02562: Ray Casting Exercises

Jeppe Revall Frisvad

August 2011

Exercises

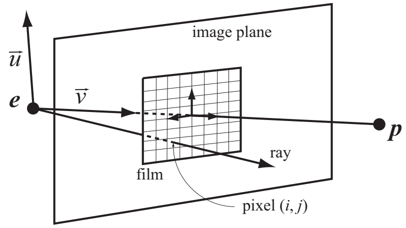
- Generate rays using a (modified) pinhole camera model.
- Compute ray-plane, ray-triangle, and ray-sphere intersection.
- Implement the main loop that traces a ray through each pixel.
- Compute shading of diffuse surfaces by point lights. Use Kepler's inverse square law and Lambert's cosine law.

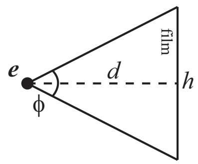
Ray generation

- Camera description:

Extrinsic parameters		Intrinsic parameters	
$\mathbf{e}$	Eye point	ϕ	Vertical field of view
$\mathbf{p}$	View point	d	Camera constant
$\vec{u}$	Up direction	W, H	Camera resolution

- Sketch of ray generation:

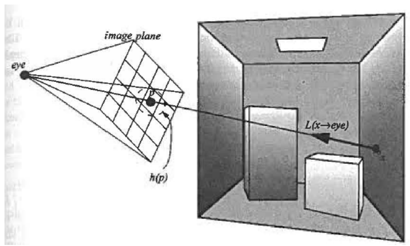


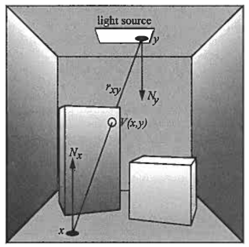


- Given pixel index (i, j), find the direction $\vec{\omega}$ of a ray through that pixel.

Rays in theory and in practice

- Parametrisation of a line: $\mathbf{r}(t) = \mathbf{o} + t\vec{\omega}$.
- Camera provides origin ($\mathbf{o}$) and direction ($\vec{\omega}$) of "eye rays".





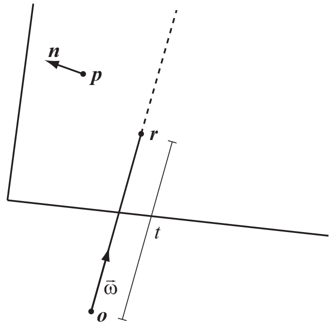
- Rays in code (OptiX library):

```
struct Ray
{
        Ray() {}
        Ray(float3 origin_, float3 direction_, unsigned int ray_type_, float tmin_, float tmax_ =
        RT_DEFAULT_MAX);

        float3 origin, direction;
        unsigned int ray_type;
        float tmin, tmax;
};
```

Ray-plane intersection

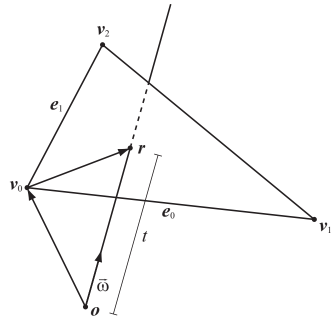
- The ray: $\mathbf{r}(t) = \mathbf{o} + t\,\vec{\omega}$.
- The plane: $ax + by + cz + d = 0 \quad \Leftrightarrow \quad \mathbf{p} \cdot \mathbf{n} + d = 0$.



- Setting $\mathbf{p} = \mathbf{r}$, we find the distance t to the intersection point:

$$(\mathbf{o} + t\,\vec{\omega}) \cdot \mathbf{n} + d = 0 \quad \Leftrightarrow \quad t = -\frac{\mathbf{o} \cdot \mathbf{n} + d}{\vec{\omega} \cdot \mathbf{n}} \; .$$

Ray-triangle intersection



- The ray: $\mathbf{r}(t) = \mathbf{o} + t\,\vec{\omega}$.
- Triangle: $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2$.
- Edges and normal:

$$\begin{aligned} \mathbf{e}_0 &= \mathbf{v}_1 - \mathbf{v}_0 \, , \\ \mathbf{e}_1 &= \mathbf{v}_0 - \mathbf{v}_2 \, , \\ \mathbf{n} &= \mathbf{e}_0 \times \mathbf{e}_1 \; . \end{aligned}$$

- Barycentric coordinates:

$$\begin{aligned} \mathbf{r}(u, v, w) &= u\mathbf{v}_0 + v\mathbf{v}_1 + w\mathbf{v}_2 = (1 - v - w)\mathbf{v}_0 + v\mathbf{v}_1 + w\mathbf{v}_2 \\ &= \mathbf{v}_0 + v\mathbf{e}_0 - w\mathbf{e}_1 \; . \end{aligned}$$

- Find $\mathbf{r}(t) - \mathbf{v}_0$ and decompose it into portions along the edges $\mathbf{e}_0$ and $\mathbf{e}_1$ to get v and w. Then check

$$v \geq 0 \quad , \quad w \geq 0 \quad , \quad v + w \leq 1 \; .$$

$\mathbf{a} \cdot \mathbf{b}$	$=$	$\mathbf{b} \cdot \mathbf{a}$	(dot product commutation)
$\mathbf{a} \times \mathbf{b}$	$=$	$-\mathbf{b} \times \mathbf{a}$	(cross product anticommutation)
$\mathbf{a} \cdot (\mathbf{b} \times \mathbf{c})$	$=$	$(\mathbf{a} \times \mathbf{b}) \cdot \mathbf{c}$	(triple scalar product)
$\mathbf{a} \times (\mathbf{b} \times \mathbf{c})$	$=$	$\mathbf{b}(\mathbf{a} \cdot \mathbf{c}) - \mathbf{c}(\mathbf{a} \cdot \mathbf{b})$	(triple vector product)
$(\mathbf{a} \times \mathbf{b}) \times (\mathbf{c} \times \mathbf{d})$	$=$	$((\mathbf{a} \times \mathbf{b}) \cdot \mathbf{d})\mathbf{c} - ((\mathbf{a} \times \mathbf{b}) \cdot \mathbf{c})\mathbf{d}$	

Ray-sphere intersection

- The ray: $\mathbf{r}(t) = \mathbf{o} + t\,\vec{\omega}$.
- The sphere: $(x - c_x)^2 + (y - c_y)^2 + (z - c_z)^2 = r^2$.
- With $\mathbf{p} = (x, y, z)$ and $\mathbf{c} = (c_x, c_y, c_z)$, the sphere is

$$(\mathbf{p} - \mathbf{c}) \cdot (\mathbf{p} - \mathbf{c}) = r^2 \; .$$

- Setting $\mathbf{p} = \mathbf{r}$, we find the distance t to the intersection point:

$$(\mathbf{o} - \mathbf{c} + t\,\vec{\omega}) \cdot (\mathbf{o} - \mathbf{c} + t\,\vec{\omega}) = r^2 \; .$$

- This is a second degree polynomial, $at^2 + bt + c = 0$, with

$$\begin{aligned} a &= \vec{\omega} \cdot \vec{\omega} = 1 \, , \\ b/2 &= (\mathbf{o} - \mathbf{c}) \cdot \vec{\omega} \, , \\ c &= (\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c}) - r^2 \; . \end{aligned}$$

- The distances to the intersection points are

$$t_1 = -b/2 - \sqrt{(b/2)^2 - c} \quad , \quad t_2 = -b/2 + \sqrt{(b/2)^2 - c} \; .$$

- There is no intersection if $(b/2)^2 - c < 0$.

Conditional acceptance of intersection

- When computing ray-object intersection, we must always check that the distance to the intersection t is within the limits:

$$t_{\text{min}} \leq t \leq t_{\text{max}} \; .$$

- When searching for the closest hit, modifying t_{max} ensures that intersections further away will no longer be considered.
- When searching for any hit, terminate the search as soon as an intersection is found.

The main loop in ray tracing

```
// Starting in RenderEngine.cpp
select center of projection (eye point) and window on viewplane (film);
for (each scan line in image) {
  for (each pixel in scan line) {
    // Calling tracer.compute_pixel(...) in RayCaster.cpp
    // which uses the Camera and the Accelerator through the Scene.
    determine ray from center of projection (eye) through pixel;
    for (each object in scene) { // in Accelerator.cpp
      if (object is intersected and is closest considered thus far)
        record intersection and object name (material);
    }
    // Using the intersected material to call the shader.
    set pixel's color (shade) to that at closest object intersection;
  }
}
```

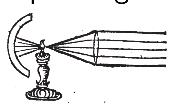
Shading pixels (local illumination)

```
// Starting in Lambertian.cpp with a ray that hit a surface
for (each light source) {
  construct a variable for accumulating light from this source;
  for (each light source sample) {
    // Handle the following three lines by calling the function sample(...)
    // associated with the light source.
    if (ray from surface position to sample point is not occluded) {
      get the direction toward the sample point on the light;
      compute the amount of radiance incident from the sample point;
      if (cosine of angle between surface normal and direction is positive)
        accumulate incident light multiplied by cosine term;
    }
    add accumulated light divided by number of samples to final result;
  }
}
multiply final result by diffuse reflectance and add emission;
```

Kepler's inverse square law

As the relation of a spherical surface, which has its centre in the origin of the light, is from a larger to a smaller one: such is the relation of the strength or density of light rays in a smaller to that in a more spacious spherical surface, that is, conversely. [Kepler 1604]

- Light emitted from a point light falls off with the square of the distance to the point.
- Kepler struggled with this law since he only had the notion of rays of light (light was not considered to spread in a volume).
- Neither did he have a precise law of refraction, but he did observe that the inverse square law only works for point lights. He did this by generating collimated/directional light using concave mirrors and convex lenses.



- If a point light at $\mathbf{p}$ has intensity I, the light incident at a surface point $\mathbf{x}$ is

$$L_i = \frac{I}{\|\mathbf{p} - \mathbf{x}\|^2} \ .$$

Lambert's cosine law

brightness decreases in the same ratio by which the sine of the angle of incidence decreases [Lambert 1760]

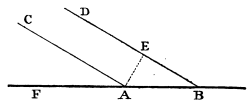


Fig. 1.

- Lambert uses the angle $(\mathrm{CAF} = \mathrm{DBF})$ between the direction of the rays (CA and DB) and the surface tangent plane (AB) as the angle of incidence.
- If we instead measure the angle of incidence θ from the normalised surface normal $\vec{n}$ to the direction toward the incident light $\vec{\omega}'$, sine becomes cosine.
- Then the diffusely reflected light is

$$L_r = \frac{\rho_d}{\pi} L_i \cos\theta = \frac{\rho_d}{\pi} L_i \, (\vec{n} \cdot \vec{\omega}') \ .$$

where ρ_d is the diffuse reflectance.