Report for Assignment 2

Basic Ray Tracer

Jiajia Fan ---2016/11/13
fanjj@shanghaitech.edu.cn

Structure

The program is designed to create a Ray Tracer, creating image with several 3d objects using Phong lighting model. It creates camera to control what the image shows, and a renderer to set every pixel color. Here the program use QT library.

Class Name	Purpose
raytracer	Create a window and set the buttons
SceneRenderer	Create things needed in a scene and render the image
Camera	Determine the image plane and objects and light sources
Ray	Hit objects and determines the color
LightSource	Lighting in the scene
AbObject	Abstract Object class, deriving several kinds of object

The core class for creating the image is SceneRenderer.

Detail in Core Class

SceneRenderer

structure of vertexes in Bezier:

```
typedef struct { GLfloat x, y, z; }MyVertex;
```

Variant

```
private:
    // members in a scene
    Camera*                 s_camera_;
    QVector<LightSource*>*  s_light_list_;
    QVector<AbObject*>*     s_object_list_;
    QVector3D               s_amibentcolor_;
```

Functions

```
    // obtain the things in my scene
    Camera* SceneCamera(){ return s_camera_; }
    QVector<LightSource*>* SceneLightList(){ return s_light_list_; }
    QVector<AbObject*>* SceneObjectList(){ return s_object_list_; }

    // rendering operations
    QVector3D GetColor(double in_x, double in_y);
```

```
void initScene();
// intersections
bool RayHitObject(Ray in_ray, IntersectPoint& out_point);
bool isShadow();

// check color range
void validColor(QVector3D& in_color);
```

Core Codes

Sampling:

```
void RayTracer::RenderImage(){
    for (int i = 0; i < m_imagesize_.width(); i++)
    {
        for (int j = 0; j < m_imagesize_.height(); j++)
        {
            QVector3D color_vector(0, 0, 0);
            if (is_antialiasing_)
            {

                for (double sample_i = -0.5; sample_i < 0.501; sample_i += 0.5)
                {
                    for (double sample_j = -0.5; sample_j < 0.501; sample_j += 0.5)
                    {
                        color_vector += m_scenrenderer_->GetColor(i + sample_i, j + sample_j);
                    }
                }
                color_vector /= 9;
            }
            else
            {
                color_vector = m_scenrenderer_->GetColor(i, j);
            }

            m_image_->setPixel(i, j, QColor(color_vector.x(), color_vector.y(), color_vector.z()).rgb());
        }
    }
```

Ray tracing:

```cpp
bool SceneRenderer::RayHitObject(Ray in_ray, IntersectPoint& out_point){
    bool ishit = false;
    QVector3D ray_origin = in_ray.OriginPoint();
    double out_dist = MAX_FLOAT_VALUE;
    IntersectPoint temp_out;
    // check each object for intersection, and find the closed
    for (int ob_num = 0; ob_num < s_object_list_->size(); ob_num++)
    {
        AbObject* cur_ob = s_object_list_->at(ob_num);
        if (cur_ob->RayHitTest(in_ray, temp_out))
        {
            ishit = true;
            double temp_dist = (temp_out.out_point - ray_origin).length();
            if (temp_dist < out_dist)
            {
                out_dist = temp_dist;
                out_point = temp_out;
            }
        }
    }
    // check the light
    if (s_light_list_->at(0)->RayHitTest(in_ray))
    {
        double dist = (s_light_list_->at(0)->LightCenter() - in_ray.OriginPoint()).length();
        if (dist < out_dist)
        {
            out_point.isLight = true;
        }
    }

    return ishit;
}
```

Shadow Ray & Lighting Model:

```cpp
QVector3D SceneRenderer::GetColor(double in_x, double in_y){
    QVector3D out_color(0, 0, 0);
    Ray* in_ray = s_camera_->GetRay(in_x, in_y);
    IntersectPoint hit_point;
    // when hit one object, set the color
    bool ishit = RayHitObject((*in_ray), hit_point);
    if (ishit)
    {
        if (hit_point.isLight)
        {
            out_color = QVector3D(255, 255, 255);
        }
```

```
        else
        {
            // todo: add Phong model
            QVector3D light_dir = s_light_list_->at(0)->LightCenter() - hit_point.out_point;
            Ray shadow_ray(hit_point.out_point, light_dir);
            IntersectPoint temp_p;
            bool is_shadow = RayHitObject(shadow_ray, temp_p);
            if (!is_shadow || temp_p.isLight)
            {
                light_dir.normalize();
                float diff_ratio = QVector3D::dotProduct(light_dir, hit_point.out_normal);
                //out_color += 0.2 * s_amibentcolor_;
                out_color += 0.1 * hit_point.out_color;
                if (diff_ratio > 1e-2)
                {
                    out_color += diff_ratio * hit_point.out_color;
                }
                validColor(out_color);
            }
        }
    }

    return out_color;
}
```

Algorithms

The process of ray-tracing can be divided into 3 parts listing below:

1. Sampling—anti aliasing:

   Using 3*3 samples for every pixel.

2. Shooting Ray:

   When a ray hits an object, get the intersection (point and normal). If it is a light source, make it color of light. Then shoot a ray from the point towards light source(s), and if it hits some object else, turn it into shadow.

3. Calculate the color of each pixel:

   Using Phong Model to determine the color-

$$I_\mathrm{p} = k_\mathrm{a} i_\mathrm{a} + \sum_{m \in \text{ lights}} (k_\mathrm{d} (\hat{L}_m \cdot \hat{N}) i_{m,\mathrm{d}} + k_\mathrm{s} (\hat{R}_m \cdot \hat{V})^\alpha i_{m,\mathrm{s}})$$

Example Images