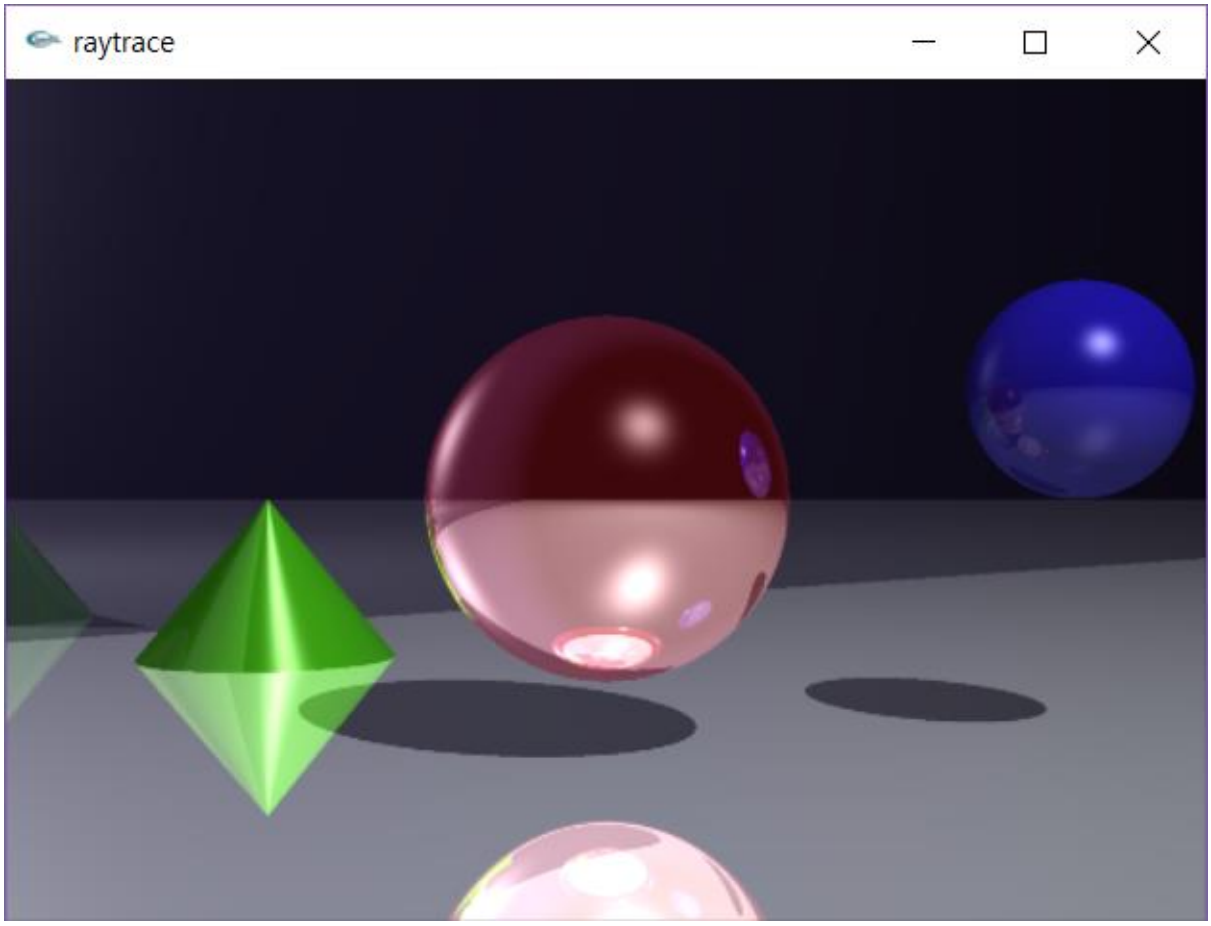
Computer Graphics Final Homework: A Simple Ray Tracer

1642041

이경연



[raytraced image]

- Source files and executable
- Visual studio project file
- Image files to capture the raytraced image (the more, the better)
- Description of your project including the features that you implemented

1. lowerlevel.h : add corn, plane, entire(object) models and lights component struct.

```
/* data structures */

typedef struct point {
  GLfloat x;
  GLfloat y;
  GLfloat z;
  GLfloat w;
} point;


/* a vector is just a point */
typedef point vector;

/* a ray is a start point and a direction */
typedef struct ray {
 point* start;
 vector* end;
} ray;

void normalize(vector* v);

typedef struct material {
 /* color */
 GLfloat r;
 GLfloat g;
 GLfloat b;
 /* ambient reflectivity */
 GLfloat amb;
 /* Diffuse Light */
 GLfloat kd;
 /* Specular Highlights*/
 GLfloat kss;
 /* Specular Highlight ns*/
 GLfloat ns;
 /* Reflection */
 GLfloat ksr;
 /* Refracted Rays */
 GLfloat kr;
} material;

typedef struct color {
 GLfloat r;
 GLfloat g;
 GLfloat b;
 /* these should be between 0 and 1 */
} color;

struct sphere {
 point* c;  /* center */
 GLfloat r;  /* radius */
};

struct plane {
 point* c;
 vector* n;

 plane() {}
 plane(GLfloat x, GLfloat y, GLfloat z, GLfloat xr, GLfloat yr, GLfloat zr) {
          c = new point;
          c->x = x;
```

```
62.             c->y = y;
63.             c->z = z;
64.             c->w = 1;
65.             n = new vector;
66.             n->x = xr;
67.             n->y = yr;
68.             n->z = zr;
69.             n->w = 0;
70.             normalize(n);
71.  }
72.  };
73.
74.
75.
76.  struct cone {
77.  point* c;
78.  vector* n;
79.  GLfloat r;
80.  GLfloat uBound;
81.  GLfloat bBound;
82.  cone() {}
83.  cone(GLfloat x, GLfloat y, GLfloat z, GLfloat xr, GLfloat yr, GLfloat zr, GLfloat daR, GLfloat
     daUBound, GLfloat daBBound) {
84.             c = new point;
85.             c->x = x;
86.             c->y = y;
87.             c->z = z;
88.             c->w = 1;
89.             n = new vector;
90.             n->x = xr;
91.             n->y = yr;
92.             n->z = zr;
93.             n->w = 0;
94.             normalize(n);
95.             r = daR;
96.             uBound = daUBound;
97.             bBound = daBBound;
98.  }
99.  };
100.
101. struct object {
102.  int objectType;
103.  sphere *theSphere;
104.  plane *thePlane;
105.  cone *theCone;
106.  material* m;
107.  object() {}
108.  object(sphere *daSphere, plane *daPlane, cone *daCone, material* daM) {
109.             if (daSphere != NULL) {
110.                       objectType = SPHERE;
111.                       theSphere = daSphere;
112.             }
113.             if (daPlane != NULL) {
114.                       objectType = PLANE;
115.                       thePlane = daPlane;
116.             }
117.
118.             if (daCone != NULL) {
119.                       objectType = CONE;
120.                       theCone = daCone;
121.             }
122.             m = daM;
123.  }
124. };
125.
```

```
126. struct objectNode {
127. object *thisObject;
128. objectNode* next;
129. };
130.
131. class light {
132. public:
133. point *p;
134. color *c;
135. light() {};
136. light(GLfloat daX, GLfloat daY, GLfloat daZ, GLfloat daR, GLfloat daG, GLfloat daB) {
137.           p = new point;
138.           p->x = daX;
139.           p->y = daY;
140.           p->z = daZ;
141.           p->w = 1.0;
142.           c = new color;
143.           c->r = daR;
144.           c->g = daG;
145.           c->b = daB;
146. }
147. };
148.
149. struct lightNode {
150. light *theLight;
151. lightNode* next;
152. lightNode() {};
153. lightNode(lightNode* daNext, GLfloat daX, GLfloat daY, GLfloat daZ, GLfloat daR, GLfloat daG,
     GLfloat daB) {
154.           theLight = new light(daX, daY, daZ, daR, daG, daB);
155.           next = daNext;
156. }
157. };
158.
```

2. shader.cpp

:Make color by shading recursively.

(ambient + diffuse + specular + reflection + refraction)

```
        // ambient component of color
        c->r = m->amb * m->r;
        c->g = m->amb * m->g;
        c->b = m->amb * m->b;

        vector nIn;
        nIn.x = -in->x;
        nIn.y = -in->y;
        nIn.z = -in->z;
        nIn.w = in->w;

        color reflectLight;
        reflectLight.r = 0.0;
        reflectLight.g = 0.0;
        reflectLight.b = 0.0;

        // diffuse
```

```
        lightNode* aLight;
        aLight = lightList->next;
        while (aLight != NULL) {
                calculateDirection(p, aLight->theLight->p, &toLightVector);
                normalize(&toLightVector);
                offsetPoint.x = p->x + toLightVector.x*0.0001;
                offsetPoint.y = p->y + toLightVector.y*0.0001;
                offsetPoint.z = p->z + toLightVector.z*0.0001;
                toLightRay.dir = &toLightVector;
                toLightRay.start = &offsetPoint;
                distToLight = sqrt(pow((offsetPoint.x - aLight->theLight->p->x), 2) +
pow((offsetPoint.y - aLight->theLight->p->y), 2) + pow((offsetPoint.z - aLight-
>theLight->p->z), 2));
                firstHit(&toLightRay, &intersectPoint, &emptyVec, &emptyMat,
&distToIntersect);

                if ((intersectPoint.w == 0.0) || (intersectPoint.w == 1.0 &&
distToIntersect > distToLight)) {
                        numLightDot = dot(&toLightVector, n);
                        if (numLightDot > 0.0) {
                                // Diffuse Lighting
                                c->r += m->kd * aLight->theLight->c->r * numLightDot *
m->r;
                                c->g += m->kd * aLight->theLight->c->g * numLightDot *
m->g;
                                c->b += m->kd * aLight->theLight->c->b * numLightDot *
m->b;
                                // For Specular Highlights (next)
                                vector Hi;
                                Hi.x = toLightVector.x + nIn.x;
                                Hi.y = toLightVector.y + nIn.y;
                                Hi.z = toLightVector.z + nIn.z;
                                normalize(&Hi);
                                numDotHi = dot(n, &Hi);
                                if (numDotHi > 0.0) {
                                        reflectLight.r += pow(numDotHi, m->ns) * aLight-
>theLight->c->r;
                                        reflectLight.g += pow(numDotHi, m->ns) * aLight-
>theLight->c->g;
                                        reflectLight.b += pow(numDotHi, m->ns) * aLight-
>theLight->c->b;
                                }
                        }
                }
                aLight = aLight->next;
        }
```

```
        // specular
        c->r += m->kss * reflectLight.r;
        c->g += m->kss * reflectLight.g;
        c->b += m->kss * reflectLight.b;
```

```
// reflections
vector rv;
normalize(n);
normalize(&nIn);
GLfloat nDotV = 2.0*dot(n, &nIn);
rv.x = (nDotV*n->x);
rv.y = (nDotV*n->y);
rv.z = (nDotV*n->z);
rv.x -= nIn.x;
rv.y -= nIn.y;
rv.z -= nIn.z;
rv.w = 0.0;
normalize(&rv);
offsetPoint.x = p->x + rv.x*0.0001;
offsetPoint.y = p->y + rv.y*0.0001;
offsetPoint.z = p->z + rv.z*0.0001;
offsetPoint.w = 1.0;
ray newRay;
newRay.dir = &rv;
newRay.start = &offsetPoint;
color traceColor;
if (m->ksr > 0.0) {
        traceColor = traceRay(&newRay, depth - 1);
        c->r += m->ksr*traceColor.r;
        c->g += m->ksr*traceColor.g;
        c->b += m->ksr*traceColor.b;
}
```

```
// refractions
offsetPoint.x = p->x + in->x*0.0001;
offsetPoint.y = p->y + in->y*0.0001;
offsetPoint.z = p->z + in->z*0.0001;
newRay.dir = in;
newRay.start = &offsetPoint;
if (m->kr > 0.0) {
        traceColor = traceRay(&newRay, --depth);
        c->r += m->kr*traceColor.r;
        c->g += m->kr*traceColor.g;
        c->b += m->kr*traceColor.b;
}
```

3. Tracer.cpp : normalize vectors and control rays (control intersects according to model)

```
void trace(ray* r, point* p, vector* n, material* *m, objectNode* objectList ,double*
overallT) {
        double t = 0;     /* parameter value at first hit */
        *overallT = -1.0;
        int hit = FALSE;
        int newHit = FALSE;
        objectNode* otherObject;
        otherObject = objectList->next;
        while (otherObject != NULL) {
                switch (otherObject->thisObject->objectType) {
                case SPHERE:
                        newHit = raySphereIntersect(r, otherObject->thisObject->theSphere,
&t);
                        break;
                case PLANE:
                        newHit = rayPlaneIntersect(r, otherObject->thisObject->thePlane,
&t);
                        break;
                case CONE:
                        newHit = rayConeIntersect(r, otherObject->thisObject->theCone, &t);
                        break;
                        }
                if (newHit) {
                        if (t < *overallT || *overallT == -1.0) {
                                findPointOnRay(r, t, p);
                                switch (otherObject->thisObject->objectType) {
                                case SPHERE:
                                        findSphereNormal(otherObject->thisObject-
>theSphere, p, n);
                                        break;
                                case PLANE:
                                        findPlaneNormal(otherObject->thisObject->thePlane,
p, n);
                                        break;
                                case CONE:
                                        findConeNormal(otherObject->thisObject->theCone,
p, n);
                                        break;
                                }
                                        *m = otherObject->thisObject->m;
                                newHit = FALSE;
                                *overallT = t;
                        }
                        hit = TRUE;
                }
                otherObject = otherObject->next;
        }

        if (!hit) p->w = 0.0;
}
```

4. raytrace.cpp (main) : drawing 2 spheres, 1 cone, 2 plane.

```
void initScene() {
        objectList = new objectNode;
        objectList->next = NULL;

        //Original Sphere
        objectNode* tempObject = new objectNode;
        tempObject = new objectNode;
        tempObject->next = objectList->next;
        sphere* s1 = makeSphere(0.0, 0.0, -2.0, 0.25);
        object* o = new object(s1, NULL, NULL, makeMaterial(0.8, 0.1, 0.15, 0.3, 0.0, 1.0,
100.0, 1.0, 0.0));
        tempObject->thisObject = o;
        objectList->next = tempObject;


        //Corn
        tempObject = new objectNode;
        tempObject->next = objectList->next;
        cone* con = new cone(-0.75, 0.0, -3.2, 0.0, 0.0, 0.0, 0.8, 0.0, 1.0);
        o = new object(NULL, NULL, con, makeMaterial(0.3, 0.8, 0.1, 0.3, 0.8, 1.0, 100.0,
0.2, 0.0));
        tempObject->thisObject = o;
        objectList->next = tempObject;


        //Second Sphere
        tempObject = new objectNode;
        tempObject->next = objectList->next;
        s1 = makeSphere(0.65, 0.15, -2.0, 0.15);
        o = new object(s1, NULL, NULL, makeMaterial(0.15, 0.1, 0.8, 0.3, 0.8, 1.0, 100.0,
0.2, 0.0));
        tempObject->thisObject = o;
        objectList->next = tempObject;


        //plane bottom
        tempObject = new objectNode;
        tempObject->next = objectList->next;
        plane* p = new plane(0.0, -0.35, -1.0, 0.0, 1.0, 0.0);
        o = new object(NULL, p, NULL, makeMaterial(0.6, 0.62, 0.62, 0.3, 1.0, 1.0, 50.0,
0.8, 0.0));
        tempObject->thisObject = o;
        objectList->next = tempObject;

        //plane top
        tempObject = new objectNode;
        tempObject->next = objectList->next;
        p = new plane(0.0, 0.0, -5.0, 1.0, 0.0, 1.0);
        o = new object(NULL, p, NULL, makeMaterial(0.15, 0.12, 0.26, 0.0, 1.0, 1.0, 50.0,
0.5, 0.0));
        tempObject->thisObject = o;
        objectList->next = tempObject;
```

```
        //lights nodes
        lightList = new lightNode;
        lightList->next = NULL;

        lightNode* tempLight = new lightNode(lightList->next,
                0.0, 0.0, -2.0,
                0.7, 0.7, 0.7);
        lightList->next = tempLight;

        tempLight = new lightNode(lightList->next,
                1.0, 2.0, 0.0,
                0.6, 0.6, 0.6);
        lightList->next = tempLight;
}
```