CG Project 1: Implementation of Global Illumination Model

Yongxi Huang 119033910011

1 Project Overview

1.1 Description

Implement an algorithm for global illumination models:

1) Ambient, diffuse, specula high light and shadows must be taken into account.
2) Transparency should be included. Refraction is optional.
3) Polyhedral and spherical objects should be included.
4) Existing graphics packages or libraries, e.g., OpenGL or D3D, are encouraged to be used, but only functions, like primitive drawing, vertices array, matrix operation and color feature, are allowed to be used in your work.

1.2 Result

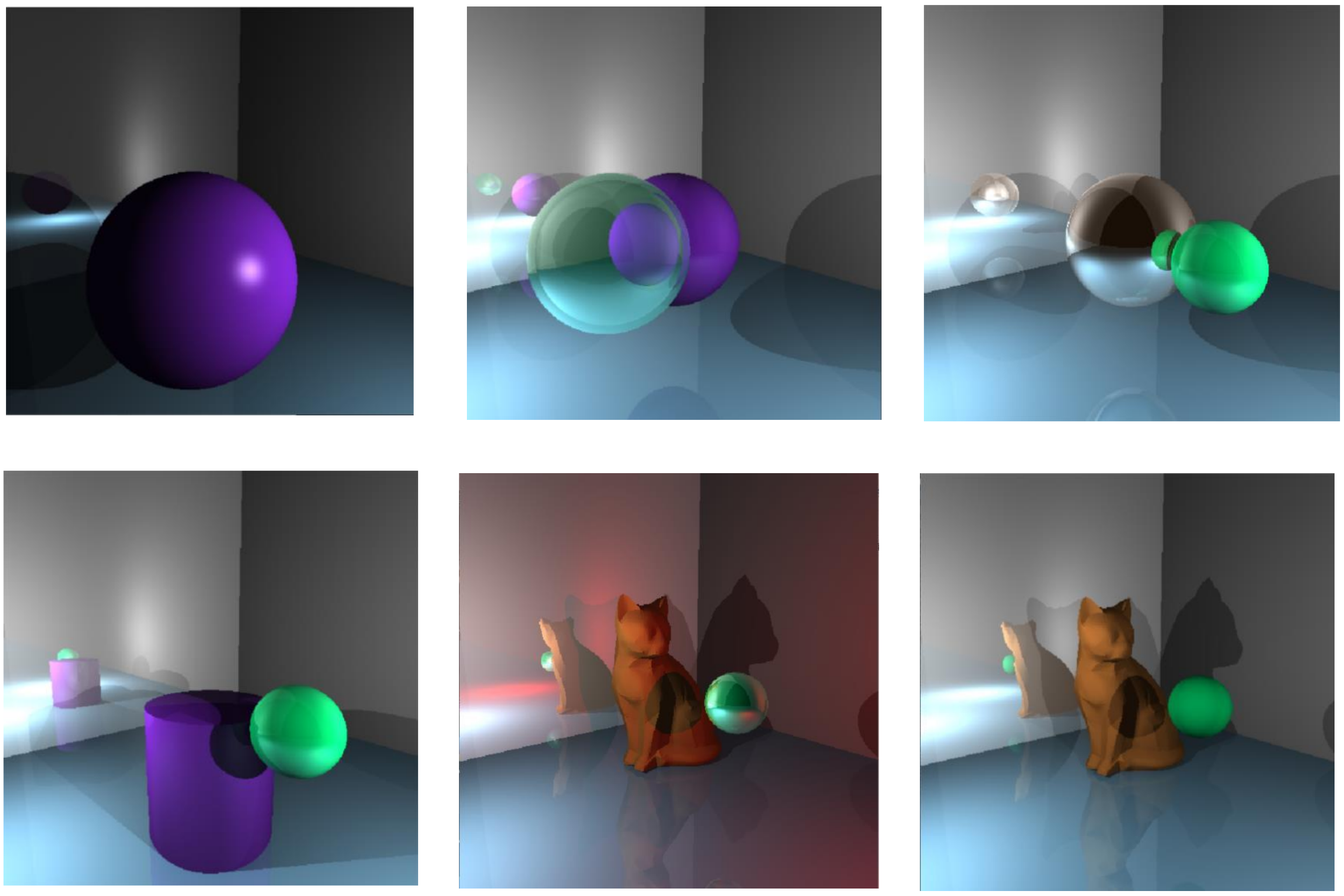


Fig 1. Samples of Results

1) A ray tracing based global illumination model is implemented. Both ambient,

diffuse, specula high light and shadows are taken into account. Besides, specular reflection is also implemented based on the feature of ray tracing.

2) Both transparency and refraction is included.
3) Support rendering for some basic geometric objects and mesh model.
4) Environment: Codes are in pure c++. OpenGL is used to draw pixels on window.

Several samples are shown in Fig1.

2 Design

2.1 Recursive Backward Ray-Tracing Algorithm

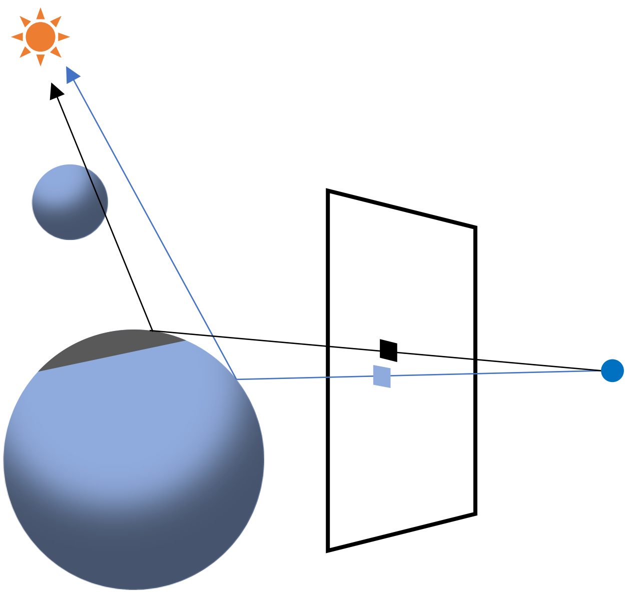


Fig 2. Ray Tracing

Ray tracing algorithm renders the whole scene by tracing rays from sight view. The Basic process of ray tracing is:

1) For each pixel, a sight ray is generated from the view point.
2) Trace that ray until it reaches some objects.
3) For each light source, generate a shadow ray pointing to it from the intersection. The color at the intersection is determined by the "visible" light sources according to the shadow ray.
4) If the material at the intersection is transparent or reflective, a refraction ray or reflection ray would be further generated and traced. Back to process 2） until the maximum tracing depth reaches or the ray is intersected with a diffuse

object.

The color of the pixel is determined by the ray traced back to this pixel. That is the reason why this algorithm is a recursive and backward algorithm. Based on ray tracing algorithm, an illumination model supporting diffuse reflection, specular reflection, specular high light, and refraction can be implemented.

2.2 Diffuse Reflection

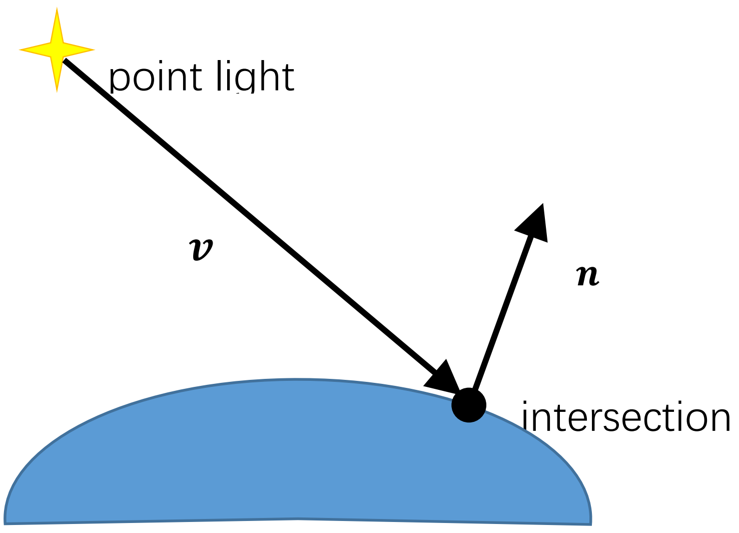


Fig 3. Diffuse Reflection

Color of diffuse reflection is computed when the ray intersects with an object. As show in Fig.3, the intensity of diffuse color is computed based on the light vector $\boldsymbol{v}$ (also works as shadow ray) and surface norm $\boldsymbol{n}$. The equation for diffuse reflection computation is

$$I_{diffuse} = I_{light} \cdot \frac{\vec{v}}{|\vec{v}|} \cdot \vec{n} \cdot \frac{1}{|\vec{v}|^2}$$

Note that not only the angle between light vector and norm vector, but the distance attenuation is taken into account as well.

And the color from ambient light is also computed here. The color traced back with ray is computed by following equation where k is the diffuse coefficient of the object's material. Fig.4 shows the render result of diffuse only object.

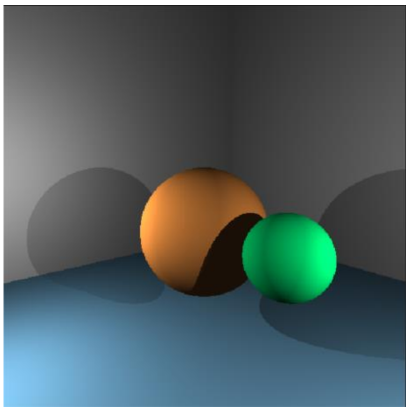


Fig 4. Diffuse Result

$$color = k(color_{light} + color_{object}) \times I_{diffuse} + k(color_{ambient} + color_{object}) \times I_{ambient}$$

2.3 Specular Reflection

If the object is reflective, we can implement specular reflecting by create a reflected ray from intersection and do ray tracing recursively. Based on the normalized incoming ray L and surface norm N, the reflected ray R can be computed by

$$\vec{X} = 2(\vec{L} \cdot \vec{N})\vec{N}$$

$$\vec{R} = \vec{X} - \vec{L}$$

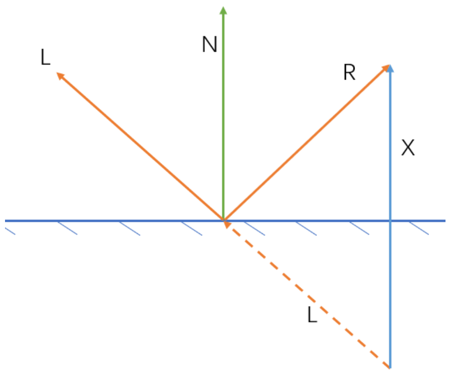


Fig 5. Reflected Ray

The reflected ray will be further traced and the color from the reflected ray will be traced back. We multiply the color from reflected ray by the material's specular reflection coefficient and add it to the color traced back by the incoming ray.

Fig.6 shows some results of specular reflection. Note that the diffuse coefficient of the right large sphere is set to be 0 so that it looks like a mirror.

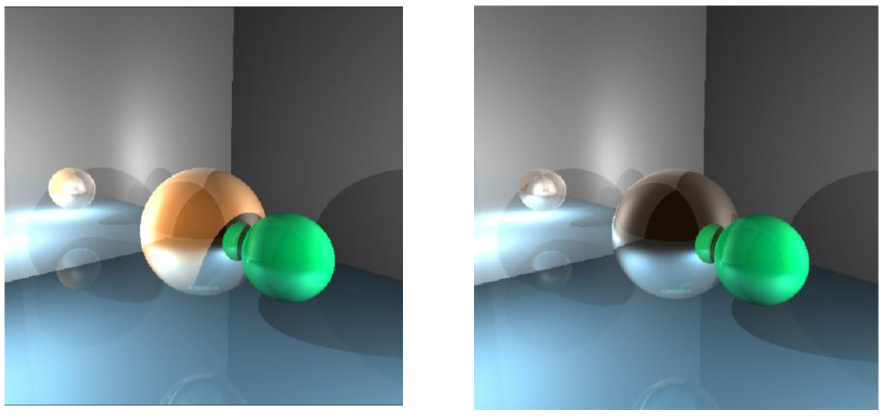


Fig 6. Specular Reflection Results

2.4 Transparency and Refraction

Transparency and refraction is implemented just like reflection. If the object is transparent, an refract ray would be generated and traced recursively. The refract ray is computed according to:

$$T = \left(\frac{\eta_L}{\eta_T} N \cdot L - \sqrt{1 - \frac{\eta_L^2}{\eta_T^2}[1 - (N \cdot L)^2]} \right) N - \frac{\eta_L}{\eta_T} L$$

Fig.7 shows some results of refraction.

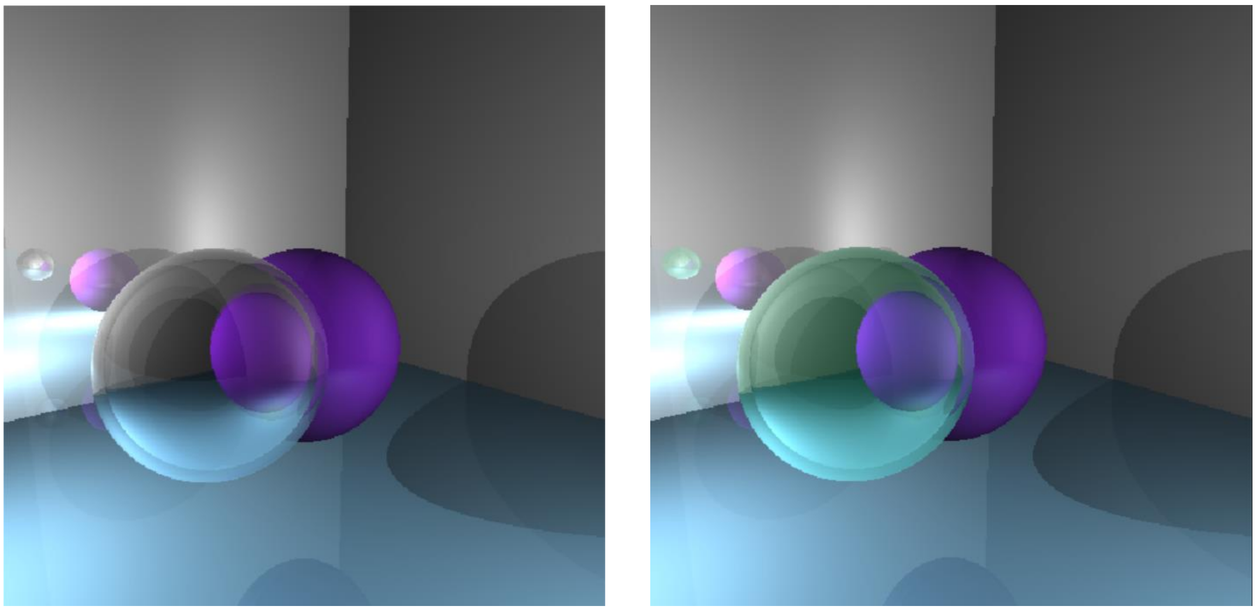
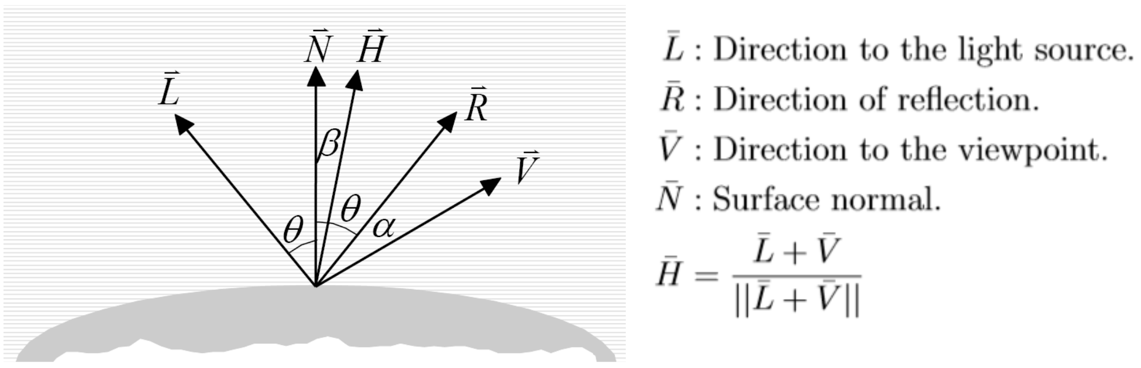


Fig 7. Refraction Results

2.5 Specular Highlight



$\bar{L}$: Direction to the light source.
$\bar{R}$: Direction of reflection.
$\bar{V}$: Direction to the viewpoint.
$\bar{N}$: Surface normal.

$$\bar{H} = \frac{\bar{L} + \bar{V}}{||\bar{L} + \bar{V}||}$$

Fig 8. Halfway Vector

In assignment 4, we have talk about the difference between specular highlight of Phong illumination model and Blinn-Phong illumination model. Here we use Blinn-illumination model to compute specular highlight which means we use halfway vector and surface norm to calculate the intensity of highlight.

Halfway vector H is defined as shown in Fig.8. And the specular highlight is computed by:

$$Highlight = k(N \cdot H)^n$$

There are two parameters affect specular highlight, k and n. How these two variables influence highlight is shown in Fig.9.

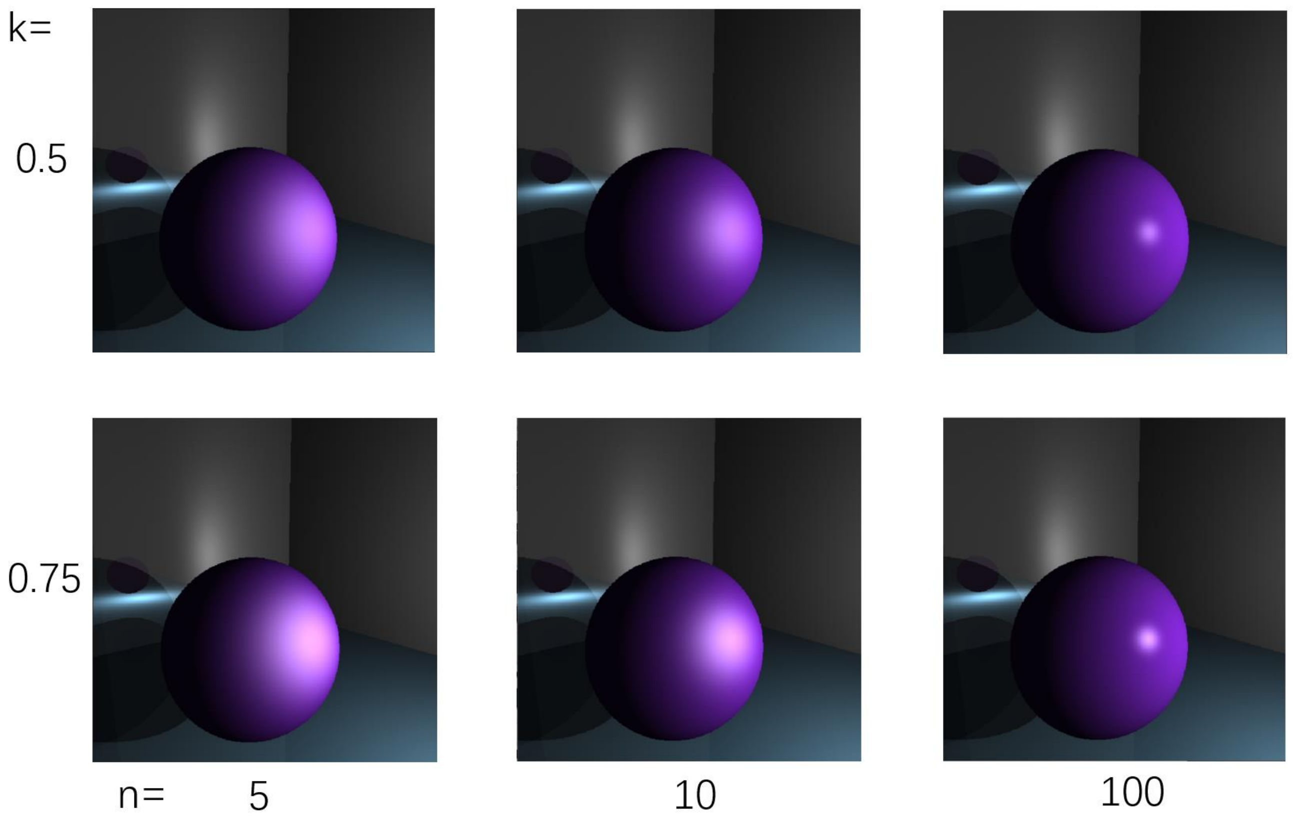


Fig 9. Specular Highlight

We can see that k affects the intensity of highlight and n affects the area size of highlight. Note that diffuse and ambient light also exists in Fig.9. But the diffuse coefficient and ambient light intensity are very low.

As for the implementation of specular highlight, it is computed together with diffuse color in process 3) of ray tracing algorithm which we have introduced in section 2.1. That is because the direction of light source, which serves as shadow ray as well, is also required when computing specular highlight.

3 Implementation Details

The whole system is implemented in pure C++. And OpenGL is used to draw pixels on the window.

3.1 Class Definition

In order to render the whole scene, several basic classes are designed.

- **GVector3:** 3D vector. It contains 3 coordinates with several overloaded operators.
- **Color:** A class represents color. It contains several static inline function that can return some common color and several overloaded operators managing color computation.
- **Ray:** Defined as $\boldsymbol{e}+t\boldsymbol{d}$. $\boldsymbol{e}$ is the start point of ray and $\boldsymbol{d}$ is its direction. Function `GVector3 getPoint(double t);` can output a position on it given distance t.
- **Material:** Contains object material information such as color, reflectivity, diffuse coefficient, and transparency.
- **Intersection:** Contains all information about intersection of a ray and a object such as position of intersection, distance from ray origin, surface normal at intersection and intersected object.
- **BaseObj:** An virtual class for all kinds of objects. The core method of it is `virtual Intersection intersect(Ray& RAY);` which calculates the intersection with a ray. All object inherited from BaseObj should implements its own intersect function. Besides, BaseObj have a Material member variable that defines the material of object.
- **objUnion:** A container of objects. All the objects in the scene is within the same objUnion. Function `virtual Intersection intersect(Ray& _ray);` can compute the closest intersection with a ray.
- **Light:** A virtual class for all kinds of lights. The core method of it is `virtual Color intersect(objUnion& scene, Intersection& result, GVector3& highray);` which calculates the color at intersection. scene is used to calculate shadow and highray is used to calculate specular highlight.
- **lightUnion:** Same as objUnion, a container of lights. It is used to calculate the color at the intersection affected by all the lights at the same time.
- **perspectiveCamera:** A camera class used to generate sight rays from the view point.
- **PLY:** A decoder for ply file. Decode a ply object file as face objects.

3.2 Objects Implementation

In order to render different kinds of objects, several object classes inherited from BaseObj are designed.

3.2.1 Plane

Plane is defined by $\{P|n \cdot (P - P_0) = 0\}$, where $\boldsymbol{n}$ is the normal of the plane and P_0 is one point on it. Two member variable determine a Plane:

- **GVector3 n :** Normal of plane.
- **double d :** Distance to origin.

As shown in Fig.10, the intersection with ray is computed by:

d:$distance\ from\ origin\ to\ Plane$

$\boldsymbol{n}$:$normal\ of\ Plane.$

$\boldsymbol{o}$:$position\ of\ camera.$

$\boldsymbol{if\ nd} < 0$: $ray\ is\ intersected\ with\ plane$

$\boldsymbol{There\ is}\ (\boldsymbol{nd} - \boldsymbol{o}) \cdot \boldsymbol{n} = \boldsymbol{r} \cdot \boldsymbol{n}$

We can get $\boldsymbol{r}$ from it.

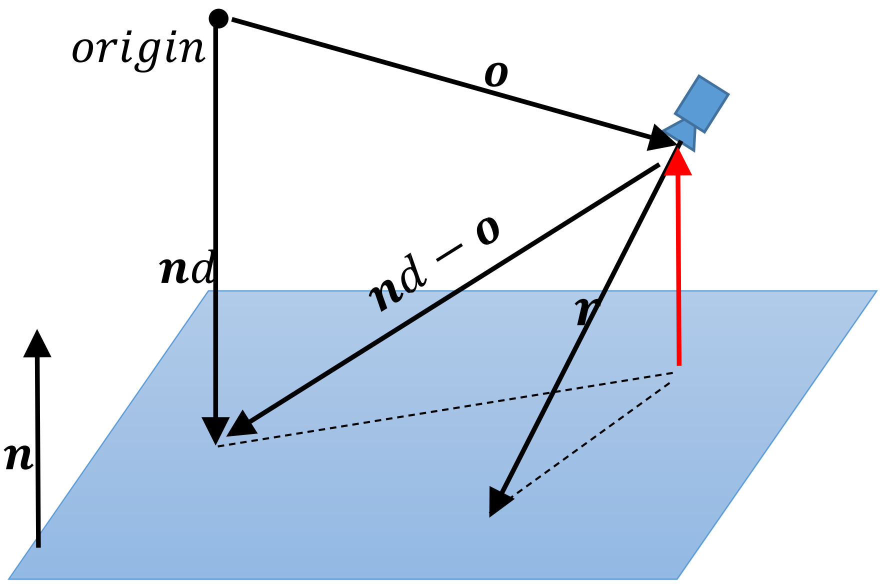


Fig 10. Plane

3.2.2 Sphere

Sphere is defined by $\{P||P - c| = radius\}$ where c is the center of sphere and radius is its radius. Two member variable determine a sphere:

- **GVector3 center**: center of sphere
- **double radius**: radius of sphere

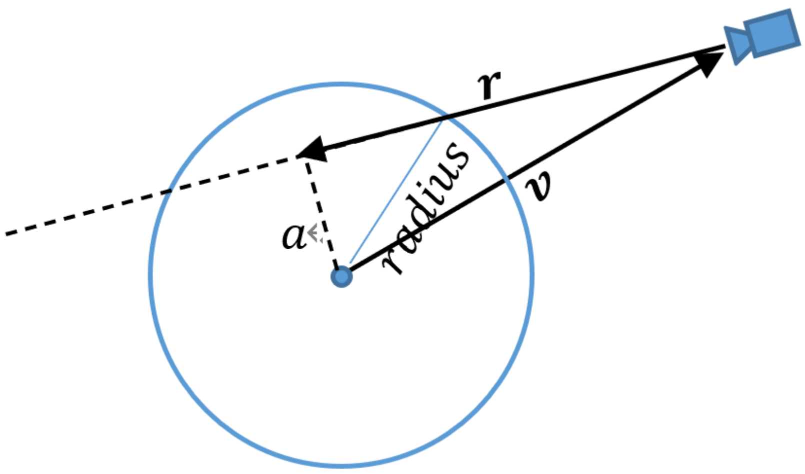


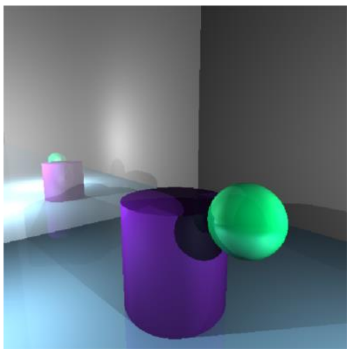
Fig 11. Sphere

As shown in Fig.11, the intersection with ray is computed by

$$\boldsymbol{v} = ray.origin - sphere.center$$
$$r = ray.direction \cdot \boldsymbol{v}.$$
$$a^2 = \boldsymbol{v}^2 - \boldsymbol{r}^2$$
$$if\ \ r < 0\ \ and\ \ a < r:\ \ the\ \ ray\ \ is\ \ intersected\ \ with\ \ s$$

$$Intersection.distance = |r| - \sqrt[2]{radius^2 - a^2}$$
$$Intersection.position = ray.getPoint(Intersection.distance)$$

3.2.3 Cylinder



Cylinder is determined by three member variables:

- **GVector3 center**: center of bottom
- **double radius**: radius of bottom
- **double height**: height

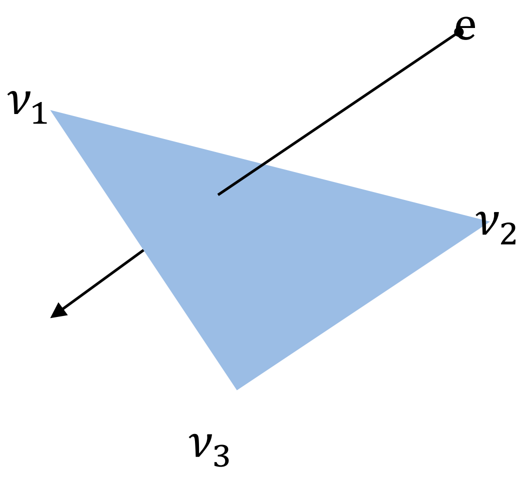
The intersection with bottom can be computed same way as plane. The intersection with side can be computed by projecting the ray to bottom plane.

3.2.4 Face

Face is the basic unit of mesh model. It is defined as $\{P|P = (1-\beta-\gamma)v_1 + \beta v_2 + \gamma v_3,\ 0 < \beta < 1, 0 < \gamma < 1\}$. And three member variables determine a face:

- **GVector3 v1,v2,v3** : Vertex of face.

The intersection is computed by:



```
$\boldsymbol{e} + t\boldsymbol{d} = (1-\beta-\gamma)v_1 + \beta v_2 + \gamma v_3$
$if\ 0 < \beta < 1\ and\ 0 < \gamma < 1\ and\ t > 0:$
    $the\ ray\ is\ intersected\ with\ face.$
    $Intersection.distance = t$
    $Intersection.position = ray.getPoint(t)$
$else:$
    $The\ ray\ is\ not\ intersected\ with\ face.$
```

4 Future Work

- Use multithreading to accelerate rendering.
- Supporting for GPU rendering.

5 Codes

Codes have been upload to https://github.com/Riften/SJTU-Computer-Graphics-2020-Assignments