Programming Assignment III

Tutorial on RayTracing

Submission

- Deadline:

  Due Date: 23:59 Dec 05th, 2022

- Submission:

  main.cpp & Hittable.cpp

If you want to change any other files or implement the program without the template, please email me(qindafei@connect.hku.hk) before submission.

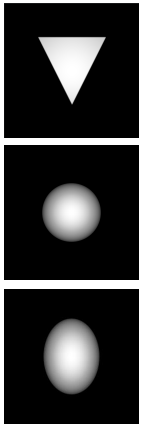
Outline

- RayTracing – Programming Assignment III
  - About the Template
  - About the Task
    - Ray trace
    - Shade
    - Intersection
      - Sphere intersection
      - Quadrics intersection
      - Triangle intersection

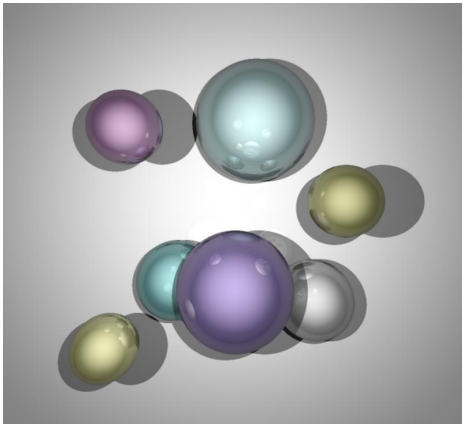
Outline

- RayTracing – Programming Assignment III
  - About the Template
  - About the Task
    - Ray trace
    - Shade
    - Intersection
      - Sphere intersection
      - Quadrics intersection
      - Triangle intersection

Expected rendering result



ray-surface intersection



reflection

Phong interpolation

Scene file

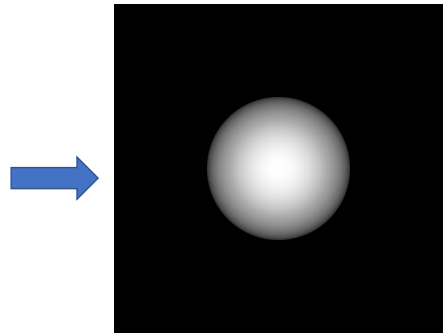
```
[camera]
extrinsic = [
   [1.0, 0.0, 0.0, 0.0],
   [0.0, 1.0, 0.0, 0.0],
   [0.0, 0.0, 1.0, 0.0],
   [0.0, 0.0, 0.0, 1.0]
]

fov = 90.0
width = 800
height = 800
```

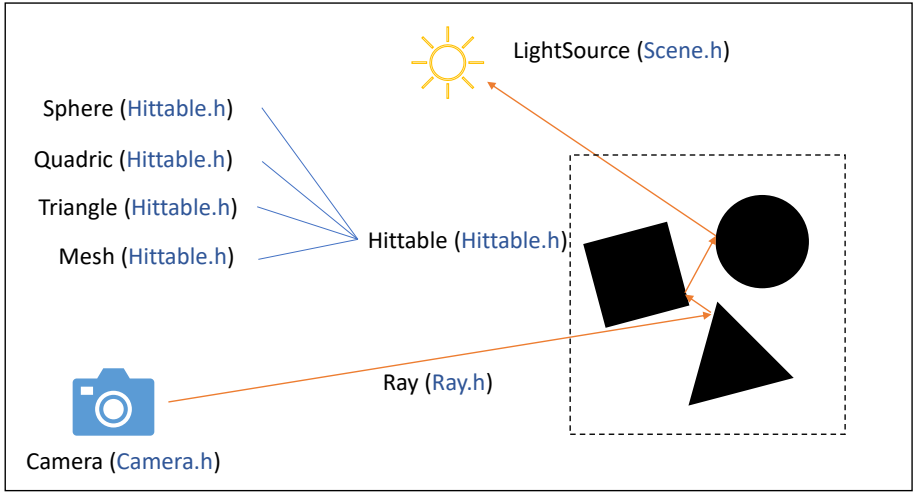
```
[[hittable]]
type = "sphere"
position = [0.0, 0.0, -1.5]
radius = 0.6
ambient = [1.0, 1.0, 1.0]
diffuse = [1.0, 1.0, 1.0]
specular = [1.0, 1.0, 1.0]
k_a = 0.1
k_d = 0.8
k_s = 0.1
sh = 1.0

[[light_source]]
position = [0.0, 0.0, 0.0]
intensity = 1.0
```

sphere.toml



Template Structure



Outline

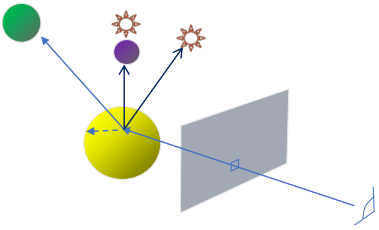
- RayTracing – Programming Assignment III
  - About the Template
  - About the Task
    - Ray trace
    - Shade
    - Intersection
      - Sphere intersection
      - Quadrics intersection
      - Triangle intersection

Your Task

Complete five functions to implement ray tracing.

- TraceRay(…) (in main.cpp)
  - ❑ Given a ray, calculate the color at the nearest intersection point.
- Shade(…) (in main.cpp)
  - ❑ Given the ray-surface intersection information, calculate the corresponding color.
- Triangle::Hit(…) (in hittable.cpp)
  - ❑ Perform intersection calculation between a ray and a triangle.
- Sphere::Hit(…) (in hittable.cpp)
  - ❑ Perform intersection calculation between a ray and a sphere.
- Quadric::Hit(…) (in hittable.cpp)
  - ❑ Perform intersection calculation between a ray and a quadric surface.

Relationship between functions



❑Color=ambient+$\sum S_i$(diffuse + specular)+reflection+refraction

main(…)
❑Calculate the color of each pixel and save it as png file

call

TraceRay(…)
❑Given a ray, calculate the color at the nearest intersection point.

Second call

Shade(…)
❑Given a point, calculate the color of this point.

If depth<kMaxTraceDepth
call with depth+1

First call

call

hittable_objects.Hit(…)
❑**Provided by the template**.
❑Visit each hittable objects in the scene. Find the nearest intersection point.

call

call

call

Sphere::Hit(…)
Do intersection calculation of a ray and a sphere.

Quadric::Hit(…)
Do intersection calculation of a ray and a quadric surface.

Triangle::Hit(…)
Do intersection calculation of a ray and a triangle.

Outline

- RayTracing – Programming Assignment III
  - About the Template
  - About the Task
    - Ray trace
    - Shade
    - Intersection
      - Sphere intersection
      - Quadrics intersection
      - Triangle intersection

Function: TraceRay(…)

Interface

```
Color TraceRay(const Ray& ray,
               const std::vector<LightSource>& light_sources,
               const Hittable& hittable_collection,
               int trace_depth)
```

Task

Trace one particular ray and calculate the color at the closest intersection point.

Parameters

ray: query ray information, represented by ray.o + t * ray.d

light_sources: vector of light sources

hittable_collection: the set of all hittable objects in the scene

trace_depth: current depth of ray tracing

Function: TraceRay(…)

Implementation

IF hittable_collection.Hit(…) Then

- ❑Call shade function to calculate the color of the intersection point.
- ❑Return this color.

ELSE

- ❑Return the background color(0,0,0).

Outline

- RayTracing – Programming Assignment III
  - About the Template
  - About the Task
    - Ray trace
    - Shade
    - Sphere intersection
    - Quadrics intersection
    - Triangle intersection

Function: Shade()

Interface

```
Color Shade(const std::vector<LightSource>& light_sources,
            const Hittable& hittable_collection,
            const HitRecord& hit_record,
            int trace_depth);
```

Task

Calculate the color at a given intersection point.

Parameters

light_sources: vector of light sources

hittable_collection: the set of all hittable objects in the scene

hit_record: the information of the closest intersection point.

trace_depth: current depth of ray tracing

Shading formula

- Intensity (color) at a point P on an object is given by

$$I = k_a O_a + \sum S_i I_{pi} [\, k_d O_d (N \cdot L_i) + k_s O_s (R \cdot V)^{sh} \,] + k_s O_r + k_t O_t$$

Ambient term

Diffuse and specular terms due to each light source

Reflected ray contribution

Refracted ray contribution
This assignment does not require implementation of refraction. So we simply discard this term.

where

O_a = object ambient color,

S_i = 0 means light i is blocked at intersection, 1 means light is not blocked at intersection.
If N.dot(L_i) >0 , call Intersect function to see whether it's blocked.

I_{pi} = intensity of light i

k_d = surface diffuse coefficient

O_d = surface diffuse color

N = normal at surface intersection

L_i = (normalized) direction from the surface intersection to light i

k_s = object specular coefficient

O_s = object specular color

R = direction of the reflection of **shadow ray**

V = ***reversed*** direction of the shoot in ray

sh = object shininess (Tip: use function 'pow')

O_r = intensity of reflected ray

O_t = intensity of refracted ray

Shading

- Intensity (color) at a point P on an object is given by

$$I = k_a O_a + \sum S_i I_{pi} [\, k_d O_d (N \cdot L_i) + k_s O_s (R \cdot V)^{sh} \,] + k_s O_r + k_t O_t$$

```
struct HitRecord {
    Vec normal;
    Vec in_direction;
    Vec reflection;
    Point position;
    float distance;
    Material material;
};
```

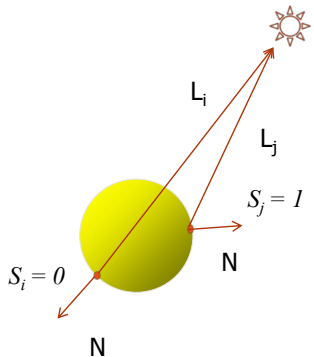
reversed

```
struct Material {
    Color ambient, diffuse, specular;
    float k_a, k_d, k_s;
    float sh = 1.f;
};
```

Get the value of S_i

$$I = O_a + \sum S_i I_{pi} [\, (1 - k_t)\, O_d\, (N \cdot L_i) + k_s\, O_s\, (R \cdot V)^n \,] + k_s\, I_r + k_t\, I_t$$

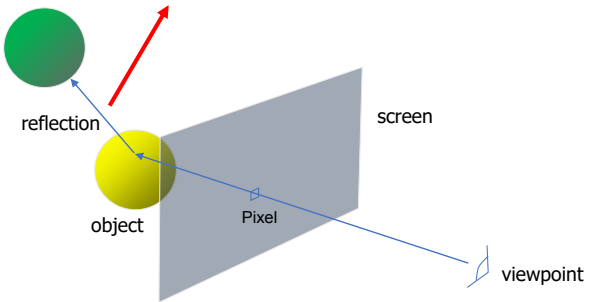
- If dot(N, L_i) <0 , no need to check.
- Otherwise, call Hit(...) function to see whether it's blocked.



Reflection

$$I = O_a + \sum S_i I_{pi} [(1 - k_t) O_d (N \cdot L_i) + k_s O_s (R \cdot V)^n] + k_s I_r + k_t I_t$$

- If depth < kMaxTraceDepth,
  - Get the reflected ray whose start point is the intersection point and direction is the reflection direction.
  - Call TraceRay(..) function to recursively trace this ray and get the color of the closest intersection point I_r
    - Trace(reflected_ray, light_sources, hittable_collection, depth + 1);



Function: Shade()

Ambient term

Diffuse and specular terms due to each light source

Reflected ray contribution

```
Void  Shade(light_sources, hittable_collection, hit_record, trace_depth)
{
    color := ambient color;
    for each light source do {
        shadow_ray := ray to light from intersection point;
        if N.dot(L_i) >0  then{
            call hittable_collection.Hit() to see whether shadow_ray is blocked;
            if(shadow_ray is not blocked by other objects)
                compute the second part of the shading formula and add it to color;
        }
    }
    if depth < MaxTraceDepth then
        if object is reflective then {      // i.e., k_s > 0
            reflected_ray := ray in reflection direction from intersection;
            r_color = TraceRay(reflected_ray, light_sources, hittable_collection, depth + 1);
            scale r_color by reflectance and add to color;
        }
     if color > 1.0 then        clamp color to 1.0
}
```

$$S_i I_{pi} [k_d O_d (N \cdot L_i) + k_s O_s (R \cdot V)^{sh}]$$

Outline

- RayTracing – Programming Assignment III
  - About the Template
  - About the Task
    - Ray trace
    - Shade
    - Intersection
      - Sphere intersection
      - Quadrics intersection
      - Triangle intersection

Function: Hit(…)

Interface

```
bool Hit(const Ray& ray, HitRecord* hit_record)
```

Task

Do ray-object intersection.

If there is no intersection, return false. Otherwise, return true and store the information of the closest intersection point.

Parameters

ray: the query ray.

hit_record: the information of the closest intersection point (output).

```
struct HitRecord {
    Vec normal;         // normal vector of the intersected surface
    Vec in_direction;   // direction of the shoot in ray
    Vec reflection;     // direction of the reflected ray
    Point position;     // intersected position
    float distance;     // distance from the ray's origin to the intersected position
    Material material;  // data structure storing the shading parameters
};
```

Function: Hit(…)

```
Bool Hit(const Ray& ray, HitRecord *hit_record) const {
    if no intersection {
        return false;
    }
    else {
        // Do some calculation
        // …

        hit_record->position = intersected position;
        hit_record->normal = intersected surface normal vector;
        hit_record->distance = distance from ray's origin to intersection;
        hit_record->in_direction = ray.d;
        hit_record->reflection = reflected direction (normalized)
        hit_record->material = material_;
    }
    return true;
}
```

Outline

- RayTracing – Programming Assignment III
  - About the Template
  - About the Task
    - Ray trace
    - Shade
    - Intersection
      - Sphere intersection
      - Quadrics intersection
      - Triangle intersection

Function: Sphere::Hit()

Interface

- `bool Sphere::Hit(const Ray& ray, HitRecord *hit_record)`

Task

- Do ray-sphere intersection. If there is no intersection, return false. Otherwise, return true and store the intersection information in hit_record.

Parameters

- ray: the query ray.
- hit_record: the information of the closest intersection point (output)

Class members

- o_: sphere center
- r_: sphere radius

```
class Sphere : public Hittable {
    Point o_;
    float r_;
    Material material_;
};
```

Function: Sphere::Hit()

Consider a sphere centered as the origin:

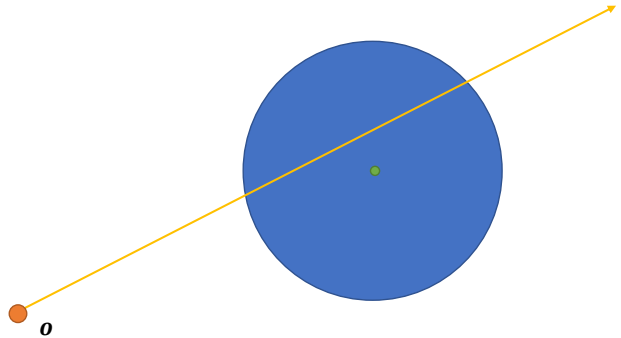
$$x^2 + y^2 + z^2 - r^2 = \boldsymbol{x} \cdot \boldsymbol{x} - r^2 = 0$$

We substitute the ray equation:

$$(\boldsymbol{o} + t\boldsymbol{d}) \cdot (\boldsymbol{o} + t\boldsymbol{d}) - r^2 = 0$$
$$\boldsymbol{o} \cdot \boldsymbol{o} + 2t\boldsymbol{o} \cdot \boldsymbol{d} + t^2 \boldsymbol{d} \cdot \boldsymbol{d} - r^2 = 0$$

Which gives us a quadratic equation in t:

$$At^2 + Bt + C = 0$$
$$A = \boldsymbol{d} \cdot \boldsymbol{d} = 1$$
$$B = 2\boldsymbol{o} \cdot \boldsymbol{d}$$
$$C = \boldsymbol{o} \cdot \boldsymbol{o} - r^2$$



• the smaller equation root is the closest ray intersection

Outline

- RayTracing – Programming Assignment III
  - About the Template
  - About the Task
    - Ray trace
    - Shade
    - Intersection
      - Sphere intersection
      - Quadrics intersection
      - Triangle intersection

Function: Quadric::Hit()

Interface

- `bool Sphere::Hit(const Ray& ray, HitRecord *hit_record)`

Task

- Do ray-quadric intersection. If there is no intersection, return false. Otherwise, return true and store the intersection information in hit_record.

Parameters

- ray: the query ray.
- hit_record: the information of the closest intersection point (output)

Class members

- A_: the coefficient matrix of the quadratic equation.

```
class Quadric : public Hittable {
    glm::mat4x4 A_;
    Material material_;
};
```

Function: Quadric::Hit()

General form of quadrics:

$$ax^2 + by^2 + cz^2 + 2dxy + 2eyz + 2fxz + 2gx + 2hy + 2iz + j = 0$$

$$[x, y, z, 1]\begin{bmatrix} a & d & f & g \\ d & b & e & h \\ f & e & c & i \\ g & h & i & j \end{bmatrix}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = 0$$

Using homogeneous coordinates.

$$X^T AX = 0$$

A is given as the class member.

```
class Quadric : public Hittable {
    glm::mat4x4 A_;
    Material material_;
};
```

Function: Quadric::Hit()

The parametric ray is $R(t) = O + Dt, t \geq 0$

Where O = (ray.o[0], ray.o[1], ray.o[2], 1)^T

D = (ray.d[0], ray.d[1], ray.d[2], 0)^T

Substituting it in the quadric surface, we have

$$(O + Dt)^T A(O + Dt) = 0$$
$$(D^T AD)t^2 + 2(O^T AD)t + O^T AO = 0$$
$$at^2 + bt + c = 0$$

The determinant of this equation is

$$\Delta = b^2 - 4ac$$

If $\Delta > 0$, $t_0 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, t_1 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$.

If $\Delta = 0$, $t = -\frac{b}{2a}$.

Choose the smallest t.

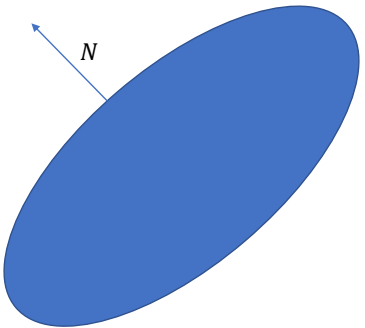
If there is no feasible t, return false.

Function: Quadric::Hit()

Calculate surface normal vector:

$$N = \frac{d(\boldsymbol{x}^T A \boldsymbol{x})}{d\boldsymbol{x}} = (A + A^T)\boldsymbol{x}$$

Remember to normalize it!



Outline

- RayTracing – Programming Assignment III
  - About the Template
  - About the Task
    - Ray trace
    - Shade
    - Intersection
      - Sphere intersection
      - Quadrics intersection
      - Triangle intersection

Function: Triangle::Hit

Interface

- `bool Triangle::Hit(const Ray& ray, HitRecord *hit_record)`

Task

- Do ray-triangle intersection. If there is no intersection, return false. Otherwise, return true and store the intersection information in hit_record.

Parameters

- ray: the query ray.
- hit_record: the information of the closest intersection point (output)

Class members

- a_, b_, c_: vertices of the triangle.
- n_a_, n_b_, n_c_: corresponding vertex normals.
- phong_interpolation: flag of using Phong shading / Flat shading

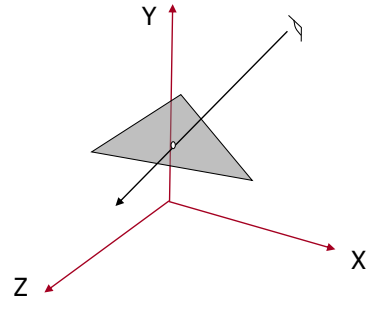
```
class Triangle : public Hittable {
    Point a_, b_, c_;
    Vec n_a_, n_b_, n_c_;
    bool phong_interpolation_ = true;
};
```

Function: Triangle::Hit

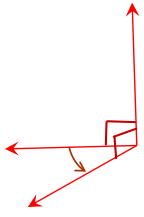
Find the intersection point P of the ray and the plane which contains the triangle;

- The equation of the plane is $n \cdot x = d$
- The parametric ray is $R(t) = S + Dt, t \geq 0$
- If N.dot(D)!=0
  - Substituting it in the plane, you can get t.

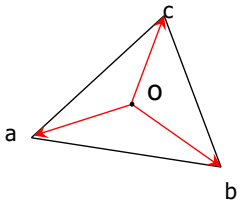
Check whether the intersection point is inside the triangle.

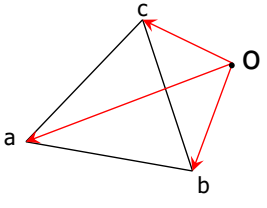


Function: Triangle::Hit



cross product





Check whether the intersection point is inside the triangle

Compute cross product in the following sequence.

- cross(oa, ob), cross(ob, oc), cross(oc, oa)

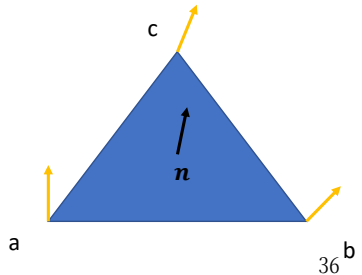
- When C is inside the triangle, the three result vectors have the same direction.
- Compute dot product to check whether two vectors have the same direction.

Function: Triangle::Hit

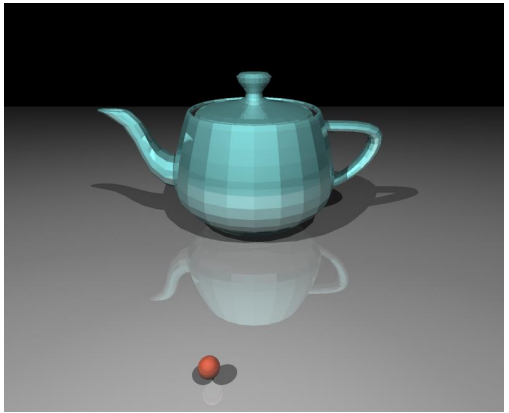
Calculate normal vector

Flat shading (if phong_interpolation_ == false):
cross product of two face edges (note the direction)
Phong shading (if phong_interpolation_ == true):
Interpolate normal vector by barycentric coordinates

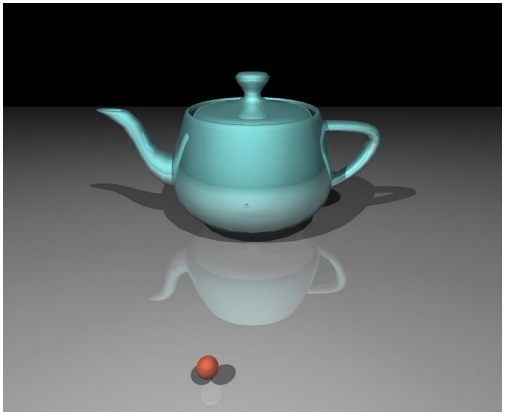
$$\boldsymbol{n} = \alpha_1 \boldsymbol{n}_1 + \alpha_2 \boldsymbol{n}_2 + \alpha_3 \boldsymbol{n}_3$$
$$\alpha_1 + \alpha_2 + \alpha_3 = 1$$



Function: Triangle::Hit



flat shading



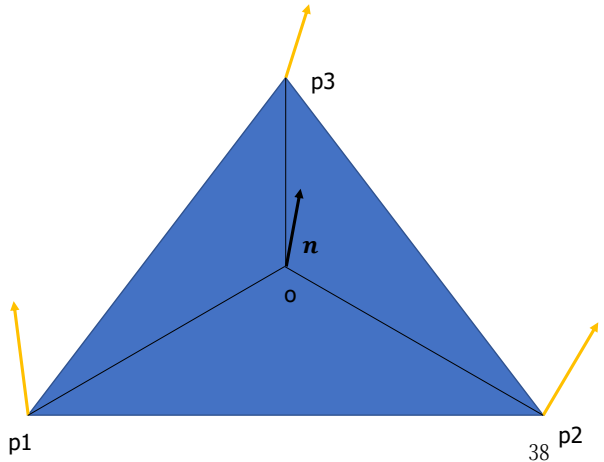
Phong shading

Function: Triangle::Hit

Calculate Barycentric coordinate

$$\boldsymbol{n} = \alpha_1 \boldsymbol{n}_1 + \alpha_2 \boldsymbol{n}_2 + \alpha_3 \boldsymbol{n}_3$$
$$\alpha_1 + \alpha_2 + \alpha_3 = 1$$

$$a_1 = \frac{\Delta o p_2 p_3}{\Delta p_1 p_2 p_3} \quad a_2 = \frac{\Delta o p_3 p_1}{\Delta p_1 p_2 p_3} \quad a_3 = \frac{\Delta o p_1 p_2}{\Delta p_1 p_2 p_3}$$



Other implementation details

We use glm for vector/matrix calculation.

Colors, 3D vectors and points are all represented by glm::vec3

```
using Color = glm::vec3;
using Vec   = glm::vec3;
using Point = glm::vec3;
```

Calculate $x^T Ax$ using glm:
glm::dot(x, A * x)

Remember to note float precision issue

Hand-in

- Hand in main.cpp and Hittable.cpp if you use the template.
- Ensure that your files can be compiled and run successfully.
- Submit your file through Moodle.

- Late Policy
  - 50% off for the delay of each working day.
  - Re-submission after deadline is treated as late submission.
- NO PLAGIARISM!

Thanks

Q & A