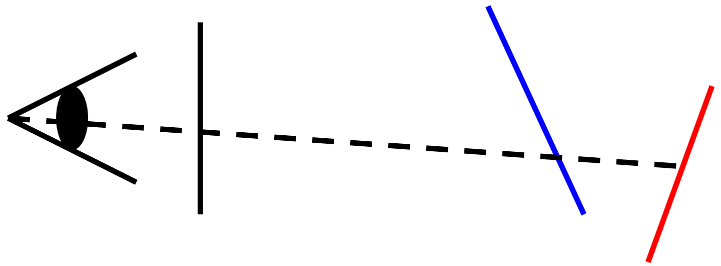
# Basic Visibility Algorithms

# The Visibility Problem



- What is the nearest surface seen at any point in the image?

- How would YOU solve this problem?

# Three of the Simplest Algorithms

### Painter’s

```
sort objects by z (back-to-front)
loop objects
    loop y
        loop x
            write pixel
```

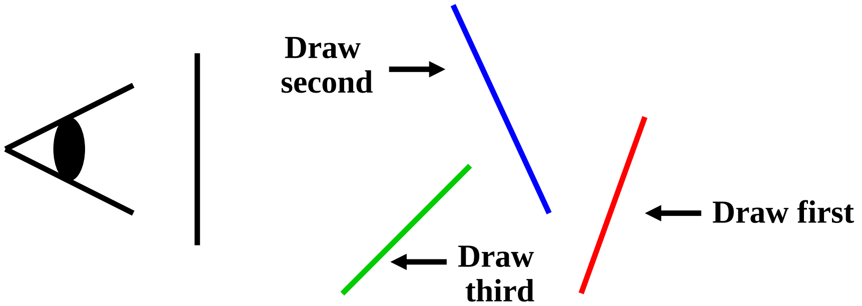
### Ray Casting

```
loop y
    loop x
        loop objects
            find object with min z
        write pixel
```

### Z-buffer

```
initialize z-buffer
loop objects
    loop y
        loop x
            if z(x,y) < zbuf[x,y]
                zbuf[x,y] = z(x,y)
                write image pixel
```

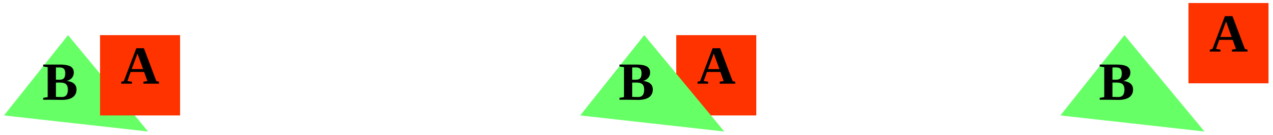
# Painter’s Algorithm



- Sort objects by depth (Z)
- Loop over objects in back-to-front order
  - – Project to image
    - » **scan convert:** image[x,y] = shade(x,y)

# Sorting Objects by Depth

Depth ordering is a *partial ordering.* Outcomes are

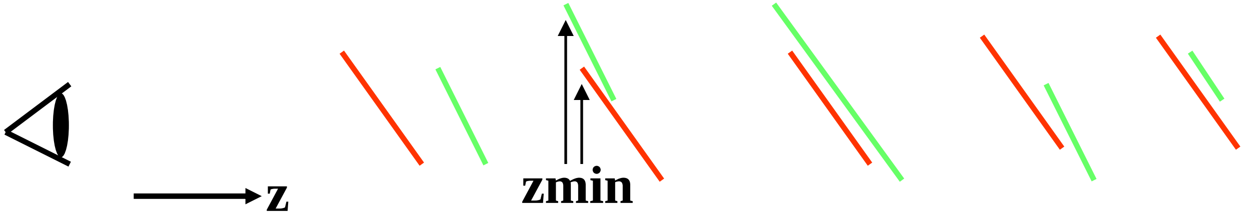


A occludes B — *draw B before A*

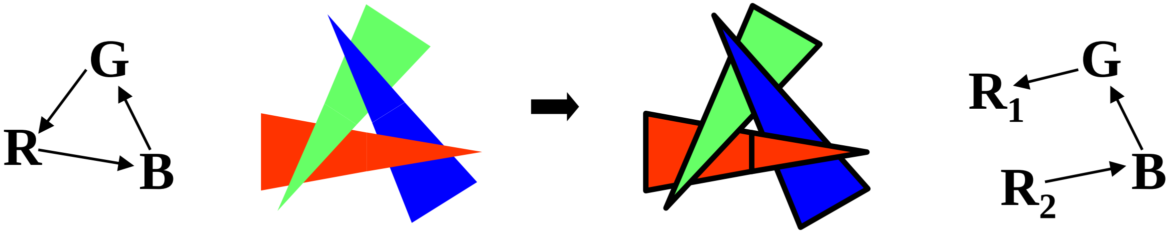
B occludes A — *draw A before B*

don't care — *draw in either order*

Sorting objects by their zmin doesn't always work! (same for zmax)



Sometimes ordering is cyclic! What to do? Split objects!



# Painter’s Algorithm

- Strengths
  - Simplicity:  draw objects one-at-a-time, scan convert each
  - Handles transparency well
- Drawbacks
  - Sorting can be expensive (slower than linear in the number of objects)
  - Clumsy when ordering is cyclic, because of need to split
  - Interpenetrating polygons need to be split, too
  - Hard to sort non-polygonal objects
- Sometimes no need to sort, or trivial
  - If objects are arranged in a grid, e.g. triangles in a height field z(x,y), such as a triangulated terrain

- Who uses it?
  - Postscript interpreters
  - OpenGL, if you don’t glEnable(GL_DEPTH_TEST); objects need to be sorted first.

# Z-Buffer Algorithm

- **Initialization**

```
loop over all x,y

        zbuf[x,y] = infinity
```

- **Drawing steps**

```
loop over all objects

    scan convert object (loop over x,y)
          if z(x,y) < zbuf[x,y]               /* compute z of this object at this pixel & test */
              zbuf[x,y] = z(x,y)   /* update z-buffer */
              image[x,y] = shade(x,y)      /* update image (typically RGB) */
```

# Z-Buffer Algorithm

- Strengths
  - Simple, no sorting or splitting
  - Easy to mix polygons, spheres, other geometric primitives
- Drawbacks
  - Can't handle transparency well
  - Need good Z-buffer resolution or you get depth ordering artifacts
    - » In OpenGL, this resolution is controlled by choice of clipping planes and number of bits for depth
    - » Choose ratio of clipping plane depths (zfar/znear) to be as small as possible
- Who uses it?
  - OpenGL, if you glEnable(GL_DEPTH_TEST);

# Ray Casting

- A very flexible visibility algorithm

```
loop y
  loop x
      shoot ray from eye point through pixel (x,y) into scene
      intersect with all surfaces, find first one the ray hits
      shade that surface point to compute pixel (x,y)'s color
```

# Comparison of Visibility Algorithms

**Painter's:**

Implementation: moderate to hard if sorting & splitting needed

Speed: fast if objects are pre-sorted, otherwise slow

Generality: sorting & splitting make it ill-suited for general 3-D rendering

**Z-buffer:**

Implementation: moderate, it can be implemented in hardware

Speed: fast, unless depth complexity is high

Generality: good but won't do transparency

**Ray Casting:**

Implementation: easy, but hard to make it run fast

Speed: slow if many objects: cost is O((#pixels)×(#objects))

Generality: excellent, can even do CSG, transparency, shadows

# *Really* Hard Visibility Problems

- Extremely high scene complexity
  - a building walkthrough
  - A fly-by of any outdoor scene
- Z-buffering requires drawing EVERY triangle for each image
  - Not feasible in real time
- Usually Z-buffering is combined with spatial data structures
  - BSP trees are common (similar concept to octrees)
- For *really* complex scenes, visibility isn’t always enough
  - Objects WAY in the distance are too small to matter
  - Might as well approximate far-off objects with simpler primitives
  - This is called geometry ***simplification*** – another *big* subject!