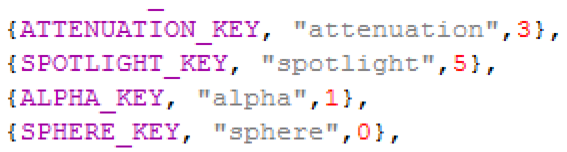
Report about Assignment#4 for Graphics

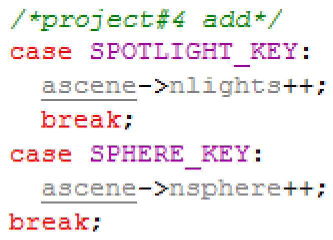
Qucheng Shen

1: About source code:

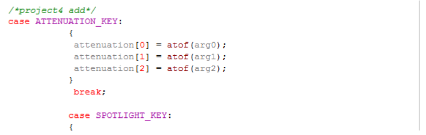
Assignment#4 is a practice of ray tracing method. In the project, it adds some new definition in the ssd.h and ssd.c file, such as the following code:



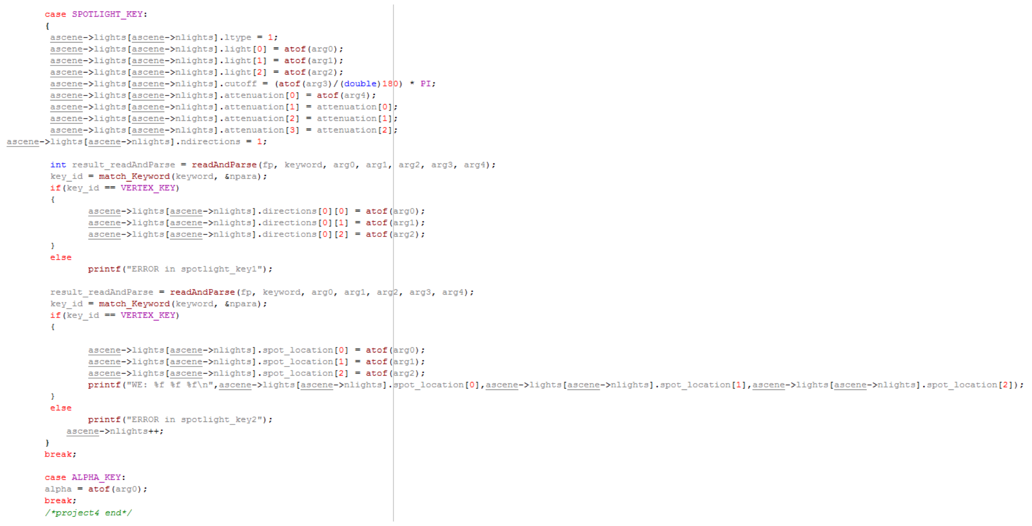
```
{ATTENUATION_KEY, "attenuation",3},
{SPOTLIGHT_KEY, "spotlight",5},
{ALPHA_KEY, "alpha",1},
{SPHERE_KEY, "sphere",0},
```



```
/*project#4 add*/
case SPOTLIGHT_KEY:
  ascene->nlights++;
  break;
case SPHERE_KEY:
  ascene->nsphere++;
break;
```



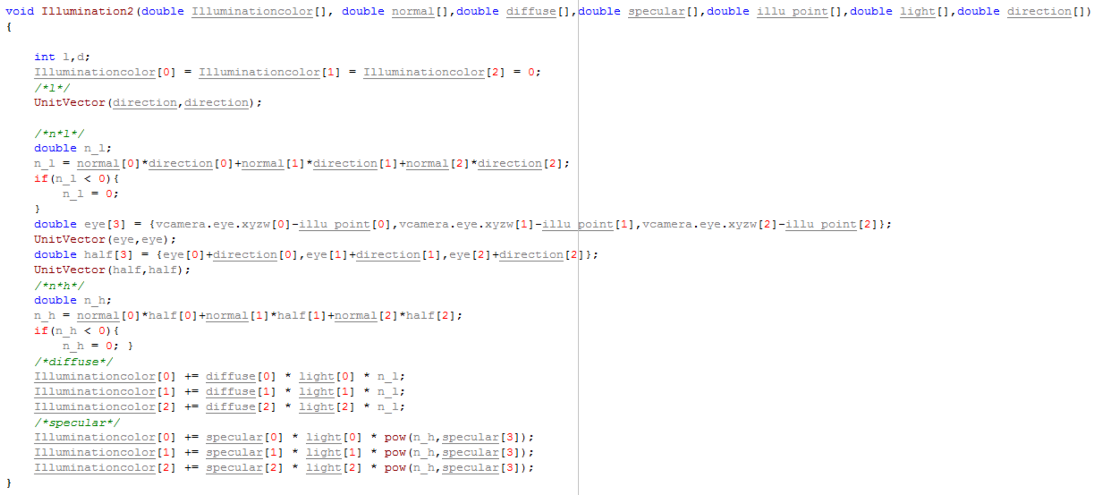
```
/*project4 add*/
case ATTENUATION_KEY:
          {
           attenuation[0] = atof(arg0);
           attenuation[1] = atof(arg1);
           attenuation[2] = atof(arg2);
          }
           break;

          case SPOTLIGHT_KEY:
          {
```



```
  case SPOTLIGHT_KEY:
  {
   ascene->lights[ascene->nlights].ltype = 1;
   ascene->lights[ascene->nlights].light[0] = atof(arg0);
   ascene->lights[ascene->nlights].light[1] = atof(arg1);
   ascene->lights[ascene->nlights].light[2] = atof(arg2);
   ascene->lights[ascene->nlights].cutoff = (atof(arg3)/(double)180) * PI;
   ascene->lights[ascene->nlights].attenuation[0] = atof(arg4);
   ascene->lights[ascene->nlights].attenuation[1] = attenuation[0];
   ascene->lights[ascene->nlights].attenuation[2] = attenuation[1];
   ascene->lights[ascene->nlights].attenuation[3] = attenuation[2];
ascene->lights[ascene->nlights].ndirections = 1;

   int result_readAndParse = readAndParse(fp, keyword, arg0, arg1, arg2, arg3, arg4);
   key_id = match_Keyword(keyword, &npara);
   if(key_id == VERTEX_KEY)
   {
          ascene->lights[ascene->nlights].directions[0][0] = atof(arg0);
          ascene->lights[ascene->nlights].directions[0][1] = atof(arg1);
          ascene->lights[ascene->nlights].directions[0][2] = atof(arg2);
   }
   else
          printf("ERROR in spotlight_key1");

   result_readAndParse = readAndParse(fp, keyword, arg0, arg1, arg2, arg3, arg4);
   key_id = match_Keyword(keyword, &npara);
   if(key_id == VERTEX_KEY)
   {

          ascene->lights[ascene->nlights].spot_location[0] = atof(arg0);
          ascene->lights[ascene->nlights].spot_location[1] = atof(arg1);
          ascene->lights[ascene->nlights].spot_location[2] = atof(arg2);
          printf("WE: %f %f %f\n",ascene->lights[ascene->nlights].spot_location[0],ascene->lights[ascene->nlights].spot_location[1],ascene->lights[ascene->nlights].spot_location[2]);
   }
   else
          printf("ERROR in spotlight_key2");
     ascene->nlights++;
  }
  break;

  case ALPHA_KEY:
  alpha = atof(arg0);
  break;
  /*project4 end*/
```

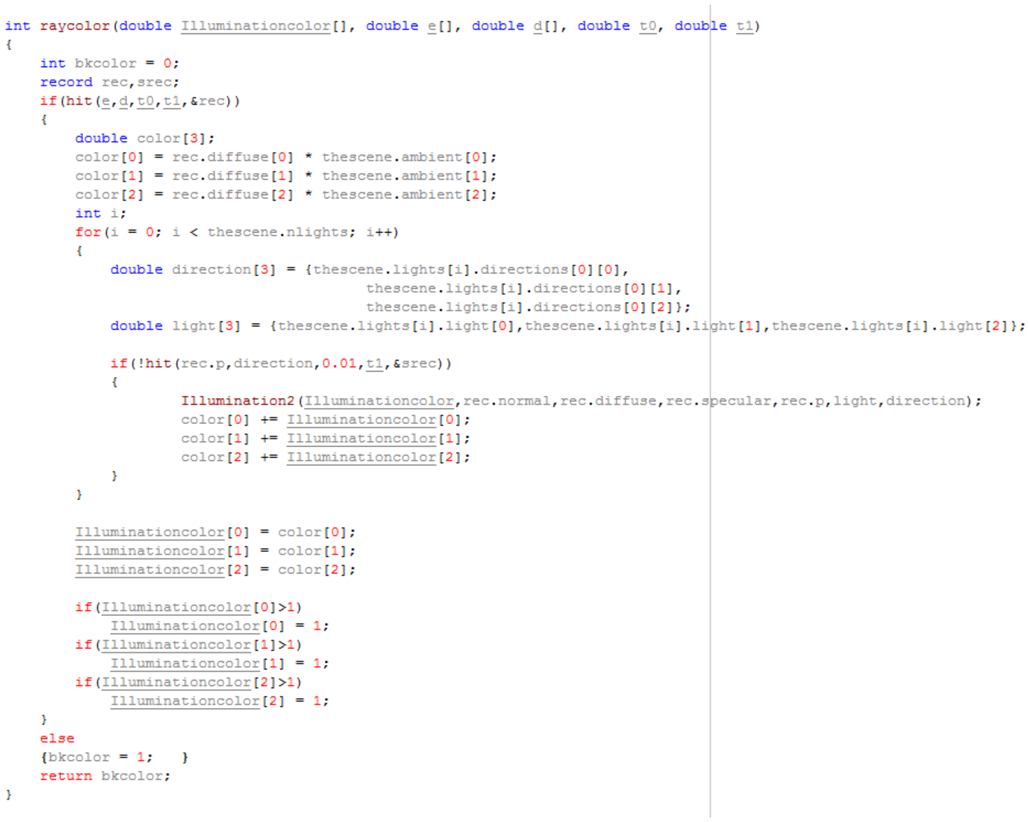
In the lab4.c, I add three new functions compared to the project#2 and project#3.

Firstly, I change the detail implementation in the Illumination function, and the new function is as followed:



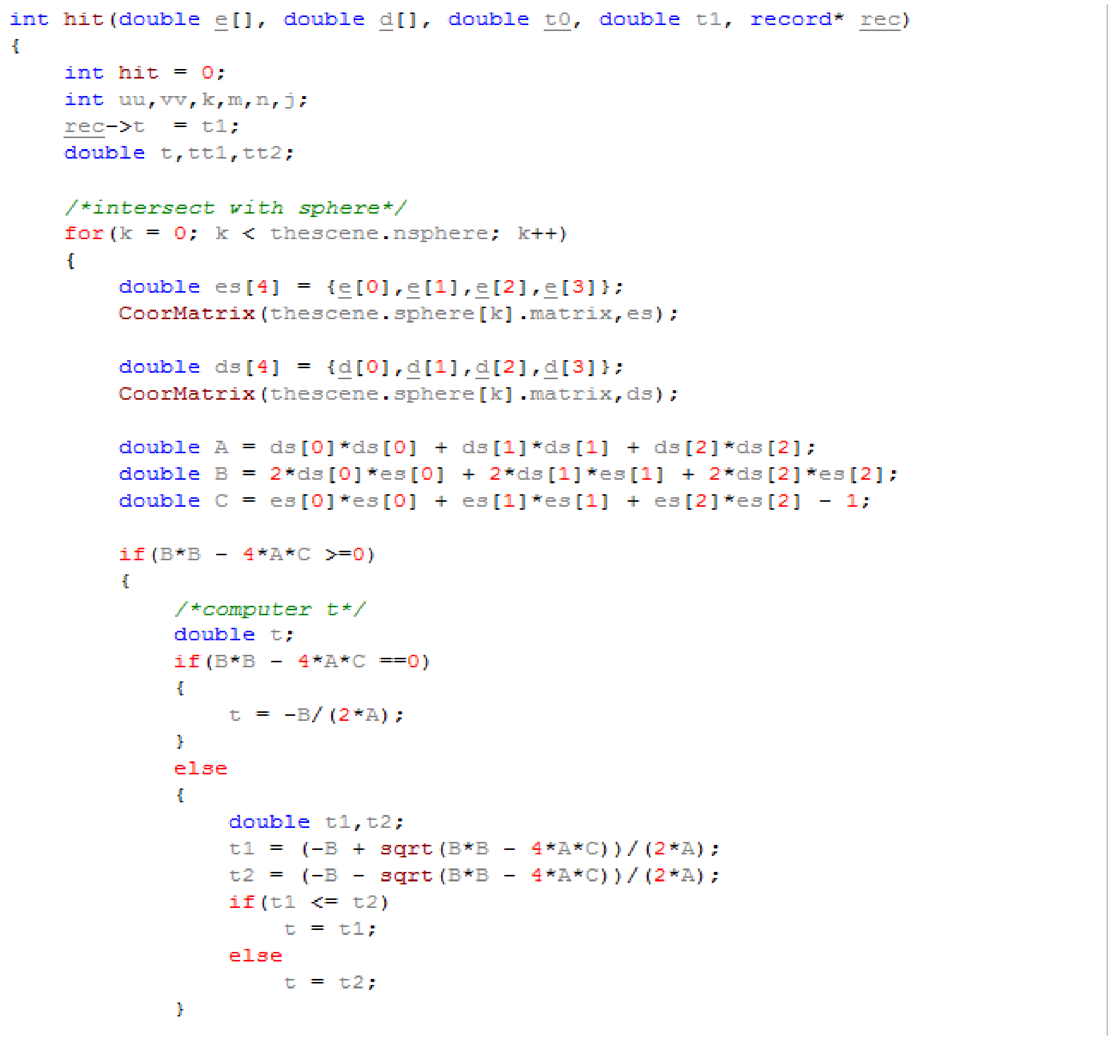
```
void Illumination2(double Illuminationcolor[], double normal[],double diffuse[],double specular[],double illu_point[],double light[],double direction[])
{

    int l,d;
    Illuminationcolor[0] = Illuminationcolor[1] = Illuminationcolor[2] = 0;
    /*l*/
    UnitVector(direction,direction);

    /*n*l*/
    double n_l;
    n_l = normal[0]*direction[0]+normal[1]*direction[1]+normal[2]*direction[2];
    if(n_l < 0){
        n_l = 0;
    }
    double eye[3] = {vcamera.eye.xyzw[0]-illu_point[0],vcamera.eye.xyzw[1]-illu_point[1],vcamera.eye.xyzw[2]-illu_point[2]};
    UnitVector(eye,eye);
    double half[3] = {eye[0]+direction[0],eye[1]+direction[1],eye[2]+direction[2]};
    UnitVector(half,half);
    /*n*h*/
    double n_h;
    n_h = normal[0]*half[0]+normal[1]*half[1]+normal[2]*half[2];
    if(n_h < 0){
        n_h = 0; }
    /*diffuse*/
    Illuminationcolor[0] += diffuse[0] * light[0] * n_l;
    Illuminationcolor[1] += diffuse[1] * light[1] * n_l;
    Illuminationcolor[2] += diffuse[2] * light[2] * n_l;
    /*specular*/
    Illuminationcolor[0] += specular[0] * light[0] * pow(n_h,specular[3]);
    Illuminationcolor[1] += specular[1] * light[1] * pow(n_h,specular[3]);
    Illuminationcolor[2] += specular[2] * light[2] * pow(n_h,specular[3]);
}
```

Another function is raycolor function. It is also same as the PPT:

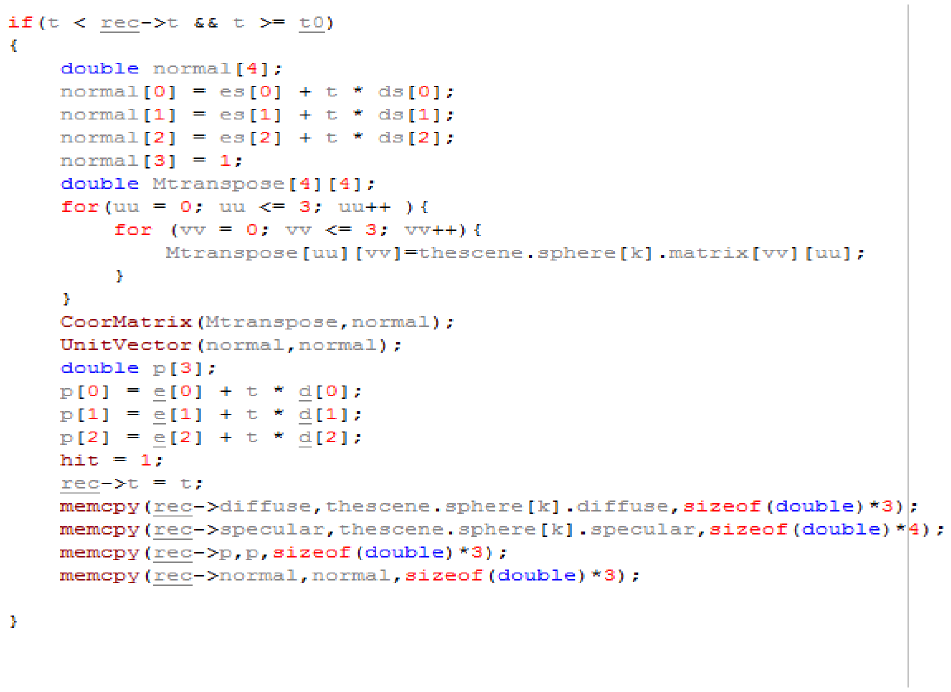


```
int raycolor(double Illuminationcolor[], double e[], double d[], double t0, double t1)
{
    int bkcolor = 0;
    record rec,srec;
    if(hit(e,d,t0,t1,&rec))
    {
        double color[3];
        color[0] = rec.diffuse[0] * thescene.ambient[0];
        color[1] = rec.diffuse[1] * thescene.ambient[1];
        color[2] = rec.diffuse[2] * thescene.ambient[2];
        int i;
        for(i = 0; i < thescene.nlights; i++)
        {
            double direction[3] = {thescene.lights[i].directions[0][0],
                                   thescene.lights[i].directions[0][1],
                                   thescene.lights[i].directions[0][2]};
            double light[3] = {thescene.lights[i].light[0],thescene.lights[i].light[1],thescene.lights[i].light[2]};

            if(!hit(rec.p,direction,0.01,t1,&srec))
            {
                    Illumination2(Illuminationcolor,rec.normal,rec.diffuse,rec.specular,rec.p,light,direction);
                    color[0] += Illuminationcolor[0];
                    color[1] += Illuminationcolor[1];
                    color[2] += Illuminationcolor[2];
            }
        }

        Illuminationcolor[0] = color[0];
        Illuminationcolor[1] = color[1];
        Illuminationcolor[2] = color[2];

        if(Illuminationcolor[0]>1)
            Illuminationcolor[0] = 1;
        if(Illuminationcolor[1]>1)
            Illuminationcolor[1] = 1;
        if(Illuminationcolor[2]>1)
            Illuminationcolor[2] = 1;
    }
    else
    {bkcolor = 1;   }
    return bkcolor;
}
```

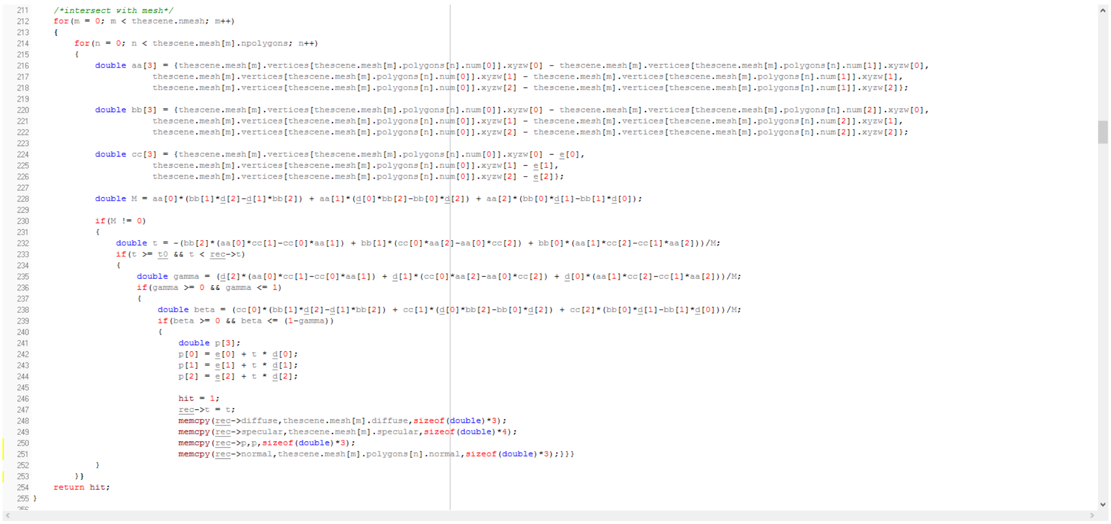
The last function I add is hit function. It was used to get whether intersect with the sphere or the mesh. The code is as followed:



```
int hit(double e[], double d[], double t0, double t1, record* rec)
{
    int hit = 0;
    int uu,vv,k,m,n,j;
    rec->t  = t1;
    double t,tt1,tt2;

    /*intersect with sphere*/
    for(k = 0; k < thescene.nsphere; k++)
    {
        double es[4] = {e[0],e[1],e[2],e[3]};
        CoorMatrix(thescene.sphere[k].matrix,es);

        double ds[4] = {d[0],d[1],d[2],d[3]};
        CoorMatrix(thescene.sphere[k].matrix,ds);

        double A = ds[0]*ds[0] + ds[1]*ds[1] + ds[2]*ds[2];
        double B = 2*ds[0]*es[0] + 2*ds[1]*es[1] + 2*ds[2]*es[2];
        double C = es[0]*es[0] + es[1]*es[1] + es[2]*es[2] - 1;

        if(B*B - 4*A*C >=0)
        {
            /*computer t*/
            double t;
            if(B*B - 4*A*C ==0)
            {
                t = -B/(2*A);
            }
            else
            {
                double t1,t2;
                t1 = (-B + sqrt(B*B - 4*A*C))/(2*A);
                t2 = (-B - sqrt(B*B - 4*A*C))/(2*A);
                if(t1 <= t2)
                    t = t1;
                else
                    t = t2;
            }
```



```
        if(t < rec->t && t >= t0)
        {
            double normal[4];
            normal[0] = es[0] + t * ds[0];
            normal[1] = es[1] + t * ds[1];
            normal[2] = es[2] + t * ds[2];
            normal[3] = 1;
            double Mtranspose[4][4];
            for(uu = 0; uu <= 3; uu++ ){
                for (vv = 0; vv <= 3; vv++){
                    Mtranspose[uu][vv]=thescene.sphere[k].matrix[vv][uu];
                }
            }
            CoorMatrix(Mtranspose,normal);
            UnitVector(normal,normal);
            double p[3];
            p[0] = e[0] + t * d[0];
            p[1] = e[1] + t * d[1];
            p[2] = e[2] + t * d[2];
            hit = 1;
            rec->t = t;
            memcpy(rec->diffuse,thescene.sphere[k].diffuse,sizeof(double)*3);
            memcpy(rec->specular,thescene.sphere[k].specular,sizeof(double)*4);
            memcpy(rec->p,p,sizeof(double)*3);
            memcpy(rec->normal,normal,sizeof(double)*3);

        }
    }
}
```



```
    /*intersect with mesh*/
    for(m = 0; m < thescene.nmesh; m++)
    {
        for(n = 0; n < thescene.mesh[m].npolygons; n++)
        {
            double aa[3] = {thescene.mesh[m].vertices[thescene.mesh[m].polygons[n].num[0]].xyzw[0] - thescene.mesh[m].vertices[thescene.mesh[m].polygons[n].num[1]].xyzw[0],
                    thescene.mesh[m].vertices[thescene.mesh[m].polygons[n].num[0]].xyzw[1] - thescene.mesh[m].vertices[thescene.mesh[m].polygons[n].num[1]].xyzw[1],
                    thescene.mesh[m].vertices[thescene.mesh[m].polygons[n].num[0]].xyzw[2] - thescene.mesh[m].vertices[thescene.mesh[m].polygons[n].num[1]].xyzw[2]};

            double bb[3] = {thescene.mesh[m].vertices[thescene.mesh[m].polygons[n].num[0]].xyzw[0] - thescene.mesh[m].vertices[thescene.mesh[m].polygons[n].num[2]].xyzw[0],
                    thescene.mesh[m].vertices[thescene.mesh[m].polygons[n].num[0]].xyzw[1] - thescene.mesh[m].vertices[thescene.mesh[m].polygons[n].num[2]].xyzw[1],
                    thescene.mesh[m].vertices[thescene.mesh[m].polygons[n].num[0]].xyzw[2] - thescene.mesh[m].vertices[thescene.mesh[m].polygons[n].num[2]].xyzw[2]};

            double cc[3] = {thescene.mesh[m].vertices[thescene.mesh[m].polygons[n].num[0]].xyzw[0] - e[0],
                    thescene.mesh[m].vertices[thescene.mesh[m].polygons[n].num[0]].xyzw[1] - e[1],
                    thescene.mesh[m].vertices[thescene.mesh[m].polygons[n].num[0]].xyzw[2] - e[2]};

            double M = aa[0]*(bb[1]*d[2]-d[1]*bb[2]) + aa[1]*(d[0]*bb[2]-bb[0]*d[2]) + aa[2]*(bb[0]*d[1]-bb[1]*d[0]);

            if(M != 0)
            {
                double t = -(bb[2]*(aa[0]*cc[1]-cc[0]*aa[1]) + bb[1]*(cc[0]*aa[2]-aa[0]*cc[2]) + bb[0]*(aa[1]*cc[2]-cc[1]*aa[2]))/M;
                if(t >= t0 && t < rec->t)
                {
                    double gamma = (d[2]*(aa[0]*cc[1]-cc[0]*aa[1]) + d[1]*(cc[0]*aa[2]-aa[0]*cc[2]) + d[0]*(aa[1]*cc[2]-cc[1]*aa[2]))/M;
                    if(gamma >= 0 && gamma <= 1)
                    {
                        double beta = (cc[0]*(bb[1]*d[2]-d[1]*bb[2]) + cc[1]*(d[0]*bb[2]-bb[0]*d[2]) + cc[2]*(bb[0]*d[1]-bb[1]*d[0]))/M;
                        if(beta >= 0 && beta <= (1-gamma))
                        {
                            double p[3];
                            p[0] = e[0] + t * d[0];
                            p[1] = e[1] + t * d[1];
                            p[2] = e[2] + t * d[2];

                            hit = 1;
                            rec->t = t;
                            memcpy(rec->diffuse,thescene.mesh[m].diffuse,sizeof(double)*3);
                            memcpy(rec->specular,thescene.mesh[m].specular,sizeof(double)*4);
                            memcpy(rec->p,p,sizeof(double)*3);
                            memcpy(rec->normal,thescene.mesh[m].polygons[n].normal,sizeof(double)*3);}}}
            }
        }}
    return hit;
}
```

2: Performance analysis and evaluation:

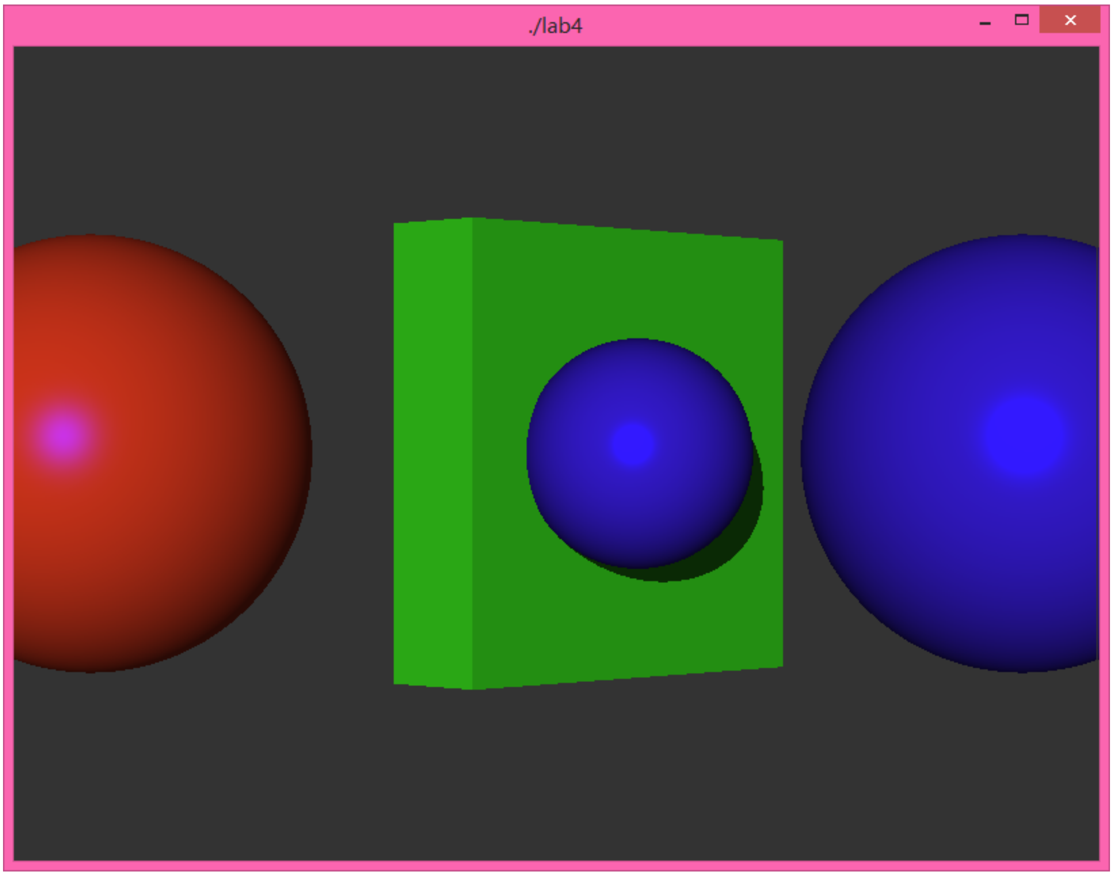
As we know, if there is no highlight, the image the ray tracing method gets is same as flat shading gets. But the ray tracing is more complex and spend much more time. It will calculate the intersections for each triangles while if the number of the triangles are very large, it is not effective.

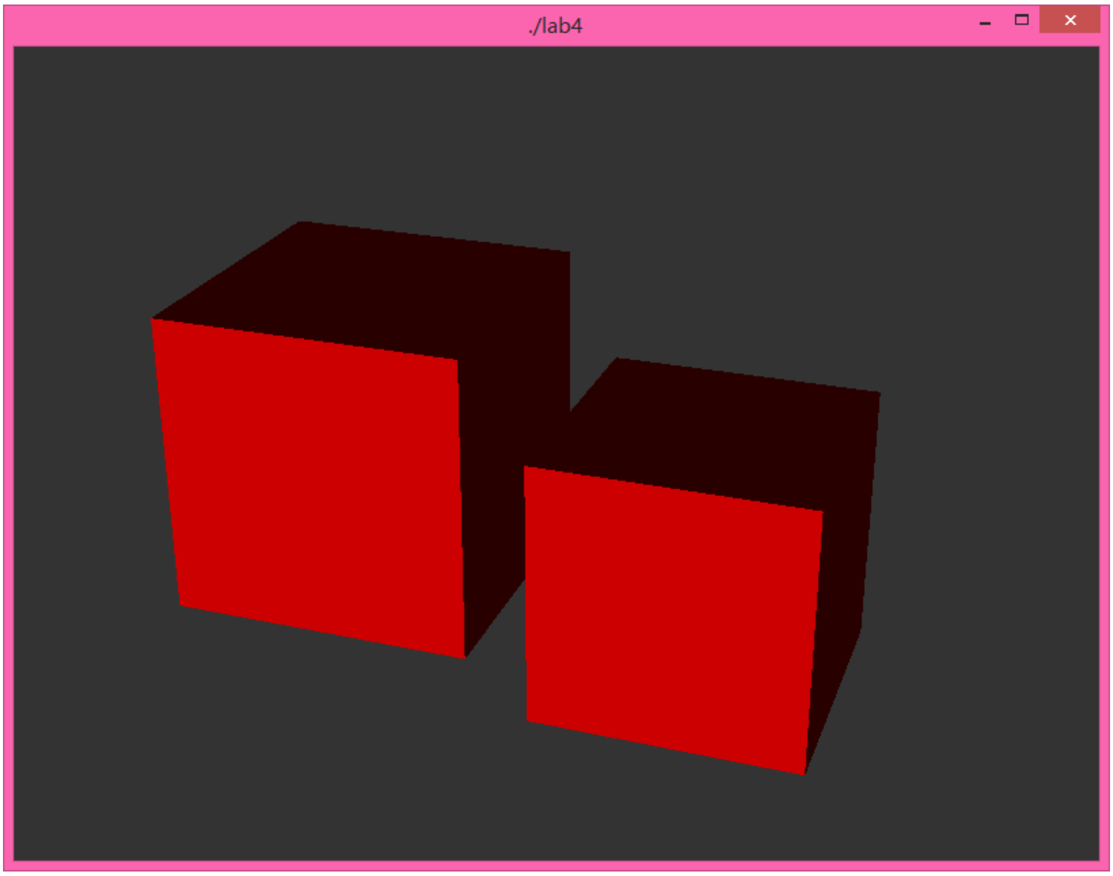
Another ray tracing method is to calculate the intersection with the sphere. It is more convenient than the above method because it only need be calculated for once.

3: Experience

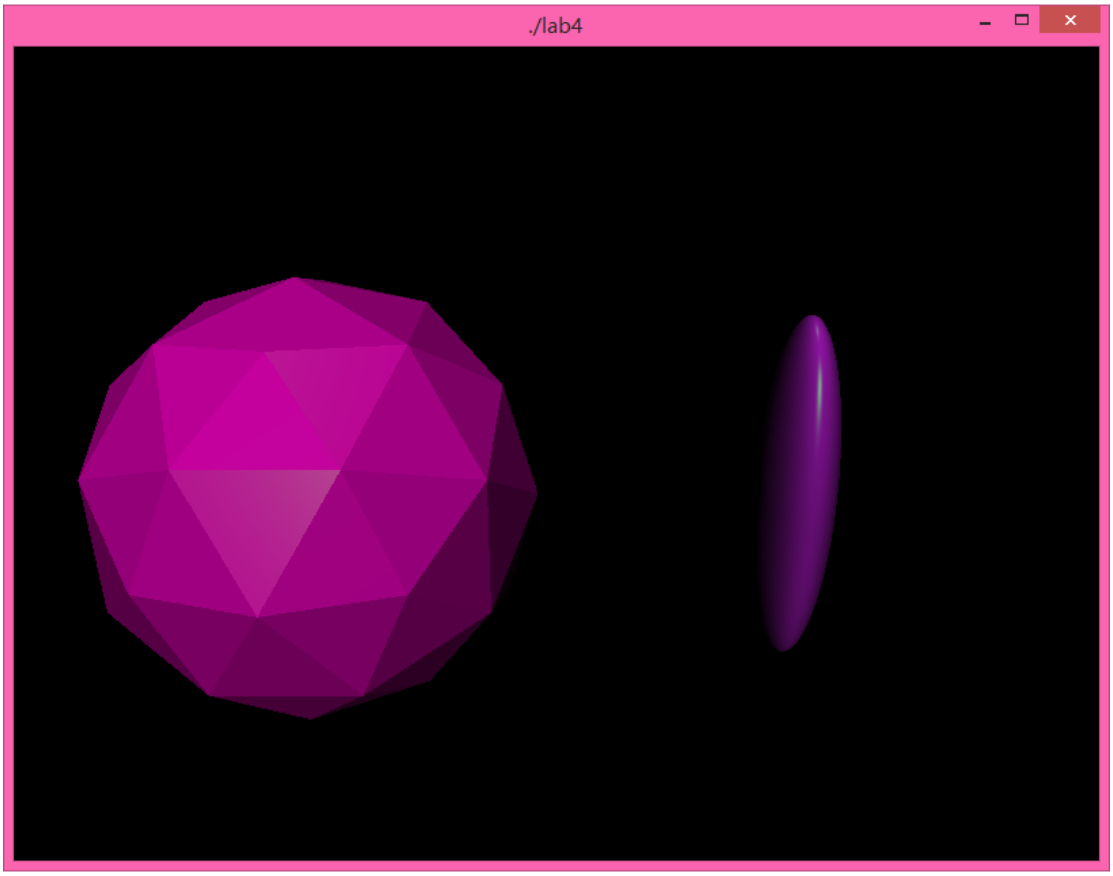
The knowledge of this assignment is all learned from the class while what we need do is only learn about the PPT and improve the functions we have made before and add some new functions which have been taught in class. Compared to the last assignment, it is easier to implement but hard to get the image.

4: Result:





./lab4



./lab4