Computer Graphics (cse167)
Ray Tracing Part I

Albert Chern
Rendering

- Effects for realistic images
  - (Soft) shadows
  - Reflections (mirror and glossy)
  - Transparent (water, glass)
  - Inter-reflections (color bleeding)
  - Realistic materials

- Difficult in OpenGL pipeline
- Easy in raytracing framework
Outline

- Basic ray casting
- Shadow and reflection
- Ray-surface intersection
Rasterization v.s. Ray Casting

- Two different approaches to turn 3D scenes into pixels
  - Rasterization
    
    ```
    For each geometry in the scene
    For each pixel from the screen
      IntersectionTest(geometry, pixel)
      Store fragments into a Z-buffer
    EndFor
    EndFor
    ```
  
  - Ray casting
    
    ```
    For each pixel from the screen
    For each geometry in the scene
      IntersectionTest(geometry, pixel)
      Keep the nearest intersection
    EndFor
    ```

- These two approaches give the same images as long as the shading models are the same
• These two approaches give the same images as long as the shading models are the same

• Shadows etc are much easier in ray casting

• OpenGL is a rasterizer. Rasterizer pipeline is fixed, easy to optimize in hardware for realtime rendering
History of ray casting/tracing

- Appel 1968
- Whitted 1980
  recursive ray tracing
- Lots of work on
  photorealism, accelerations.

- Real time ray tracing
  - 2009 Nvidia OptiX API
  - 2020 PlayStation5, Xbox series X and S
Image Raytrace(Camera cam, Scene scene, int width, int height){

    Image image = new Image(width, height);
    for (int i=0; i<height; i++)
        for (int j=0; j<width; j++){
            Ray ray = RayThruPixel(cam, i, j);
            Intersection hit = Intersect(ray, scene);
            image[i][j] = FindColor(hit);
        }
    return image;
}
• Take the closest hit for each ray.

• Shade the ray according to a shading model.

• Shade the ray only when the secondary ray connecting to the light source does not hit other objects.
  ▶ To avoid self-shadowing, the secondary ray is shot off slightly above the hitting point. (This is more straightforward than the rasterization-based shadow casting)
Shading, shadow and mirror reflection

- Take the closest hit for each ray.
- Shade the ray according to a shading model.
- Shade the ray only when the secondary ray connecting to the light source does not hit other objects.
- In addition to the diffuse+specular shading model we can include a recursive specular reflection.
Recursive ray tracing

ForEach pixel

- Trace the primary eye ray
  - Find intersection
- Trace secondary shadow ray to all lights
  - Color = Visible ? IlluminationModel : 0 ;
- Trace reflected ray
  - Color += reflectivity * (Color of the reflected ray)
  - Recursion might never stop, so set a max recursion depth
Detail: RayThruPixel
Image Raytrace(Camera cam, Scene scene, int width, int height){
    Image image = new Image(width, height);
    for (int i=0; i<height; i++)
        for (int j=0; j<width; j++){
            Ray ray = RayThruPixel(cam, i, j);
            Intersection hit = Intersect(ray, scene);
            image[i][j] = FindColor(hit);
        }
    return image;
}
• A **ray** is a described by a point $\mathbf{p}_0 \in \mathbb{R}^3$ and a direction $\mathbf{d} \in \mathbb{R}^3$.

• Mathematically, the ray is a continuous set of points parametrized as

$$\mathbf{p}(t) = \mathbf{p}_0 + t\mathbf{d} \quad t > 0$$

\[\text{Ray}\]
A camera is described by
\[ \text{eye} \in \mathbb{R}^3 \quad \text{center} \in \mathbb{R}^3 \quad \text{up} \in \mathbb{R}^3 \]
which can also be rewritten in terms of a coordinate frame
\[ \text{eye} \in \mathbb{R}^3 \quad \mathbf{u} \in \mathbb{R}^3 \quad \mathbf{v} \in \mathbb{R}^3 \quad \mathbf{w} \in \mathbb{R}^3 \]

\[ \mathbf{w} = \frac{\text{eye} - \text{center}}{|\text{eye} - \text{center}|} \quad \mathbf{u} = \frac{\text{up} \times \mathbf{w}}{|\text{up} \times \mathbf{w}|} \quad \mathbf{v} = \mathbf{w} \times \mathbf{u} \]
Ray through pixel

• Other relevant parameters:
  - Window resolution: width, height
  - Field of view (angle): $\text{fovx}, \text{fovy}$
  - $\tan\left(\frac{\text{fovx}}{2}\right) = \tan\left(\frac{\text{fovy}}{2}\right) \frac{\text{width}}{\text{height}}$

• Given camera $\text{eye} \in \mathbb{R}^3$ $u, v, w \in \mathbb{R}^3$
  - Eye position
  - And pixel $i \in \{0, \ldots, \text{height} - 1\}$
  - $j \in \{0, \ldots, \text{width} - 1\}$

  the ray is given by… (next page, see whiteboard for derivation)
Ray through pixel

\[ p_0 = \text{eye} \quad d = \frac{\alpha u + \beta v - w}{|\alpha u + \beta v - w|} \]

where

\[
\alpha = \tan \left( \frac{\text{fov}_x}{2} \right) \left( \frac{(j+.5)-\frac{\text{width}}{2}}{\frac{\text{width}}{2}} \right) \\
\beta = \tan \left( \frac{\text{fov}_y}{2} \right) \left( \frac{(i+.5)-\frac{\text{height}}{2}}{\frac{\text{height}}{2}} \right)
\]
Detail: Ray-Scene Intersect
```java
Image Raytrace(Camera cam, Scene scene, int width, int height) {
    Image image = new Image(width, height);
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            Ray ray = RayThruPixel(cam, i, j);
            Intersection hit = Intersect(ray, scene);
            image[i][j] = FindColor(hit);
        }
    }
    return image;
}
```
Ray-scene intersection

```cpp
Intersection Intersect(Ray ray, Scene scene){
    Distance mindist = INFINITY;
    Object hitobject = NULL;
    foreach (object in scene){ // Find closest intersection; test all objects
        Distance t = Intersect(ray, object);
        if (t>0 && t< mindist){ // closer than previous closest object
            mindist = t;
            hitobject = object;
        }
    }
    return IntersectionInfo(ray, mindist, hitobject);
}
```
Ray-object intersection

```c
Intersection Intersect(Ray ray, Scene scene) {
    Distance mindist = INFINITY;
    Object hitobject = NULL;
    foreach (object in scene) {
        Distance t = Intersect(ray, object);
        if (t>0 && t< mindist) {
            mindist = t;
            hitobject = object;
        }
    }
    return IntersectionInfo(ray, mindist, hitobject);
}
```

- Compute the intersection (in terms of the parameter \( t \) for the ray) between the ray and a primitive.
• Compute the intersection (in terms of the parameter $t$ for the ray) between the ray and a primitive.

• We will study ray-sphere (and ray-ellipsoids) and ray-triangle on whiteboard.
Next time

- Lighting