



Computer Graphics

Lecture 7

Ray-Sphere Intersection

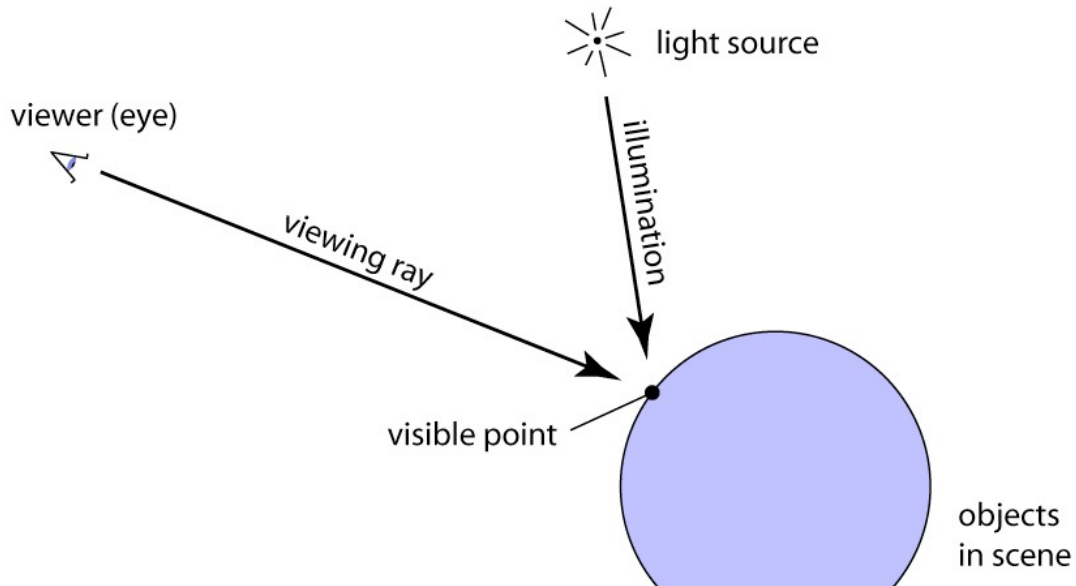
Announcements

- Don't forget to vote for your favorite artifact!
See my canvas announcement for details.
- HWO grades released
- AO grades coming soon (TM)
- No videos for tomorrow

Ray Tracing: Pseudocode

for each pixel:

- last time* → generate a viewing ray for the pixel
- to obj (spaces)* → find the closest object it intersects
- next time* → determine the color of the object



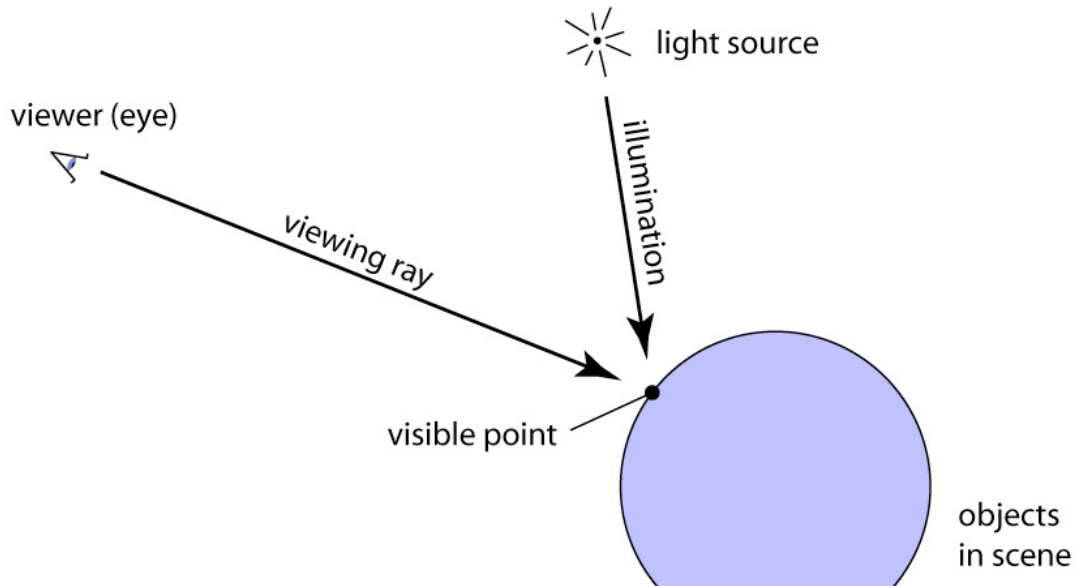
Ray Tracing: Pseudocode

for each pixel:

generate a viewing ray for the pixel

find the closest object it intersects

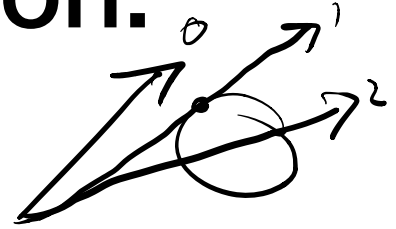
determine the color of the object



Reminder: Implicit vs Parametric

- Implicit equations: a property true at all points
 - e.g., $ax + by + c = 0$, for a line
- Parametric equations: use a free parameter variable to *generate* all points:
 - e.g., $r(t) = \mathbf{p} + t\mathbf{d}$, for a line
- Intersecting parametric with implicit is usually cleanest.

Ray-Sphere Intuition: Geometric




- How many times ~~will~~ can ^a ray intersect a sphere?
- For now, consider a unit sphere at the origin.
- What's an implicit equation for a sphere?
or: What's true of all points on a sphere?

$$x^2 + y^2 + z^2 = 1$$

Ray-Sphere Intuition: Geometric

- How many times will can ray intersect a sphere?
- An implicit equation for a sphere centered at the origin:

$$\sqrt{x^2 + y^2 + z^2} = r$$


Geometric Intuition: LHS is the distance from the origin.

Ray-Sphere Intuition: Geometric

- How many times will can ray intersect a sphere?
- An implicit equation for a sphere centered at the origin:

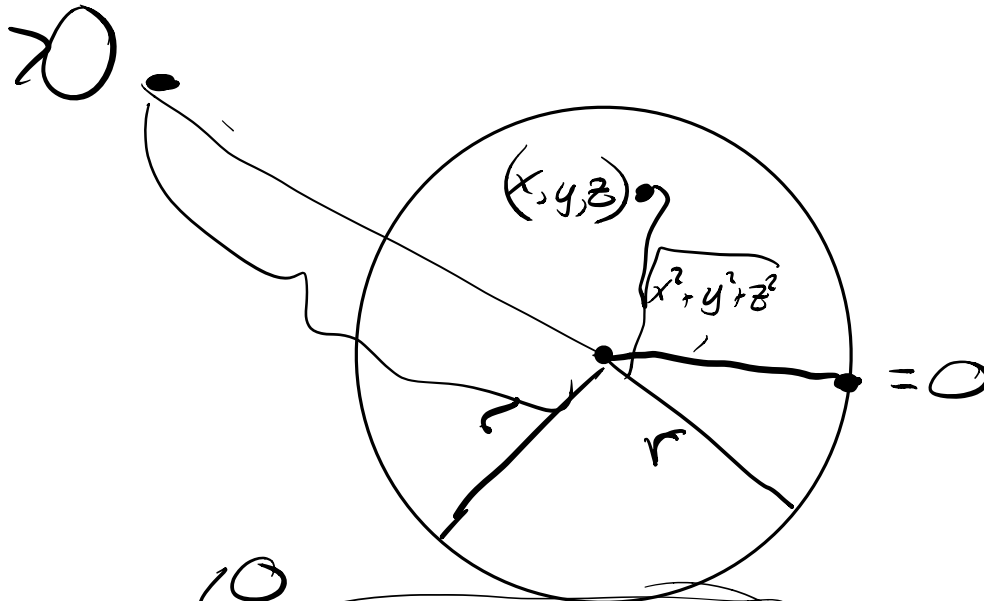
$$\sqrt{x^2 + y^2 + z^2} = r$$

or:

$$x^2 + y^2 + z^2 - r^2 = 0$$

Geometric Intuition: LHS relates to a 3D point's **signed squared distance** from sphere's surface.

Ray-Sphere Intuition: Geometric



$$\rightarrow x^2 + y^2 + z^2 - r^2 = 0$$

<https://www.google.com/search?client=firefox-b-1-d&q=plot+x%5E2+%2B+y%5E2+-+1>

Ray-Sphere Intersection: Algebraic

Whiteboard / notes.

Ray-Sphere intersection

- For now, assume unit sphere centered at the origin. See 4.4.1 for general derivation.

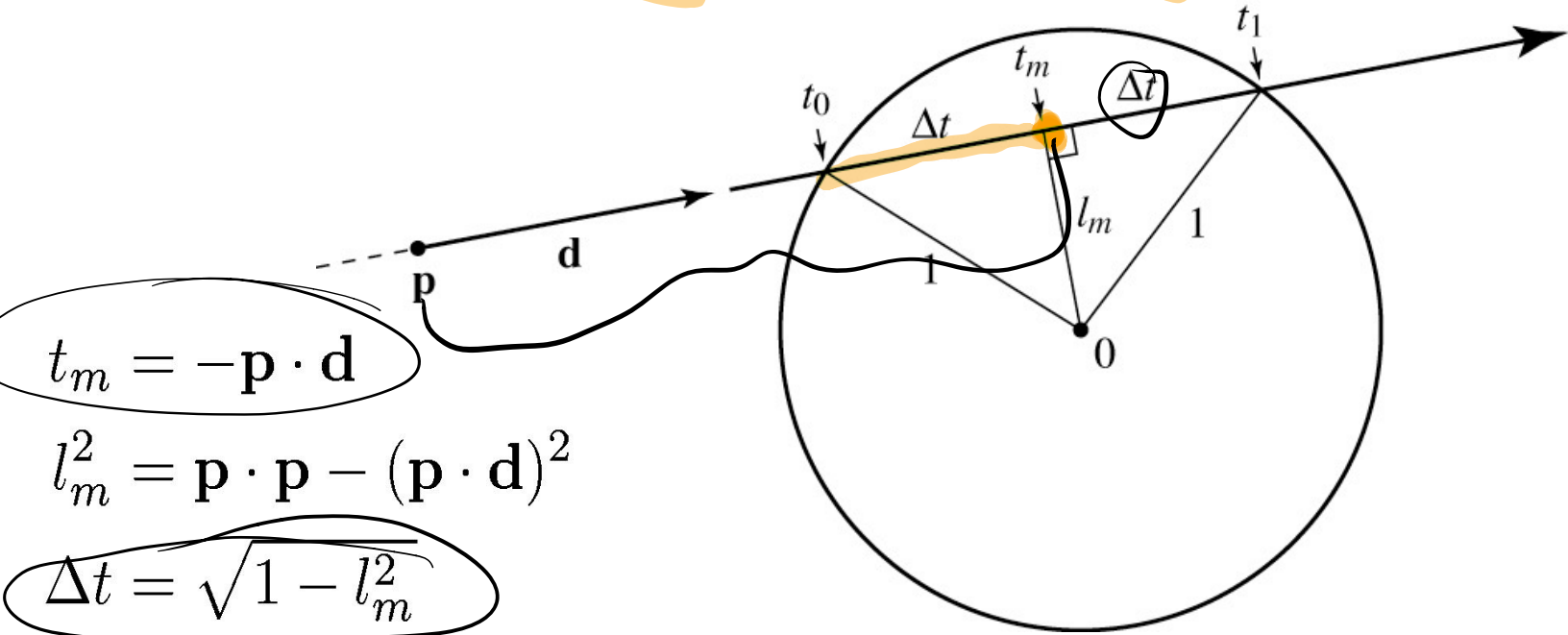
$$\nearrow t = \frac{-\mathbf{d} \cdot \mathbf{p} \pm \sqrt{(\mathbf{d} \cdot \mathbf{p})^2 - (\mathbf{d} \cdot \mathbf{d})(\mathbf{p} \cdot \mathbf{p} - 1)}}{\mathbf{d} \cdot \mathbf{d}}$$

If \mathbf{d} is unit-length:

$$t = -\mathbf{d} \cdot \mathbf{p} \pm \sqrt{(\mathbf{d} \cdot \mathbf{p})^2 - \mathbf{p} \cdot \mathbf{p} + 1}$$

Geometric Intuition

$$t = -\mathbf{d} \cdot \mathbf{p} \pm \sqrt{(\mathbf{d} \cdot \mathbf{p})^2 - \mathbf{p} \cdot \mathbf{p} + 1}$$



$$t_m = -\mathbf{p} \cdot \mathbf{d}$$

$$l_m^2 = \mathbf{p} \cdot \mathbf{p} - (\mathbf{p} \cdot \mathbf{d})^2$$

$$\Delta t = \sqrt{1 - l_m^2}$$

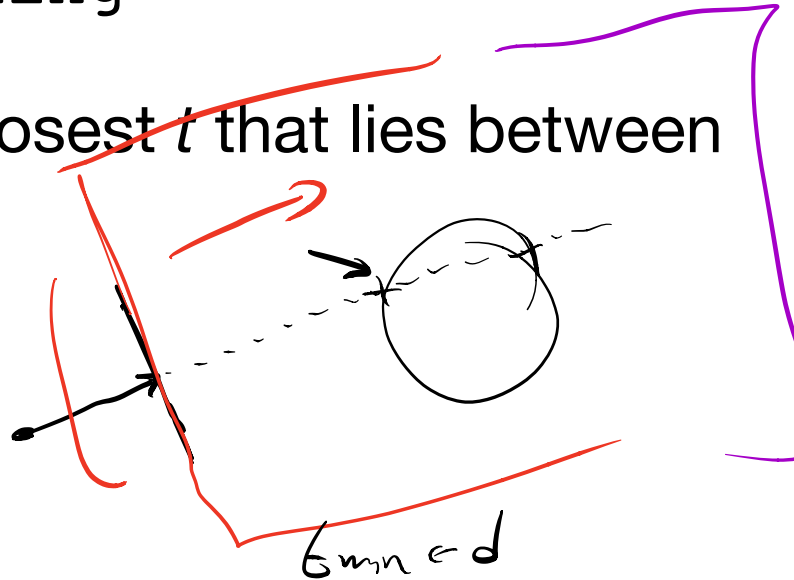
$$= \sqrt{(\mathbf{p} \cdot \mathbf{d})^2 - \mathbf{p} \cdot \mathbf{p} + 1}$$

$$t_{0,1} = t_m \pm \Delta t = -\mathbf{p} \cdot \mathbf{d} \pm \sqrt{(\mathbf{p} \cdot \mathbf{d})^2 - \mathbf{p} \cdot \mathbf{p} + 1}$$

Ray-Sphere: Code Sketch

```
function ray_intersect(ray, sphere, tmin, tmax):
```




- Use above math to find $\pm t$
- If none, return nothing
- Otherwise, return closest t that lies between $tmin$ and $tmax$



Ray-Scene: Code Sketch

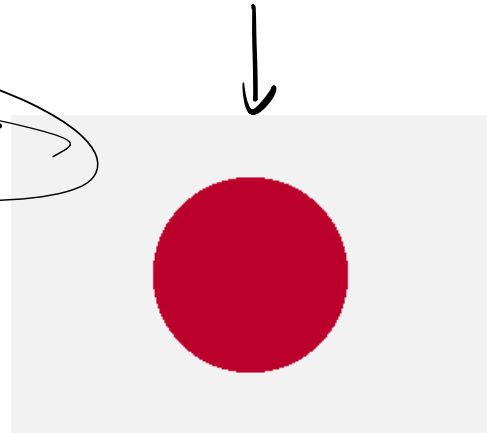
Brute force: check all objects.

There are better ways - more on this later.

```
find_intersection(ray, scene):  
    closest_t = Inf   
    closest_obj = nothing   
    for obj in scene:  
         t = ray_intersect(ray, obj, 1, closest_t)  
        if obj != nothing:  
            closest_t = t  
            closest_obj = surf  
    return closest_t, closest_obj
```

Ray Tracing: Code Sketch

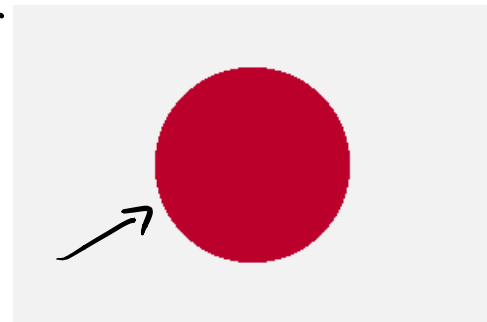
```
scene = model_scene() ←  
for each pixel (i,j):  
    ray = get_view_ray(i, j) ←  
    t, obj = find_intersection(ray, scene)  
    if obj != nothing: ↙  
        canvas[i,j] = obj.color  
    else: ↘  
        canvas[i,j] = scene.bgcolor
```



Next time...

```
scene = model_scene()  
for each pixel (i,j):  
    ray = get_view_ray(i, j)  
    t, obj = find_intersection(ray, scene)  
    if obj != nothing:  
        canvas[i,j] = obj.color  
    else:  
        canvas[i,j] = scene.bgcolor
```

Let's work on this.



Problems

- Write ray intersection code for axis-aligned rectangles.
- Model an empty Cornell box.

