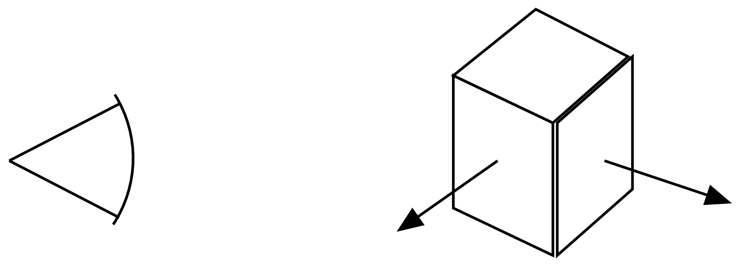
More visible surface detection

sometimes applicable to entirely different problems!

Backface culling

Object space method

Removes all polygons that are "looking away" from the camera.

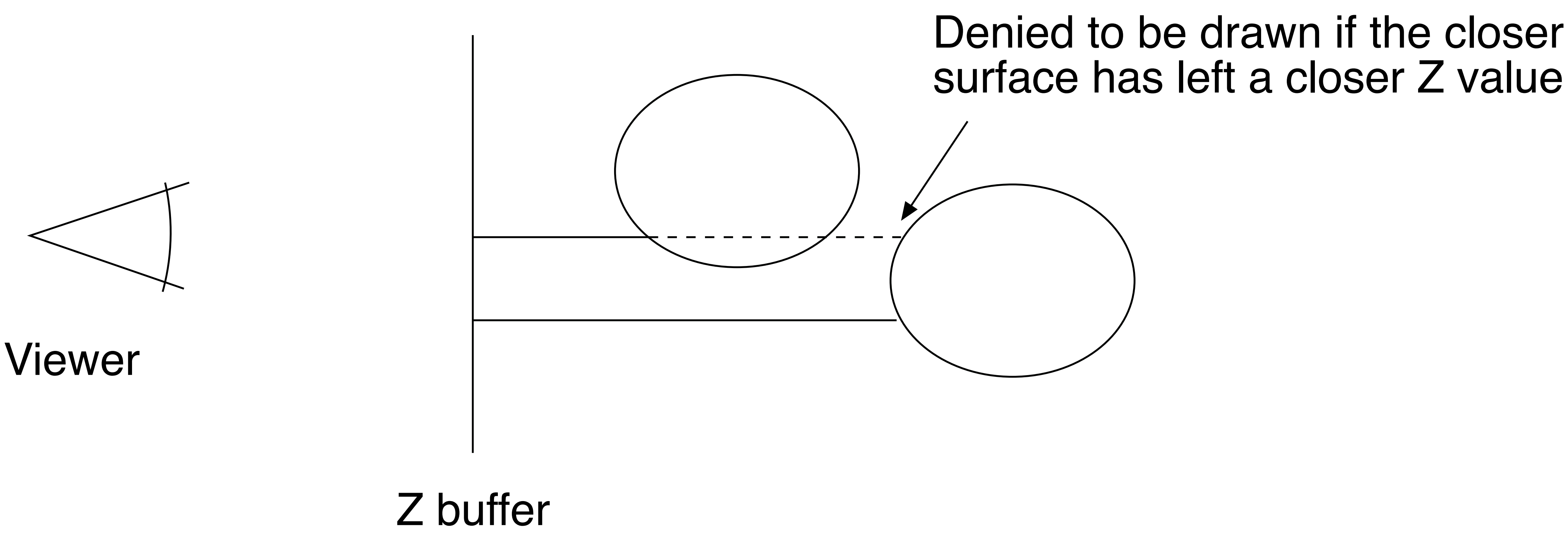


Removes ≈50% of all polygons that would otherwise be in view!

Z-buffer

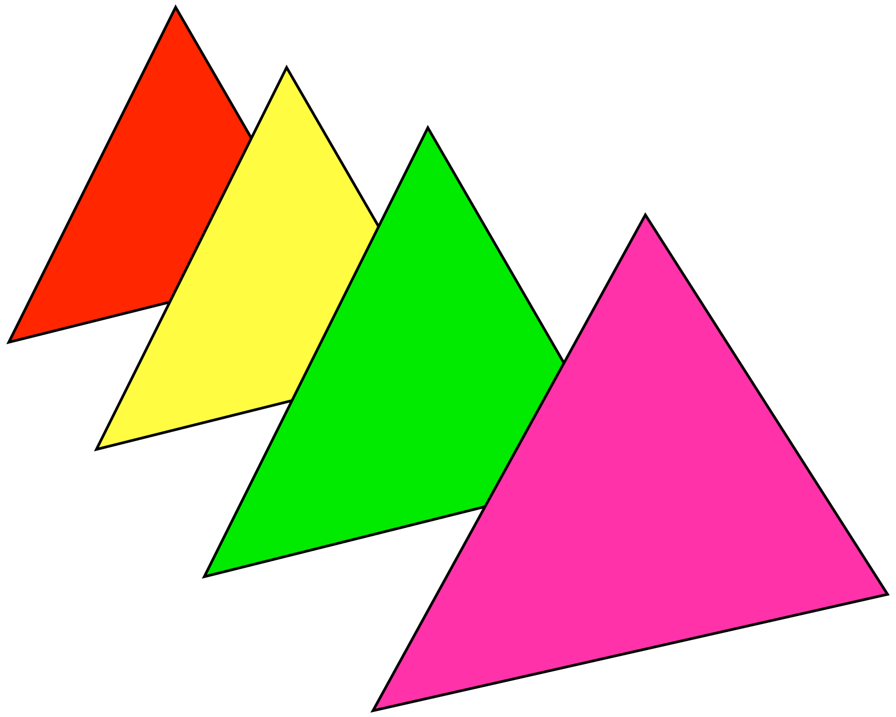
Depth-buffer method

Only draw if a pixel is closer than the closest drawn before. Z value saved in the depth buffer, the "Z buffer"


Denied to be drawn if the closer surface has left a closer Z value
Viewer
Z buffer

Painter’s algorithm

Depth-sorting method



Render from back to front

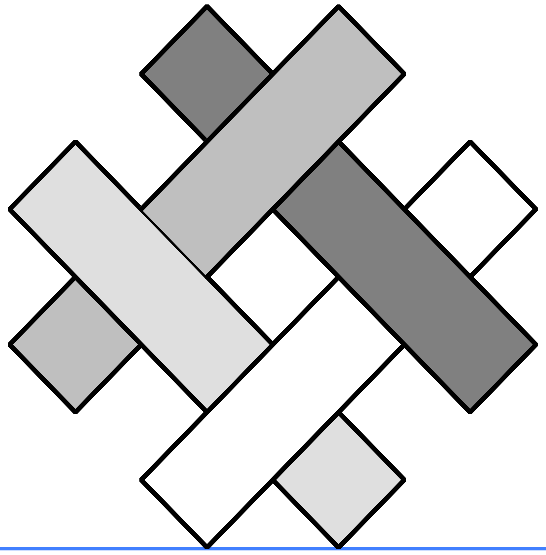
Both image and object space method

Painter’s algorithm

Sorting on polygon level.

But some scenes can not be sorted at all!

Solution: Figure out a way to split polygons to resolve the sort. But how?



Painter’s algorithm

- Slow – may paint many pixels more than once

- Slow and complicated in its full form - can be solved with BSP trees

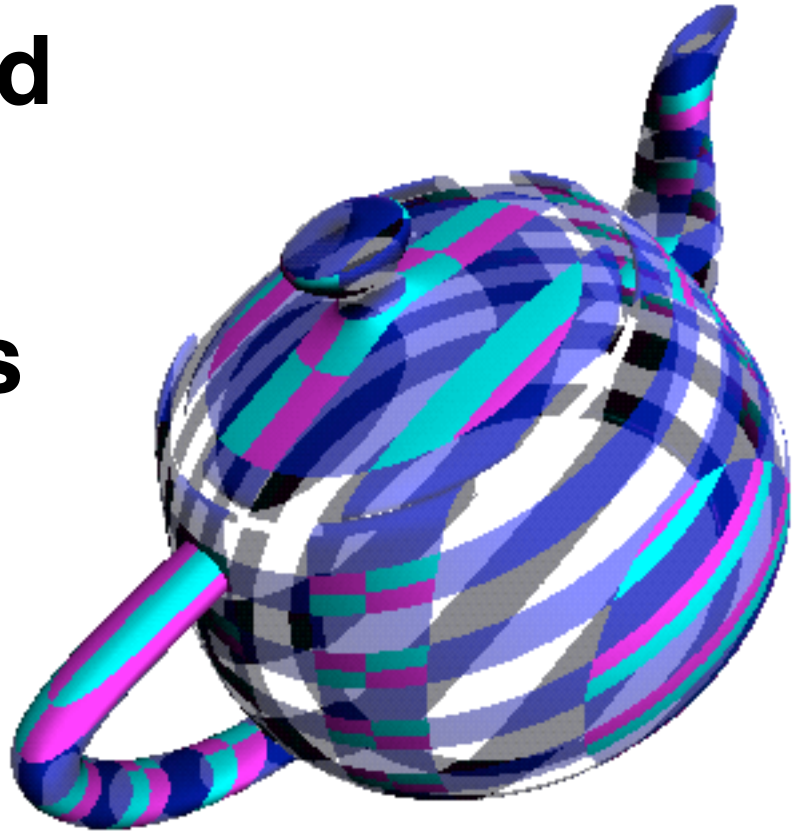
- Practically useful at object level, sorting transparent objects only

- Approximative sorting is often sufficient

Drawing with transparency

A (alpha) in RGBA can be used for transparency

Alpha values exist in textures as well as color etc



```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA,
GL_ONE_MINUS_SRC_ALPHA);
```

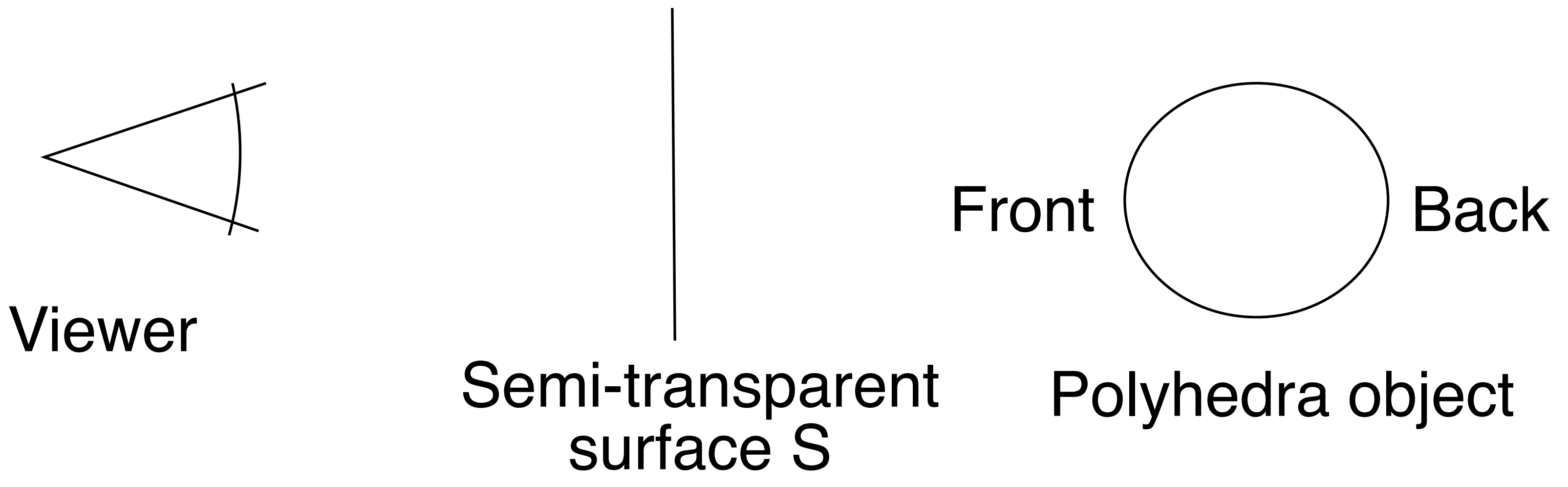
(glBlendEquation for even more control)

$$\text{dest} = \text{source}*\alpha + \text{dest}*(1-\alpha)$$

Note that alpha does not have to be taken from the source!

Problem: Drawing order causes problems with Z-buffer!

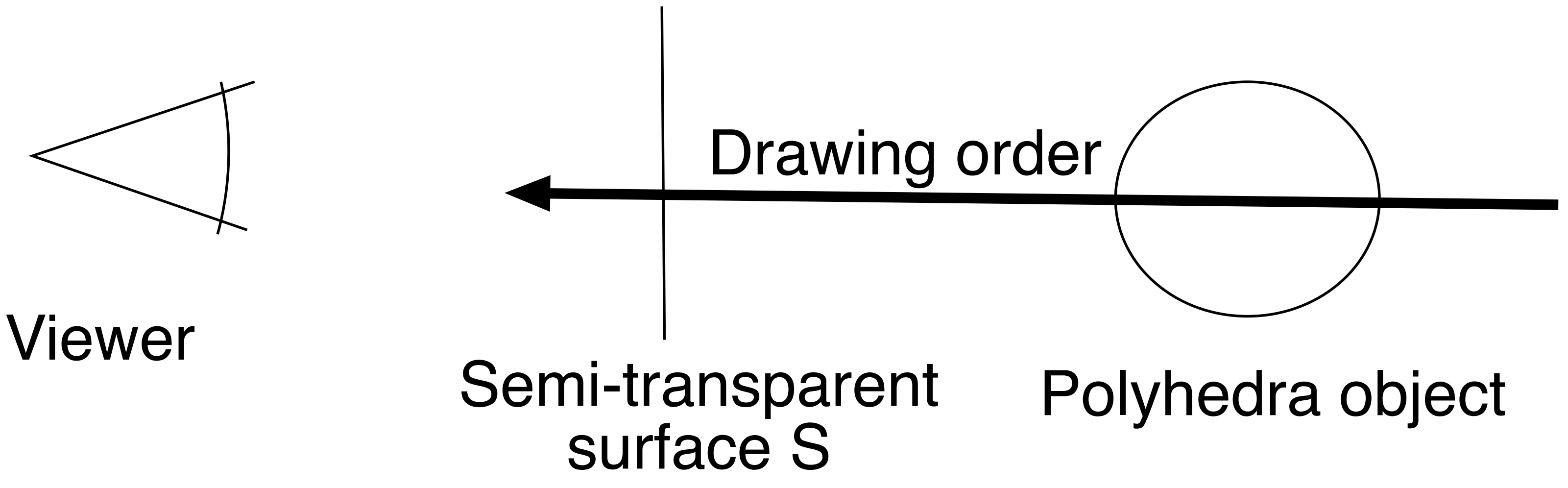
The Z-buffer problem with transparency



If S is drawn first, the other scenery will not be drawn!

For a single object, its inside will be obscured by its front.

The Z-buffer problem, solutions

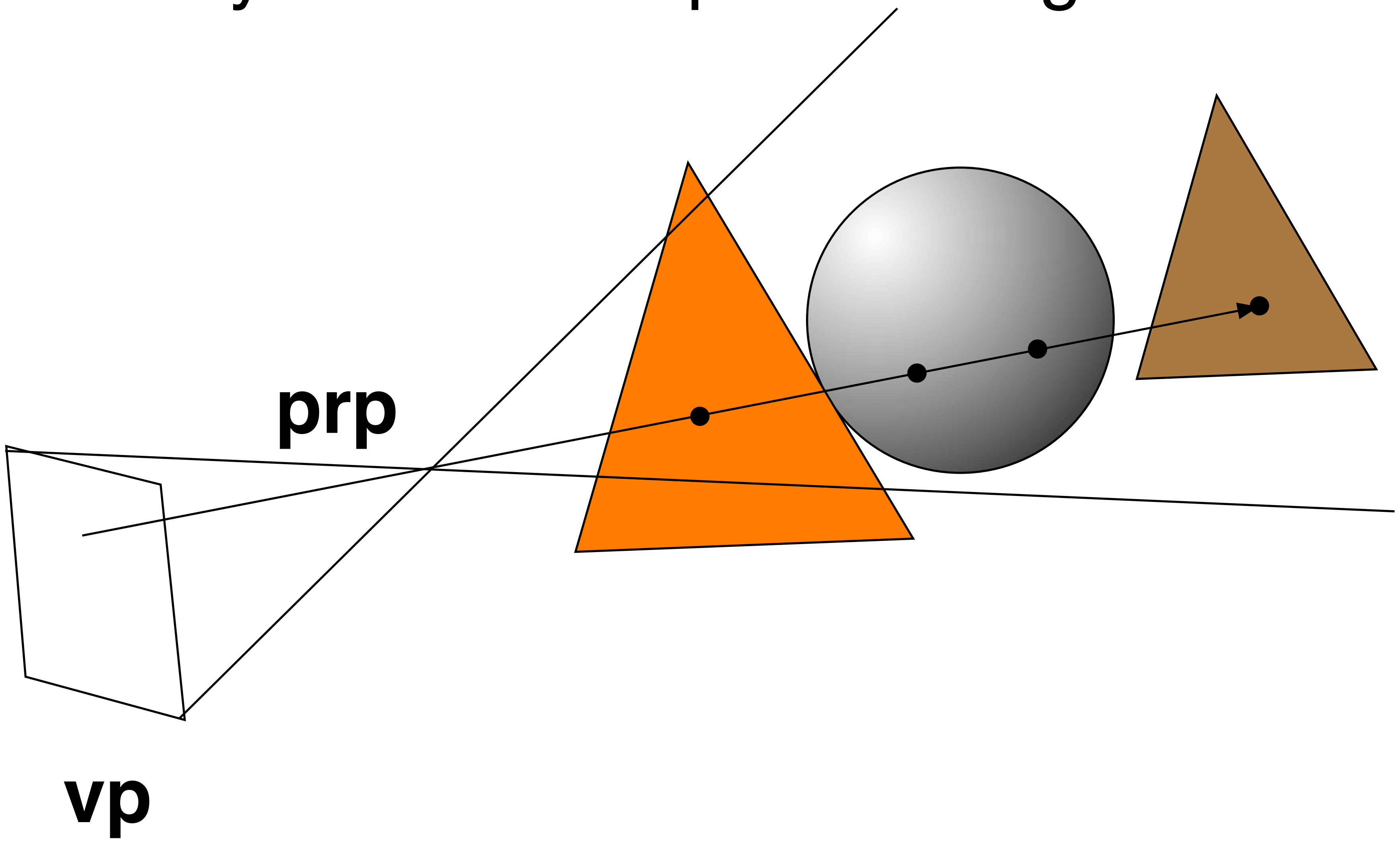


Solution for entire scene: Draw the scene back-to-front. ”Painter’s algorithm”

For a single object, draw its inside first, front later. Can be done with culling.

Ray-casting

Follow rays from each pixel through the scene



Full 3D raycasting

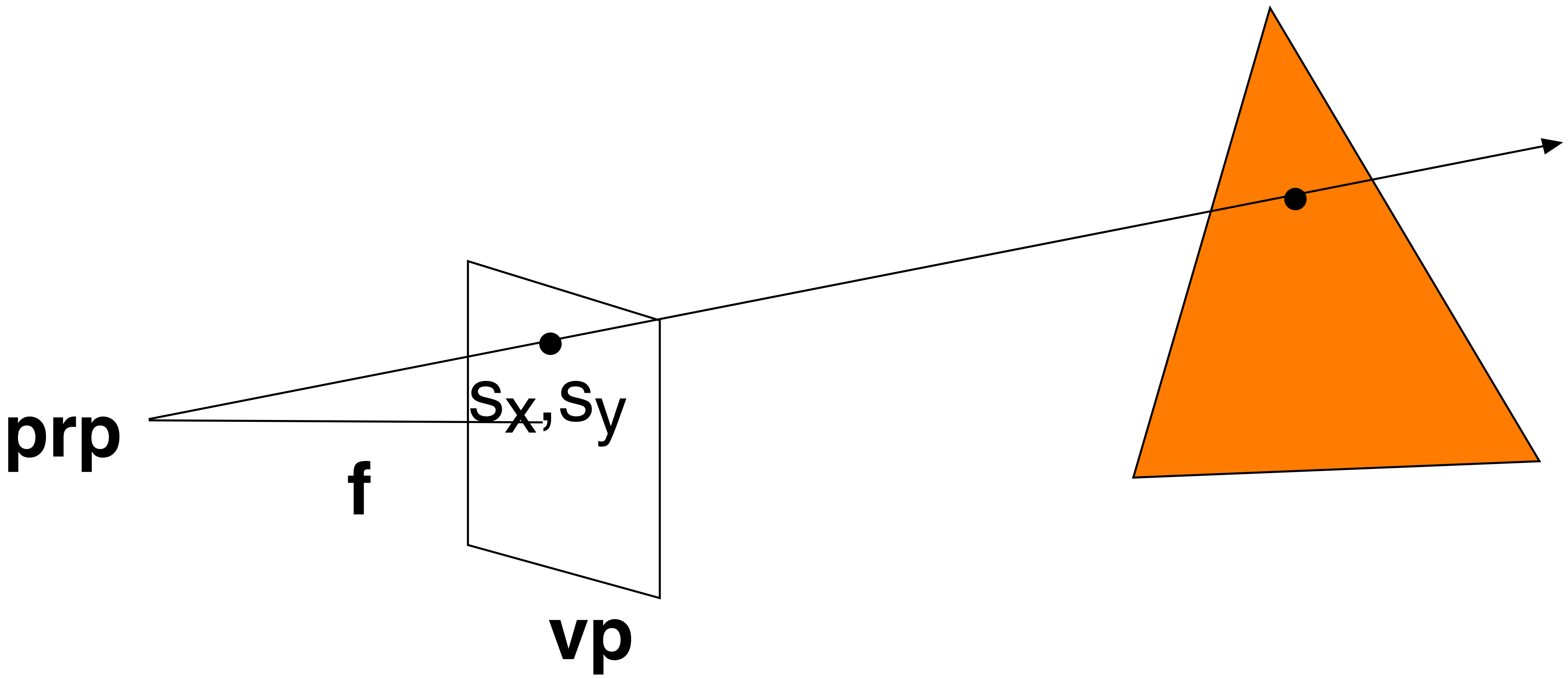
for every pixel (x,y) in the image

calculate a ray from the pixel through the camera (prp) and through the scene

calculate intersections with all objects in the scene

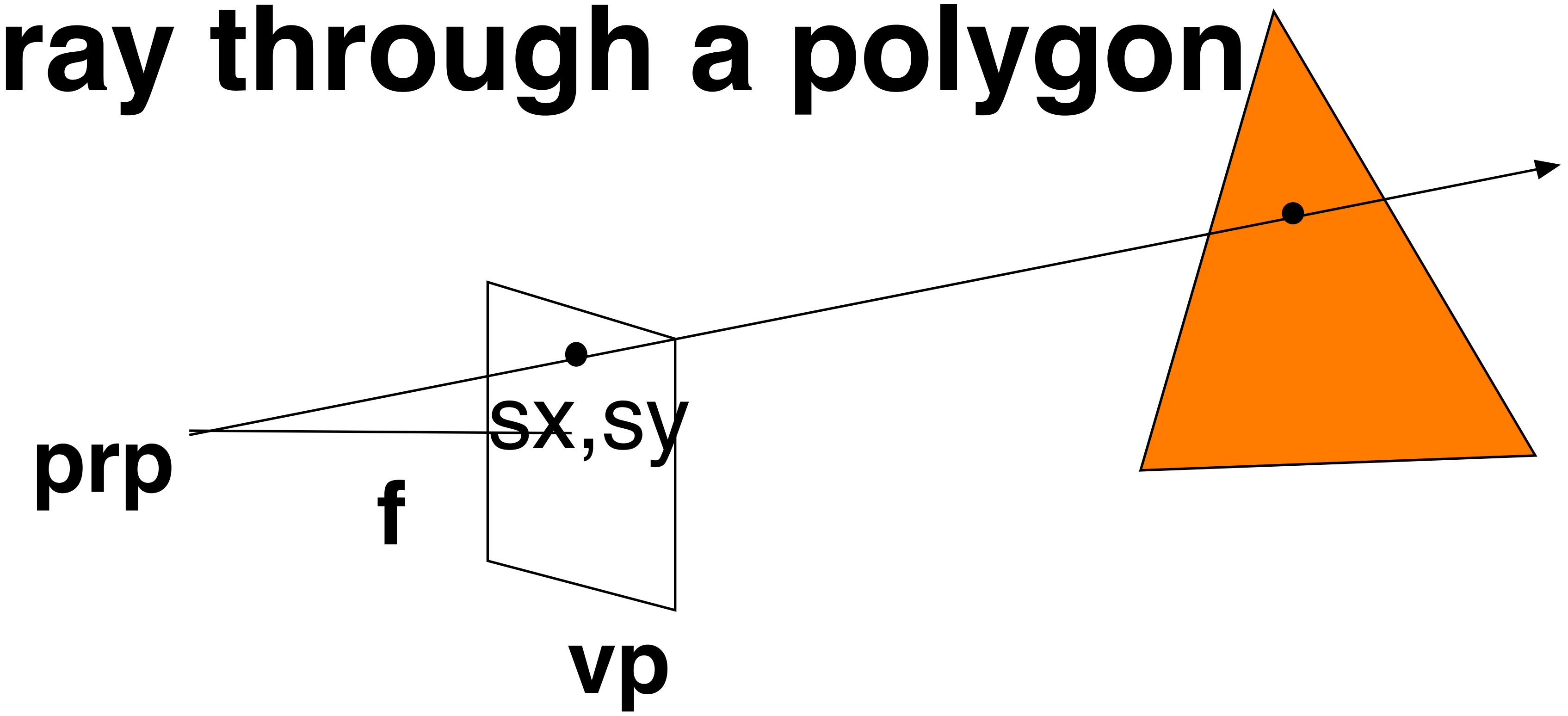
the pixel value is calculated from the closest intersection found

The ray



Line equation: prp + $\mu(s_x,s_y,-f)$

A ray through a polygon



Take the case prp = (0,0,0)

Insert (0,0,0) + $\mu(s_x,s_y,-f)$ in plane equation:

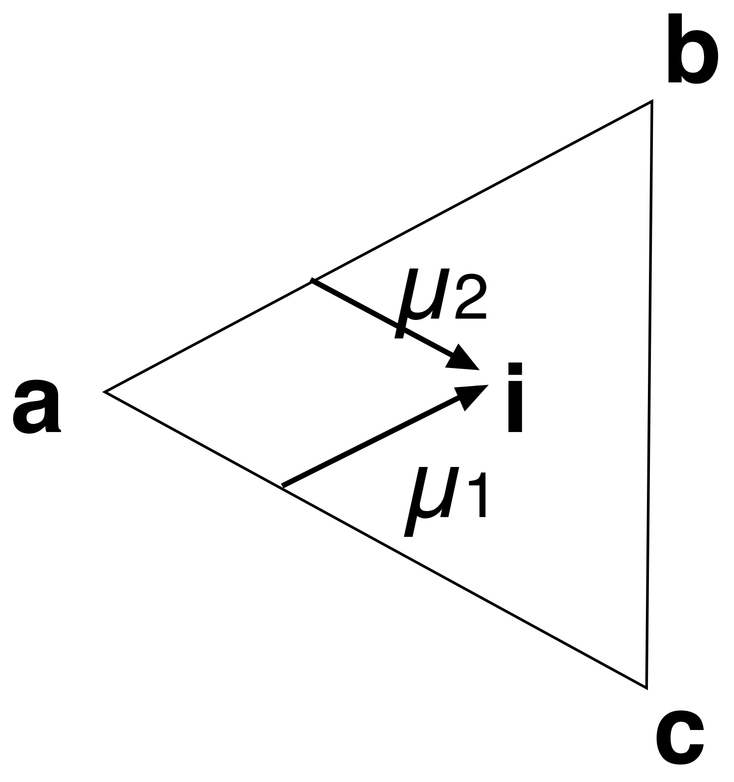
$$A\mu s_x + B\mu s_y - C\mu f + D = 0$$

$$\mu = -D / (As_x + Bs_y - Cf)$$

Is the point in the polygon?

Triangle: Straight-forward

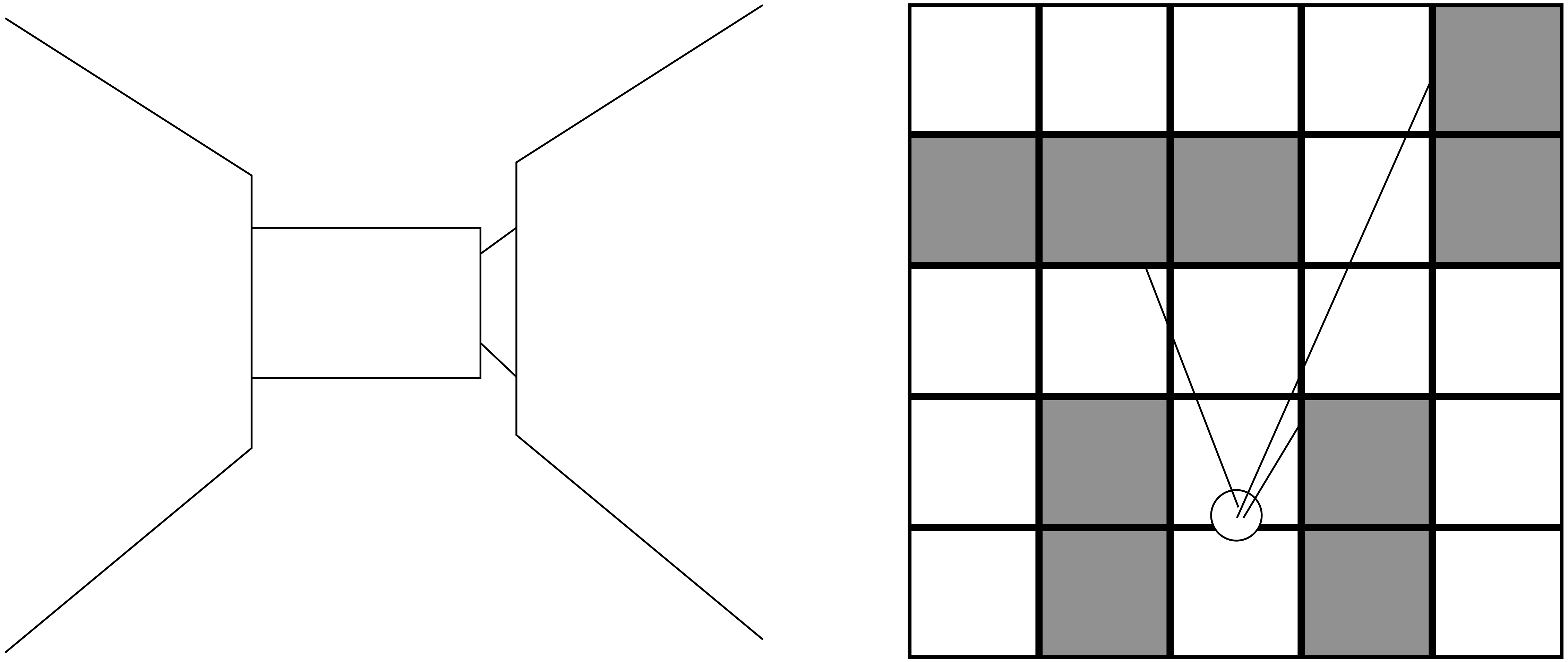
Several methods possible.



$$\mathbf{i} = \mathbf{a} + \mu_1 * \mathbf{ab} + \mu_2 * \mathbf{ac}$$

$$0 < \mu_1$$
$$0 < \mu_2$$
$$\mu_1 + \mu_2 < 1$$

Raycasting in a grid: *Ray marching*

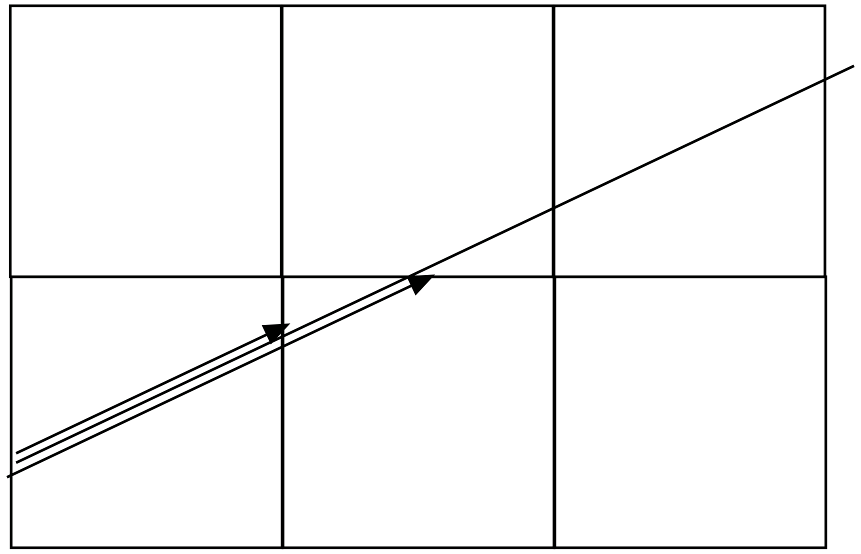


Ray marching relatively easy

Step to next potential voxel wall (3 possible in 3D)

Pick the closest, check neighbor space

Repeat until filled space is found.



Essentially a line drawing algorithm!

Ray-casting applications

- VSD in 2D or 3D grids
- Visibility tests for AI
- Visibility tests for global illumination
- First step of ray-tracing
- Picking