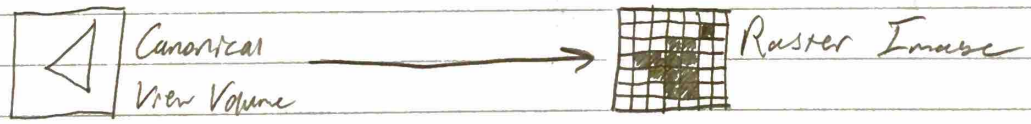
L03 - Rendering Algs

- Visibility Problem
  ↳ "What parts of the object in world falls on which pixels of the screen?"

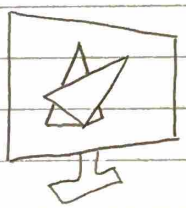
Rendering Algs

- Rasterization Algs
  - Painter's Alg → easiest/simplest
  - Z-Buffer → most common.
  - A-Buffer
  - REYES
- Ray Tracing
  - ray casting
  - Path tracing
  - ...

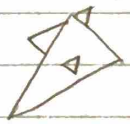
Rasterization



Painter's Alg



- "Paints" objects from far end to near end.
- Problems:
  - needs sorting
  - Cannot handle intersecting Geometry



Z-Buffer Rasterization

- Stores a "Z" or depth value for each pixel → RGBA + Z
  ↳ as it renders more objects, it determines pixel-by-pixel, which one goes infront of the other
- Can still do anti-aliasing
- What if one triangle is semi-transparent?
  ↳ Can handle this if draw the semi-transparent ones last
  ↳ GPU's can do alpha-blending

→ If you render a transparent primitive first, then an opaque one behind it, the Z-buffer Alg will say, since the opaque primitive is behind the transparent one (in depth/z val), it is not going to show up, even though the primitive in front of it is transparent

Summary:
- ✓ can handle intersecting geometry
- × needs sorting for transparency (opaque then transparent primitives)

A-Buffer Rasterization

- ✓ can handle intersecting geometry
- ✓ Supports order-dependent transparency
- × requires more (dynamic) memory

- generally, GPU's are designed for Z-buffer rasterization (not A-buffer)
  - ↳ Can implement on GPU still, but would need custom software
- for each pixel, store a list of "fragments" (not just a single color & depth value) which contains everything that the pixel encounters
  - ↳ when new objects need to be rendered, their fragments are placed accordingly in the fragment list.
  - ↳ fragment list: [depth + RGBA + Coverage] ... [depth + RGBA + Coverage]

REYES

↳ Render Everything You Ever Saw

- Splits objects into smaller & smaller pieces, until they become smaller than a pixel
- Can get lots of high detail realistic images using REYES
- Pixar uses REYES for a while up until recently

Rasterization vs. Ray Tracing

Rasterization:
- For each primitive:
  - Find Pixel samples

Ray Tracing:
- For each pixel sample
  - Find the closest primitive

- Can still employ some tricks with Rasterization to get some secondary effects

Ray Tracing

- for each pixel, trace a ray from it into the view volume & see where it lands. Depending on where it lands, will determine it's pixel sample
- Why Ray Tracing?
  - ↳ can do a lot more with it like reflections
  - ↳ primary ray from camera, secondary ray off of reflection (both rays are treated the same)
  - ↳ can generate much more realistic images @ high res (usually offline)

Rasterization vs. Ray Tracing

Rasterization; (linear complexity)
- for each primitive → linear mem. access
- find pixel samples → fast

Ray Tracing; (logarithmic complexity)
- for each pixel sample → Random mem. access
- find closes primitive → Slow

- although ray tracing in theory has faster complexity, still struggles to catch up w/ Rasterization even in large scenes because of random mem. access
- Nonetheless, Rasterization can only handle primary visibility, while ray tracing can also handle reflections

Rasterization + Ray Tracing → a common concept today

- for primary visibility → Rasterization
- for secondary effects; → Ray Tracing
  - reflections/refractions
  - shadows
  - realistic illumination
  - ...

- for offline rendering, might as well use Ray Tracing for everything, primary visibility is the fast part anyways so doesn't matter too much which you choose