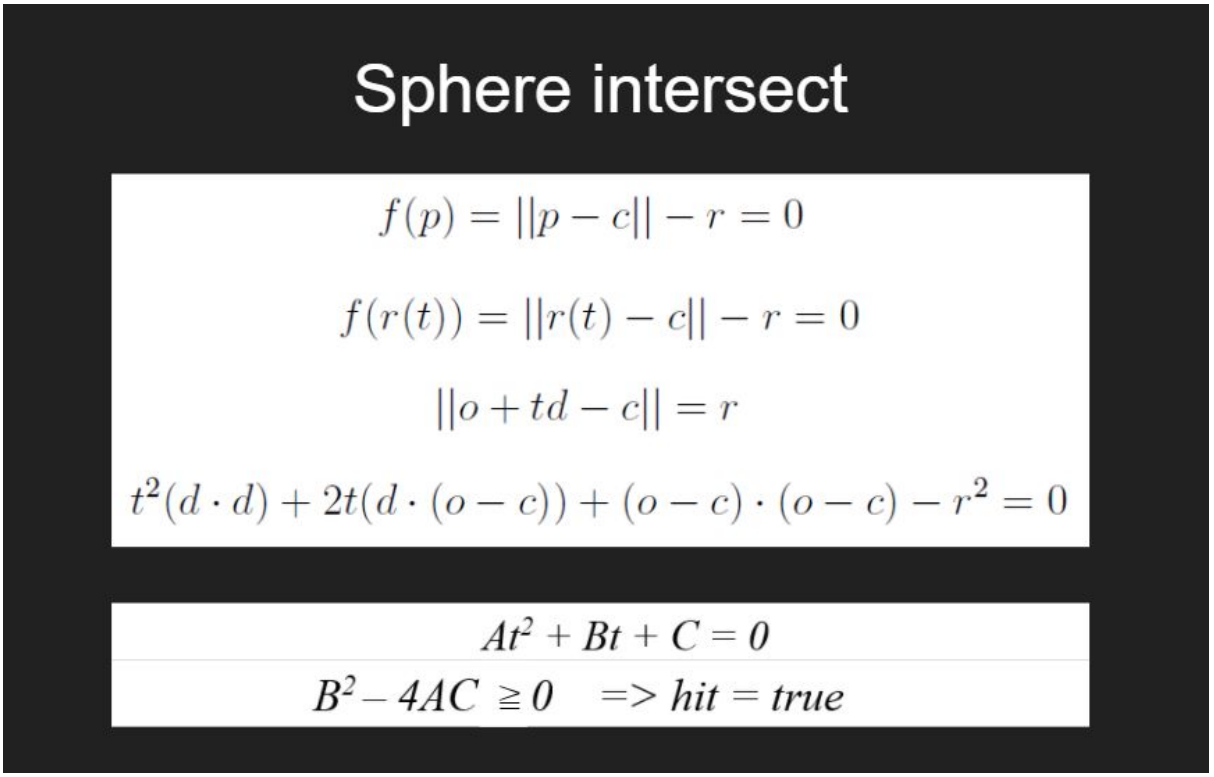
計算機圖學 HW3

309553012 黃建洲

1. Code Explanation and how I implement the works.

(1) ray.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; }
    inline vec3 point_at_parameter(float t) const{
        /*
        To-do:
            compute the position at t
        */


        vec3 pos;
        pos = origin() + t * direction();
        return pos;
    }

    vec3 O; //center(origin) point
    vec3 D; //direction vector
};

#endif
```



這部分只需要完成point_at_parameter的function，簡單套用課本公式: Position = O + t * D。

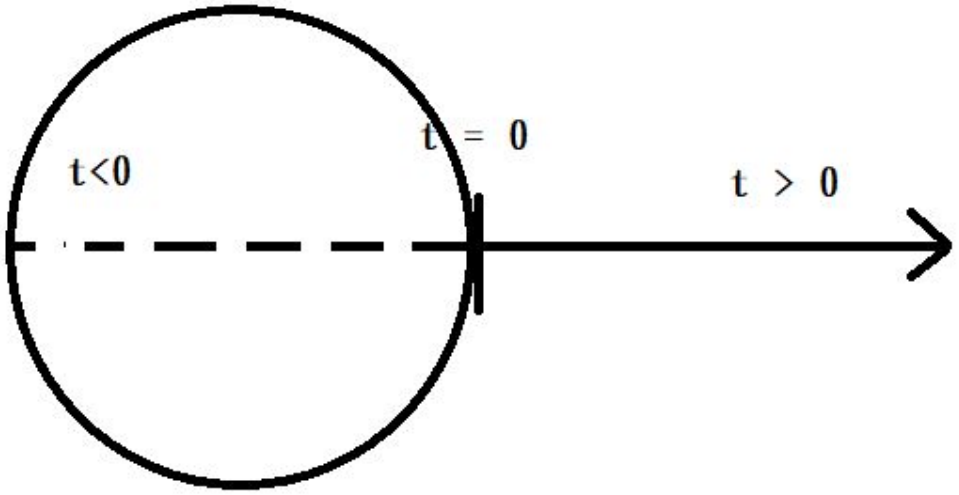
(2) geo.h -- sphere.hit

```
bool sphere::hit(const ray &r, float tmin, float tmax, hit_record & rec) const {
    /*
    To-do:
        compute whether the ray intersect the sphere
    */

    vec3 o = r.origin();
    vec3 d = r.direction();
    vec3 ce = center;

    float rc = radius;
    float A = dot(d, d);
    float B = 2 * dot(d, (o - ce));
    float C = dot((o - ce), (o - ce)) - rc * rc;
    float root = B * B - 4 * A * C;

    if (root > 0) {
        //cout << -B - sqrt(root) << endl;
        float t1 = (-B - sqrt(root)) / (2 * A);
        float t2 = (-B + sqrt(root)) / (2 * A);

        //if (t1 == 0) t = t2;
        //if (t1 <= tmin) return false;
        if ((t1<=tmin && t2 <= tmin) || (t1>=tmax && t2>=tmax) || (t1<=tmin && t2 >= tmax)) return false;
        float t = t1;
        if (t <= 0.01f) t = 0;
        if (t <= tmin) t = t2;
        if (t <= 0.01f) t = 0;
        //if(t<= tmin) t = t2;
        if (t <= tmin || t >= tmax) { return false; }
        //cout << t << endl;
        rec.p = r.origin() + t * r.direction();
        vec3 v = rec.p - ce;
        v.make_unit_vector();
        rec.nv = v;
        rec.t = t;
        return true;
    }
    else return false;
}
```

根據講義與hint，我們定義出一個t的二元一次方程式At^2 + Bt + C = 0(A,B,C的定義根據Hint定義)，以公式解求出兩根，若兩根皆為虛數則無交點。若方程式有實數解，首先檢查兩根是否有一個在tmin, tmax的範圍內，若沒有則視同沒有交點。

接著為了折射與反射的需求，我不希望他又再次取到與球的交點(t = 0附近的，如附圖)，由於計算上會有雜訊，故我將t = 0.1以下皆視為0，並檢測若第一次取的t1不符合規則，則改取t2，一樣過濾雜訊並檢測，若仍不符合規範則視為無交點。

其餘狀況視為有交點，取得t之後，將trace function傳入的rec分別填入交點座標p、交點於圓上的normal向量nv以及交點在ray上之t。



(3) geo.h -- reflect

```
vec3 reflect(const vec3 &d, const vec3 &nv) {
    /*
    To-do:
        compute the reflect direction
    */
    vec3 tmp_d = d;
    vec3 tmp_n = nv;

    vec3 r = tmp_d - 2 * dot(tmp_d, tmp_n) * tmp_n;
    return r;
}
```

$$r = d - 2(d \cdot n)n$$

反射部分一樣套用課本公式，其中d為經過normalize的ray direction，n則為在hit裡面取得的rec.nv。計算得到反射向量。

(4) geo.h -- refract

```
vec3 refract(const vec3& L, const vec3& N, float e) {
    /*
    To-do:
        compute the refracted(transmitted) direction
    */
    vec3 tmpL = L;
    vec3 tmpN = N;
    tmpL.make_unit_vector();
    tmpN.make_unit_vector();
    float cosi = -dot(tmpL , tmpN);
    float e2sini2 = e*e*(1 - cosi * cosi);
    float cost2 = 1.0f - (e2sini2);
    //cout << cost2 << endl;
    if (cost2 <= 0) return reflect(tmpL, tmpN);
    vec3 t = e * tmpL + (e * cosi - sqrt(abs(cost2))) * tmpN;
    return t;
}
```

```
float3 refract( float3 i, float3 n, float eta )
{
  float cosi = dot(-i, n);
  float cost2 = 1.0f - eta * eta * (1.0f - cosi*cosi);
  float3 t = eta*i + ((eta*cosi - sqrt(abs(cost2))) * n);
  return t * (float3)(cost2 > 0);
}
```

refract的部分則參考下列網址
https://www.itdaan.com/tw/5b68d5572d7cb5724bd4d2260dc9e33f
的教學，其中L為入射向量、N為normal向量、e為折射率。計算得到折射向量，當全反射發生時則計算反射向量。

(5) main.cpp -- shading

```
vec3 shading(vec3 &lightsource, vec3 &intensity, hit_record ht, vec3 kd, const vector<sphere> &list) {
    /*
    To-do:
        define L, N by yourself
    */
    vec3 L;
    vec3 N;
    L = lightsource - ht.p;
    L.make_unit_vector();
    N = ht.nv;
    ray shadowRay(ht.p, L);

    int intersect = -1;
    hit_record rec;
    float closest = FLT_MAX;
    /*
    To-do:
        To find whether the shadowRay hit other object,
        you should run the function "hit" of all the hitable you created
    */
    for (int i = 0; i < list.size(); i++) {
        if (list[i].hit(shadowRay, 0, 500, rec)) {
            //cout << rec.t << endl;
            if (rec.t < closest) {
                closest = rec.t;
                intersect = i;
```

```
    for (int i = 0; i < list.size(); i++) {
        if (list[i].hit(shadowRay, 0, 500, rec)) {
            //cout << rec.t << endl;
            if (rec.t < closest) {
                closest = rec.t;
                intersect = i;

            }
        }
    }

    if (intersect == -1) {
        return kd*intensity*MAX(0, dot(N, unit_vector(L)));
    }
    else {
        //cout << dot(rec.nv, shadowRay.direction()) << endl;
        float shadow_factor = -dot(rec.nv, shadowRay.direction());
        if (shadow_factor <= 0) shadow_factor = 0;
        //cout << shadow_factor << endl;
        return kd * intensity * MAX(0, dot(N, unit_vector(L))) * (1-shadow_factor) ;
    }
}
```

初始給入的L為交點到光源的方向，N為交點的normal向量，用來計算沒有被影子覆蓋時的diffuse color。

之後以交點為起點， 以L為方向射出一條shadowRay，檢測交點與光源之間是否有障礙物，若有則計算影子。這邊的計算方法已經改成具有attenuation的形式，於extra的部分再敘述。

(6) main.cpp -- trace

```
vec3 trace(const ray&r, const vector<sphere> &list, int depth) {
    if (depth >= max_step) return skybox(r); //or return vec3(0,0,0);

    int intersect = -1;
    hit_record rec;
    float closest = FLT_MAX;
    float min_t = 10000;
    for (int i = 0; i < list.size(); i++) {
        if (list[i].hit(r, 0, 500, rec)) {
            //cout << rec.t << endl;
            if (rec.t < min_t) {
                min_t = rec.t;
                intersect = i;

            }
        }
    }
```

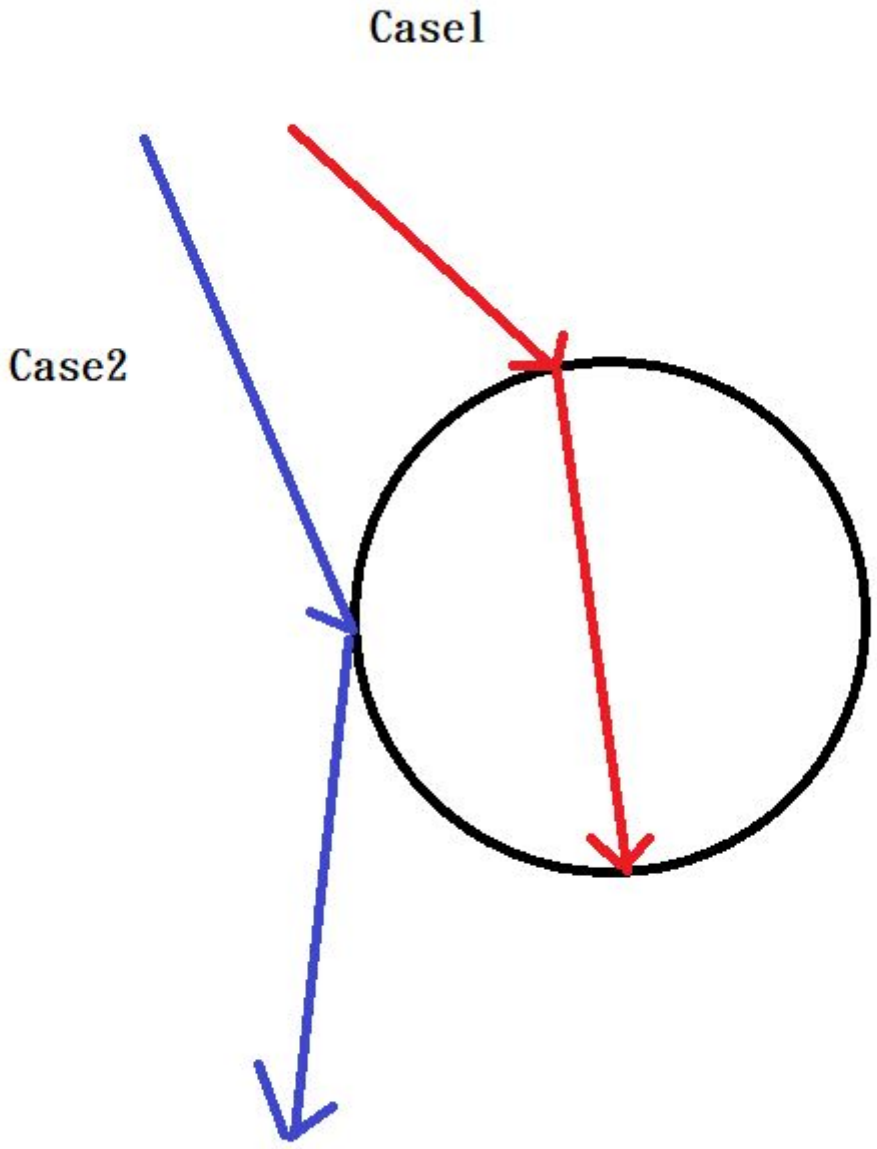
trace中，首先計算ray第一個碰到的交點位於哪一個sphere上，並記錄rec資訊，用rec.t來確定是否是最近的交點。

```
if (intersect != -1) {

    vec3 lightPosition = vec3(-10, 10, 0);
    vec3 lightIntensity = vec3(1, 1, 1);
    list[intersect].hit(r, 0, 500, rec);
    //diffuse
    vec3 diffuse_color = shading(lightPosition, lightIntensity, rec, list[intersect].kd, list);

    //reflect
    vec3 q = r.direction();
    vec3 n = rec.nv;
    q.make_unit_vector();
    n.make_unit_vector();
    vec3 reflect_param = reflect(q, n);
    reflect_param.make_unit_vector();
    ray reflectRay(rec.p, reflect_param);
    vec3 reflect_color = trace(reflectRay,list, depth + 1);
    //vec3 reflect_color = vec3(0,0,0);
```

```
//refract
vec3 L = r.direction();
vec3 N = rec.nv;
float e = 0.66f;
L.make_unit_vector();
N.make_unit_vector();
vec3 refract_param = refract(L, N, e);
refract_param.make_unit_vector();

ray refractRay(rec.p, refract_param);
vec3 refract_color;
//test
vec3 tmp_p = rec.p;
if (list[intersect].hit(refractRay, 0, 500, rec) == true) {

    vec3 L2 = refractRay.direction();
    vec3 N2 = rec.nv * -1;
    float e2 = 1.55f;
    L2.make_unit_vector();
    N2.make_unit_vector();

    vec3 refract_param2 = refract(L2, N2, e2);
    //refract_param2.make_unit_vector();
    ray refractRay2(rec.p, refract_param2);
    refract_color = trace(refractRay2, list, depth + 1);
}
```

```
    else {

        refract_color = trace(refractRay,list, depth+1);
    }
    //refract_color = trace(refractRay, list, depth + 1);
    float wr = list[intersect].w_r;
    float wt = list[intersect].w_t;

    vec3 color =  wt * refract_color + (1-wt) * (wr * reflect_color + (1 - wr) * diffuse_color);
    return color;
}
else {
    return skybox(r);
}
```

若有交點，首先再對正確的intersect index進行一次sphere.hit，以確保rec內容正確。
接著使用先前已經定義的shading function、reflection function、refraction function分別定義diffusion color、reflection color和refraction color。其中diffuse和reflection的部分只需要依照hint與講義的方式設置就能使用。
Refraction的部分則需要額外檢查折射後的光線是否在同一個球內，如下圖:



若在圓內，由於需要重新定義折射率e(空氣進球->球進空氣)，故直接計算下一次折射回到空氣的交點才進入遞迴。若折射後沒有進到球，則直接進入遞迴。
計算出refraction color。
最後根據hint設置三種color以及其參數，得到pixel color。

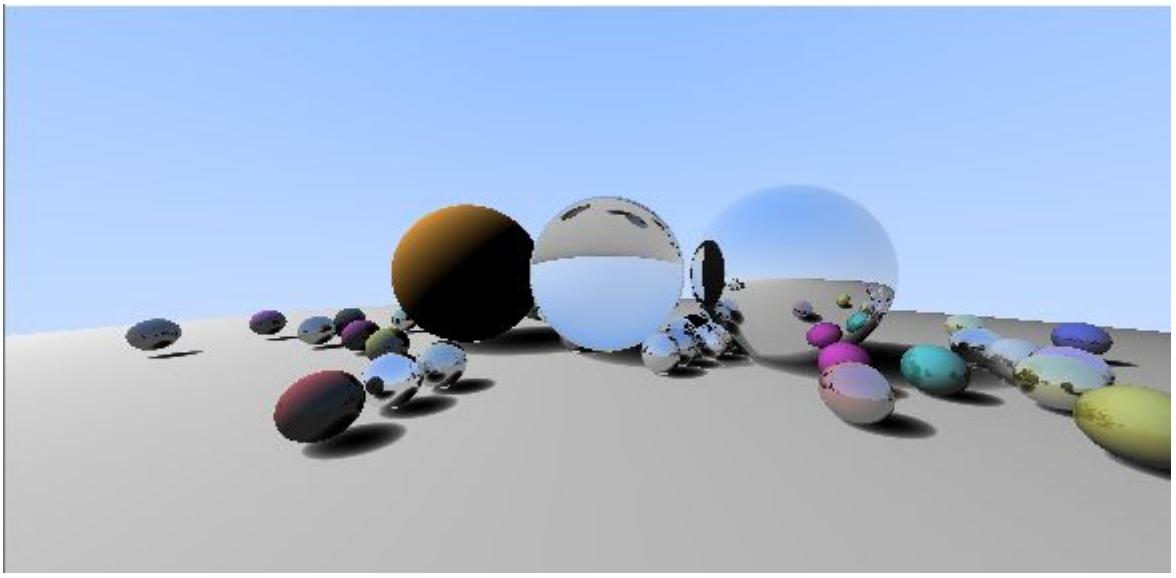
(7) main.cpp -- Main

```
fstream file;
file.open("../ppm2bmp-master/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 u = float(i) / float(width);
        float v = float(j) / float(height);
        ray r(origin, lower_left_corner + u*horizontal + v*vertical);
        vec3 c = trace(r, hitable_list, 0);

        /*Hint: Here to save each pixel after ray tracing*/



        file << int(c.r() * 255) << " " << int(c.g() * 255) << " " << int(c.b() * 255) << "\n";

        // for display window
        int index = ((j) * width + i) * 3;
        data[index + 0] = (GLbyte)(c.r() * 255);
        data[index + 1] = (GLbyte)(c.g() * 255);
        data[index + 2] = (GLbyte)(c.b() * 255);
    }
}
```

```
CImg<unsigned char> im("../ray.ppm");
im.save("../out.bmp");
```

根據範例教學，將pixel資料存入ppm檔案。
轉換圖片部分，雖然嘗試了許多的方法，但大多都轉換不出正常的圖片。最後使用了CImg的開源函式庫進行，成功進行轉換。

2. Result:



3. Describe the problems you met and how you solved them.

在這次作業中我遇到最大的問題是架構上debug的不易。在color的計算上我是在最後才做refraction。這導致雖然我的sphere.hit在前面雖然並不是這麼完整，但在diffusion和reflection上都還是有正常的表現，也因此越錯越多。之後我將整體架構重新檢查，並以邏輯與數學算式一步一步推導，才修好前面錯誤的部分。

Refraction的部分由於一開始沒有概念而卡了很久，而後才想起玻璃球會出現兩次折射，加上去後畫面才正常許多。

4. Extra features -- Shadow attenuation:

Code附於shading處，額外紀錄shadow ray的direction與障礙物的normal，內積作為陰影參數，並利用該參數將影子從裡到外呈現一個漸弱的型態。結果於Result處。