, 0, 0);
        float xMin = -FLT_MAX;
        float yMin = -FLT_MAX;
        float zMin = -FLT_MAX;
        float __p0;
        float xMax = FLT_MAX;
        float yMax = FLT_MAX;
        float zMax = FLT_MAX;
        float __p1;
        glm::vec3 pos;
        float a; // x2
        float b; // y2
        float c; // z2
        float d; // z
        float e; // y
        float f; // const

        float __padding[3];
} rt_surface;
```

9. RT Light Direct
   This RT Light Direct is defined as follows:

```
structrt_light_direct {
        glm::vec3 direction; float __p1;
        glm::vec3 color;

        float intensity;
};
```

10. RT Light Point
    This RT Light Point is defined as follows:

```
structrt_light_point {
        glm::vec4 pos; //pos + radius
        glm::vec3 color;
        float intensity;

        float linear_k;
        float quadratic_k;
        float __padding[2];
};
```

11. RT Scene
    This RT Scene is defined as follows:

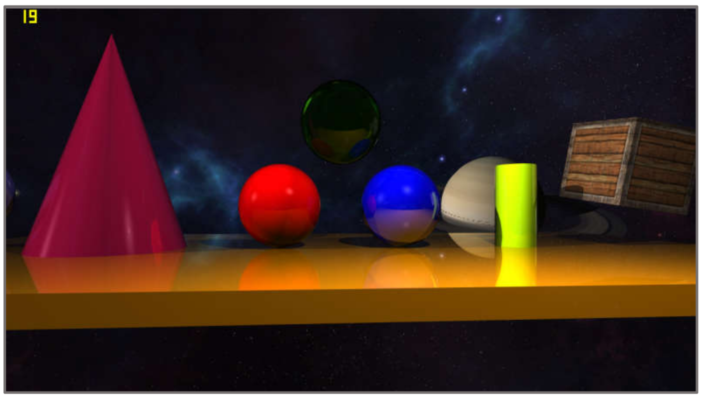
```
typedefstruct {
        glm::quat quat_camera_rotation;
        glm::vec3 camera_pos; float __p1;

        glm::vec3 bg_color;
        int canvas_width;

        int canvas_height;
        int reflect_depth;
        float __padding[2];
} rt_scene;
```

Apart from the above definitions, there are main classes in the implementation of this primitive geometry ray tracing. These classes are interrelated because the implementation of ray tracing is quite complex. There are two main classes created in this ray tracing implementation. The classes that need to be created are as follows:

1. GL Wrapper
   GL Wrapper functions to interface OpenGL with Scene Manager. This GL Wrapper has the task of rendering 3D scene images into layers. The main functions required by GL Wrapper are:
   a. initWindow()
   b. initShaders()
   c. setSkybox()
   d. draw()
   e. load_cubemap()
   f. load_texture()
   g. init_buffer()
   h. update_buffer()
2. Scene Manager
   Meanwhile, the Scene Manager class is tasked with organizing the objects that will be drawn in the 3D scene. This Scene Manager has several important functions, namely:
   a. Init()
   b. Update()
   c. Create_material()
   d. Create_sphere()
   e. Create_plane()
   f. Create_box()
   g. Create_torus()
   h. Create_ring()
   i. Create_LightPoint()
   j. Create_lightDirect()
   k. Update_scene()

RESULT AND DISCUSSION

From the implementation of the ray tracing geometric primitive above, the rendering results are obtained as follows:



The results of the rendering above produce an FPS of around 19-20 FPS. By using this ray tracing implementation, the scene rendering results obtained will be more realistic and similar to the results in the real world. There are several objects that can be rendered by ray tracing in the image above, namely: plane, sphere, cone, slider and box. Apart from that, there is a skybox rendered in the background. The reflections produced by the ray tracing implementation program above are quite realistic and similar to reflections in the real world. This is caused by the ray tracing algorithm program used in the implementation above.

CONCLUSIONS

From the results of the research that has been carried out, it can be concluded that the implementation of ray tracing can produce incredibly realistic scenes. However, these very realistic results can be produced using rather lengthy calculations. As a result of this rather long calculation, the resulting FPS is quite small compared to rendering results without ray tracing.

REFERENCES

[1]. Phillip, Slullasek., Peter, Shirley. Bill, Mark., Introduction to Real Time Ray Tracing. SIGGRAPH 2005
[2]. Hae-Won, Son., Jung-Woong, Yoo. A Fast 3D Ray Tracing Method for Wave Propagation Prediction Using a Reflection Tube Tree. IEEE.2017
[3]. Yong-Keun, Yoon., Myoung-Won, Jung., Jongho, Kim. Intelligent Ray Tracing for the Propagation Prediction. IEEE. 2012
[4] Wenbo, Li., Yunhua, Cao., Donghui, Meng. Space Target Scattering Characteristic Imaging in the Visible range Based on Ray Tracing Algorithm. IEEE. 2018
[5] Shuo, Hu., Li-Xin, Gao., Zhong-Liu, Yu. A Fast Ray Tracing Algorithm for Rugged Terrain. IEEE. 2017
[6] Carsten, Benthin., Ingo, Wald,. Michael, Scherbaum., Heiko, Friedrich. Ray Tracing on the Cell Processor. IEEE. 2006

****