Computer Graphics Coursework Report

Zl16533 yp16505

Ray Tracing

The first part of the coursework is to use ray tracing to render an image of the object stored in the "TestModel" file. The theory of ray tracing is just like simulating the process of taking a photo in the real world. A photograph is the projection of the objects on the screen. For each pixel on the screen, the value of the color is decided by the combination of every ray

First of all, a pinhole camera and a screen had been set up (the position of the pinhole camera and the size of the screen can be set by the global variable). All the objects were separated into triangles. A ray was lunched from the pinhole through the screen to find which objects should be projected on this pixel. The formula used to find the cross point of this ray and the object is:

$$v_0 + ue_1 + ve_2 = s + td$$

The formula can be transformed and used to find which of the triangles should be projected into the pixel. The result of the first step is showed as figure 1.

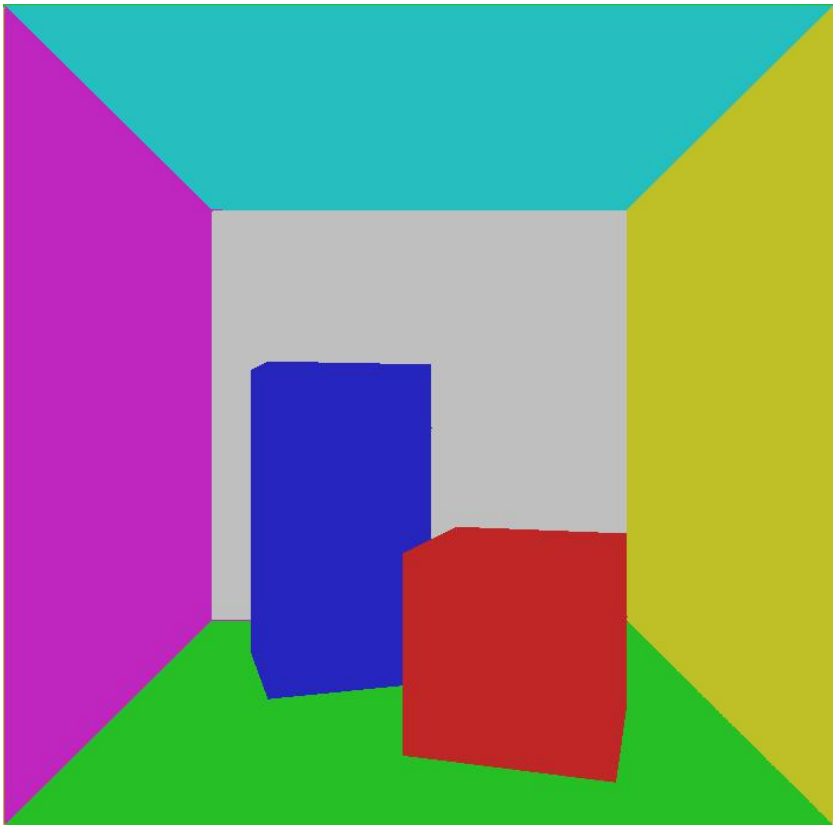

Figure 1

The second step we did is to enable the camera to move. SDLK was used to read the user's input and a variable called yaw is declared to control the rotate angle of the camera. The camera was enabled to do the following motion:

1) Move forward along the Z-axis, when the up key is pressed;
2) Move backward along the Z-axis when the down key is pressed;
3) Rotate to the left side when the left key is pressed
4) Rotate to the right side when the right key is pressed

Then, the light was added into the scene. The light source can also be moved by pressing "QWEASD". The power of the light on the objects was calculated by the following formula:

$$B = \frac{P}{A} = \frac{P}{4\pi r^2}$$

The result of adding light is like Figure 2:

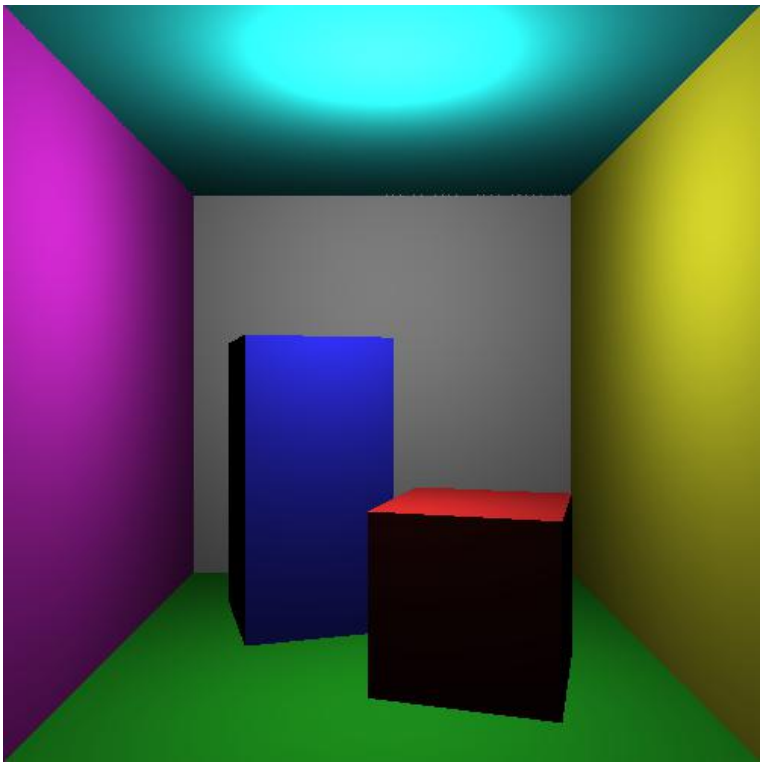

Figure 2

As showed, although light had been added, there is no shadow in the scene. So, what we did next is to add shadow. The process of finding shadows was very similar to the process of step 1. A vector was created to compute the angle between the normal of the ray which is started from the light. If the angle is negative, the surface may have shadow on it. After that we used the ray

started from the light to find whether this triangle is the closest triangle to the light or not. If there is another triangle which is closer to the light, then the surface has shadow on it. The final result of adding shadow is like Figure 3 showed.

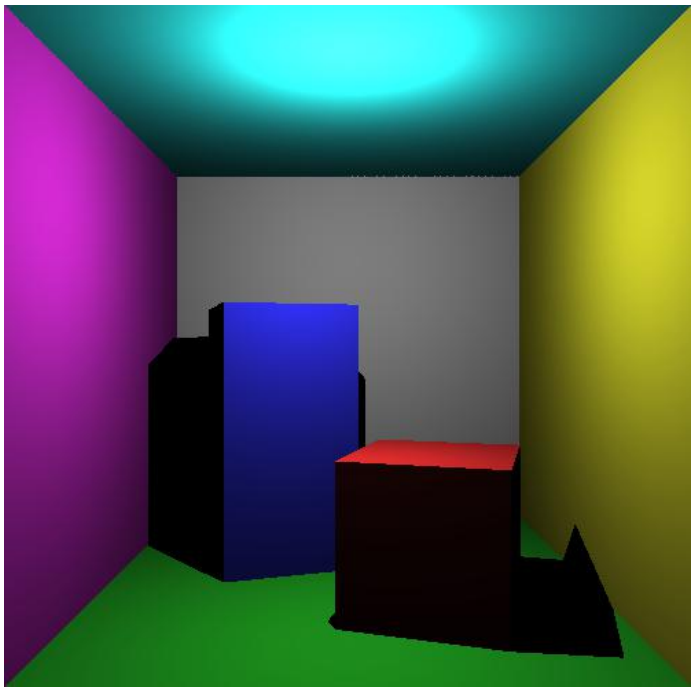

Figure 3

After that, the scene seemed to be too dark, since there is no indirect light. Therefore, we increase the light on every object so that it looks like much closer to the real. The effect was shown in figure 4.

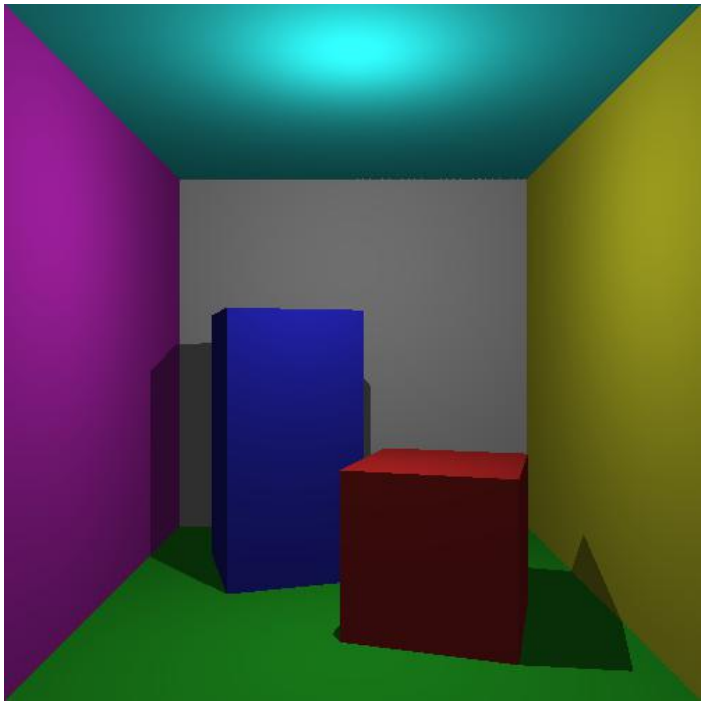

Figure 4

Extension

Soft Edge (Anti-aliasing)

We did the soft edge by double the height and width of the screen and combine the neighborhood four pixels into one pixel. The effect is shown in Figure 5.

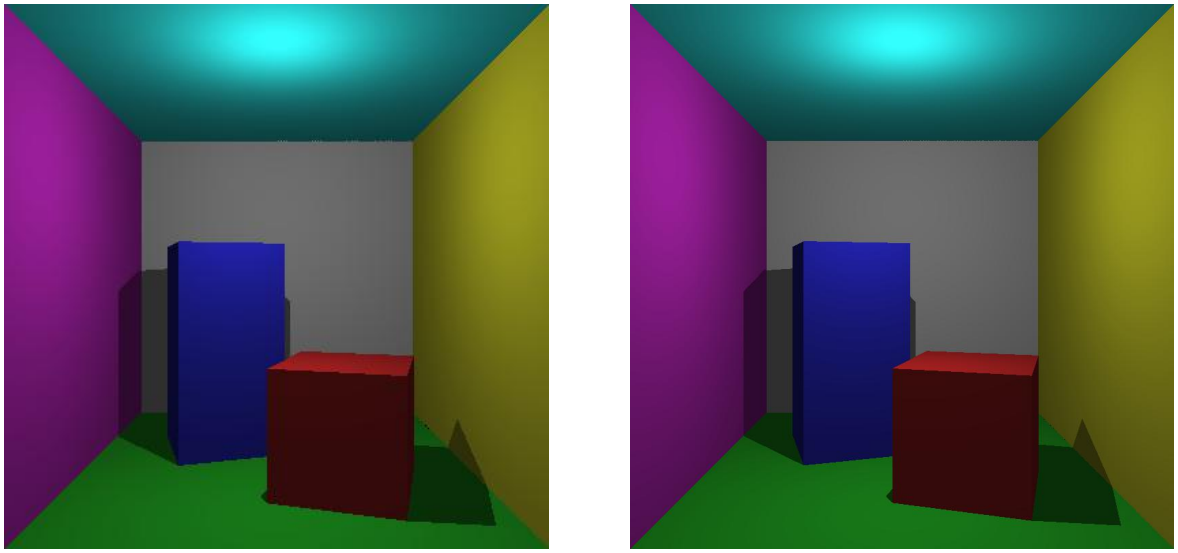

Figure 5 Left (before soft edge), right (after soft edge)

Depth of Focus

Depth of focus is like the soft edge, it just shoot several rays in the vicinity of the every pixel towards the objects and then average the value of all these rays into one pixel. This time, a focus depth is set and the objects behind or in front of this depth will be blurry, like Figure 6 shown.

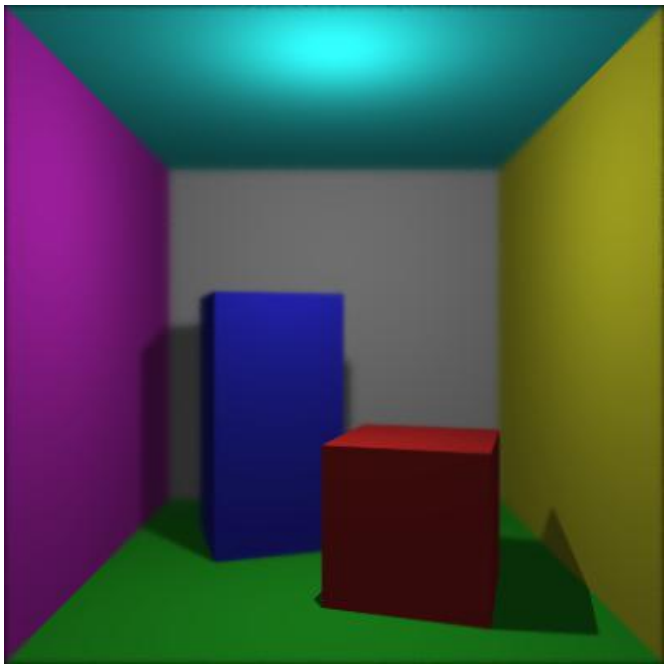

Figure 6

Soft shadow

The basic principle of soft shadow is not very complex. The idea is to shoot many test rays from the intersection point towards the point light source, but we add a bit of an offset to each one, so that it goes close to the point light, but just misses. In this case, the edge of the shadow will be blurry. (See the difference of the edge of shadow in both pictures)

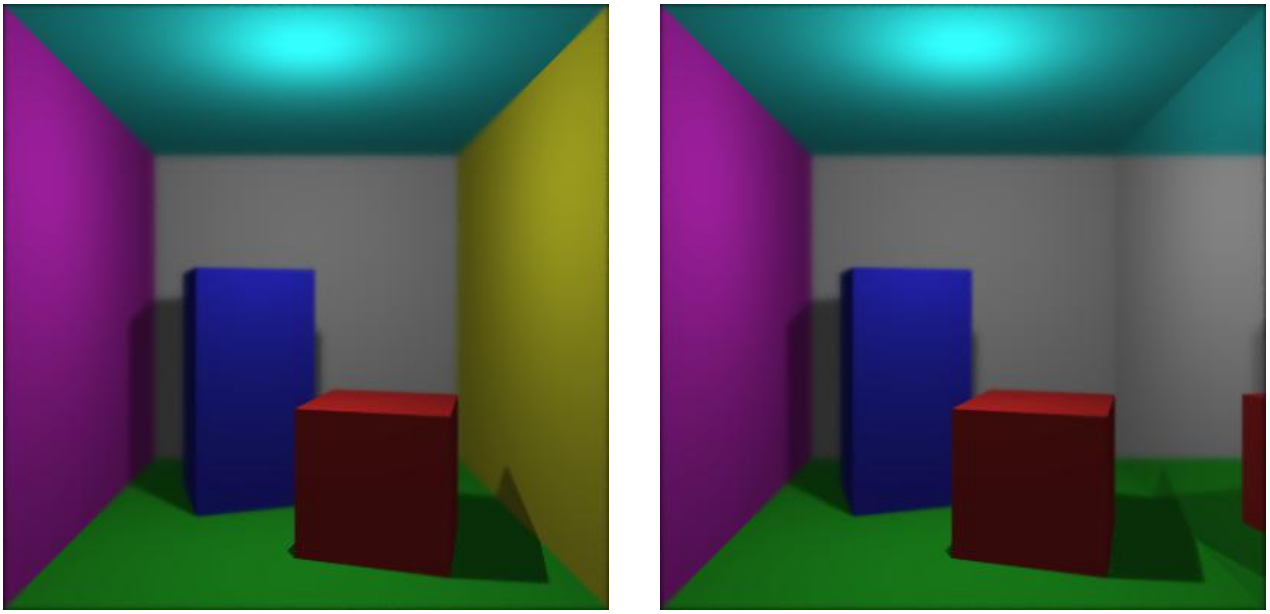

Figure 7

Specular reflection

When rendering specular reflection, we just check every intersection point whether it is a specular material or not. If it is, then the intersection point will become a new start point and a ray will be shot from here to another place to find a second intersection point. After that, the pixel value of the first intersection point will be replaced by the pixel value of the second intersection point, just like a mirror.

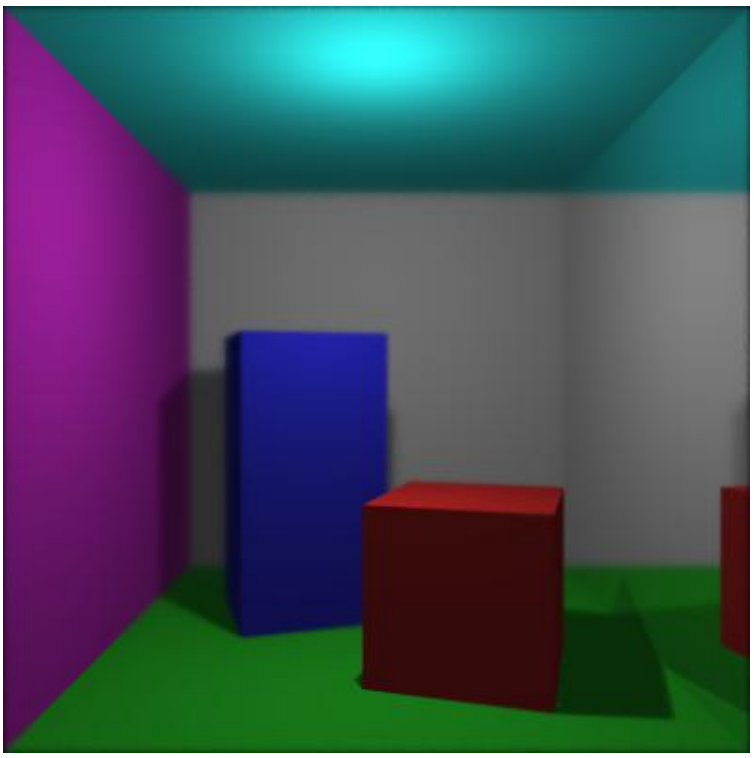

Figure 8

Storing the geometry in a hierarchical spatial structures

We rearranged the objects and created some invisible triangles to cover multiply triangles (such as the cubes in the middle of the room). If the ray did not touch the cover triangles, there is no need for this ray to check whether it can reach the triangles behind it, which can save a lot of time.

Optimisation (Parallel Ray Tracer)

In terms of optimisations, we divided the screen into four part and created four thread to deal each part, which will enable us to trace ray parallelly. The effect of applying threads was very distinct. Rendering a 300x300 images without thread would take about 33 seconds, but rendering a same size image with thread would only take about 12 seconds, which almost save more than 60% of time.

Apart from that, another measure was used to increase the render speed. When we are rendering shadow, we notice that when the position of light source is higher than the tall block, the position of shadow will be lower than the tall block. Therefore, in this case we do not need to compute shadow for pixels that are higher than the tall block, given that there is no shadow at there. after using this measure, the render time has been decreased from 12 seconds to about 9 seconds.