Useful Classes for a Ray Tracer
A possible Ray class

class Ray {
    public:
        glm::vec3 Origin;
        glm::vec3 Direction;
};
A possible Intersection class

class Intersection {
    public:
        Intersection(){Mtl=0; HitDistance=1e10;}

        // Results of intersection test
        float     HitDistance;
        glm::vec3 Position;
        glm::vec3 Normal;
        glm::vec3 Direction;
        Material* Mtl;

        // Results of shading
        Color     Shade;
};
A possible Vertex (with normal) class

class Vertex {
    public:
    glm::vec3 Position;
    glm::vec3 Normal;
};
A possible Triangle class

class Triangle {
    public:
        Triangle();

        bool Intersect(const Ray &ray, Intersection &hit);

    private:
        Vertex* Vtx[3];
        Material* Mtl;
};
A possible Camera class

class Camera {
public:
    Camera();

    // many familiar helper functions such as
    void LookAt(glm::vec3 eye, glm::vec3 cen, glm::vec3 up);

    // and of course
    void Render(Scene &s);
    void SaveImage(char *filename);

private:
    int width, height;
    glm::vec3 pos, u, v, w;
    float fovy;
    float aspect;
    Image image;
};
Additional Ray-Object Intersection Discussion
• We have learned ray-triangle intersection
• We have learned ray-sphere intersection
• Let us look at ray-transformed object intersection (e.g. ray-ellipsoid) (see whiteboard)
• Quadratic surface (quadrics) (see whiteboard)
Acceleration for Ray-Scene Intersection
Acceleration

• An object with millions of triangles will slow down the whole ray tracer even for pixels looking away from the object.

• To accelerate, we can organize the space into a partition.
  ▶ Easy to find ray-partition intersection
  ▶ We only need to perform intersection test for the primitives in the partition.

• These partitions are spatial data structures:
  ▶ Octree
  ▶ k-d tree
  ▶ (see whiteboard for illustration)
Shade the Ray/Intersection
Shade the ray

- Suppose we have a ray with direction \( \mathbf{d} \) intersecting a surface with normal \( \mathbf{n} \).
- Let the material be characterized by
  - Ambient color \( \mathbf{A} \)
  - Emission color \( \mathbf{E} \)
  - Diffuse color \( \mathbf{D} \)
  - Specular color \( \mathbf{S} \)
  - shininess \( s \)
- Then this ray is shaded with color

\[
\mathbf{I} = \mathbf{A} + \mathbf{E} + \sum_{j \in \text{lights}} \left( \text{visibility to light } j \right) \cdot \left( \text{color contributed by light } j \right) + \mathbf{SI}_{\text{RefIRay}}
\]

RGB channel-wise product
Shade the ray

- Suppose we have a ray with direction $d$ intersecting a surface with normal $n$.

- Let the material be characterized by
  - Ambient color $A$
  - Emission color $E$
  - Diffuse color $D$
  - Specular color $S$
  - Shininess $s$

- Then this ray is shaded with color

$$I = A + E + \sum_{j \in \{\text{lights}\}} \left( \text{visibility to light } j \right) \cdot \left( \text{color contributed by light } j \right) + S I_{\text{ReflRay}}$$

\[
\text{(color contributed by light } j) = \frac{\text{LightColor}_j}{c_0,j + c_1,j r_j + c_2,j r_j^2}
\]

Lambert cosine law

\[
r_j = \left( \text{distance to light}_j \right)
\]

attenuation model

Blinn–Phong phenomenological model
Global Illumination
Global illumination

• The earlier shading model computes diffuse/specular color directly from the light source. (Only the mirror reflection is recursive.)

• But the light can bounce off from some other surface. The contribution of diffuse/specular color can also come from nearby surfaces.

• Let the recursive mirror reflection take care of the specular color (forget about the Blinn–Phong term).

• Let the diffuse color also computed through recursive ray tracing!
Path tracing

- The earlier model gives the so-called direct lighting.
- We add indirect lighting, which are paths that have more bounces.
For a path that has three bounces, it may be:

- Shoot a ray through a random point in the pixel.
- Hit the first surface, specularly (mirror) reflect the ray, record the specular color as the reflection weight.
- Hit the second surface, diffusely reflect the ray. That is, take a uniformly sampled point on the hemisphere and set it as the reflection ray. Record the diffuse color multiplied by the cosine of the angle between the normal and the reflection ray.
- Hit the third surface. Just shade this final ray using the direct diffuse model.
- The color of this path at the pixel is given by the final ray color multiplied by all the recorded weights.

This type of path is denoted by “L D D S E”
- L: light, D: diffuse bounce, S: specular bounce, E: eye
Path tracing

- This type of path is denoted by “L D D S E”
  - L: light, D: diffuse bounce, S: specular bounce, E: eye

- The color of the pixel contributed by the type LDDSE is given by taking the average of all random paths going about LDDSE.

- The color from all the 3-bounce-paths is the sum of the color of the LDDDE, LDSDE, LDSSE, …
  - A good simplification is to only take the LD*S*E ones.
  - The LS [D or S]* E needs other tricks if L is only a point light. For example, photon-map.

- The color of the pixel is [the color of the 1 bounce paths] + [the color of the 2 bounce paths] + [the color of the 3 bounce paths] …
Path tracing

- The color of the pixel is [the color of the 1 bounce paths] + [the color of the 2 bounce paths] + [the color of the 3 bounce paths] …
  - The infinite series converges.
  - To see this, note that the weights in the n-bounce path has n cosines and diffuse/specular weights (between 0 and 1) multiplied together.
  - One can truncate the series to only a few bounces.
  - The Russian Roulette method: Let the ray run indefinitely until randomly terminated. At every bounce, toss a coin (with probability p) dictating whether we terminate the path. Now, a path of m bounces is weighted by (1-p)^m. So, divide the contribution from the m bounce paths by (1-p)^m.
Path tracing

- 1 sample path per pixel
- 10 sample paths per pixel
- 1000 sample paths per pixel
Next time

- Rendering equation, BRDF
- Additional topics