HW3_hint

PPM Example

```
int width = 200;
int height = 100;
fstream file;
file.open("ray.ppm", ios::out);
file << "P3\n" << width << " " << height << "\n255\n";
for (int j = height - 1; j >= 0; j--) {
        for (int i = 0; i < width; i++) {
                float r = float(i) / float(width);
                float g = float(j) / float(height);
                float b = 0.2;
                file << int(r * 255) << " " << int(g * 255) << " " << int(b * 255) << "\n";
        }
}
```

PPM format

PPM example [edit]

This is an example of a color RGB image stored in PPM format. There is a newline character at the end of each line.

```
P3
# The P3 means colors are in ASCII, then 3 columns and 2 rows,
# then 255 for max color, then RGB triplets
3 2
255
255   0   0    0 255   0    0   0 255
255 255   0  255 255 255    0   0   0
```

Image
(magnified)

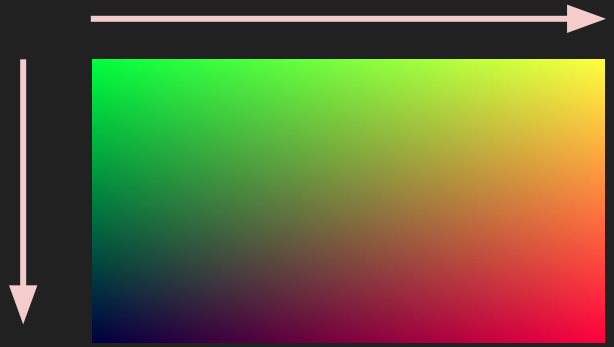
The P6 binary format of the same image represents each color component of each pixel with one byte (thus three bytes per pixel) in the order red, green, then blue. The file is smaller, but the color information is difficult to read by humans.

檔案描述子	類型	編碼
P1	點陣圖	ASCII
P2	灰度圖	ASCII
P3	像素圖	ASCII
P4	點陣圖	二進位
P5	灰度圖	二進位
P6	像素圖	二進位

ppm image v.s. OpenGL display window

- Be aware to the data direction when saving images.

ppm format



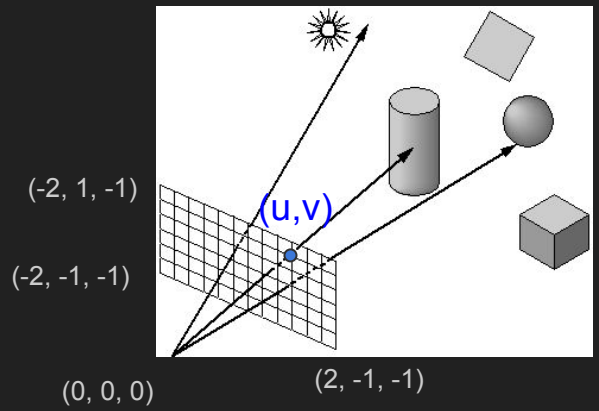
OpenGL display window

Skybox

```
vec3 skybox(const ray& r) {
    vec3 uni_direction = unit_vector(r.direction());
    float t = 0.5 * (uni_direction.y() + 1);
    return (1.0 - t) * vec3(1, 1, 1) + t * vec3(0.5, 0.7, 1.0);
}
```

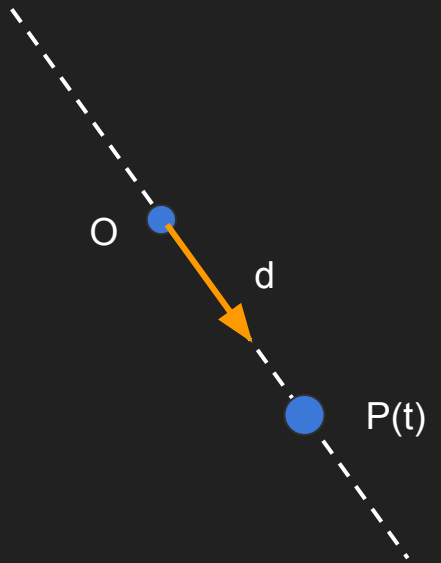
Camera (Primary Ray)

- Ray
  - Origin Point
  - Direction
- Camera
  - COP: center of projection
  - Projection plane
- Image size
  - Ex: 200x100 pixel



ray.h

```
#include "vec3.h"
class ray
{
        public:
        ray() {}
        ray(const vec3& a, const vec3& b) { O = a; D = b; }
        vec3 origin() const          { return O; }
        vec3 direction() const       { return D; }
        vec3 point_at_parameter(float t) const { … }
        vec3 O;
        vec3 D;
};
```



Sphere intersect

main.cpp

```
vec3 trace(const ray&r, const vector<sphere> &list, int depth) {
        if ( sphere(vec3(0, 0, -1), 0.5).hit(r, tmin, tmax, rec) ) {
                return vec3(1.0, 0.0, 0.0);
        }
        else return skybox(r);
}
```

geo.h

```
bool sphere::hit(const ray& r, float tmin, float tmax, hit_record& rec) const {
...
}
```

Sphere intersect

$$f(p) = ||p - c|| - r = 0$$

$$f(r(t)) = ||r(t) - c|| - r = 0$$

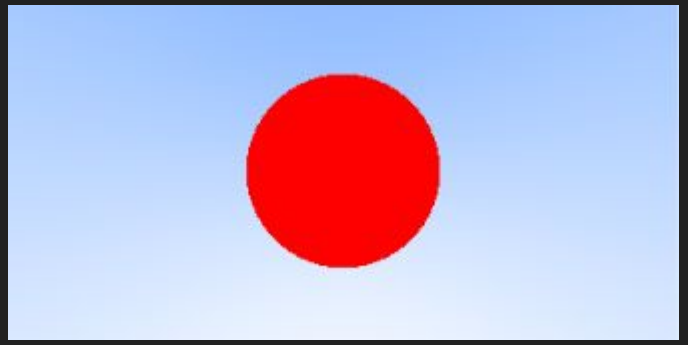
$$||o + td - c|| = r$$

$$t^2(d \cdot d) + 2t(d \cdot (o - c)) + (o - c) \cdot (o - c) - r^2 = 0$$

$$At^2 + Bt + C = 0$$

$$B^2 - 4AC \geqq 0 \quad => hit = true$$

Result



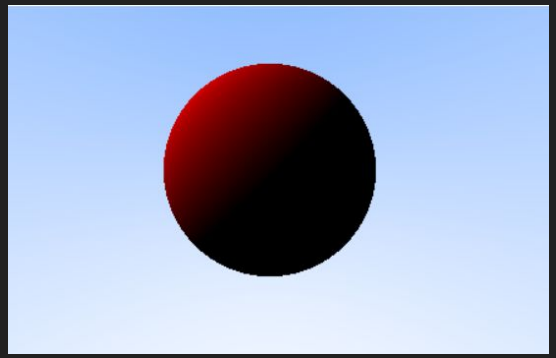
Diffuse Lighting

main.cpp

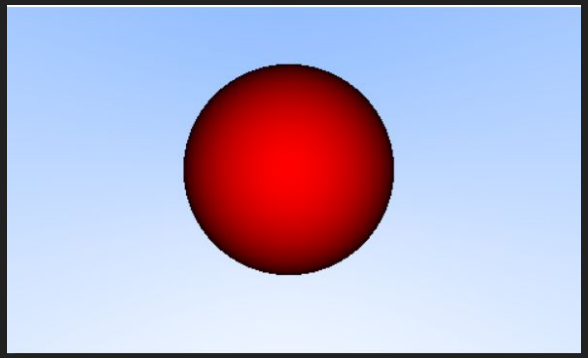
```
vec3 trace(const ray&r, const vector<sphere> &list, int depth) {
        if ( sphere(vec3(0, 0, -1), 0.5).hit(r, tmin, tmax, rec) ) {
                color = shading( … )
                return color;
        }
        else return skybox(r);
}
```

```
vec3 shading(vec3& lightsource, vec3& intensity, hit_record ht, vec3 kd, const
vector<sphere>& list) {
        return  Kd * intensity * max(0, N dot L);
}
```

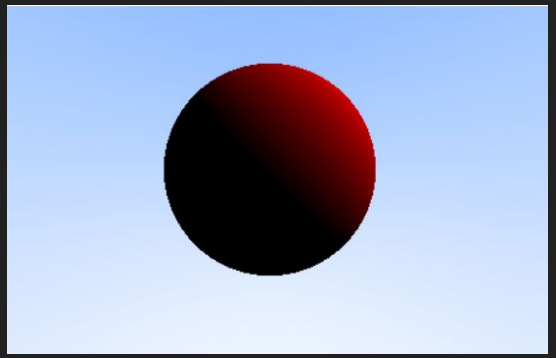
Result



Light Position
(-10, 10, 0)



Light Position
(0, 0, 0)



Light Position
(10, 10, 0)

Multi-sphere list in main.cpp

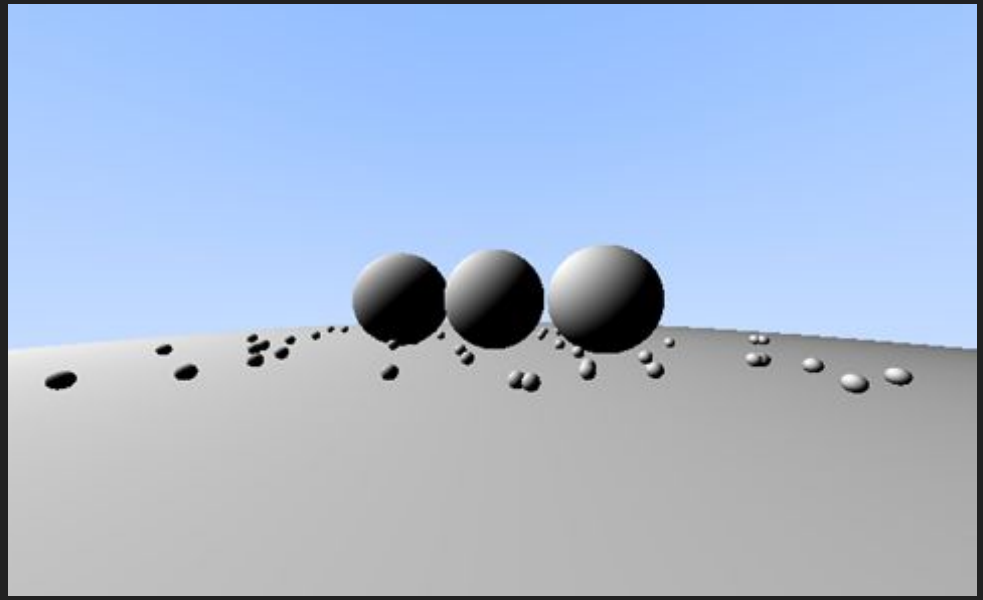
```
vector<sphere> hitable_list;
hitable_list.push_back(sphere(vec3(0, -100.5, -2), 100)); //ground
hitable_list.push_back(sphere(vec3(0, 0, -2), 0.5, vec3(1.0f, 1.0f, 1.0f), 0.0f, 0.9f)); // refract
hitable_list.push_back(sphere(vec3(1, 0, -1.75), 0.5, vec3(1.0f, 1.0f, 1.0f), 0.9f, 0.0f)); // reflect
hitable_list.push_back(sphere(vec3(-1, 0, -2.25), 0.5, vec3(1.0f, 0.7f, 0.3f), 0.0f, 0.0f)); //diffuse
for (int i = 0; i < 48; i++) {
        float xr = ((float)rand() / (float)(RAND_MAX)) * 6.0f - 3.0f;
        float zr = ((float)rand() / (float)(RAND_MAX)) * 3.0f - 1.5f;
        int cindex = rand() % 8;
        float rand_reflec = ((float)rand() / (float)(RAND_MAX));
        float rand_refrac = ((float)rand() / (float)(RAND_MAX));

         // small balls are all reflected ray.
        hitable_list.push_back(sphere(vec3(xr, -0.4, zr - 2), 0.1, colorlist[cindex], rand_reflec, 0.0f));
}                                                    center,      radius,      color,          w_r,      w_t
```

multi-sphere

main.cpp (pseudocode only)

```
vec3 trace(const ray&r, const vector<sphere> &list, int depth) {
        hit_record rec;

        for( all hitable_sphere s ) {
                if( s.hit ( …,  rec ) ) {  …  }
        }

        Find the smallest t of hit_record;

        color = shading(lightposition, lightintensity, rec, list[index], list );
}
```

Result

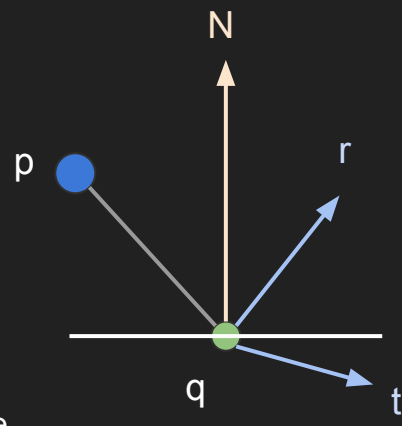


```
// camera
vec3 lower_left_corner(-2, -1, -1);
vec3 origin(0, 0, 1);
vec3 horizontal(4, 0, 0);
vec3 vertical(0, 2, 0);

// light
vec3 LightPosition(-10, 10, 0);
vec3 LightIntensity(1.0, 1.0, 1.0);
```

Recursive Ray Tracer

main.cpp (pseudocode only)

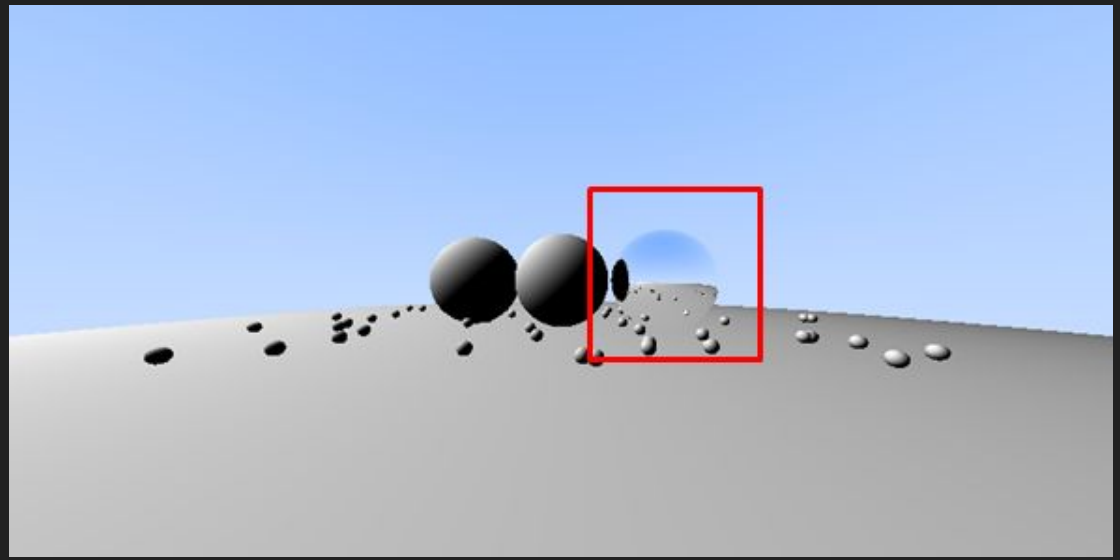
```
vec3 trace(const ray&r, const vector<sphere> &list, int depth) {
        q = intersect position
        n = normal(q);  //or get from hit_record
        r = reflect(q, n);
        t = transmit(q, n);

        diffuse = shading(…);
        reflected = trace((q, r), depth+1);
        transmitted = trace((q, t), depth+1);

        return (w_l*diffuse  + w_r*reflected  + w_t*transmitted);
}
```



assume
Mirror: w_l=0.0f, w_r=1.0f, w_t =0.0f
Plastic: w_l=0.9, w_r=0.6f, w_t =1.0f

Reflected Ray

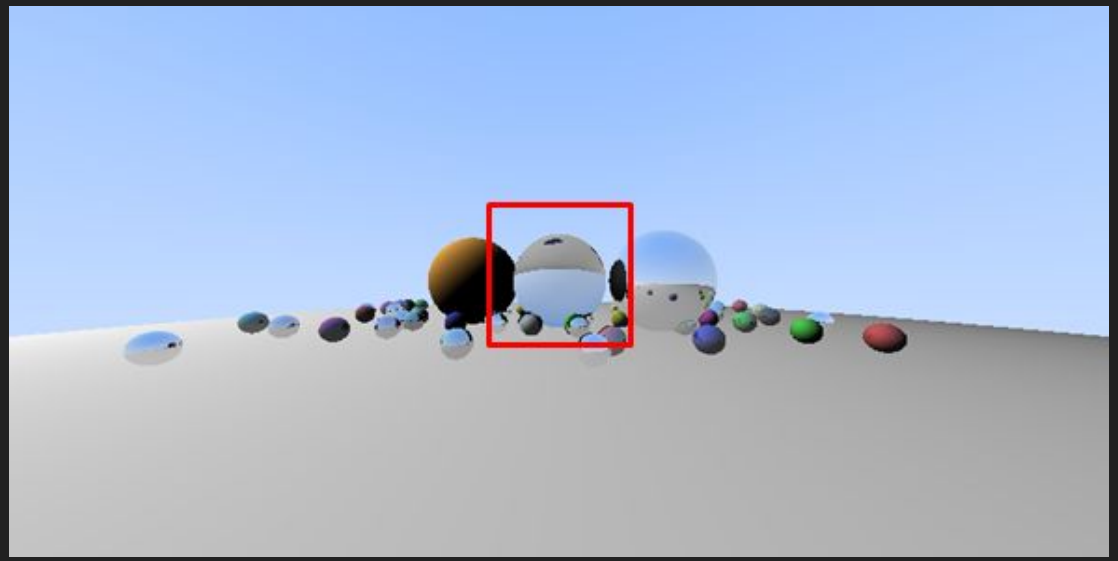
hitable_list.push_back(sphere(vec3(1, 0, -1.75), 0.5f, 0.9f));
center radius w_r



return (1.0f-rec_nearest.w_r)*diffuse_color + rec_nearest.w_r*reflected_color;

Transmitted(Refracted) Ray

hitable_list.push_back(sphere(vec3(0, 0, -2), 0.5f, 0.0, 0.9f));
center radius w_r w_t

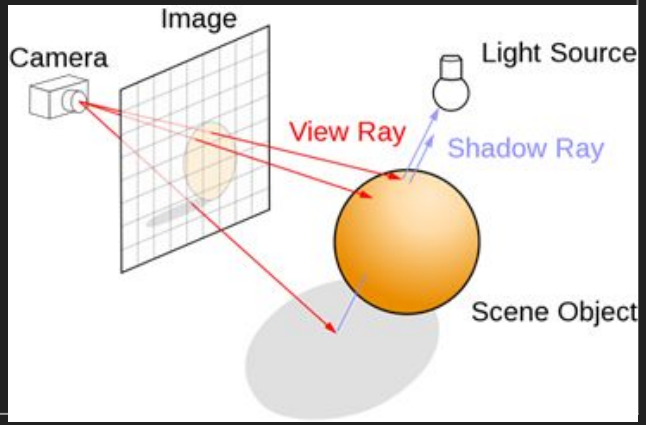


return (1.0f - w_t) * ((1.0f-w_r) * diffuse_color + w_r * reflected_color) +
w_t * transmitted_color;

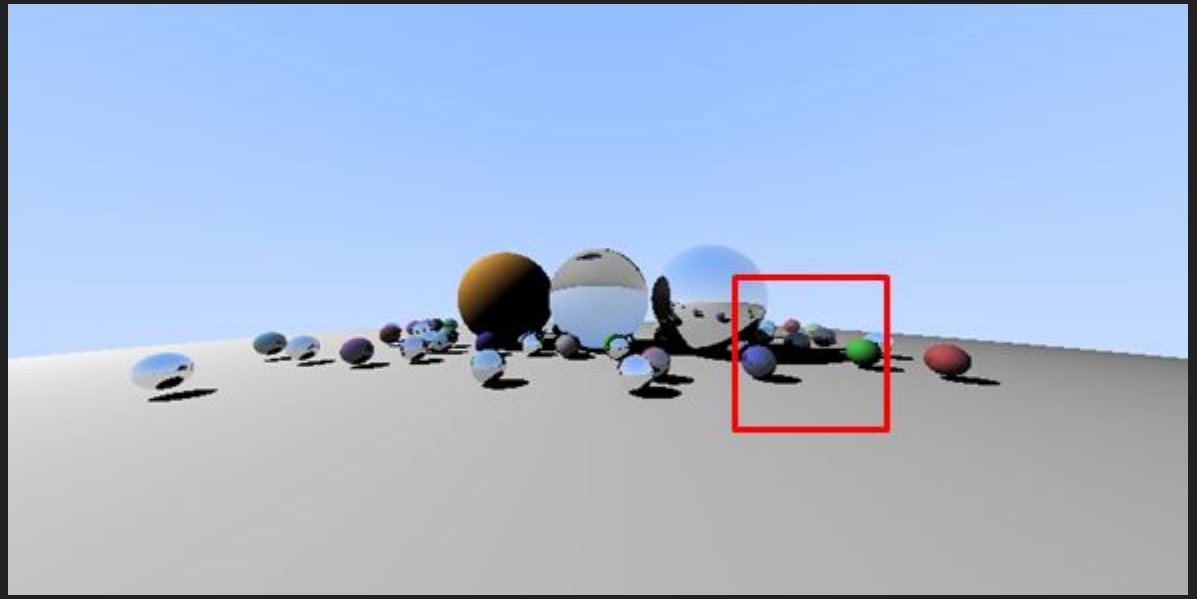
Shadow Ray in shading()

main.cpp

```
vec3 shading(vec3& lightsource, vec3& intensity, hit_record ht, vec3 kd, const
vector<sphere>& list) {

        ray ShadowRay(.....)

        if ShadowRay intersect with sphere
                // ground shading color
                return vec(1.0, 1.0, 1.0);
}
```



Shadow Ray Result



Save picture

- ppm file is not convenient for window to display, so you also need to save another general format such as bmp, jpg, or png.

- Be careful to the orientation of different format.

- Please also put the screenshot of your final result in the report.