CSC 433
Computer Graphics

Alon Efrat
Credit: Joshua Levine

Lecture 09
Ray Tracing

Today's Agenda

- Reminders:
  - A02 due!
  - A03 posted!
- Goals for today:
  - Begin our discussing of 3D rendering with Ray Tracing

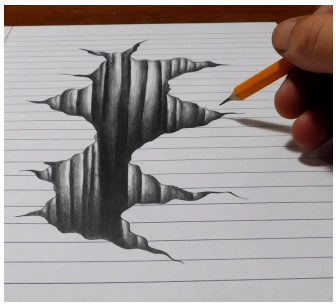
Rendering

What is Rendering?

*"**Rendering** is the task of taking three-dimensional objects and producing a 2D image that shows the objects as viewed from a particular viewpoint"*

Two Ways to Think About How We Make Images

- Drawing



- Photography

Two Ways to Think About Rendering

- Object-Ordered
- Decide, for every object in the scene, its contribution to the image

- Image-Ordered
- Decide, for every pixel in the image, its contribution from every object

Two Ways to Think About Rendering

- Object-Ordered or **Rasterization**

```
for each object {
  for each image pixel {
    if (object affects pixel)
    {
      do something
    }
  }
}
```

- Image-Ordered or **Ray Tracing**

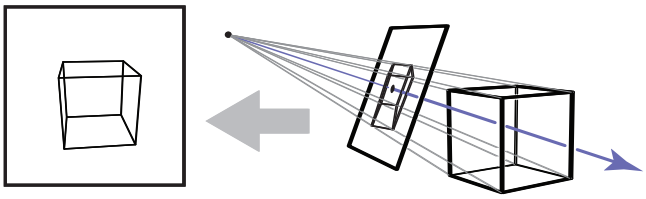
```
for each image pixel {
  for each object {
    if (object affects pixel)
    {
      do something
    }
  }
}
```

TODAY

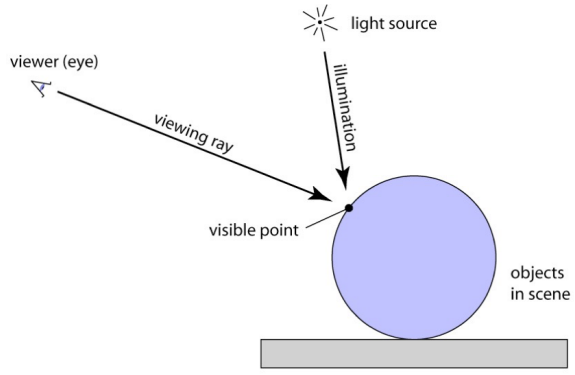
Basics of Ray Tracing

Idea of Ray Tracing

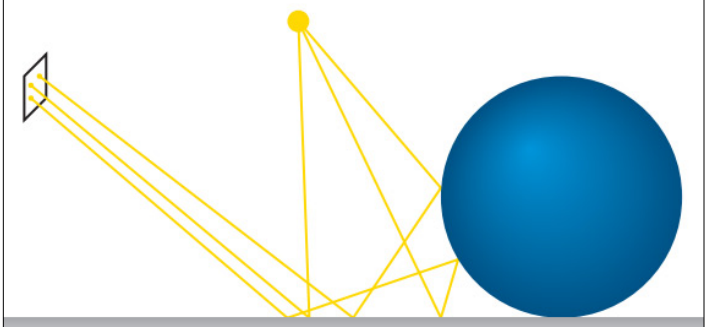
- Ask first, for each pixel: what belongs at that pixel?
- Answer: The set of objects that are visible if we were standing on one side of the image looking into the scene



Key Concepts, in Diagram

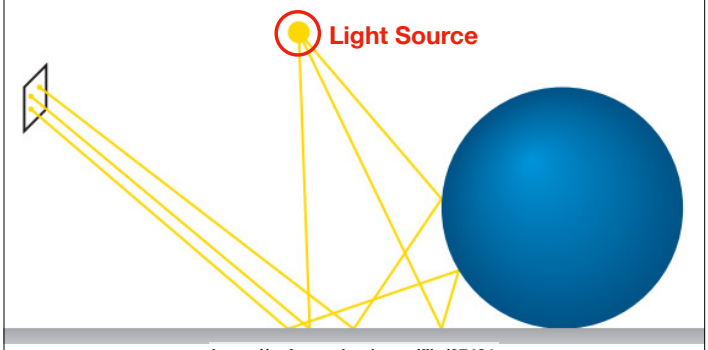


Idea: Using Paths of Light to Model Visibility



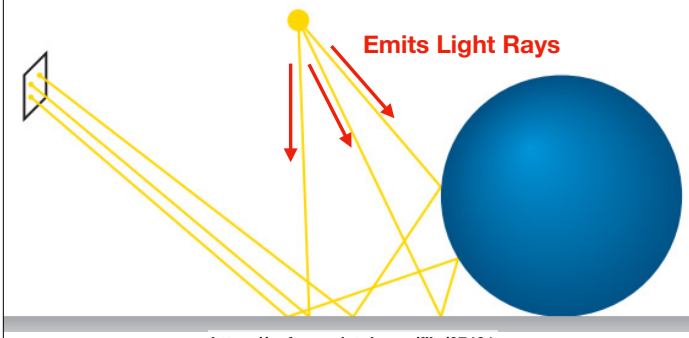
https://software.intel.com/file/37491

Using Paths of Light to Model Visibility



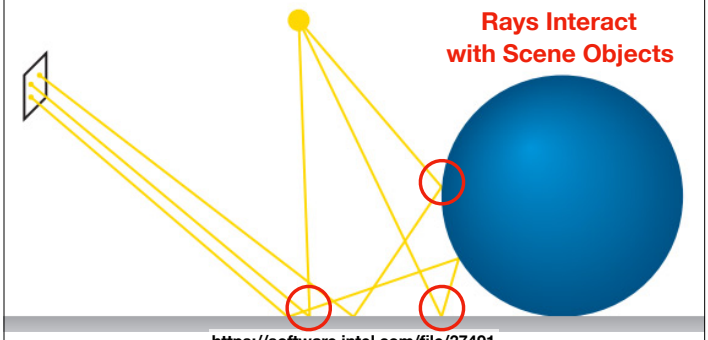
https://software.intel.com/file/37491

Using Paths of Light to Model Visibility



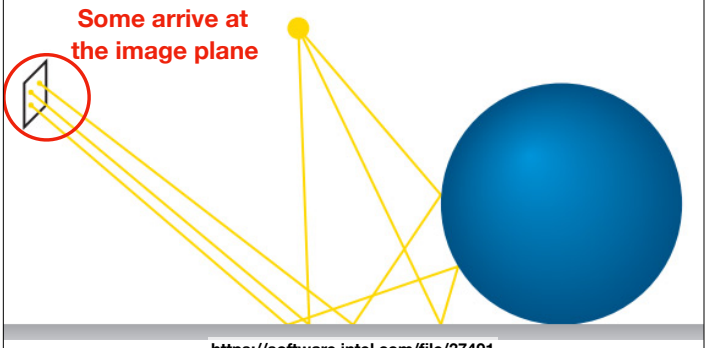
https://software.intel.com/file/37491

Using Paths of Light to Model Visibility



https://software.intel.com/file/37491

Using Paths of Light to Model Visibility



https://software.intel.com/file/37491

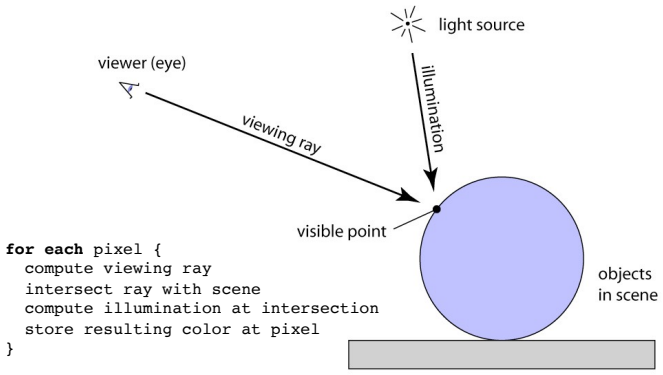
Using Paths of Light to Model Visibility

https://software.intel.com/file/37491

Forwarding vs Backward Tracing

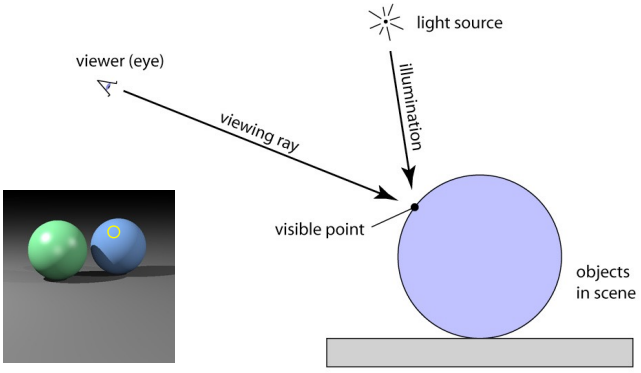
- Idea: Trace rays from light source to image
  - This is slow!
- Better idea: Trace rays **from** image **to** light source

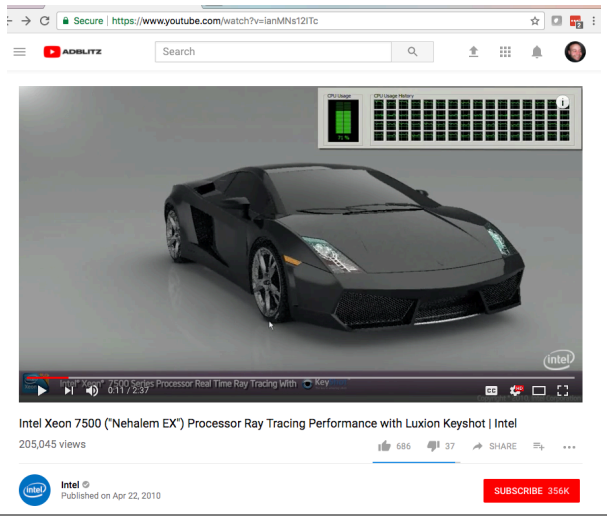
Ray Tracing Algorithm




```
for each pixel {
  compute viewing ray
  intersect ray with scene
  compute illumination at intersection
  store resulting color at pixel
}
```

Ray Tracing Algorithm





Cameras and Perspective

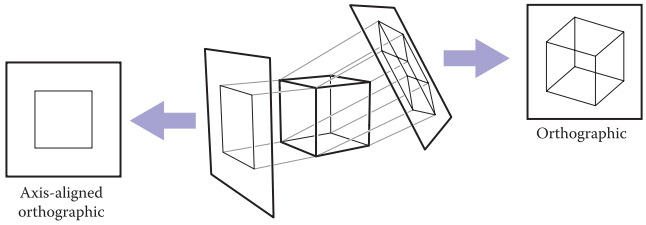
```
for each pixel {
  compute viewing ray
  intersect ray with scene
  compute illumination at intersection
  store resulting color at pixel
}
```

Linear Perspective

- Standard approach is to project objects to an image plane so that straight lines in the scene stay straight lines on the image
- Two approaches:
  - Parallel projection: Results in orthographic views
  - Perspective projection: Results in perspective views

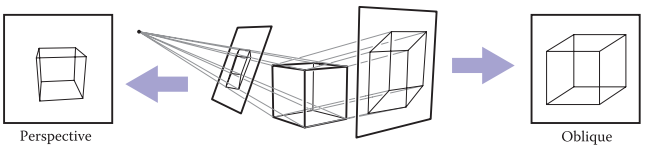
Orthographic Views

- Points in 3D are moved along parallel lines to the image plane.
- Resulting view determined solely by choice of projection direction and orientation/position of image plane



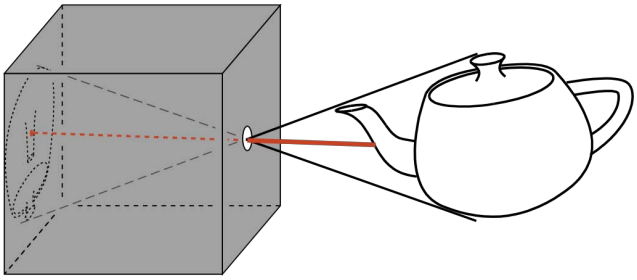
Perspective Views

- But, objects that are further away should look smaller!
- Instead, we can project objects through a single viewpoint and record where they hit the plane.



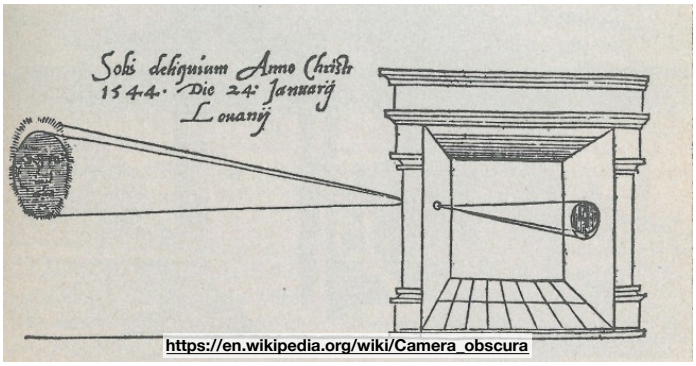
Pinhole Cameras

- Idea: Consider a box with a tiny hole. All light that passes through this hole will hit the opposite side
- Produced image inverts

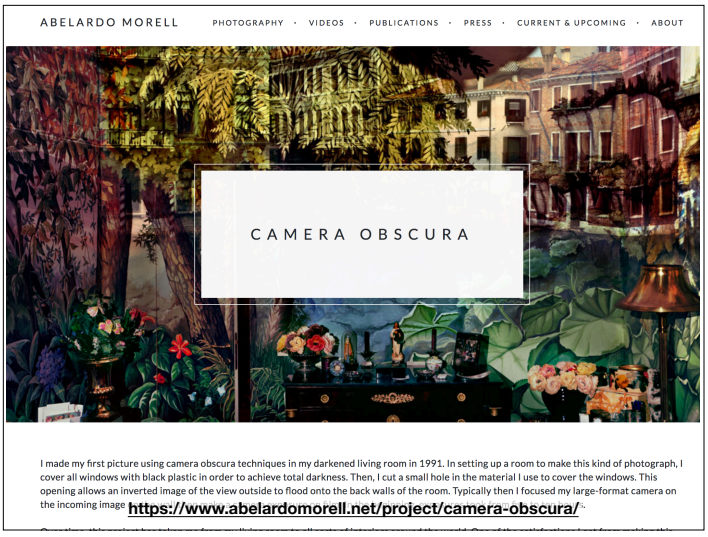


Camera Obscura

- Gemma Frisius, 16th century



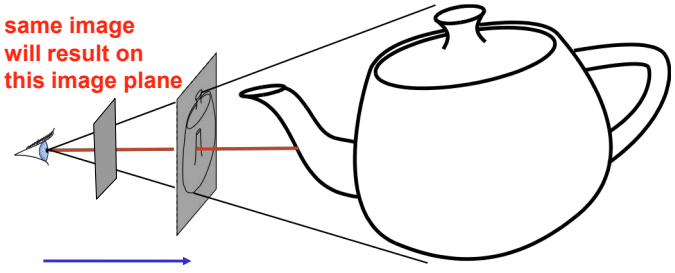
https://en.wikipedia.org/wiki/Camera_obscura



https://www.abelardomorell.net/project/camera-obscura/

Simplified Pinhole Cameras

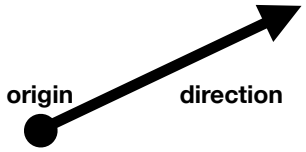
- Instead, we can place the eye at the pinhole and consider the eye-image pyramid (sometimes called **view frustum**)



Defining Rays

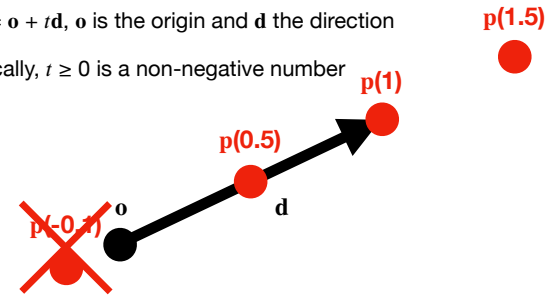
Mathematical Description of a Ray

- Two components:
  - An **origin**, or a position that the ray starts from
  - A **direction**, or a vector pointing in the direction the ray travels
    - Not necessarily unit length, but it's sometimes helpful to think of these as normalized

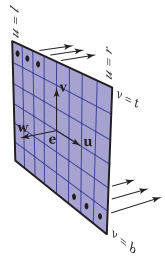


Mathematical Description of a Ray

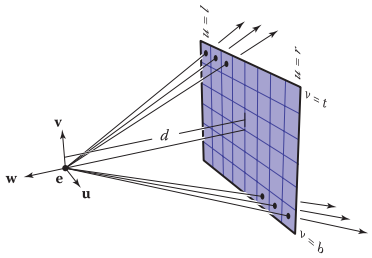
- Rays define a family of points, $\mathbf{p}(t)$, using a **parametric** definition
- $\mathbf{p}(t) = \mathbf{o} + t\mathbf{d}$, $\mathbf{o}$ is the origin and $\mathbf{d}$ the direction
- Typically, $t \geq 0$ is a non-negative number



Orthographic vs. Perspective Rays



Parallel projection
same direction, different origins



Perspective projection
same origin, different directions

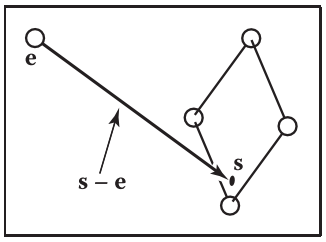
Defining o and d in Perspective Projection

- Given a viewpoint, **e**, and a position on the image plane, **s**

  $\mathbf{o} = \mathbf{e}$

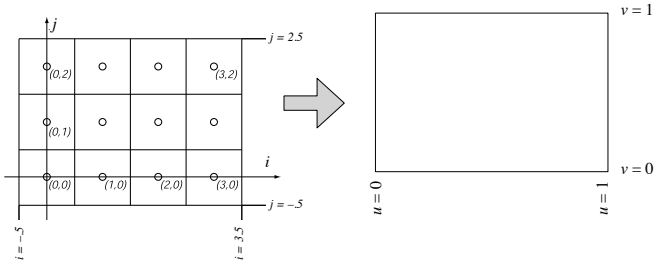
  $\mathbf{d} = \mathbf{s} - \mathbf{e}$

- And thus $\mathbf{p}(t) = \mathbf{e} + t(\mathbf{s} - \mathbf{e})$



Pixel-to-Image Mapping

- Exactly where are pixels located? Must convert from pixel coordinates (i,j) to positions in 3D space (u,v,w)
- What should w be?



$$u = (i + 0.5)/n_x$$
$$v = (j + 0.5)/n_y$$

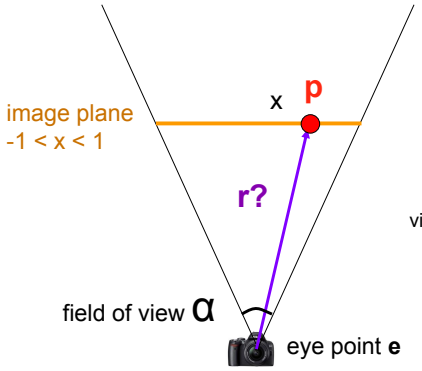
What components define a camera?

think-pair-share
(but you might want to try to draw a picture too)

Camera Components

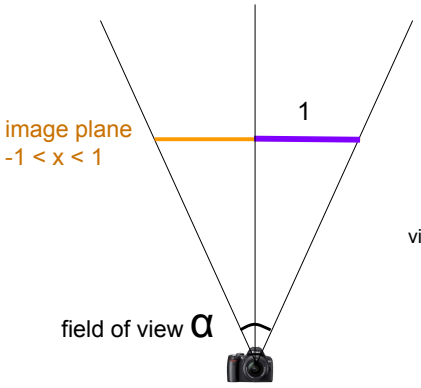
- Definition of an image plane
  - Both in terms of pixel resolution AND position in 3D space or more frequently in **field of view** and/or **distance**
- Viewpoint
- View direction
- Up vector

Ray Generation in 2D



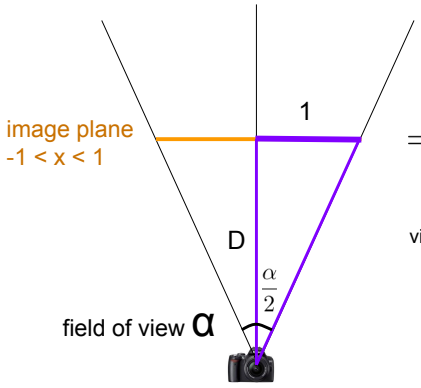
p is point on image plane at coordinate x, we want to know the direction of the ray **r**

Ray Generation in 2D



What is the distance D to the screen so that the normalized coordinates go to 1?

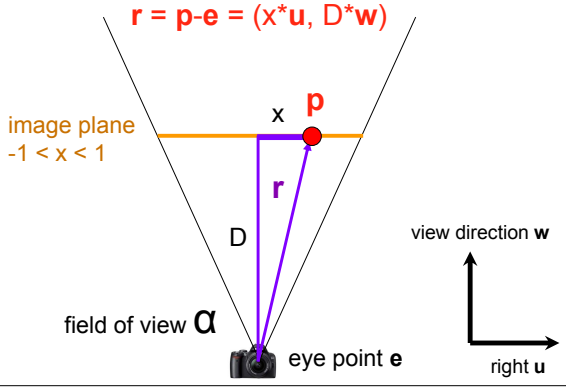
Ray Generation in 2D



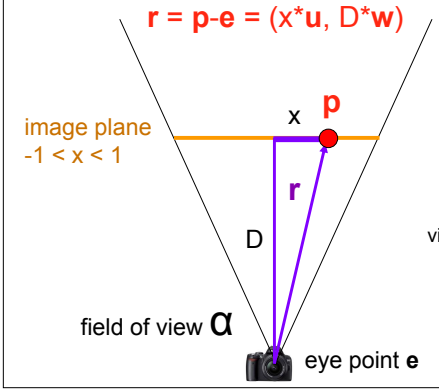
$$\tan\frac{\alpha}{2} = \frac{1}{D}$$

$$\Rightarrow D = \frac{1}{\tan(\alpha/2)}$$

Ray Generation in 2D



Ray Generation in 2D



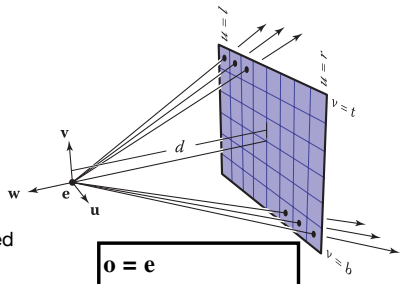
then we just normalize **r** to get the ray

$$\boldsymbol{P}(t) = \boldsymbol{e} + t\boldsymbol{d}$$

$$(\boldsymbol{d} = \boldsymbol{r}/\|\boldsymbol{r}\|)$$

From 2D to 3D

- Moving from 2D to 3D is essentially the same thing once you can define the positions of pixels in uvw space
- Following the convention of the book, **w** is the negated viewpoint vector
- The up vector, **v**, can be used to define a local coordinate space by computing **u**

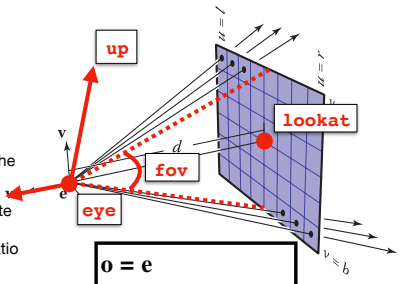


$\mathbf{o} = \mathbf{e}$

$\mathbf{d} = -d\mathbf{w} + u\mathbf{u} + v\mathbf{v}$

In the Assignment

- An `eye` position
- A position to `lookat`, which is centered in the image
  - **w** can be defined use `eye` and `lookat` as well as d
- An `up` vector, not necessarily **v**!
- A `fov_angle` for the vertical FOV
  - The FOV defines the **height** of the image plane in world space
  - You can then use this to compute the **width** of the image plane in world space using the aspect ratio (`rows/columns`) of the image
- Using the number of `rows/columns` you can then sample u, v



$\mathbf{o} = \mathbf{e}$

$\mathbf{d} = -d\mathbf{w} + u\mathbf{u} + v\mathbf{v}$

Intersecting Objects

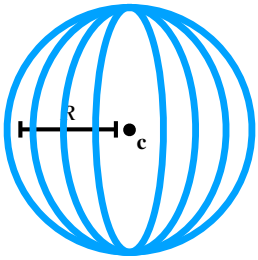
```
for each pixel {
  compute viewing ray
  intersect ray with scene
  compute illumination at intersection
  store resulting color at pixel
}
```

Defining a Sphere

- We can define a sphere of radius R, centered at position **c**, using the implicit form

$$f(\mathbf{p}) = (\mathbf{p} - \mathbf{c}) \cdot (\mathbf{p} - \mathbf{c}) - R^2 = 0$$

- Any point **p** that satisfies the above lives on the sphere



Ray-Sphere Intersection

- Two conditions must be satisfied:
  - Must be on a ray: $\mathbf{p}(t) = \mathbf{o} + t\mathbf{d}$
  - Must be on a sphere: $f(\mathbf{p}) = (\mathbf{p} - \mathbf{c}) \cdot (\mathbf{p} - \mathbf{c}) - R^2 = 0$
- Can substitute the equations and solve for t in $f(\mathbf{p}(t))$:

$$(\mathbf{o} + t\mathbf{d} - \mathbf{c}) \cdot (\mathbf{o} + t\mathbf{d} - \mathbf{c}) - R^2 = 0$$

- Solving for t is a quadratic equation

Ray-Sphere Intersection

- Solve $(\mathbf{o} + t\mathbf{d} - \mathbf{c}) \cdot (\mathbf{o} + t\mathbf{d} - \mathbf{c}) - R^2 = 0$ for t:
- Rearrange terms:

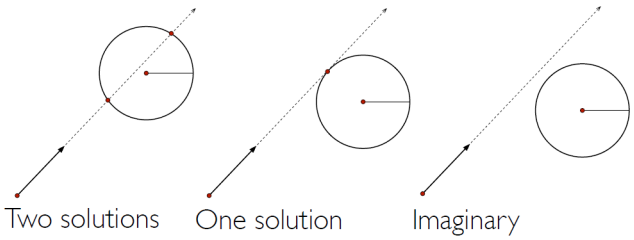
$$(\mathbf{d} \cdot \mathbf{d})t^2 + (2\mathbf{d} \cdot (\mathbf{o} - \mathbf{c}))t + (\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c}) - R^2 = 0$$

- Solve the quadratic equation $At^2 + Bt + C = 0$ where
  - $A = (\mathbf{d} \cdot \mathbf{d})$
  - $B = 2^*\mathbf{d} \cdot (\mathbf{o} - \mathbf{c})$
  - $C = (\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c}) - R^2$

Discriminant, D = B²-4*A*C
Solutions must satisfy:
t = (-B ± √(D)) / 2A

Ray-Sphere Intersection

- Number of intersections dictated by the discriminant
- In the case of two solutions, prefer the one with lower *t*



More Examples Of What Can Be Done With Ray Tracing

- https://developer.nvidia.com/optix
- https://embree.github.io/gallery.html

Next Lecture Required Reading

- FOCG, Ch. 4.5-4.9