Assignment 3 : Direct Lighting and Texturing

STUDENT NAME CHENGHUA YANG STUDENT NO. 2019533089 EMAIL YANGCHH@SHANGHAITECH.EDU

1 INTRODUCTION

This time's homework realise a simple one-time-raytracing renderer.
The raytracing only generate ray direct from the camera and the 1st ray hit point, which is aim to rebuild the projection based render with phong lighting model.
I finished the essential part including the pihole-camera, the intersection algorithm, the phong lighting intergrator and the texture mapping.
Also I do the a simple optional part, the anti-aliasing with uniform multiple sample.

2 CONTENT

2.1 The pinhole camera to generate rays.

2.2 The intersection algorithm

2.3 The texture mapping

2.4 The integrator

3 IMPLEMENTATION DETAILS

3.1 The pinhole camera to generate rays.

The pinhole camera shoot ray from a original point.

Before we generate rays, firstly we should let the camera look at the right direction.
Given a lookat point and reference up direction, we can figure out the forward, up and right direction of the camera, thus to present its look-at direction.

To shoot the ray ,we first fetch the number of the pixel (dx,dy), then convert it to the world spacepixel postion on the picture(or the screen). see Fig.1.
The essential part of this is to fix the fov and focal length to simplify the formula, then use the tan to find out the vertical/horizontal distance between center and the pixel. see Fig.2.

3.2 The intersection algorithm

This time's raytracing homework nearly skips the intersection part which should be a mainly difficult part of raytracing, and replace it with the analystic geometry intersection.

To interact with all the algorithm, the scene function intersect() simply iterate sequetially through every geometry in the scene.
Then, with each geometry(sphere or parallegram), there is a solution of polynomial equation as $Point_{onRay} = Point_{onGeometry}$. This will be a quadratic equation in the sphere case. See Fig.3.

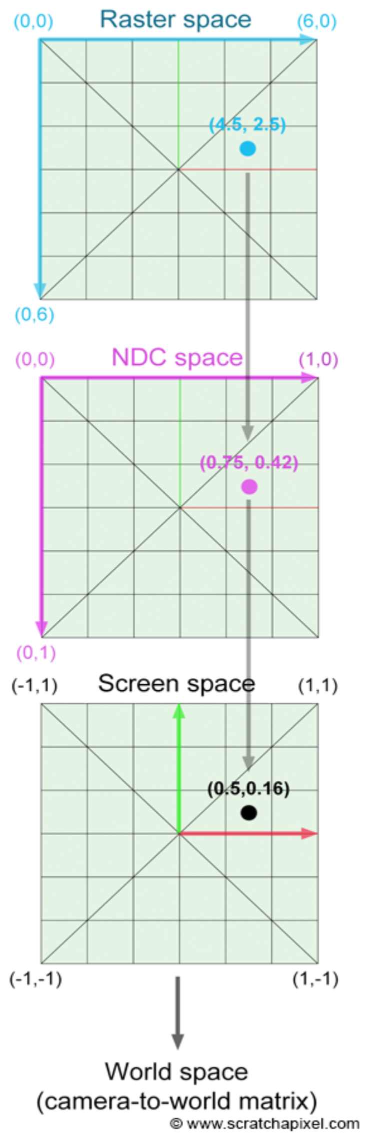


Fig. 1. The whole transform process

3.3 The texture mapping

The texure mapping process including turn a world position on geomrtry into a uv position
and validate the uv, sample it from the texture.

To map a position into uv, there will be several map to use. I choose to use the angle term directly, the phi angle represents the u and the theta angle represents the v. see Fig.4.

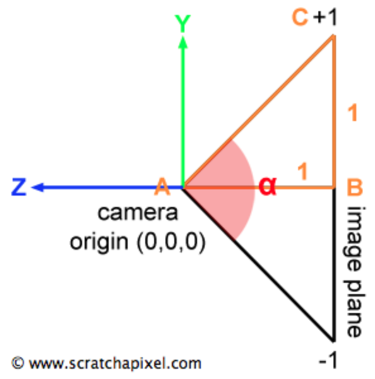
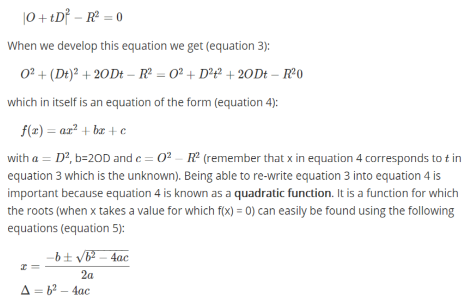


Fig. 2. from screen to world using alpha angle



$|O + tD|^2 - R^2 = 0$

When we develop this equation we get (equation 3):

$O^2 + (Dt)^2 + 2ODt - R^2 = O^2 + D^2t^2 + 2ODt - R^2 0$

which in itself is an equation of the form (equation 4):

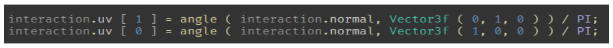
$f(x) = ax^2 + bx + c$

with $a = D^2$, b=2OD and $c = O^2 - R^2$ (remember that x in equation 4 corresponds to t in equation 3 which is the unknown). Being able to re-write equation 3 into equation 4 is important because equation 4 is known as a **quadratic function**. It is a function for which the roots (when x takes a value for which f(x) = 0) can easily be found using the following equations (equation 5):

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$
$$\Delta = b^2 - 4ac$$

Fig. 3. the quadratic equation of the sphere intersection

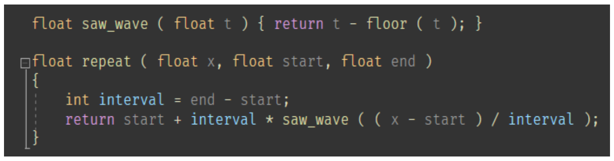
To sample a pixel on the texture, we need to specify the uv wrap



```
interaction.uv [ 1 ] = angle ( interaction.normal, Vector3f ( 0, 1, 0 ) ) / PI;
interaction.uv [ 0 ] = angle ( interaction.normal, Vector3f ( 1, 0, 0 ) ) / PI;
```

Fig. 4. the sphere coordinate map to uv

mode because the original uv coordinate may out of [0,1]x[0,1] and thus invalid. Also we need to specify the pixel interpolation mode, to make the image more smooth on texture edge.
Here I choose the repeart mode the the linear interpolation.
The repeat mode uv wrap need a saw wave as the original signal and to strech this wave in time and amplitude we will get the repeat function. See Fig.5.
The linear interpolation simply choose the 4 pixel around the uv



```
float saw_wave ( float t ) { return t - floor ( t ); }

float repeat ( float x, float start, float end )
{
    int interval = end - start;
    return start + interval * saw_wave ( ( x - start ) / interval );
}
```

Fig. 5. The repeat wrap implementation

postion and interpolate them according to the distance between each pixel position and the real uv pixel position.

There is one problem that make me confused. When the generate-checkboard using odd num-chekers arg, the texture will present line of aliasing in the image, the problem might be related to the interpolation.

3.4 The integrator

The integrator is the another main algorithm of this program.The integrator part is in charge of calculate the randiance of every ray and assign them to pixel to render a image. To generate the picuture, the intergrator generate rays from the camera point accroding to the pixel position on the image. Every pixel may use multiple rays to do the anti-aliasing.

A ray may intersects with nothing or the light or a geometry. In the first 2 case, the radiance of the ray is just black or the color of light. The 3rd case is the main problem the algorithm solves. In normal raytracing,the ray which intersect with a reflection object need to recursively generate rays again to integrate the randiance of the ray its own. However, here we simply replace the deeper ray tracing with the phong lighting model. The model tells us that a randiance of a given point on the algorithm can be calculated by the light source, the vision direction and the point info.
In general, the phong lighting model use the input: the ray , the intersection info, the light sources to generate the output: the radiance of this ray.
The phong model have been conclude from my previous homework report. However, these are still several important things to say:
1. The randiance of the original ray is determined by all the light source. More specifically, we need to generate every second ray for every light sources, if the second ray can reach the light source directly without being blocked, the light source's randiance will be considered.
2. The specular and diffuse part shouldn't be divided by the distance square.

4 RESULTS

With all the algorithms above, we now are able to generate one image represent the 1st and 0nd reflections light rays of a given scene. See Fig.6 for config 11.

5 CONCLUSION

This assignment helps me to make the concept of ray-tracing clear and gives me a comprehensive experience of how to realise a whole program of ray-tracing, which is about Object design, Multi-thread, image sampling and so on.

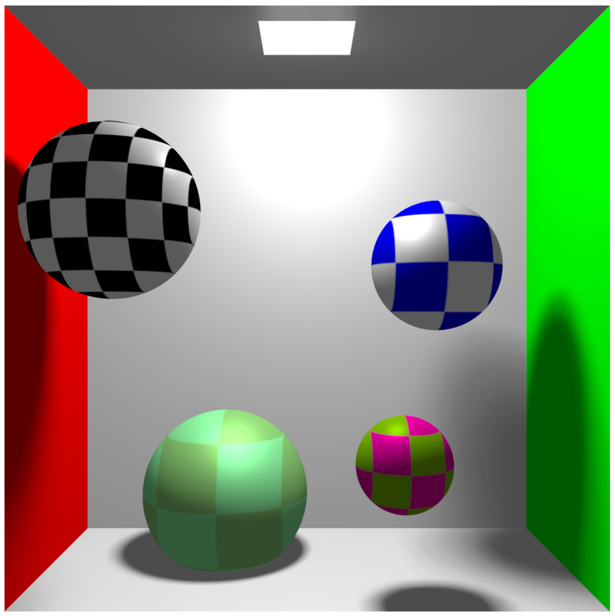


Fig. 6. The final result of config11, 1024x1024 resolution and 4X anti-aliasing