Assignement Two: A Simple Ray Tracer

Instructor: David Gu TA: Yuyao Lin
Due Date: 11:59pm March 30th, 2014

Computer Science Department,
Stony Brook University

1 Assignment Objectives

This assignment is designed to make students get familiar with the principles of ray tracing, and fundamental concepts for photo-realistic rendering. In details:

- Geometric intersection between ray and meshes.
- Diffuse rendering
- Phong Shading
- Reflection
- Shadow detection
- Vertex Color (Extra Credit)
- Texture Mapping (Extra Credit)
- Refraction (Extra Credit)
- Cube map (Extra Credit)

2 Geometric Intersection

Given a ray, with starting point $\mathbf{p}$ and direction $\mathbf{d}$,

$$\mathbf{r}(t) = \mathbf{p} + t\mathbf{d}.$$

Given a triangle $[v_0, v_1, v_2]$, the normal of the plane is given by

$$\mathbf{n} = \frac{(v_1 - v_0) \times (v_2 - v_0)}{|(v_1 - v_0) \times (v_2 - v_0)|}.$$

The plane determined by the triangle is

$$\langle \mathbf{q} - v_0, \mathbf{n} \rangle = 0.$$

From $\langle \mathbf{r}(t) - v_0, \mathbf{n} \rangle = 0$, we get

$$t\langle \mathbf{d}, \mathbf{n} \rangle = \langle v_0 - \mathbf{p}, \mathbf{n} \rangle.$$

- $\langle \mathbf{d}, \mathbf{n} \rangle = 0$, if $\langle v_0 - \mathbf{p}, \mathbf{n} \rangle = 0$, then the ray is on the plane, otherwise the ray is parallel to the plane, and never intersects the plane.

– $\langle \mathbf{d}, \mathbf{n} \rangle \neq 0$, then the intersection time is

$$t = \frac{\langle v_0 - \mathbf{p}, \mathbf{n} \rangle}{\langle \mathbf{d}, \mathbf{n} \rangle}.$$

Suppose the intersection point between the ray and the plane is $\mathbf{p}$, then we need to test if $\mathbf{p}$ is inside the triangle. The bary-centric coordinates of $\mathbf{p}$ with respect to v_0, v_1, v_2 is given by

$$\begin{aligned} \alpha_0 &= \tfrac{1}{2A} \langle (v_1 - \mathbf{p}) \times (v_2 - \mathbf{p}), \mathbf{n} \rangle \\ \alpha_1 &= \tfrac{1}{2A} \langle (v_2 - \mathbf{p}) \times (v_0 - \mathbf{p}), \mathbf{n} \rangle \\ \alpha_0 &= \tfrac{1}{2A} \langle (v_0 - \mathbf{p}) \times (v_1 - \mathbf{p}), \mathbf{n} \rangle \end{aligned}$$

where A is the area of the face

$$A = \frac{1}{2} \langle (v_1 - v_0) \times (v_2 - v_0), \mathbf{n} \rangle$$

If one of $(\alpha_0, \alpha_1, \alpha_2)$ is negative, then $\mathbf{p}$ is outside the triangle.

3 Illumination and Shading Model

In this assignment, we use Whitted's illumination equation

$$I_\lambda = I_{a\lambda} k_a O_{d\lambda} + \sum_i S_i I^i_{p\lambda} [k_d O_{d\lambda} \langle \mathbf{n}, \mathbf{l}_i \rangle + k_s \langle \mathbf{v}, \mathbf{r}_i \rangle^n] + k_s I_{r\lambda} + k_t I_{t\lambda}$$

1. Ambient Light The ambient light is given by

$$I_{a\lambda} k_a O_{d\lambda}$$

where

- k_a is the ambient reflection coefficient of the material
- O_d is the color of the material (could be vertex color or texture color)
- I_a is the global background color
- λ is the index of color channel $0, 1, 2$ representing red, green and blue.

2. Diffuse shading The diffuse shading component is given by

$$\sum_i S_i I^i_{p\lambda} [k_d O_{d\lambda} \langle \mathbf{n}, \mathbf{l}_i \rangle]$$

where

- i is the index of all point light sources
- S_i is the visibility. If the ray connecting the point p, and i-the light source center intersects any object, then p is in the shadow, S_i is 0. Otherwise, p can be lit by the i-th light source, S_i is 1.
- I^i_p is the intensity of the i-th light source
- $\mathbf{n}$ is the surface normal at the intersection point p
- $\mathbf{l}_i$ is the i-th light direction

3. Secularity The secularity shading component is given by

$$\sum_i S_i I^i_{p\lambda}[k_s \langle \mathbf{v}, \mathbf{r}_i \rangle^n]$$

where

- k_s is the specular reflectance coefficient
- $\mathbf{v}$ is the view direction
- $\mathbf{r}_i$ is the reflection direction of $\mathbf{l}_i$
- n is the power coefficient

4. Reflection The term

$$k_s I_r$$

gives the reflection component, where I_r is the color of the reflection ray.

5. Refraction The term

$$k_t I_t$$

gives the refraction component, where I_t is the color of the refraction ray.

4 Tast 1. Geometric Intersection

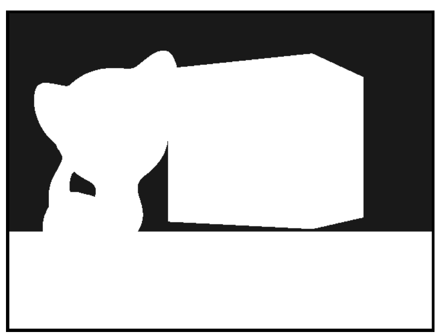

Fig. 1. Result of tast 1

Step 1. Modify the function in intersection.h

```
template<typename M>
bool __intersect_face(const CRay & ray, typename M::CFace * pF,
                           CIntersection & intersection )
```

1. Test if the current face is visible or not, if not return false.
2. Compute the intersection between the ray and the plane determined by the face q. If the intersection time is negative, return false.
3. Compute the bary-centric coordinates of q with respect to three vertices of the face
4. Test if q is inside the triangle, if bary-centric coordinates have negative component, return false.
5. Use bary-centric coordinates to linearly interpolate normal, color rgb, texture coordinates uv, record these information in the intersection object, also record the intersection time.
6. return true.

Step 2. Modify the function in raytracer.h

```
template<typename M>
CPoint CRayTracer<M>::__trace(const CRay & ray, const int &depth,
                              CShape<M> ** intersection_shape )
{

    CIntersection intersection;

    /*!
     *  find the nearest intersecting object
     */
    CShape<M>* intersect_shape  = NULL;

    intersect_shape = __intersection( ray, intersection );
    (*intersection_shape) = intersect_shape;

    // if there's no intersection return black or background color
    if (! intersect_shape )
    {
        return m_background_color;
    }
    //else return white color
    return CPoint(1,1,1);
}
```

Step 3. In the function test_phong() in test_cases.h, set the tracing depth to be 0,

```
    raytracer.render(0)
```

5 Task 2. Diffuse Shading

Step 1. Modify the function to compute the local color in raytracer.h

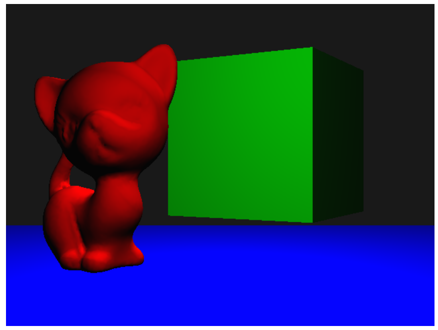

Fig. 2. Result of tast 2

```
template<typename M> CPoint CRayTracer<M>::__local_color( CShape<M>*
pShape, const CIntersection & intersection, const CRay & view_ray)
```

This function returns the summation of ambient light and the diffusion component.

6 Task 3. Specular Shading

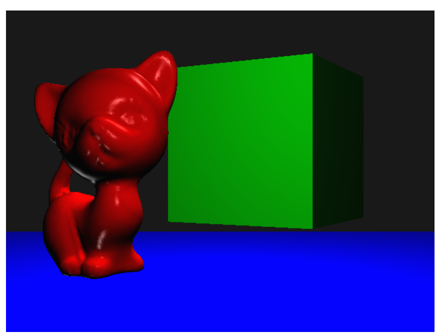

Fig. 3. Result of tast 3

Step 1. Modify the function to compute the local color in raytracer.h

```
template<typename M> CPoint CRayTracer<M>::__local_color( CShape<M>*
pShape, const CIntersection & intersection, const CRay & view_ray)
```

This function returns the summation of ambient light, the diffusion component and the specularities.

7 Task 4. Shadow

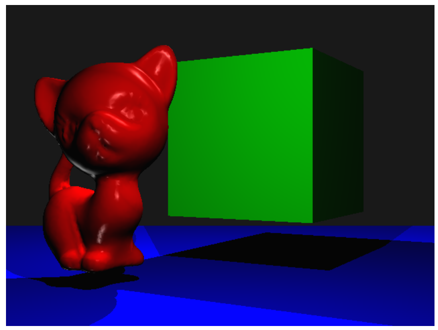

Fig. 4. Result of tast 4

Step 1. Modify the shadow testing function in raytracer.h

```
template<typename M>
bool CRayTracer<M>::_in_shadow( const CShape<M> * pShape,
const CIntersection & intersection, const CPoint light_source )
```

If the ray from the intersection point to the light source intersects any object other than the current one, return true.

Step 2. Modify the function to compute the local color in raytracer.h

```
template<typename M> CPoint CRayTracer<M>::__local_color( CShape<M>*
pShape, const CIntersection & intersection, const CRay & view_ray)
```

For each light source, detect if the current intersection is under the shadow.

8 Task 5. Reflection

step 1. Compute the reflection ray,

```
template<typename M>
CRay CRayTracer<M>::__reflection_ray( const CRay & ray,
const CIntersection & intersection )
```

step 2. Set the tracing depth to be 1 in test_phong(),

```
raytracer.render(1);
```

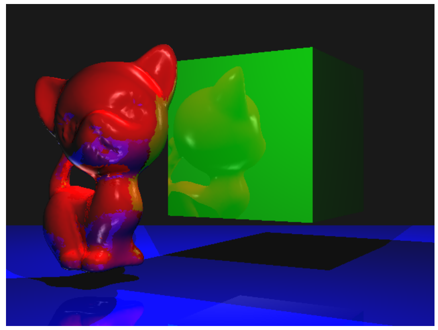

Fig. 5. Result of tast 5

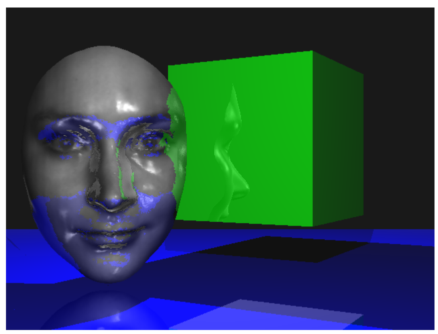

Fig. 6. Result of vertex color

9 Extra Credits

Task 1. Vertex Color In _local_color function,

```
template<typename M> CPoint CRayTracer<M>::__local_color( CShape<M>*
pShape, const CIntersection & intersection, const CRay & view_ray)
```

use the color in the intersection. In the main() function, call test_vertex_color().

Task 2. Texture Mapping In local color function,

```
template<typename M> CPoint CRayTracer<M>::__local_color( CShape<M>*
pShape, const CIntersection & intersection, const CRay & view_ray)
```

use the color from the texture image, you can directly call

```
CPoint _bilinear_interpolation( RgbImage * pImg, CPoint2 uv )
```

defined in utilities.h. In the main() function, call test_texture().

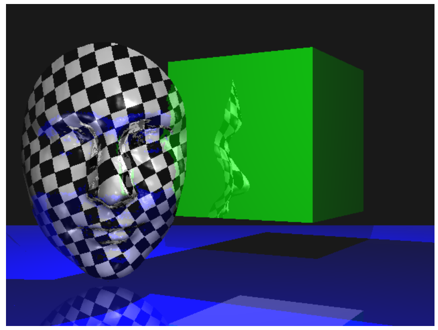

Fig. 7. Result of texture mapping.

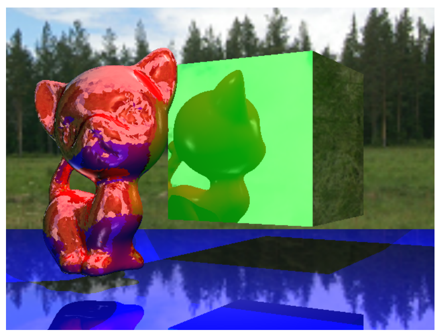

Fig. 8. Result of cubemap.

Task 3. Cube Map Implement the function

```
template<typename M>
CPoint CCubeMap<M>::getColor( const CRay & ray ).
```

In function

```
CPoint CRaytracer::__trace( const CRay & ray, const int & depth,
CShape<M>** intersection_shape );
```

if a ray r doesn't intersect any object, call

```
pCubemap->getColor( r ).
```

In the testing function test_cubempa().

Task 4. Refraction Implement the function

```
template<typename M>
CRay CRayTracer<M>::__refraction_ray( const CRay & ray,
const CIntersection & intersection,bool & total_internal_reflection )
```

Fig. 9. Result of cubemap.

In function

```
template<typename M>
bool __intersect_face(const CRay & ray, typename M::CFace * pF,
CIntersection & intersection )
```

disable back face culling.

In the main() function, call test_refraction().

10 Coding Environment

The solution code is compiled on windows platform using Visual Studio. You can compile, debug the assignment in the same environment. The student computer lab has windows machines with Visual Studio IDE.

You can use kitten_simplified.m and smaller image size for the testing purposes.

11 Submission Requirements

You need to submit the followings: source code with project file or makefile; the images you generated; Detailed Readme file,

- Name, ID, email address
- Explain how to use your solution code
- Source codes
- Project files
- Explain your algorithm for each requirement.

Your source code will be examined in details, compiled, and executed in the grading process. If you have further questions, please contact the instructor David Gu, gu@cs.stonybrook.edu, or the TA Yuyao Lin, yuylin@cs.stonybrook.edu. If you need extension, please email to the instructor as well.