Announcements

• A1 is out!

• HW1 is forthcoming

• Getting help:
  • #q-and-a channel on Discord
  • email me
Where were we?

Pseudocode for 3D graphics:

Create a model of a scene
Render an image of the model

Triangle(a, b, c)
Sphere(c, r)
meshgen.jl (A1)
Where were we?

Pseudocode for 3D graphics:

Create a model of a scene
Render an image of the model

For each pixel:
if inside triangle:
  color pixel
Two Rendering Algorithms

for each object in the scene {
    for each pixel in the image {
        if (object affects pixel) {
            do something
        }
    }
}  

for each pixel in the image {
    for each object in the scene {
        if (object affects pixel) {
            do something
        }
    }
}  

object order
or
rasterization

image order
or
ray tracing

Starting here
Two Rendering Algorithms

for each object in the scene {
  for each pixel in the image {
    if (object affects pixel) {
      do something
    }
  }
}

Q: Which of these did we do in A0?

object order
or
rasterization

for each pixel in the image {
  for each object in the scene {
    if (object affects pixel) {
      do something
    }
  }
}

image order
or
ray tracing
Today

Render an image of the model

• What does image mean? ✓

• What does render mean?

• Beginnings of image-order rendering (i.e., ray tracing)

• Where do rays come from?
How do we make images?

- camera?
- $\text{canvas}(i,j) = \text{color}$
- draw-tri, etc
- MS Paint
- Paint brushes
How do we make images?

- IRL:
  - pencils, paintbrushes, watercolors, etc
  - eyes
  - cameras

- On computers:
  - MS paint
  - manually writing pixel values into Julia arrays
  - virtual cameras
The Camera Conundrum:

The world is 3D

Images are 2D

we gotta lose a dimension somehow
Projections: ways to lose a dimension

- The picture-frame method is called *perspective projection*.

- Key property of perspective: all *viewing rays* originate at a single point, the *center of projection*, or eye.
Projections: ways to lose a dimension

- Another common one is parallel projection

- Key property of parallel projections: all viewing rays are parallel
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
A ray is half a line.

We'll describe rays using:

• An origin (p) where the ray begins
• A direction (d) in which the ray goes

\[ \mathbf{r}(t) = \mathbf{p} + t\mathbf{d} \]

- This is a parametric equation: it **generates** points on the line
- The set of points with \( t > 0 \) gives all points on the ray
Viewing Rays

are determined by the **position** and **orientation** of the camera

- For perspective projection, viewing rays originate at the **eye**.
- The direction varies depending on the pixel.
Let's start with a simple camera

- Eye is at the origin (0, 0, 0)
- Looking down the negative z axis
- Viewport is parallel to the xy plane
- \( v_h = v_w = 1 \)
- \( d = 1 \)

What is the 3D viewing ray for pixel (i, j)?
\[ X(j) = \left( j - \frac{1}{2} \right) - \frac{1}{2} \]

\[ y(i) = -\left( \frac{i - \frac{1}{2}}{H} - \frac{1}{2} \right) \]
\[ p = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \]

\[ d = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \]

\[ \rho + \epsilon \in p \]
Viewing rays for the canonical camera

\[ x = \frac{j - \frac{1}{2}}{W} - \frac{1}{2} \]
\[ y = -\left( \frac{i - \frac{1}{2}}{H} - \frac{1}{2} \right) \]

Origin \((p)\): \((0, 0, 0)\)
Direction \((d)\): \((x, y, -1)\)
Problems - in groups

1. Generate an example viewing ray

2. Intersect the ray with a plane in the scene

3. Generalize camera model by removing assumptions:
   - Eye is not at the origin (0, 0, 0)
   - vh \neq vw \neq 1
   - d \neq 1
What if I want to point the camera somewhere else?

The camera's pose is defined by a **coordinate frame**:
- **u** points right from the eye
- **v** points up from the eye
- **w** points back from the eye

Given this, we can generate a viewing ray as follows:

1. Turn (i,j) into u, v instead of x, y (same math)
2. Viewing ray in (x, y, z) world is:
   - origin = eye
   - direction = $u^*u + v^*v + -d^*w$