Other Issues with Z-buffering

Z-fighting: Due to the lack of precision of depth buffer, two fragments from different objects that are close to each other (and far from the camera) will flicker back and forth on alternating frames.

- Alleviation: use 24-bit or 32-bit depth buffers or adjust near and far clipping planes for a smaller view frustum

Take a look at the example: https://youtu.be/9AcCrF_nX-I

Z-buffering is done on a pixel-by-pixel basis. If an entire object is blocked by another, it still needs to undergo all the lighting calculations and rasterization before all its pixels are discarded.

- Solutions: object culling algorithms such as BSP (binary spatial partitioning), occlusion volumes, etc.

Painter's Algorithm

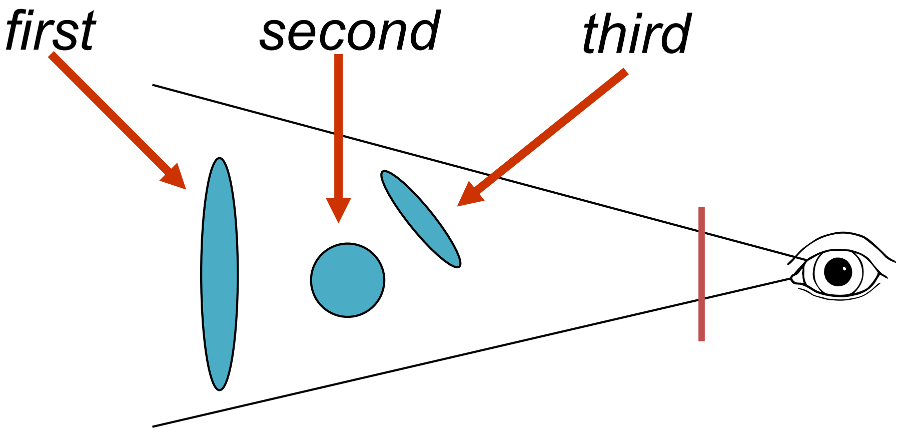
Developed thousands of years ago

- probably by cave dwellers

Draws every object in depth order

- from back to front
- near objects overwrite far objects

What could be simpler?



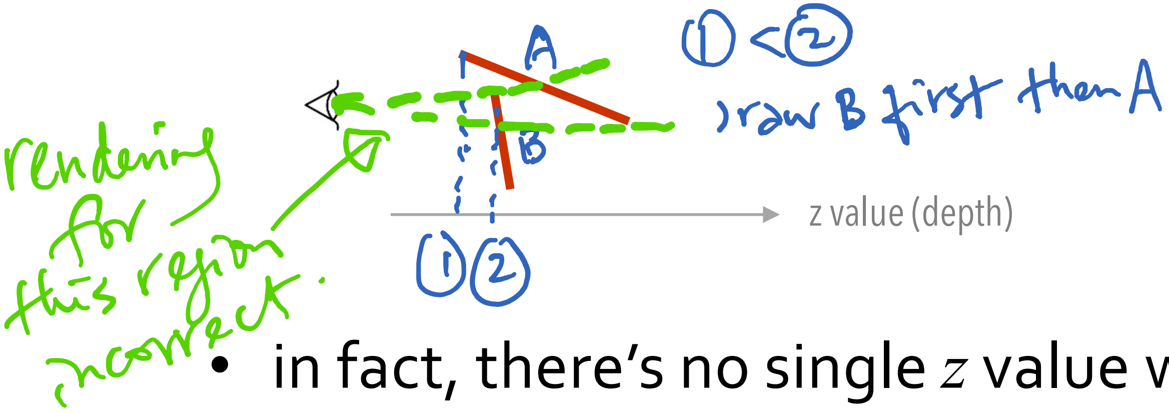
Painter's Algorithm:

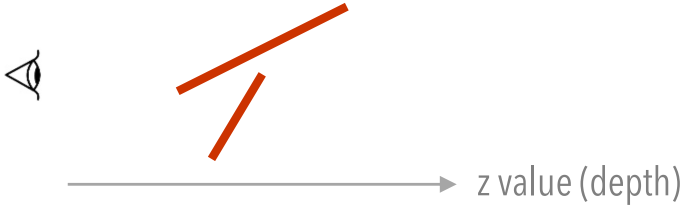
```
sort objects back to front

loop over objects
   rasterize current object
   write pixels
```

But the Catch is in the Depth Sorting

What do we sort by?

- minimum *z* value — no   maximum *z* value — no

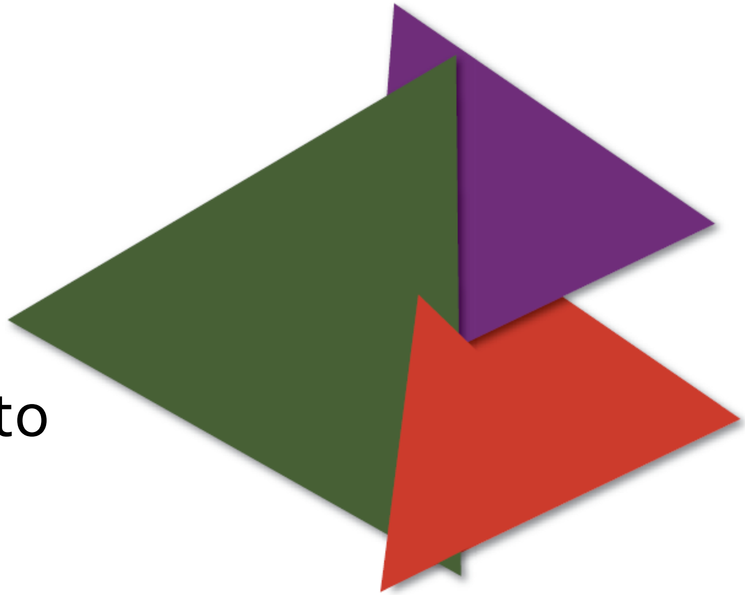




- in fact, there's no single *z* value we can sort by

Worse yet, depth ordering of objects can be cyclic

- may need to split polygons to break cycles



Looking at Painter’s Algorithm

It has some nice strengths

- the principle is very simple
- handles transparent objects nicely
  - just composite new pixels with what’s already there

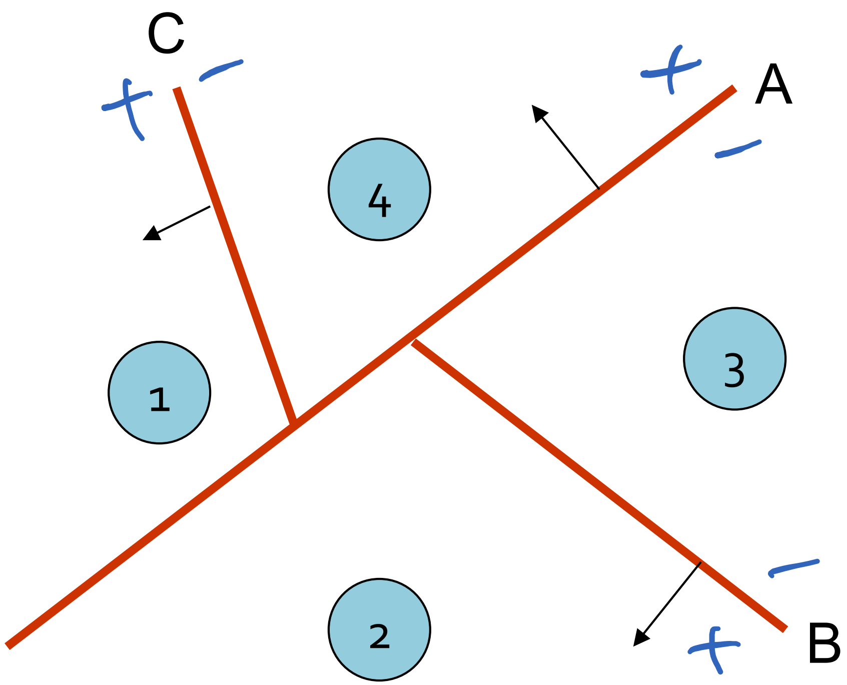
But it also has some noticeable weaknesses

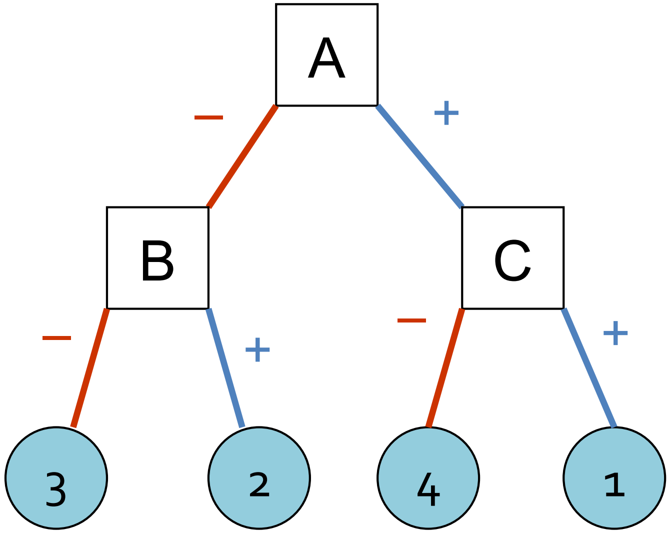
- general sorting is a little expensive — worse than *O(n)*
- need to do splitting for depth cycles, interpenetration, …

A Quick Look at BSP Trees

Recursively partition space with planes

- this defines a binary space partitioning tree
- need to split objects hit by planes

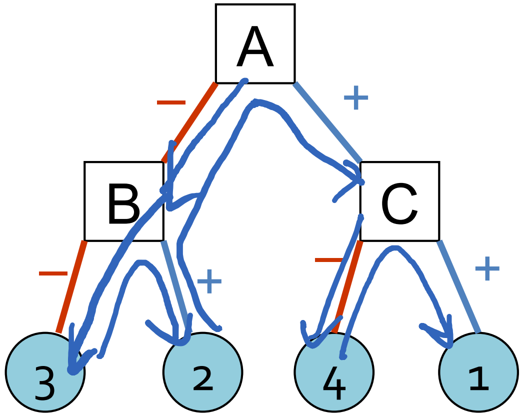




A Quick Look at BSP Trees

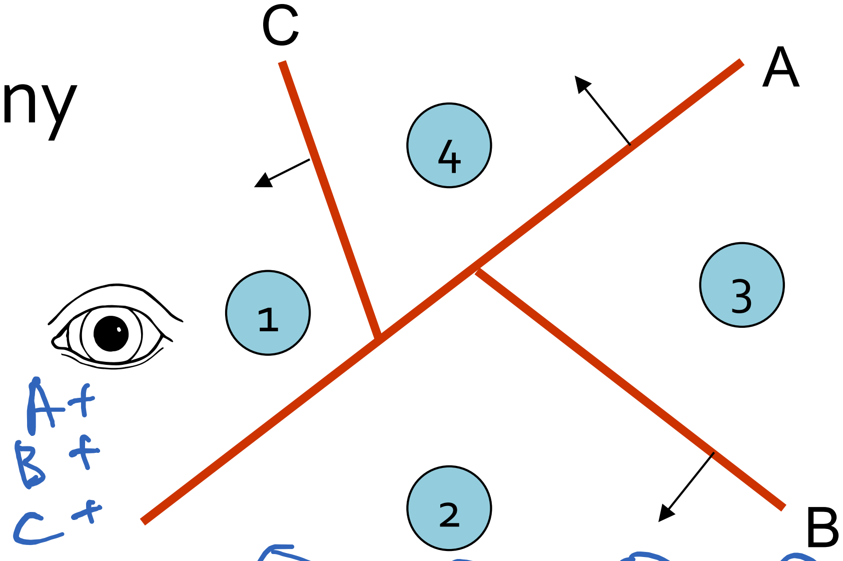
Can use this to draw scene in order (back to front for instance)

- start at root plane
- figure out which side the viewpoint is on
- descend on the opposite first
- do this recursively



Can use one BSP tree for any possible viewpoint.

View Independent Data Structures



A+
B+
C+

Order: 3 → 2 → 4 → 1

A Quick Look at BSP Trees

Can use this to draw scene in order (back to front for instance)

- start at root plane
- figure out which side the viewpoint is on
- descend on the opposite first
- do this recursively

Originally developed in early 80's

- for Painter's Algorithm, for instance
- resurrected by PC game programmers in the early 90's
  - e.g., by John Carmack for Doom
- it's quite handy if you don't have a z-buffer

Ray Casting

This is a very general algorithm

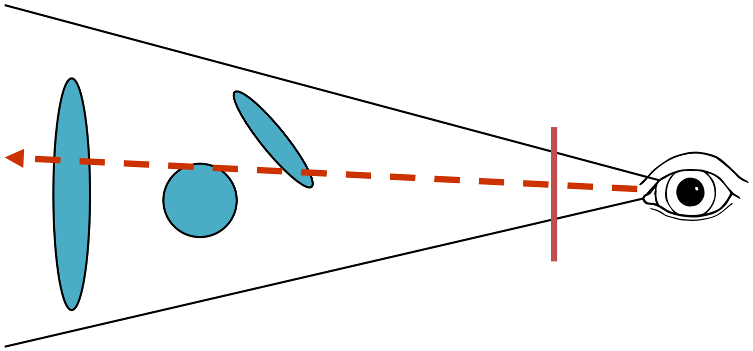
- works with any primitive we can write intersection tests for
- but it’s hard to make it run fast

We’ll come back to this idea later

- can use it for much more than visibility testing
- shadows, refractive objects, reflections, motion blur, …

Ray Casting:

```
loop over every pixel (x,y)
  shoot ray from eye through (x,y)
  intersect with all surfaces
  find first intersection point
  write pixel
```



A Classification of Visibility Algorithms

- Image-space
    - Z-buffer
    - Ray-casting
- Object-space
    - Painter’s algorithm
    - BSP

They are all view-dependent except BSP trees.