Assignment 3: Ray Tracing Basics

NAME: YIN HAIRUI
STUDENT NUMBER: 2020533028
EMAIL: YINHR@SHANGHAITECH.EDU.CN

1 INTRODUCTION

In this assignment, we are asked to render a scene by simple ray tracing with area light. Camera generates rays to pixel on the screen plane and ray sintersect with objects. After calculation, we can get the radiance of the pixel which contributes to the whole photo. The followings are all jobs that have been doen, containing basic task and texture!

- generate rays from camera
- ray geometry intersection
- Phong lighting at intersection
- light ray sampling for soft shadow
- anti-aliasing by super-resolution
- texture mapping

2 IMPLEMENTATION DETAILS

2.1 Camera Rays

There are elements that we are required to define in camera class:

- Forward: a normalized vector pointing at where the camera looks at
- Right: a normalized vector pointing at the right side of camera
- Up: a normalized vector pointing at the up side of camera

To generate a ray from the camera, two floating number 'dx' and 'dy' are given, specifying the ray's location on raster space. Then we need to calculate the coordinate in world space of this point. The 'dx' and 'dy' has range of

$$dx \in [0, resolution.x()), dy \in [0, resolution.y())$$

which give a percentage position of the current raster. To ease the calculation of ray sample, I return the lower left corner of grids that divides the raster.

$$LC = SC - H_n * s_x/2 - V_n * s_y/2$$

Where LC is lower left corner, SC is the screen center, H_n is the normalized vertor in horizon, s_x is the size of resolution in horizon direction, V_n is the normalized vertor in vertical, s_y is the size of resolution in vertical direction.

2.2 Geometry Intersection

Given a ray, the geometry intersection part gives the answer of whether the ray intersects with any geometry and returns a element called 'interaction' recording all imformation about the intersection, containing

- type: Can be "NONE", "GEOMETRY" or "LIGHT", meaning what the ray first intersects
- pos: the position where intersection happens
- dist: the distance between ray.origin and intersection position
- normal: normal vector of the intersection point
- uv: a percentage variable giving ways for texture mapping
- model: phone light model telling the properties of light in this intersecting position

There are three types of geometries we define:

2.2.1 Triangle. Trianle intersection depends on linear equation

$$o + td = (1 - b_1 - b_2)P_0 + b_1P_1 + b_2P_2$$

2.2.2 Rectangle. Rectangle has a similar formula as triangle and a little adjustment in the range of b_1, b_2.

2.2.3 Ellipsoid. Ellipsoid is a little bit difficult. Since we can transform a sphere to ellipsoid with scaling, rotating and transforming, the transmitting matrix can be generated. We use the inverse of the matrix to do transformation in our ray and find out the intersection of transformed ray and unit sphere. After finding the intersection, we use t in the transformed ray equation in untransformed one, which gives us the intersection of ray and ellipsoid. And normal has a more difficult calculating method.

$$Normal = M_inverse.eval().transpose() \times Sphere_n$$

2.3 Phone lighting

Given a light with color, phone lighting represents how the object interacts with the light. It has three components: Ambient, Diffuse and Specular. We load these components when intersect with geometries.

2.3.1 Ambient Light: Ambient light comes from the refleciton of surrounding environment. There is a constant defining the strength of ambient light.

2.3.2 Diffuse Light: Diffuse light is determined by the angle between light and normal of that position. After normalizing these two vectors, I use dot product to find the cos(angle) and mutiple it with light to calculate diffuse light. Bigger the angle is, Weaker is the diffuse light.

2.3.3 Specular Light: As light hit a surface, there is a reflect angle. If we are in the angle of outgoing light, the object will seem glossy. Specular Light calculates the angle between our view direction and the reflecting direction. It has the same calculating order as diffuse light, first normalizing then dot product.
These three light factor interacting with given sampled light altogether are added together distributing to the color in pixels. The calculating procedure in in radiance(), "intergrator.cpp".

© 2022
Computer Graphics I Course Assignment

2.4 Soft shadow

For soft shadow, the area light is divided into several points light, each has a equal weight. I sample it into rectangle grids, center of each as a single light. The sampled area light gives color to pixel where it happend to be blocked by the marginal of geometries in case the pixel becoming full black.

2.5 Super Resolution

Similar to Soft shadow, super resolution samples the pixel into many small pixels, each with a weight. Color of the original pixel is generated by the average of each small pixel. However, super resolution treats every pixel equally, which gives too much unnecessary sampling.

2.6 Texture Mapping

With lib "stb_image", we can load images like jpg, png into readable raw data. Texture divides a geometies into different part, and each part has its unique light model. I define a getPixel() function where we can use "u,v"(defined in the part of geometries intersection), to map a texture into geometries according to its percentage.

3 RESULTS

The results are pictures following.

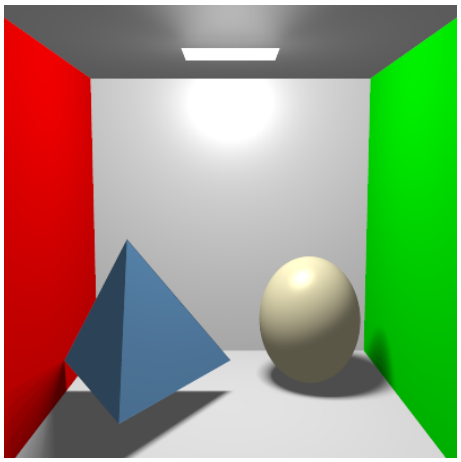

Fig. 1. Origin

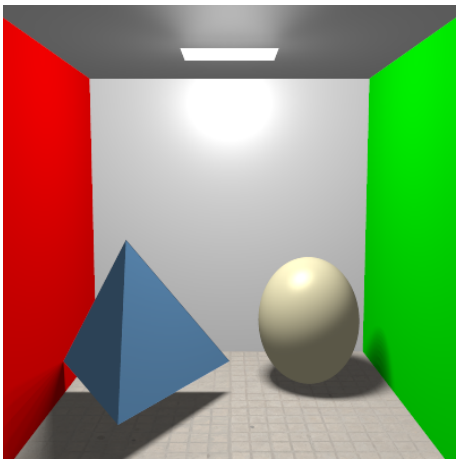

Fig. 2. Render with texture

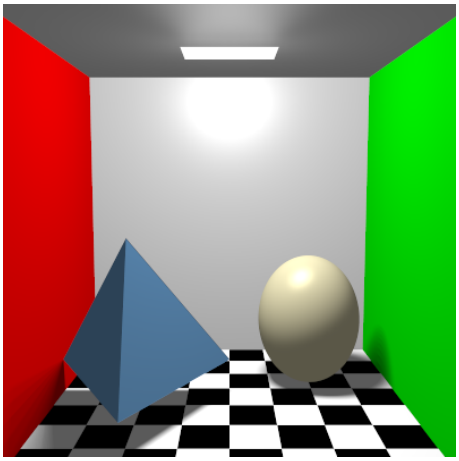

Fig. 3. Render with texture