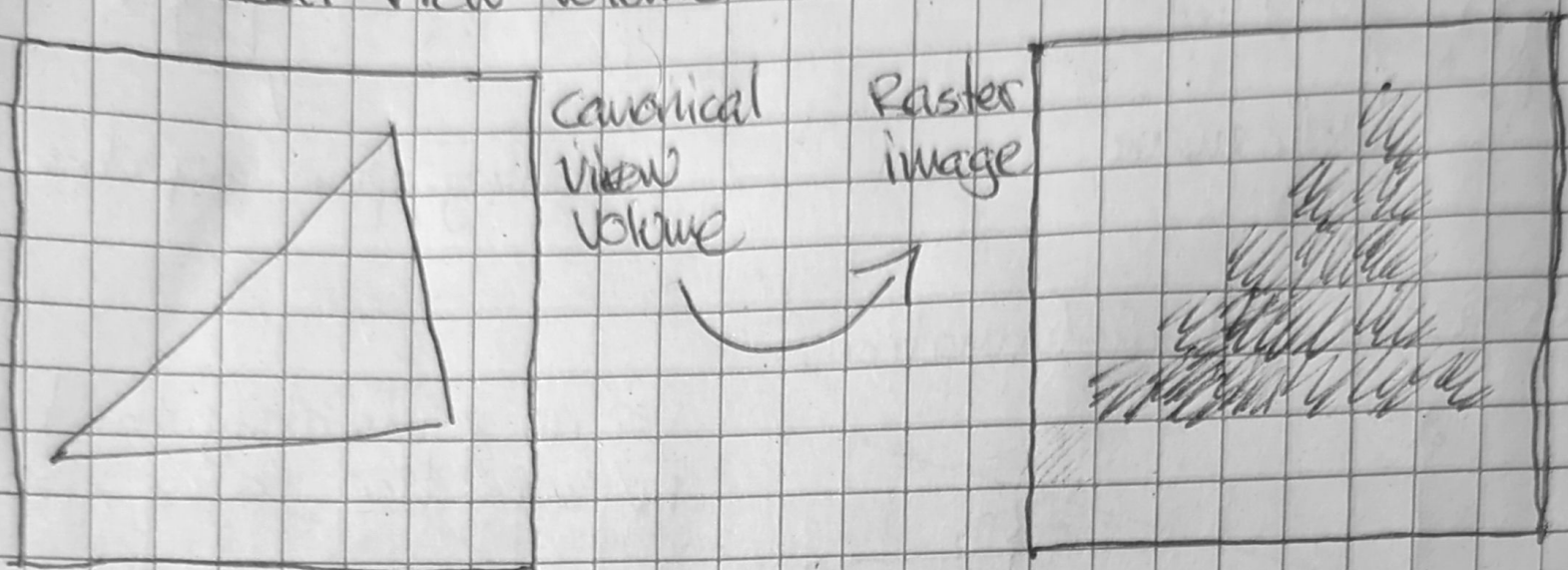
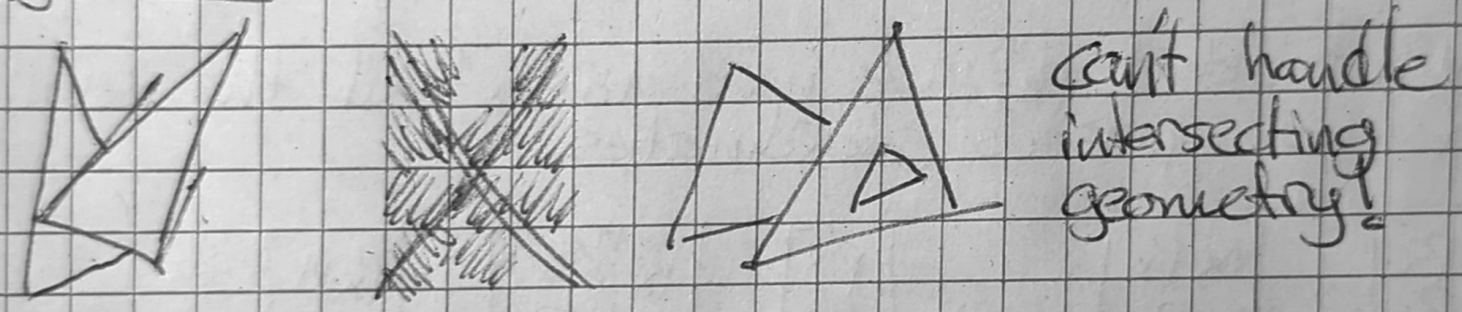
3-Rendering algorithms

Rasterization:
Main problem rasterization solves is going from canonical view volume to raster image.



A major problem rasterization needs to solve is how to draw multiple triangles. Can use painters algorithm:



→ sort then begin to draw the one farthest behind

Or z-buffer rasterization; which stores RGBA+ z/depth-value for each pixel. Solves anti-aliasing in intersecting geometries by storing multiple samples for each pixel, storing 4x RGBA+z for example (can be other like 64x), called super sampling anti-aliasing (SSAA).

SSAA is very heavy so we can use Multi-sampling anti-aliasing (MSAA), where we only store a single color for each sample group, but a different z-value like SSAA.

Z-buffer needs sorting for handling transparent triangles, because it only compares the z-value and a solid triangle can be rendered behind a transparent one and not be visible.

A-buffer rasterization can handle this better with the tradeoff that it requires much more dynamic memory, so it's not supported with hardware level on GPUs. It works by having a linked list for each pixel containing depth, RGBA and coverage for each entry/triangle.

Ray tracing

Rasterization:

```
for each primitive
    find pixel samples
```

Fast ✓

Linear memory access ✓

Linear complexity ✗

Ray tracing:

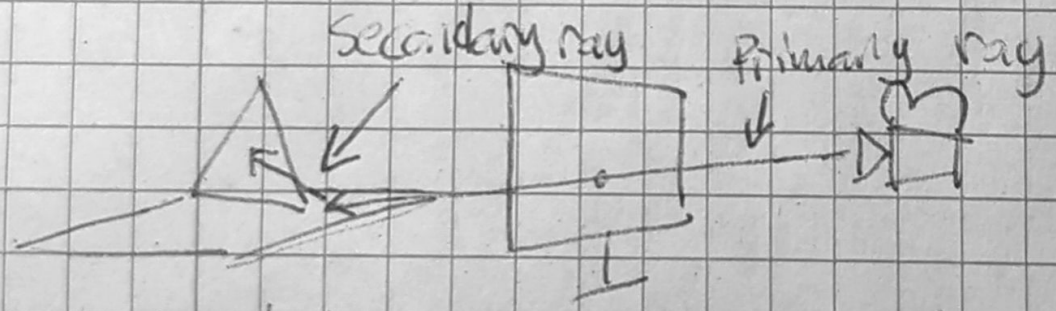
```
for each pixel sample
    find closest primitive
```

Slow ✗

Logarithmic complexity ✓

random access memory ✗

The power of ray tracing is that we can easily ~~find~~ calculate shadows, reflections, refractions and so on using the secondary ray.



Rasterization can only handle primary visibility.
Rasterization + ray tracing:

Rasterization for primary visibility and ray tracing for secondary effects. But for offline rendering only ray tracing.

Future is moving towards only ray tracing for interactive graphics.

Rasterization can simulate secondary effects with tricks.