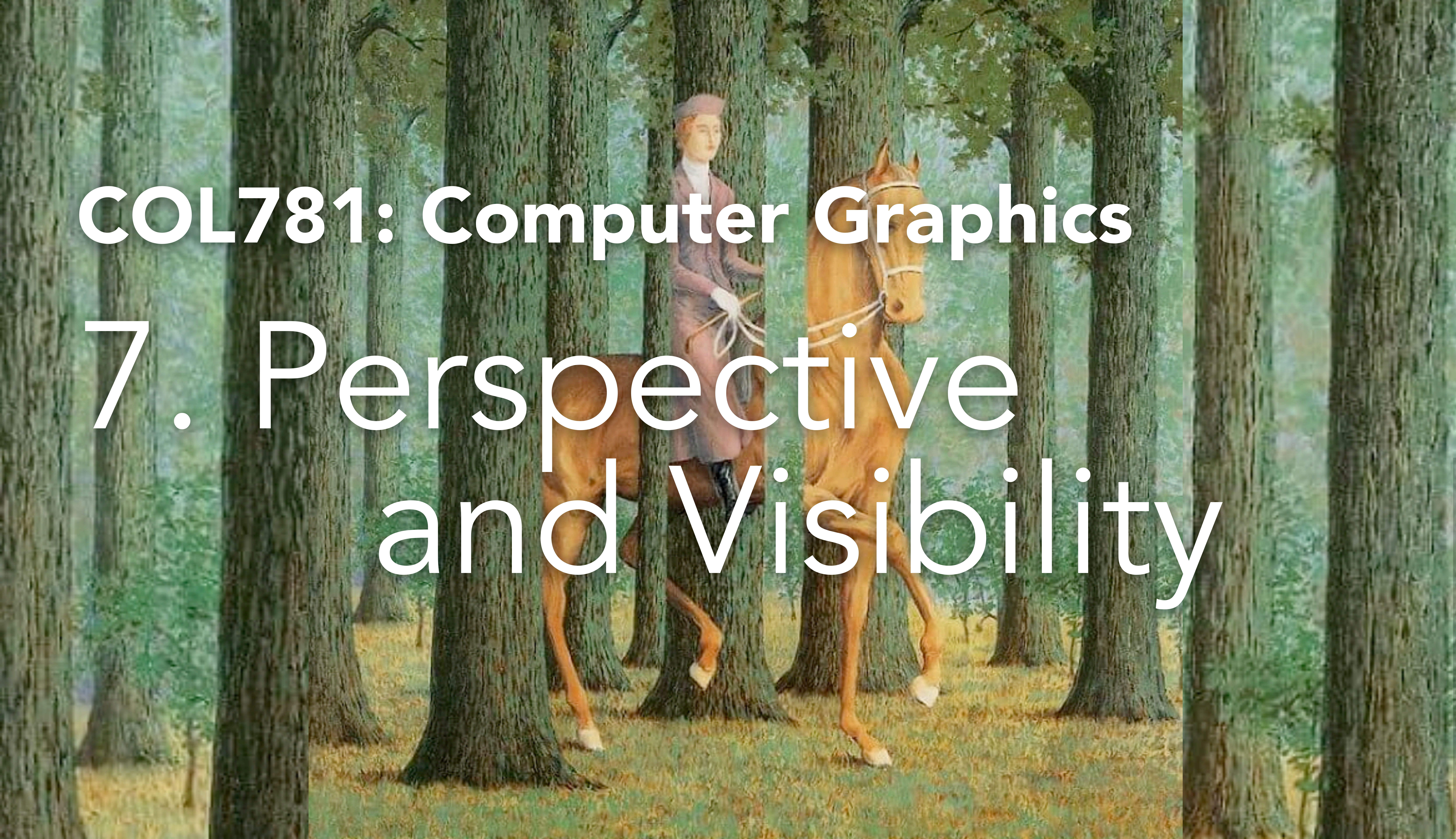
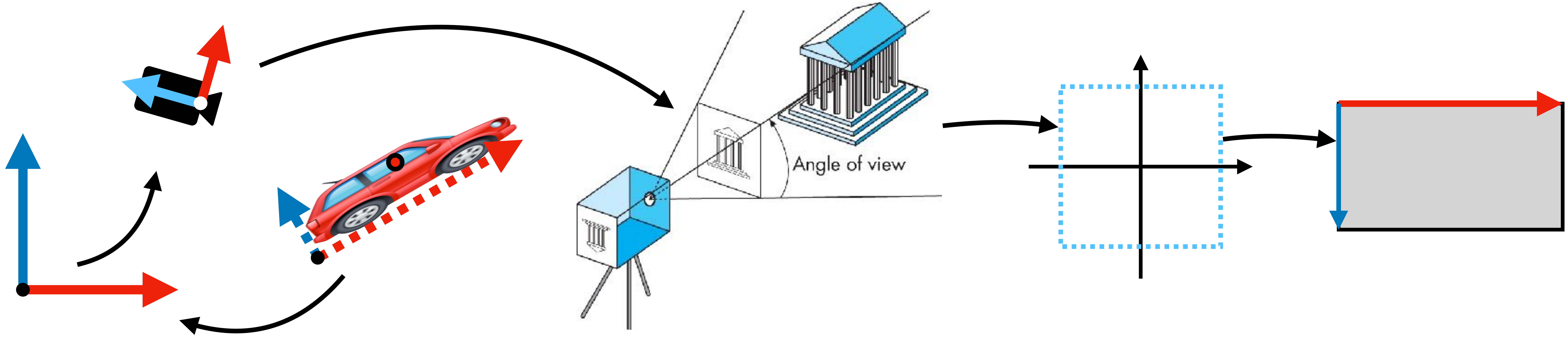


COL781: Computer Graphics

7. Perspective and Visibility

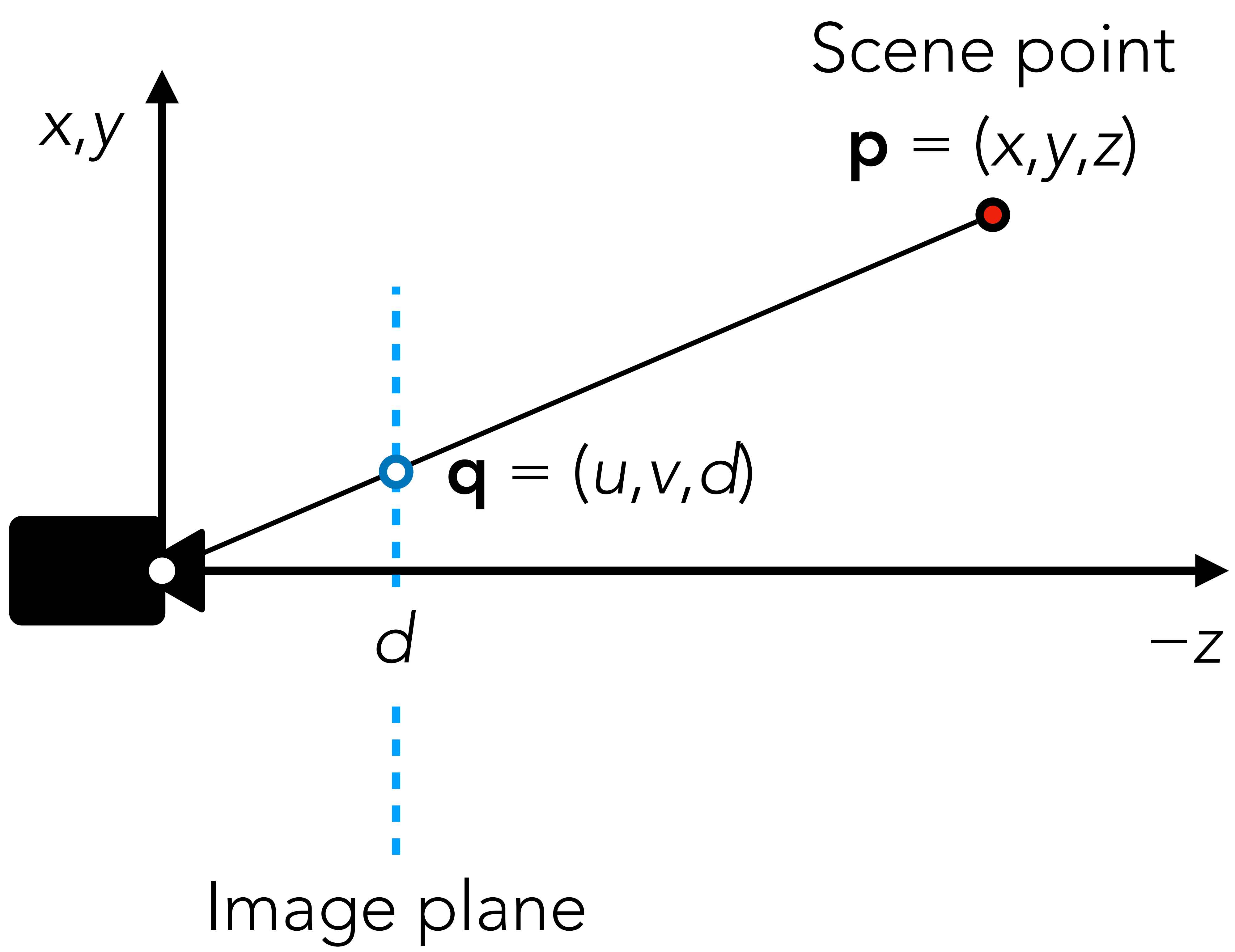
Rene Magritte, *The Blank Signature*



- Object space → world space
- World space → camera space
- Camera space → projection plane (division by *z*)
- Projection plane → NDC
- NDC → screen coordinates

Two problems:

- Every step is a matrix, except perspective division.
- Final result has lost depth information (the *z* coordinate): don't know which points are in front of which

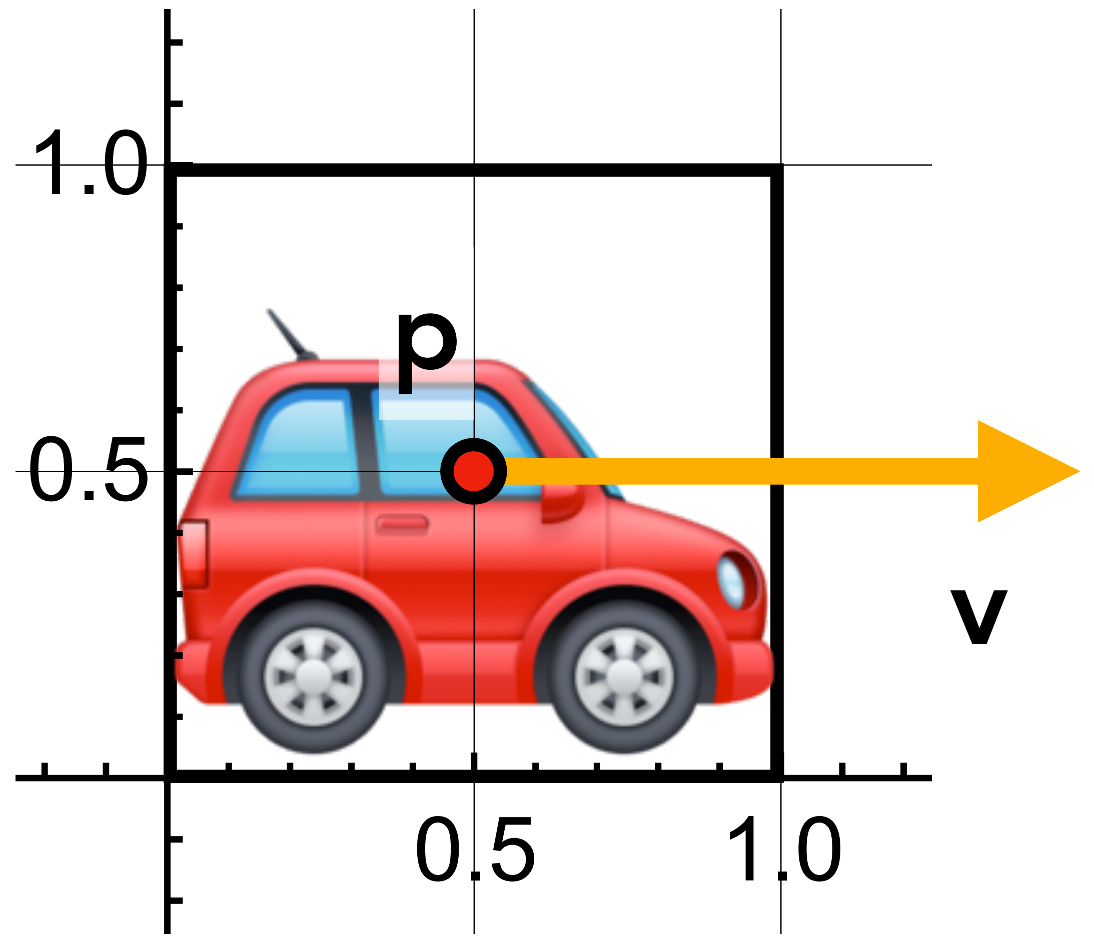


Perspective projection:
$(x,y,z) \rightarrow (xd/z,\ yd/z)$

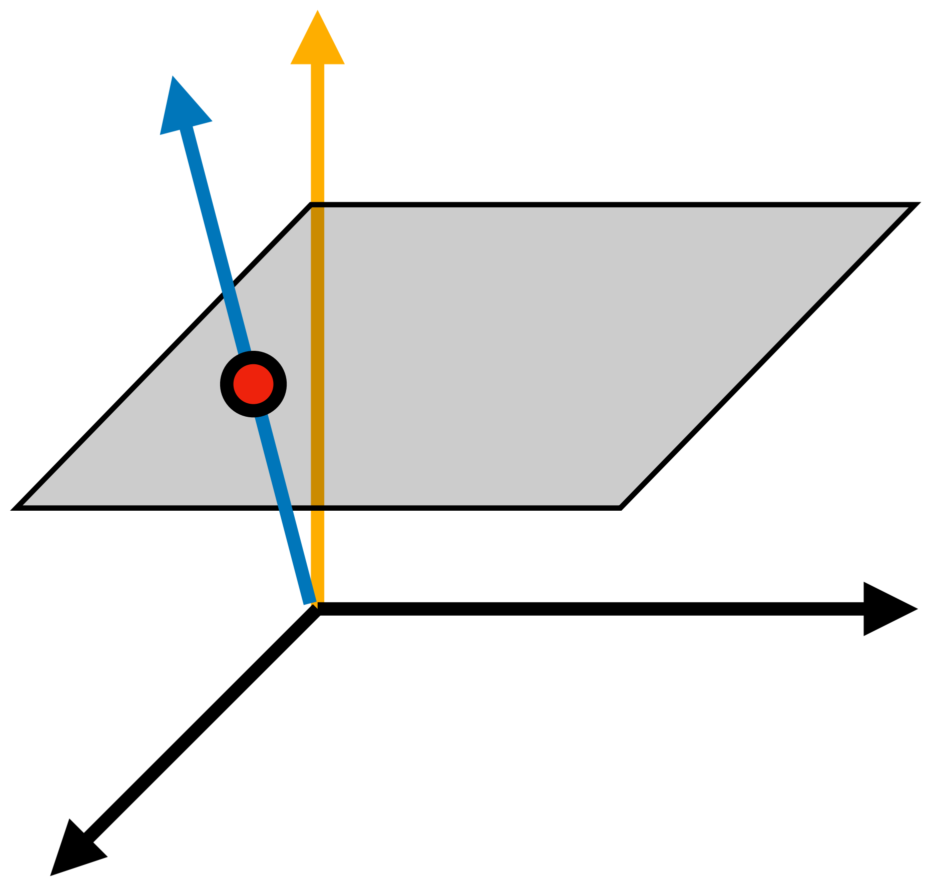
What about just $(xd/z,\ yd/z,\ z)$?

Homogeneous coordinates revisited

Recall points vs. vectors: $\mathbf{p} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$, $\mathbf{v} = \begin{bmatrix} x \\ y \\ 0 \end{bmatrix}$

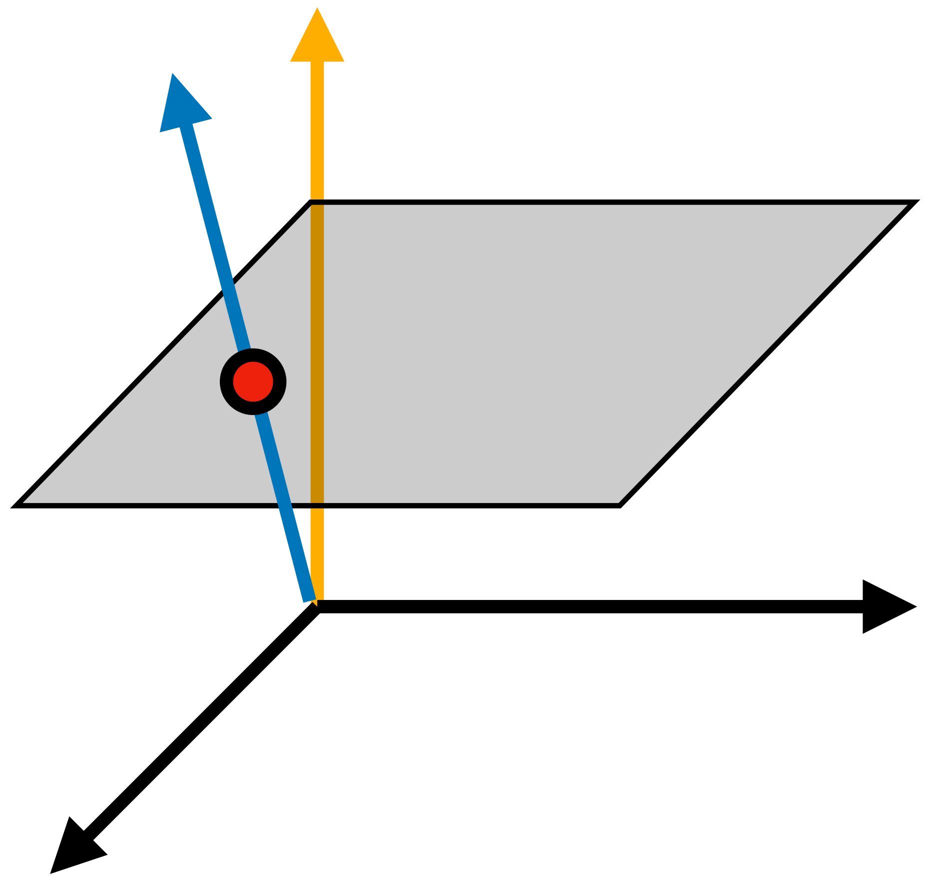


Let's generalize: points can have any $w \neq 0$



Any point in homogeneous coordinates $\hat{\mathbf{p}} = \begin{bmatrix} x \\ y \\ w \end{bmatrix}$ with $w \neq 0$

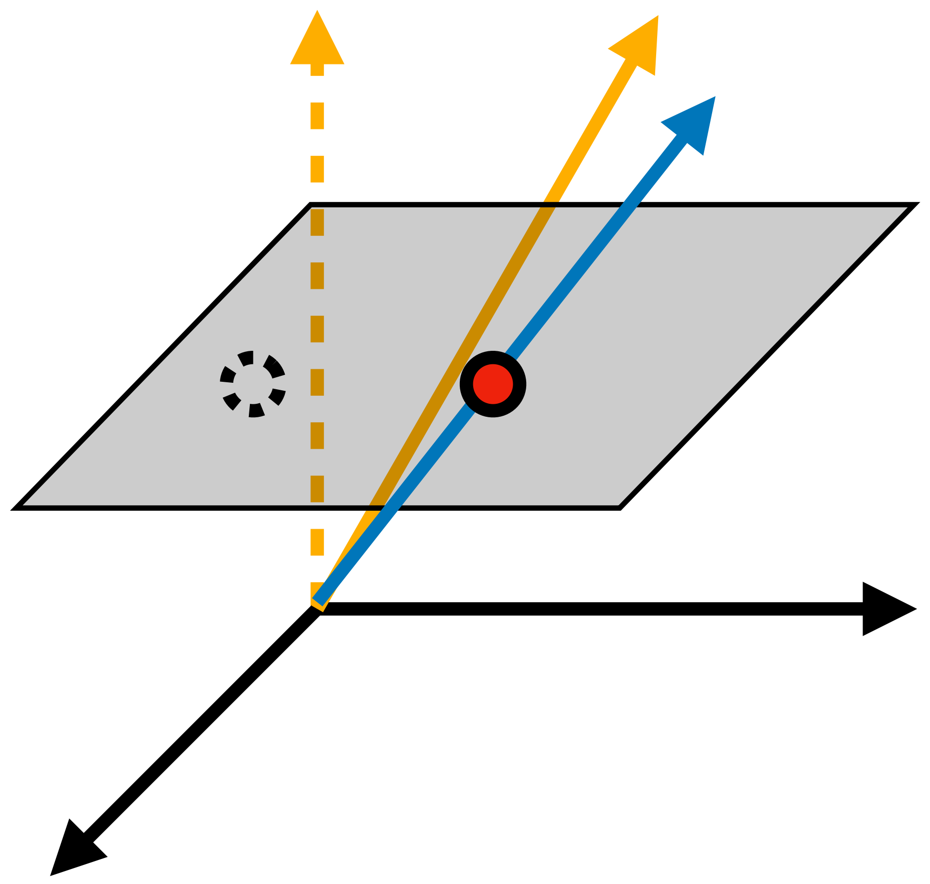
corresponds to the 2D point $\mathbf{p} = (x/w, y/w)$



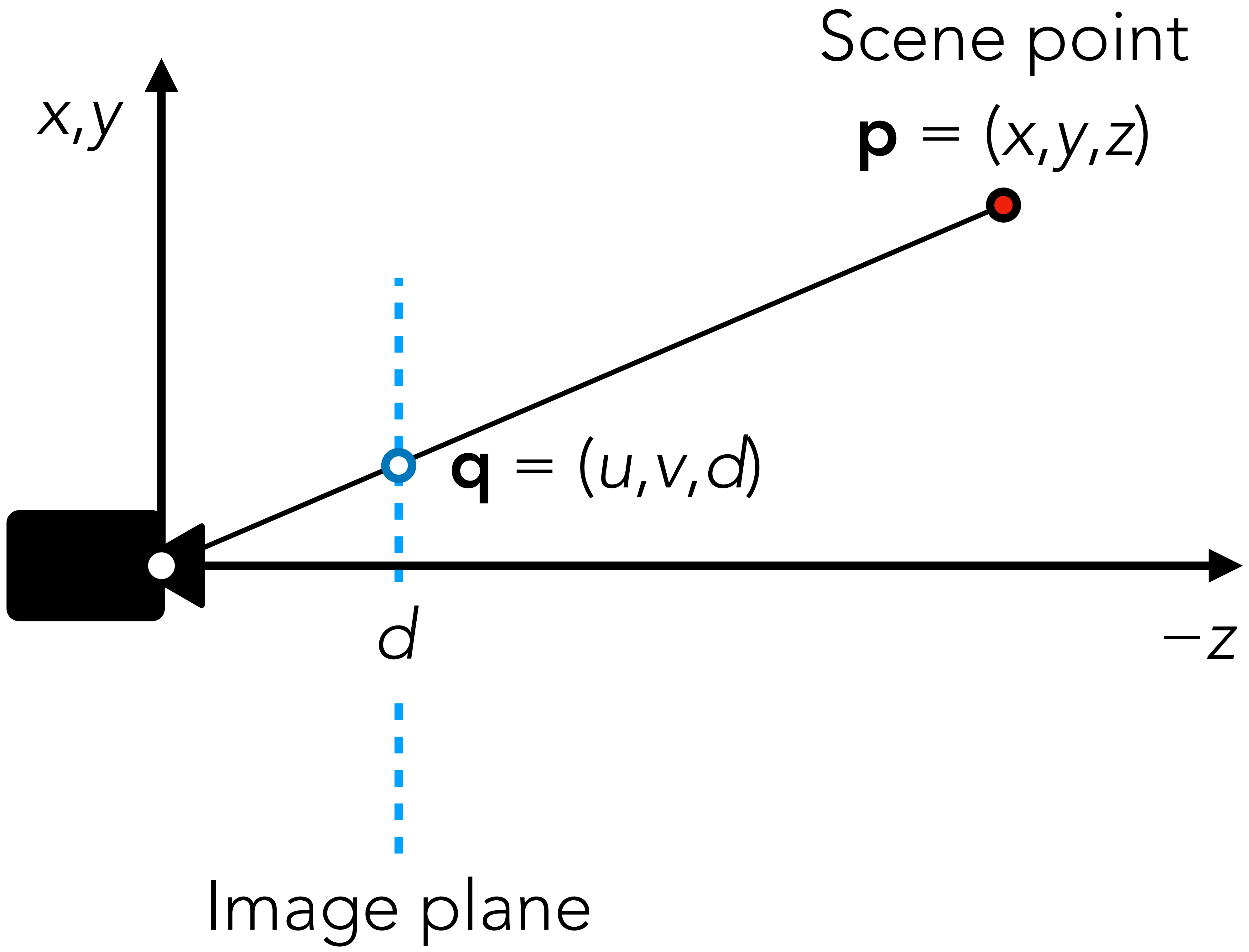
The main idea: Points in 2D correspond to lines through the origin in 3D!

All points $\hat{\mathbf{p}} = \begin{bmatrix} cx \\ cy \\ c \end{bmatrix}$ on a line represent the same point $\mathbf{p} = (x, y)$ where the line meets the plane $w = 1$

Analogy: Various tuples (2,4), (−1,−2), (5,10), … all represent the same rational number ½



Linear and affine transformations still work as before!
[Worked example on whiteboard…]



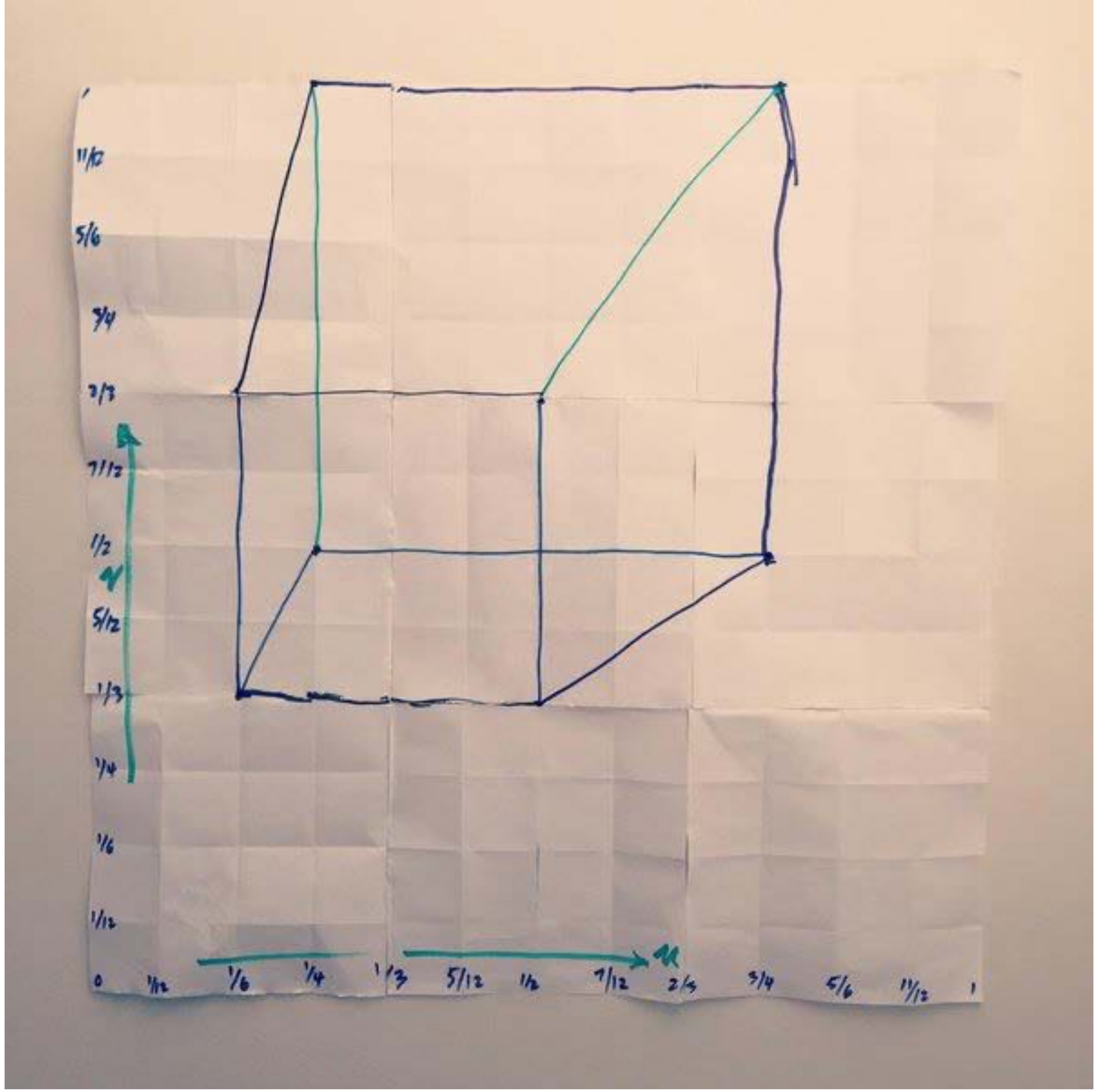
Perspective projection: $(x,y,z) \to (xd/z,\ yd/z)$

With homogeneous coordinates:

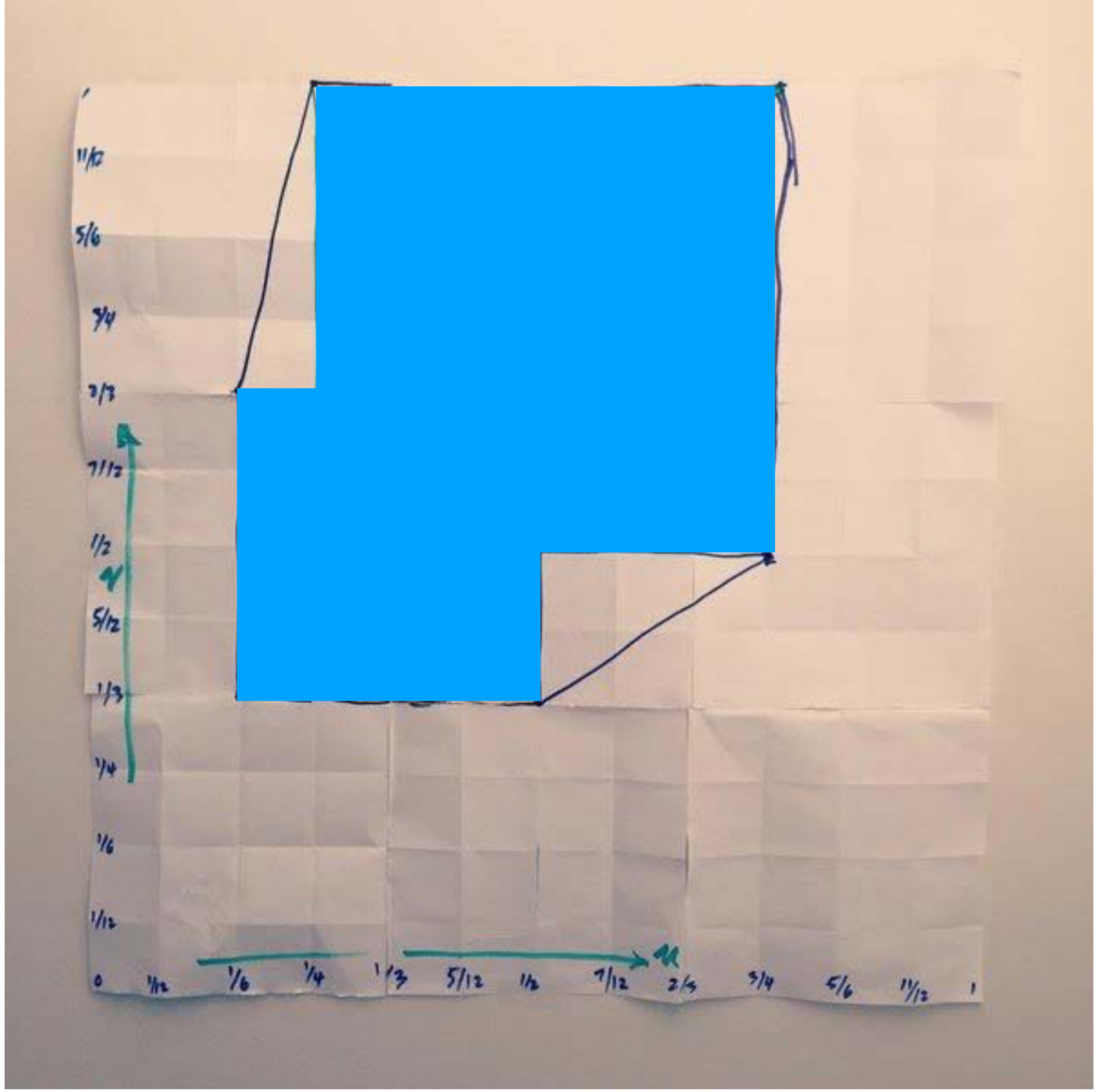
$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \to \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix} \sim \begin{bmatrix} xd/z \\ yd/z \\ d \end{bmatrix}$$

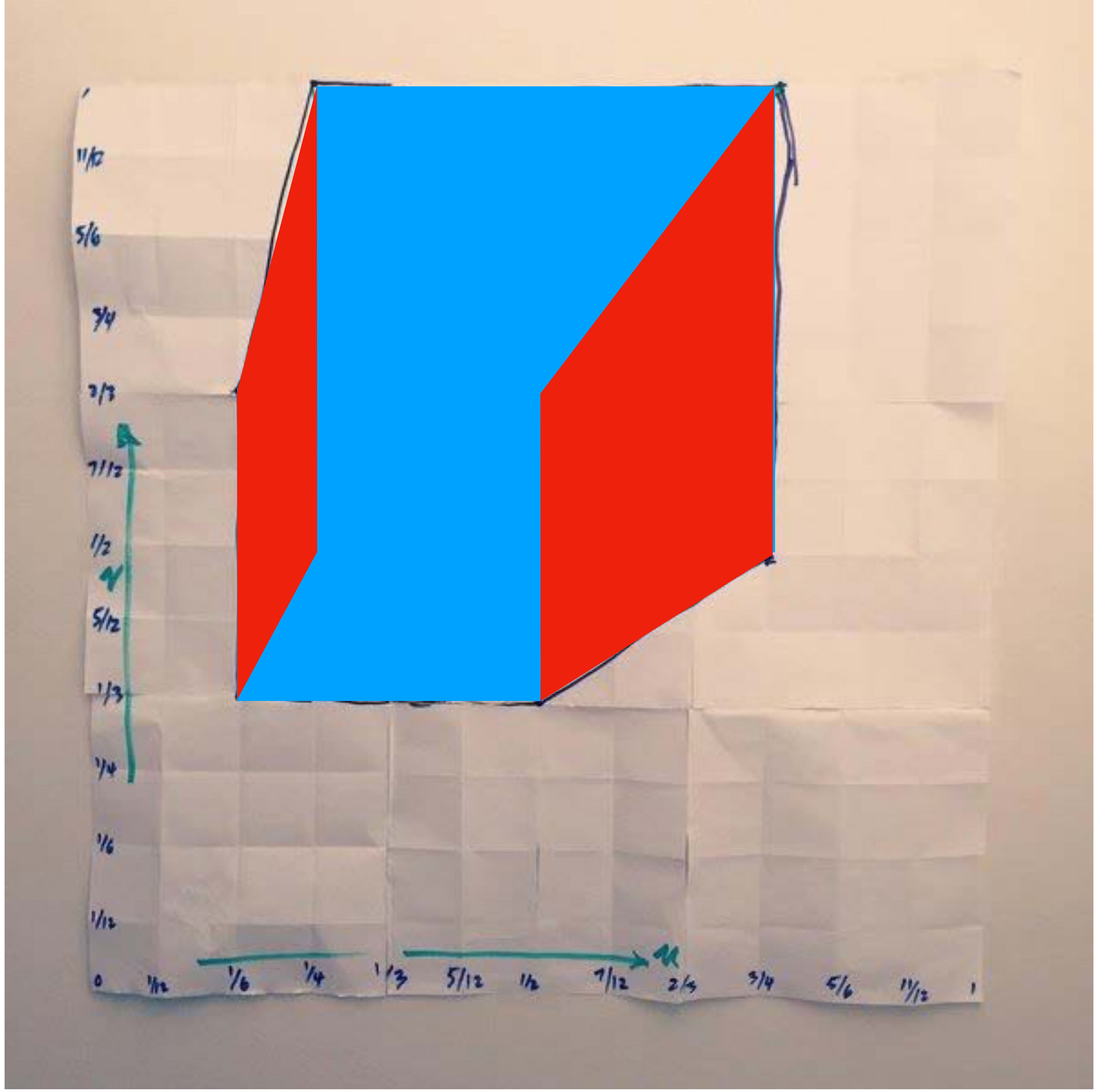
Corresponding matrix: $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$

Hang on, we've still lost depth information.

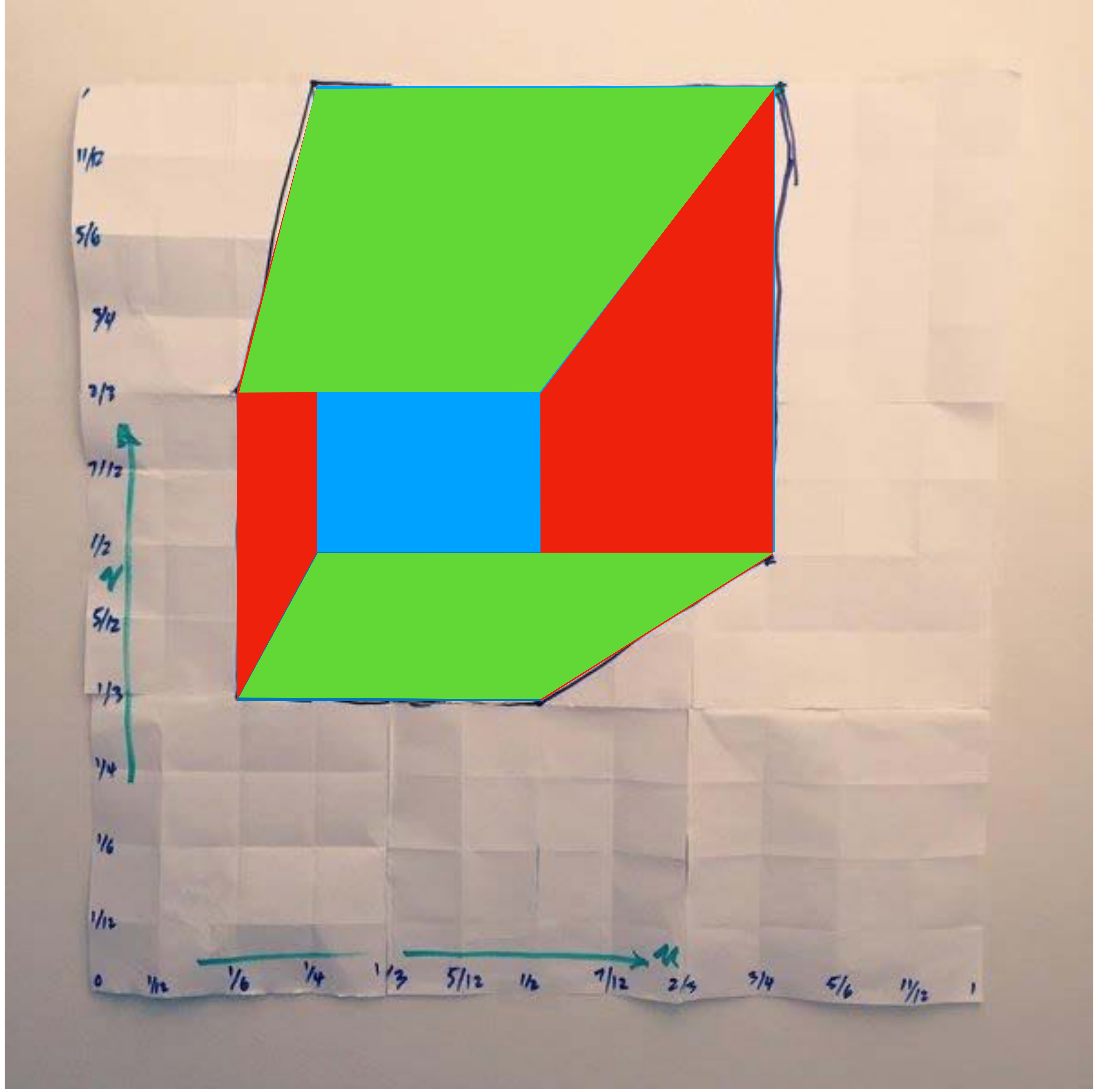


1
11/12
5/6
3/4
2/3
7/12
1/2
v
5/12
1/3
1/4
1/6
1/12
0
1/12
1/6
1/4
1/3
5/12
1/2
7/12
2/3
3/4
5/6
11/12
1
u

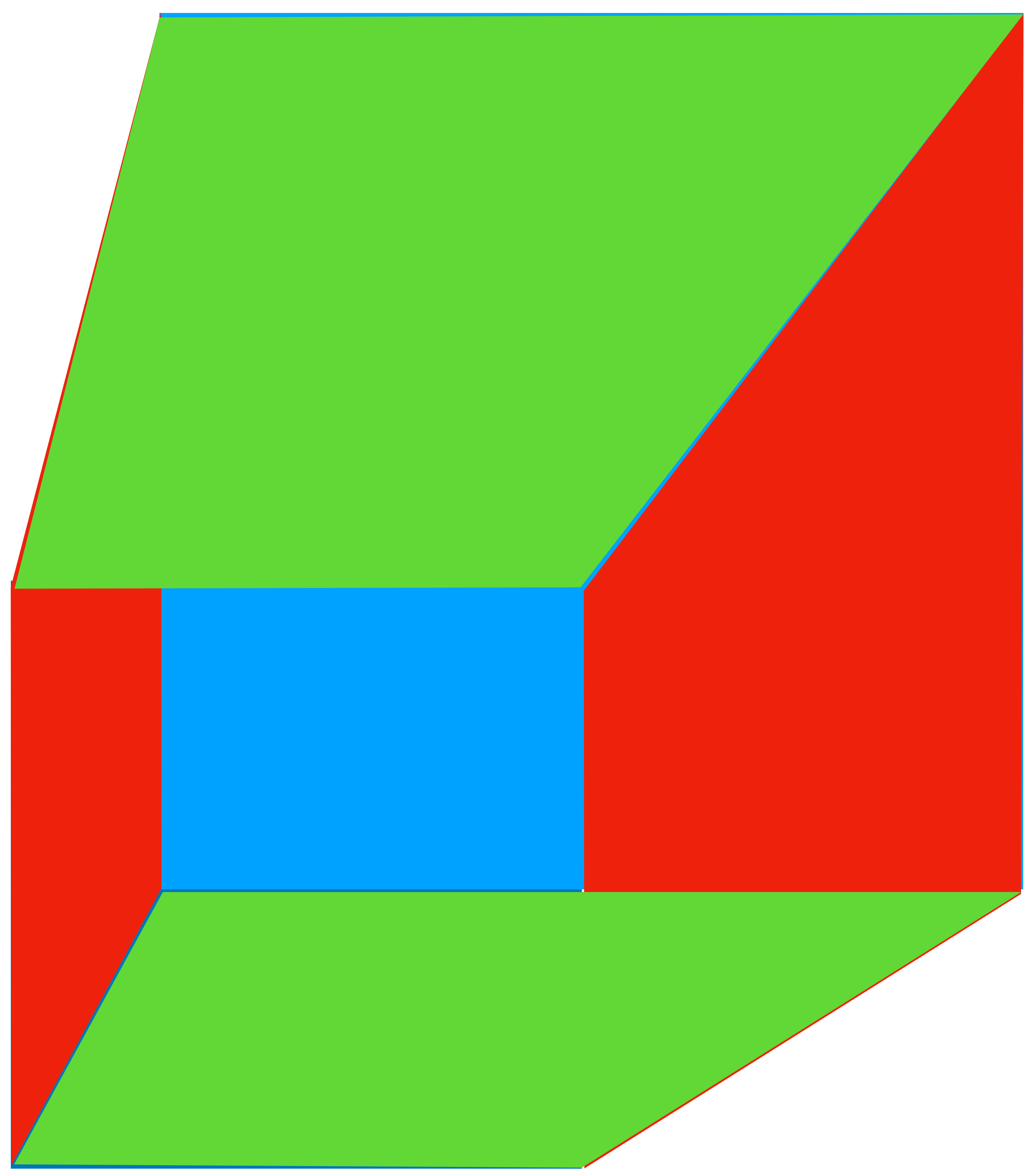

1
11/12
5/6
3/4
2/3
7/12
1/2
v
5/12
1/3
1/4
1/6
1/12
0
1/12
1/6
1/4
1/3
5/12
1/2
7/12
2/3
3/4
5/6
11/12
1
u

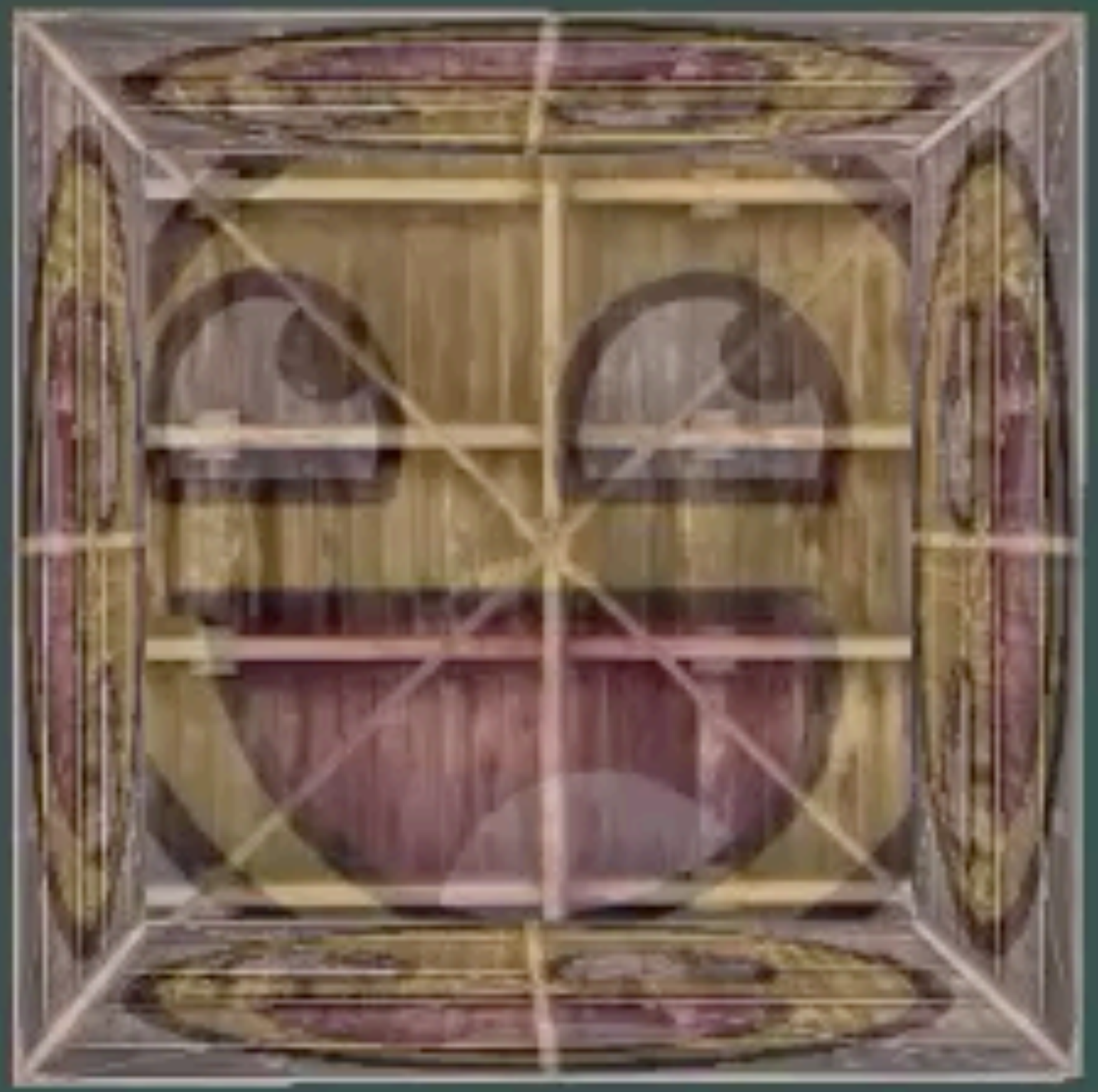


0
1/12
1/6
1/4
1/3
5/12
1/2
7/12
2/3
3/4
5/6
11/12
1



0
1/12
1/6
1/4
1/3
5/12
1/2
7/12
2/3
3/4
5/6
11/12
1
u
v

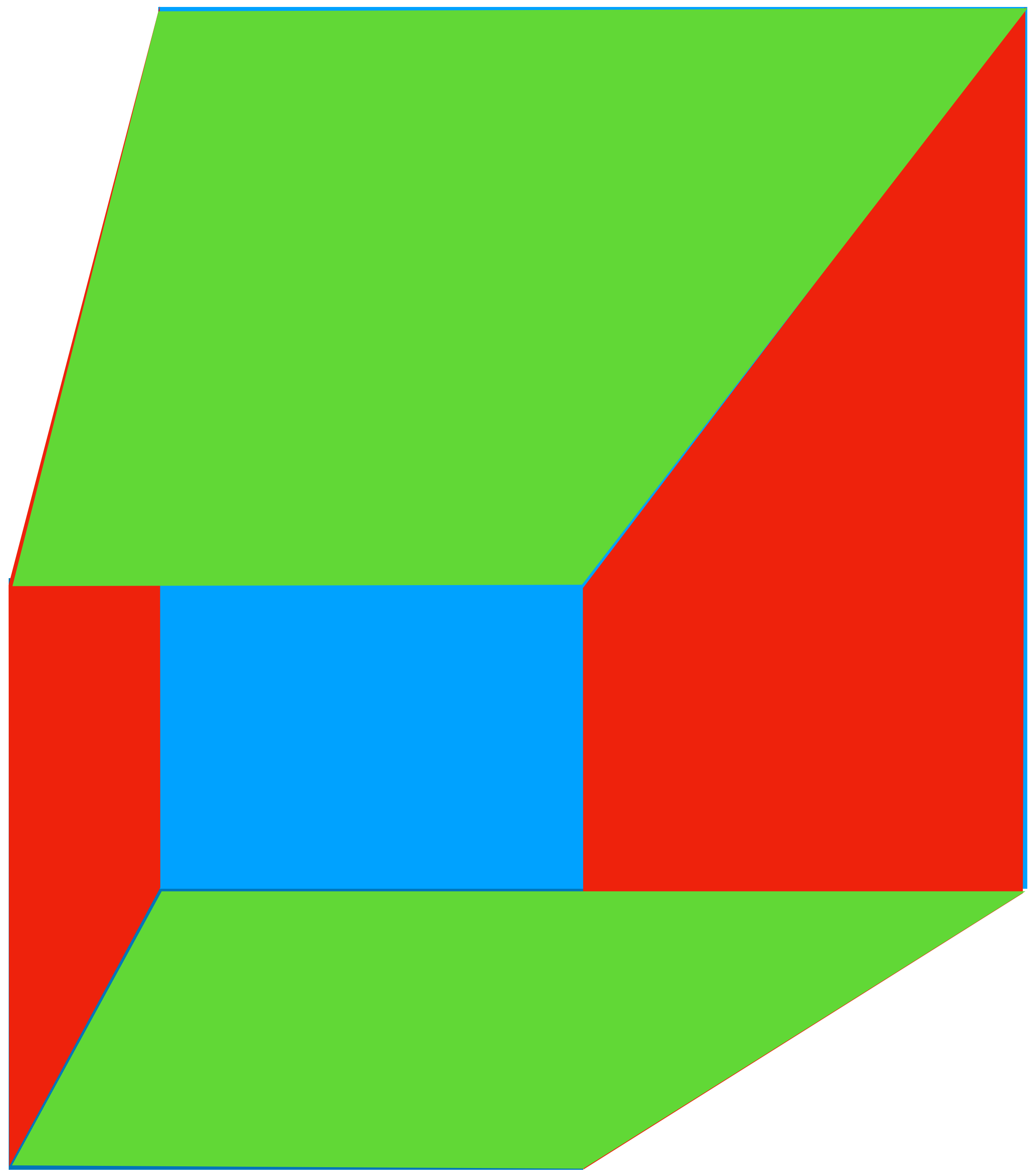




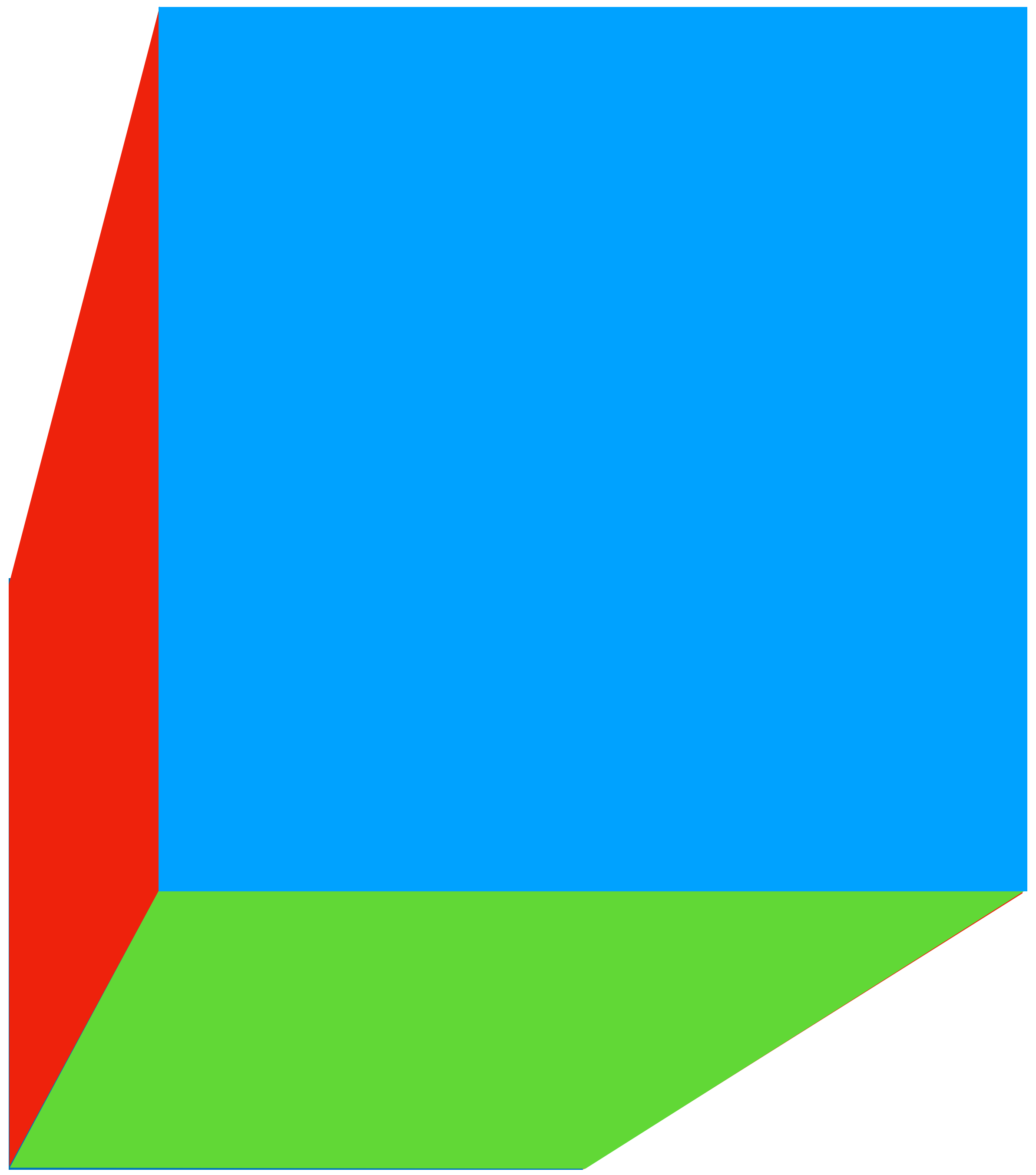
LearnOpenGL.com

Visibility a.k.a. hidden surface removal

Which surfaces are visible? Those that are not hidden by nearer surfaces.



Triangles drawn without considering depth / visibility



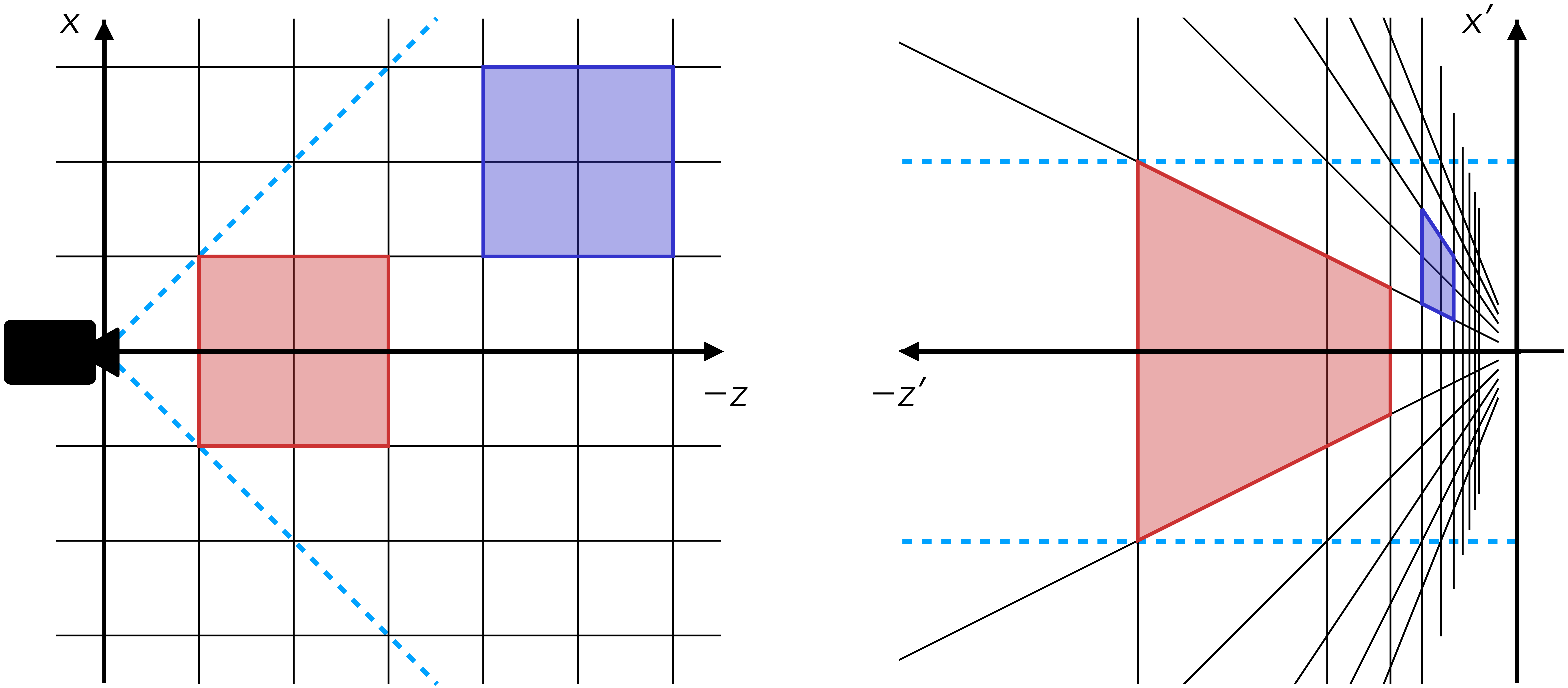
Correct result

To retain depth information, let's copy *w* into the *z*-coordinate:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix} \sim \begin{bmatrix} xd/z \\ yd/z \\ d \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ 1/d \\ z/d \end{bmatrix} \sim \begin{bmatrix} xd/z \\ yd/z \\ 1/z \end{bmatrix}$$

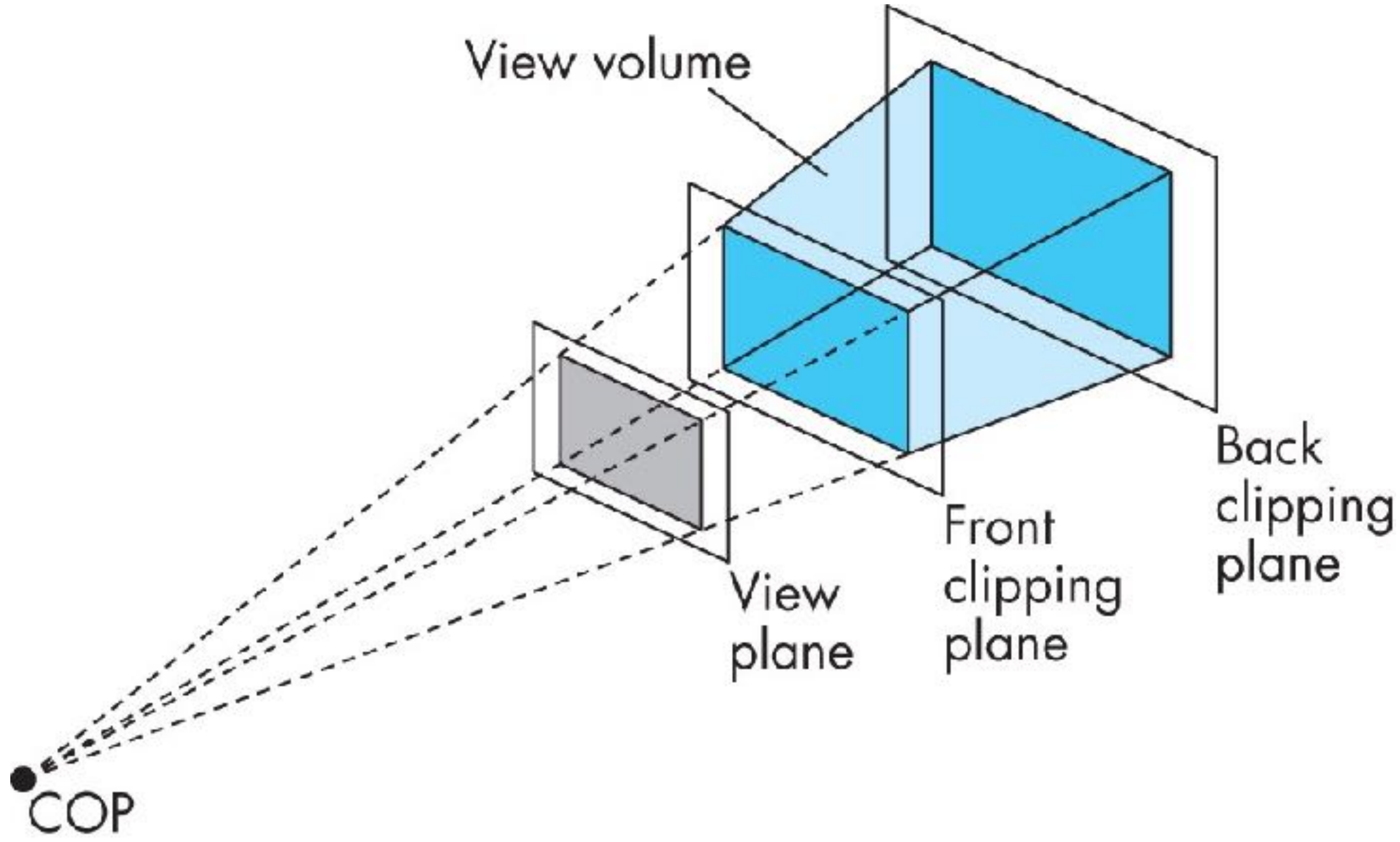
Matrix: $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1/d \\ 0 & 0 & 1/d & 0 \end{bmatrix}$



Scene in camera space
(x, y, z)

After perspective transformation
$(xd/z, yd/z, 1/z)$

The view frustum



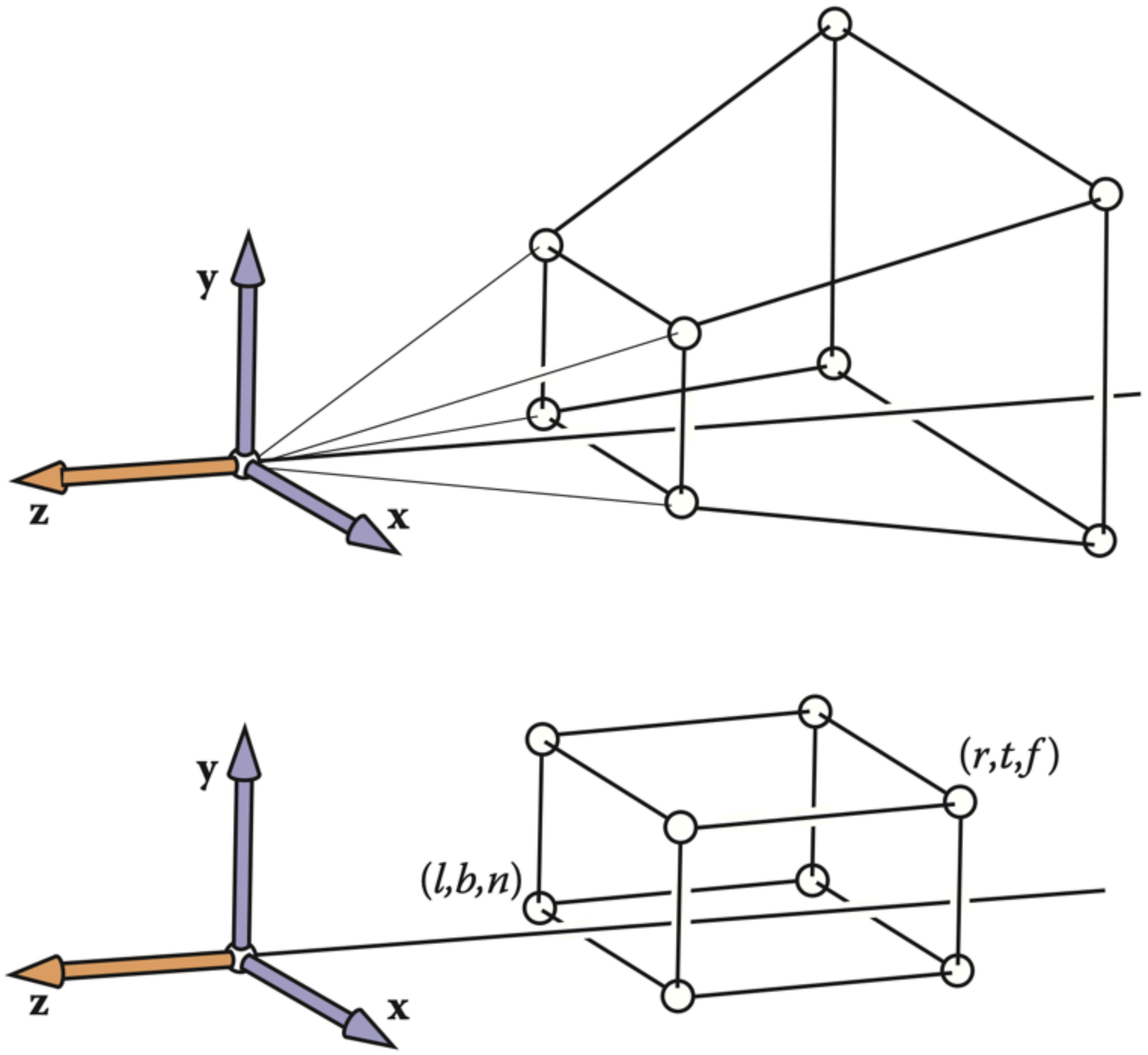
Angel & Shreiner, *Interactive Computer Graphics*

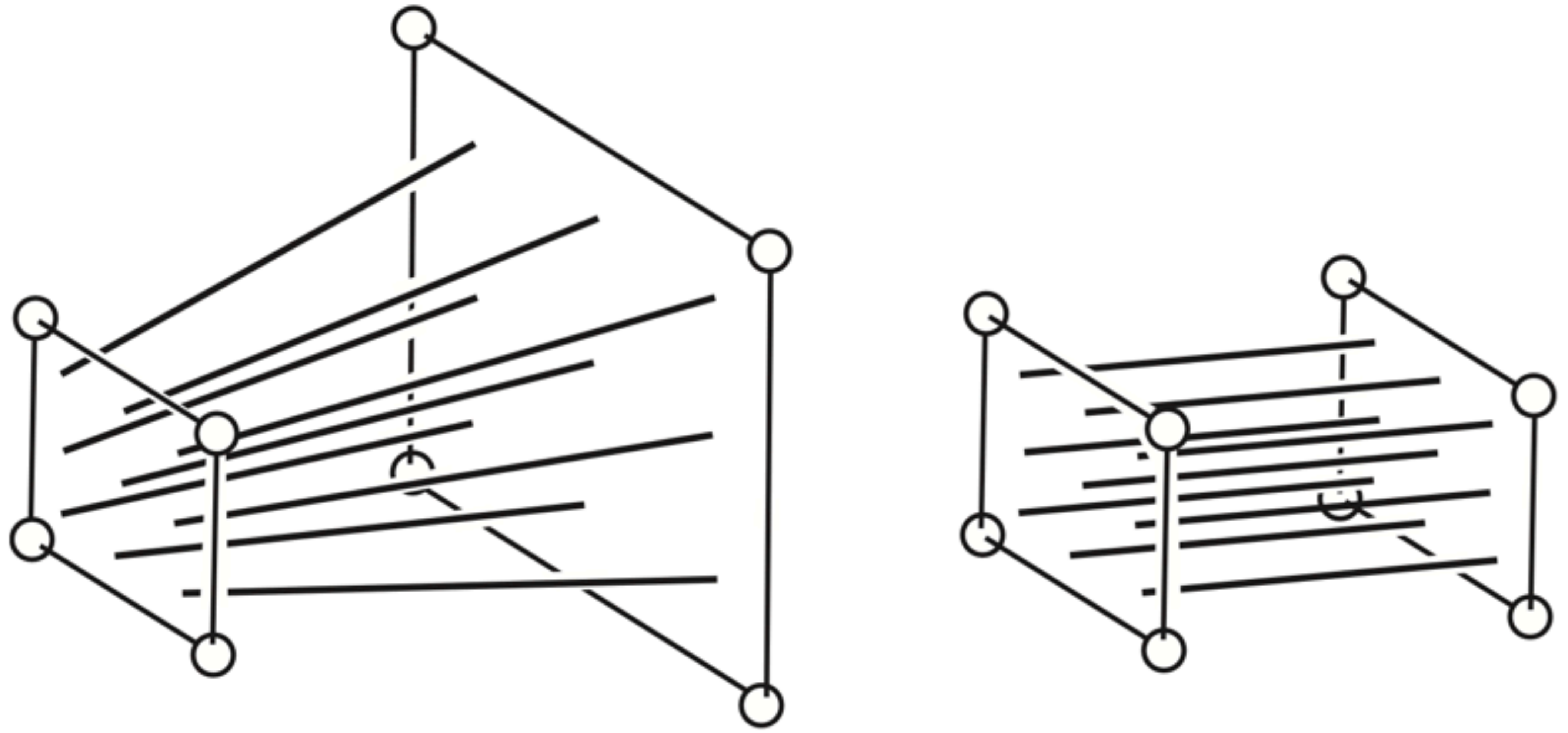
In theory, horizontal and vertical angles of view define an infinite view cone

In practice, cut off at near and far “clipping planes”: **view frustum**

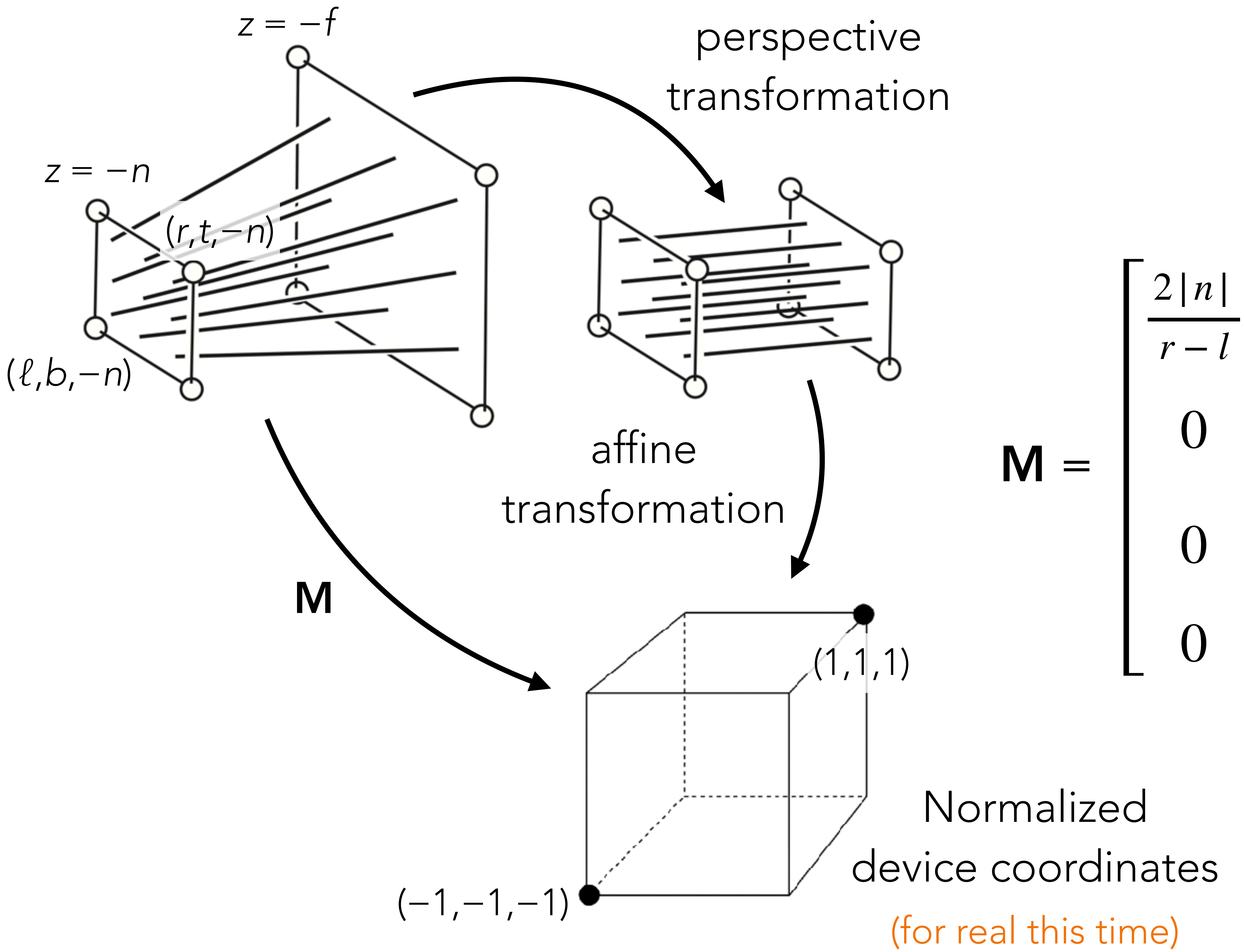
Why?

- Exclude objects behind the camera
- Finite precision of depth coordinate (we’ll see why shortly)



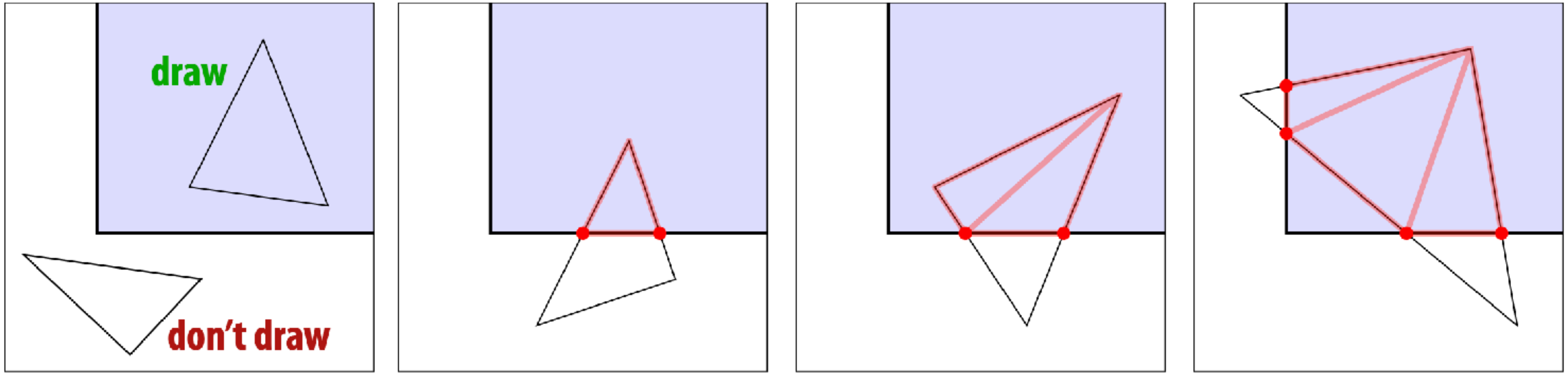


Marschner & Shirley, *Fundamentals of Computer Graphics*



$$\mathbf{M} = \begin{bmatrix} \frac{2|n|}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2|n|}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{|n|+|f|}{|n|-|f|} & \frac{2|n||f|}{|n|-|f|} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Clipping



Keenan Crane

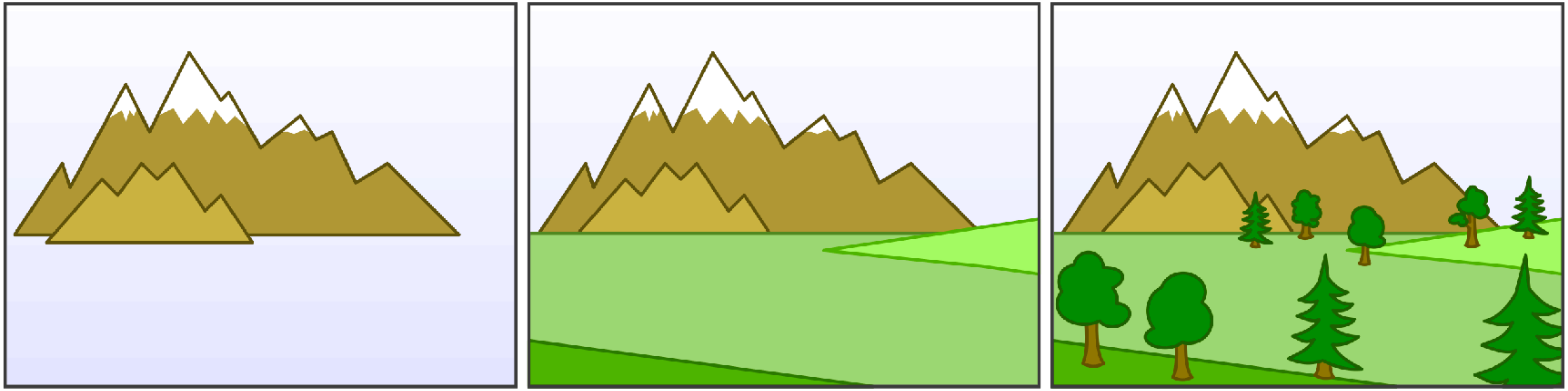
- Discard triangles outside view frustum
- Clip triangles partially intersecting view frustum

Usually implemented in homogeneous coordinates (before division)

OK, so how do we actually use z (or $1/z$) to handle visibility?

Painter's algorithm

Draw objects in "depth order" from farthest to nearest.
Nearer objects overwrite pixels painted by farther ones.



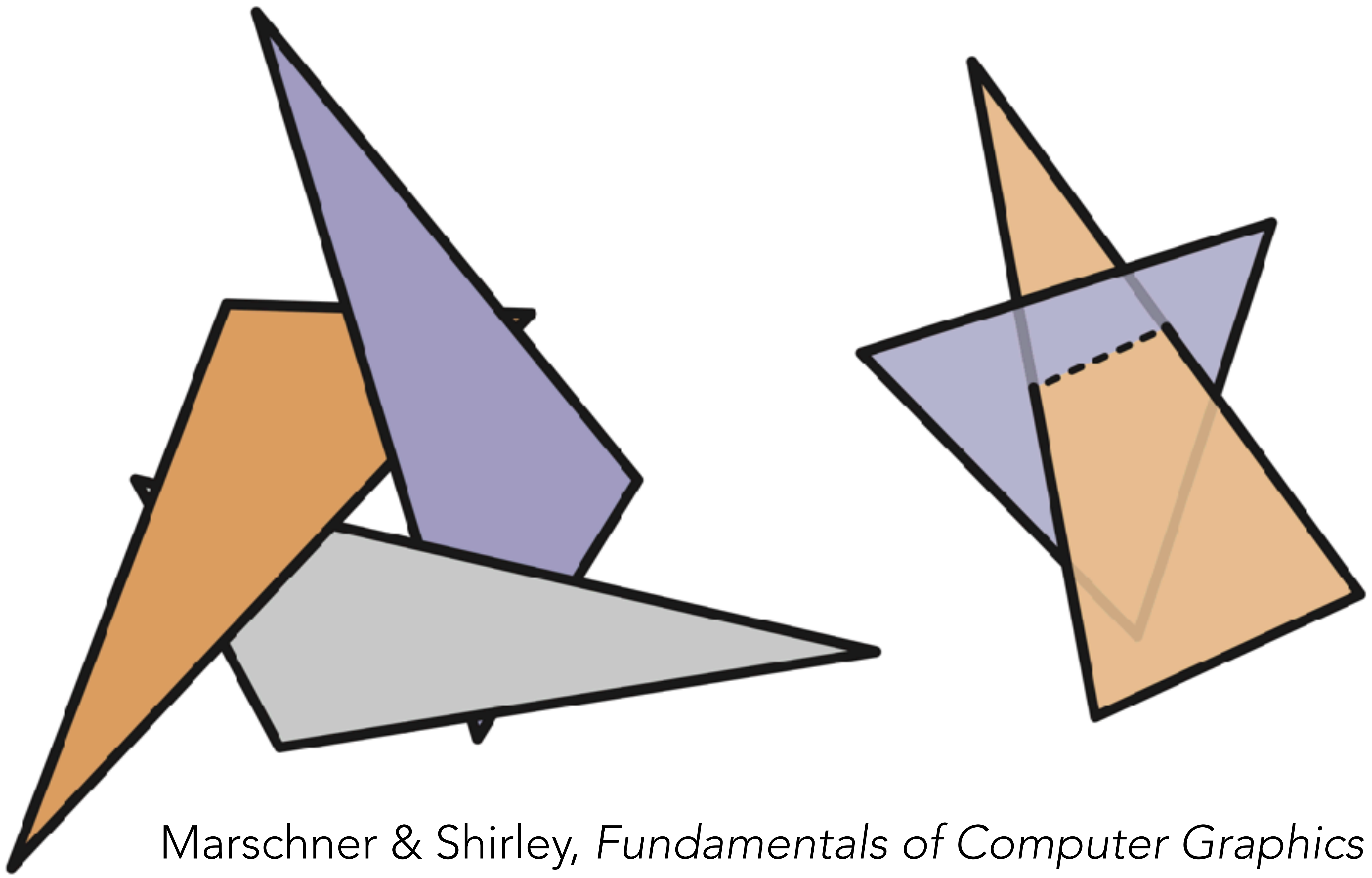
Can such a depth ordering always be found?

No:

The Lord of the Rings: The Fellowship of the Ring

Stockbusters

OK, what if we do the ordering per triangle instead of per object?



Marschner & Shirley, *Fundamentals of Computer Graphics*

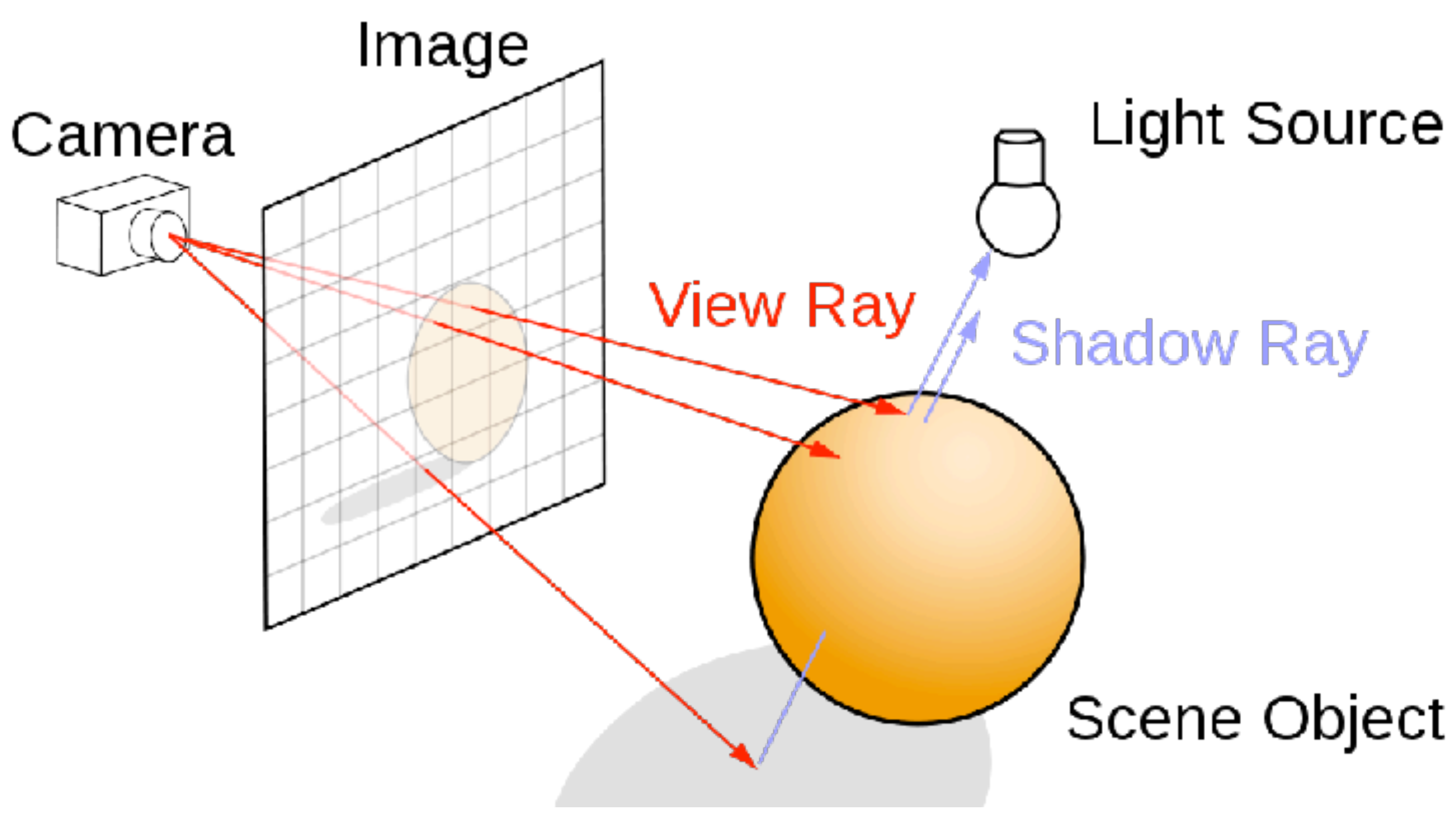
The painter's algorithm cannot handle occlusion cycles without splitting at least one of the triangles.

Practical visibility testing

Evidently we need to make visibility decisions per sample, not per triangle!

One way:

for each *sample*:
    for each *triangle* that covers it:
        if *triangle* is closest surface seen so far:
            set *sample*.colour to *triangle*.colour



This is the basic idea behind ray tracing
(covered later in the course)

Another way, more compatible with the rasterization pipeline:

for each *triangle*:
    for each *sample* that it covers:
        if *triangle* is closest surface seen by *sample* so far:
            set *sample*.colour to *triangle*.colour

This is what's actually done on the GPU!

Each sample needs to remember the closest depth it has seen, until the entire scene is rendered.

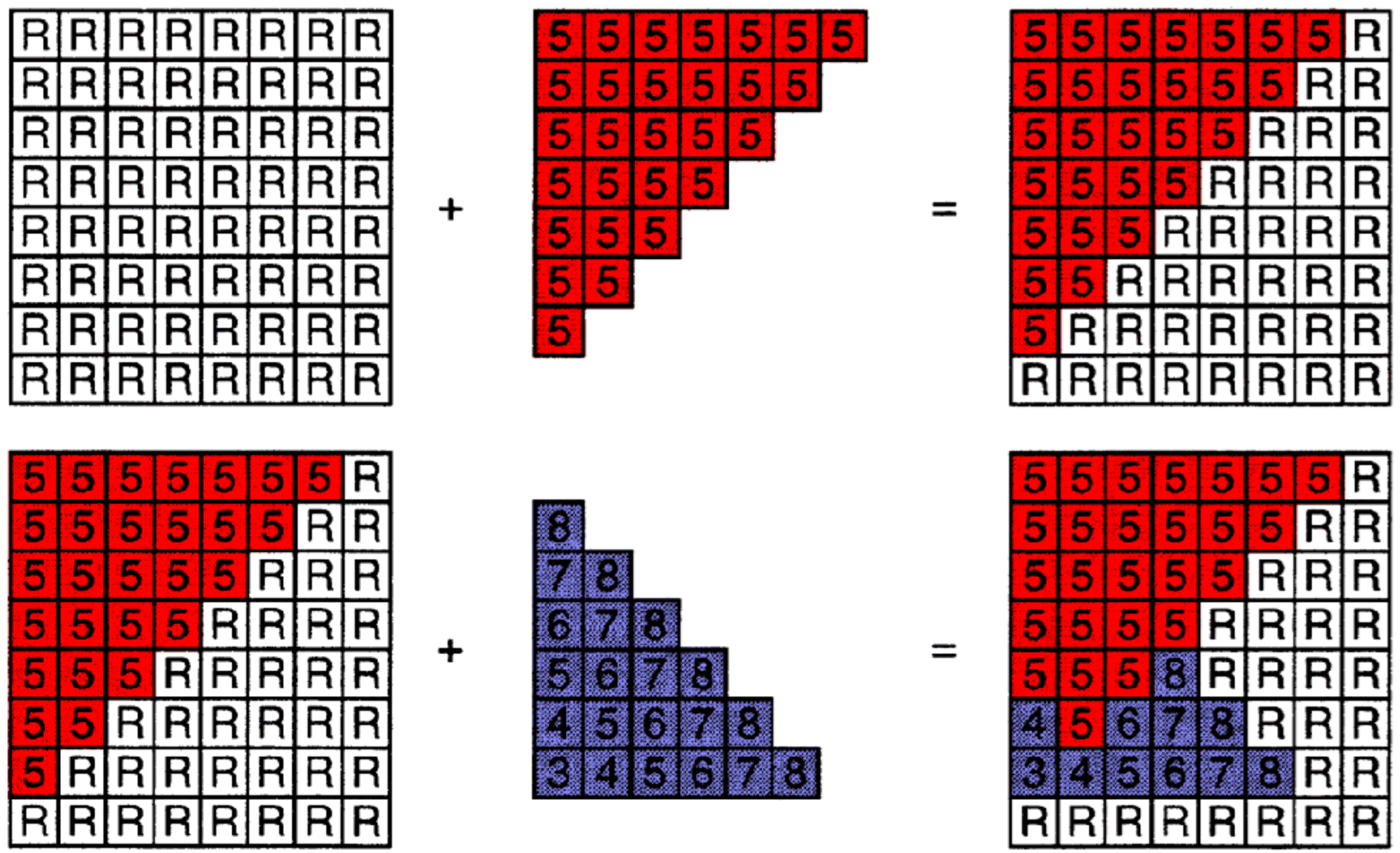
Z-buffering

Framebuffer now contains a colour buffer and a depth buffer (a.k.a. z-buffer)

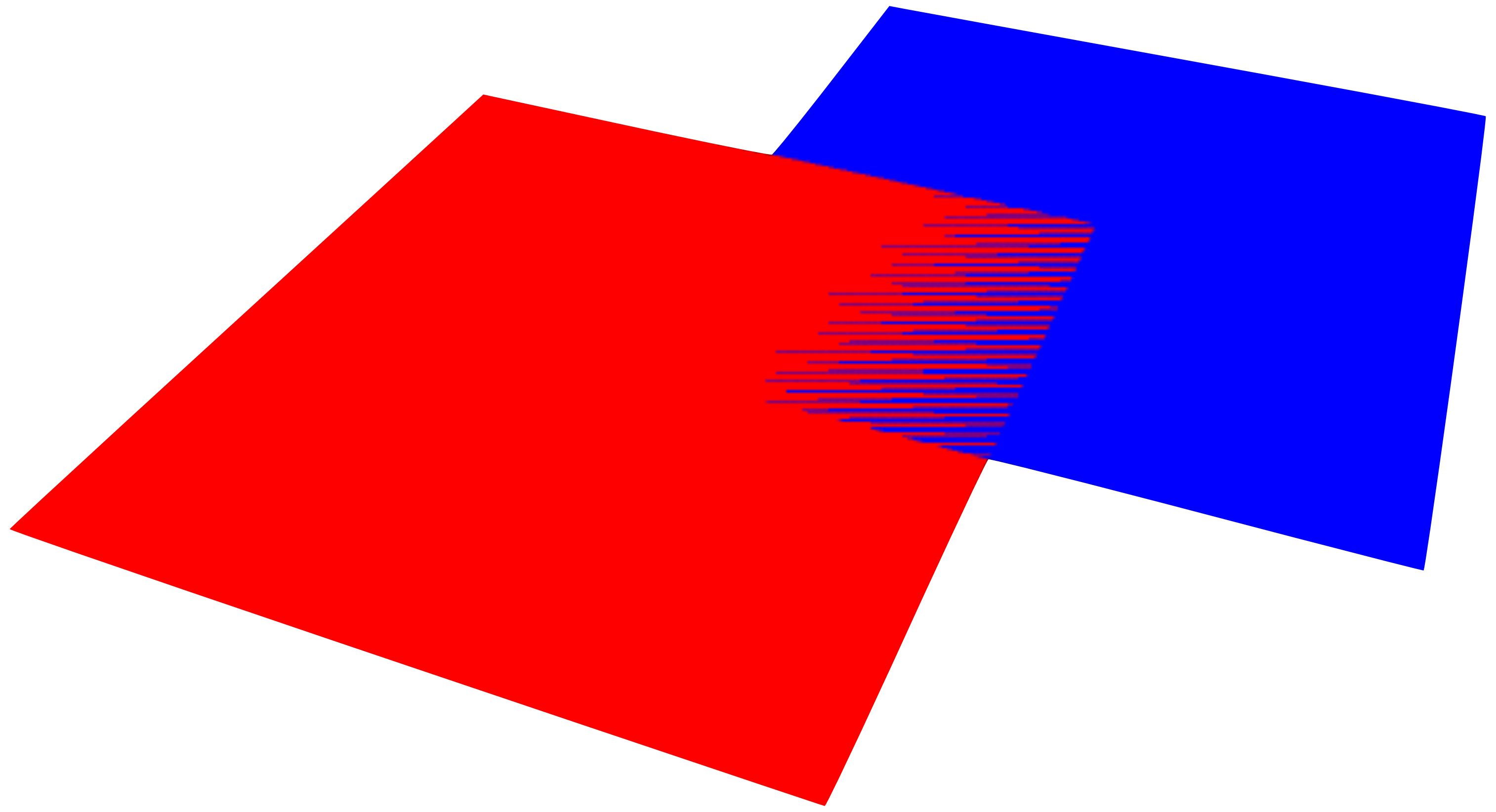
Colour

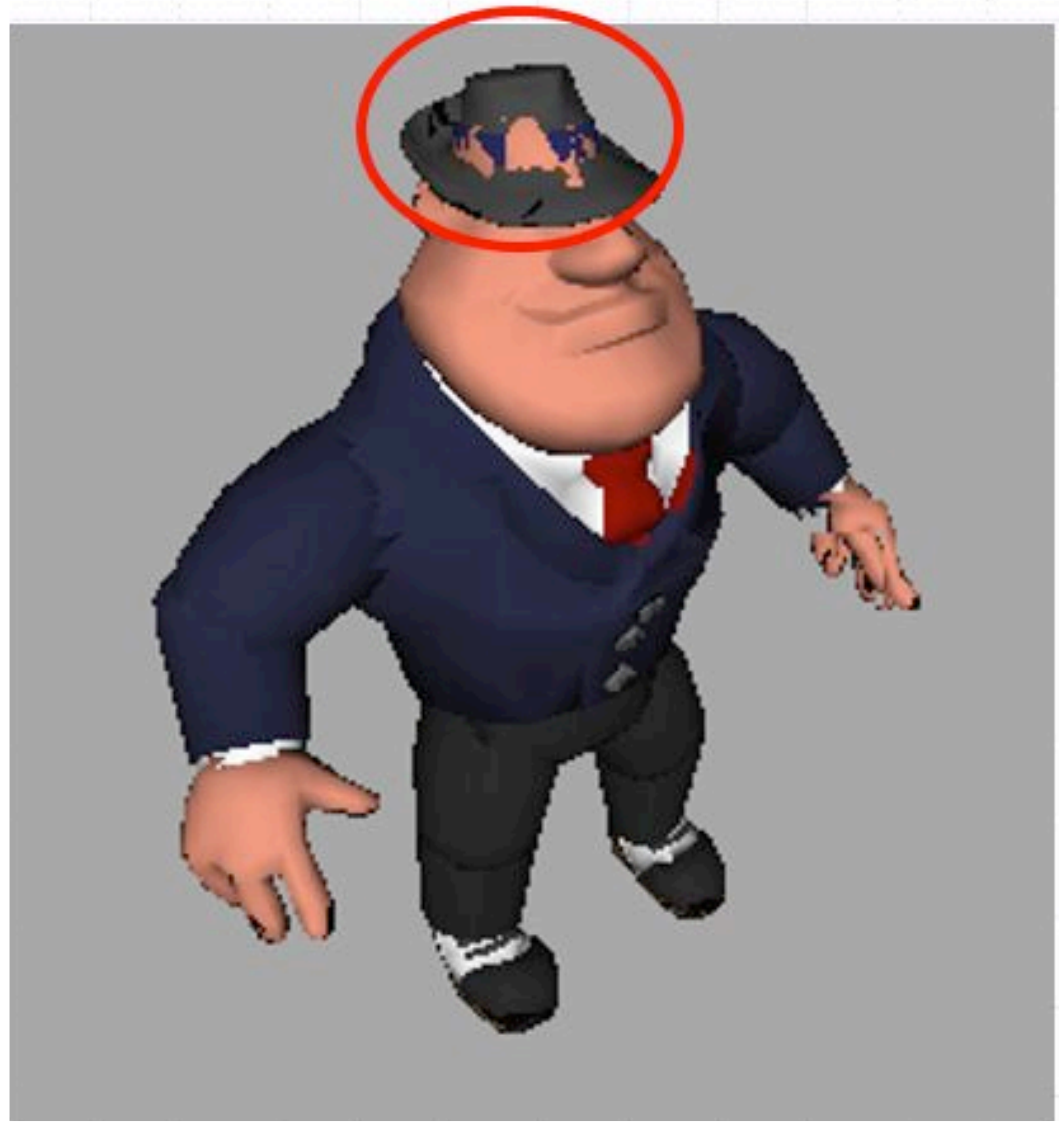
Depth

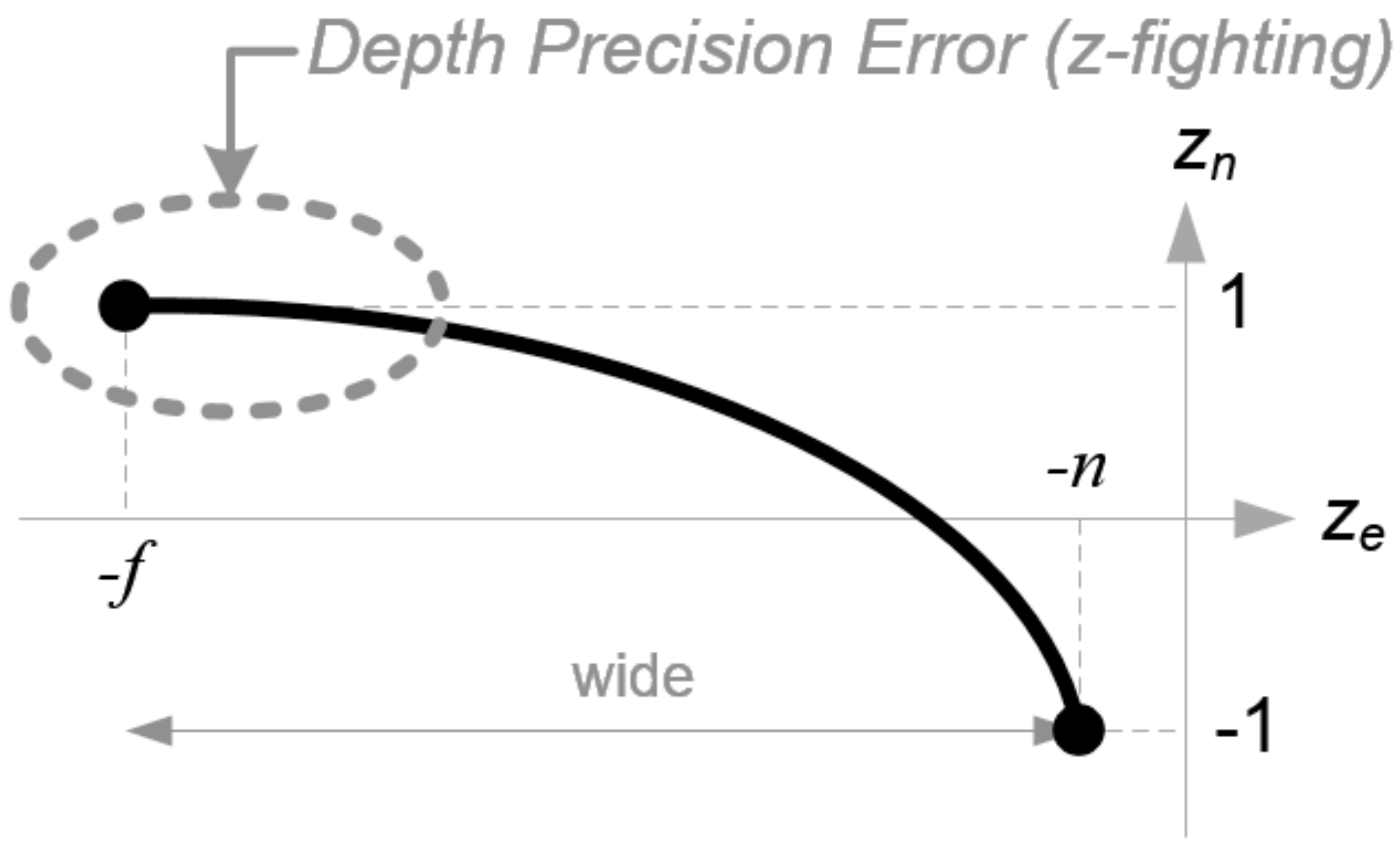
```
drawSample(x,y,z, rgb):
  if z < zbuffer[x,y]:
    color[x,y] = rgb
    zbuffer[x,y] = z
  else:
    # do nothing
```



Z-buffer can only store depth up to finite precision!



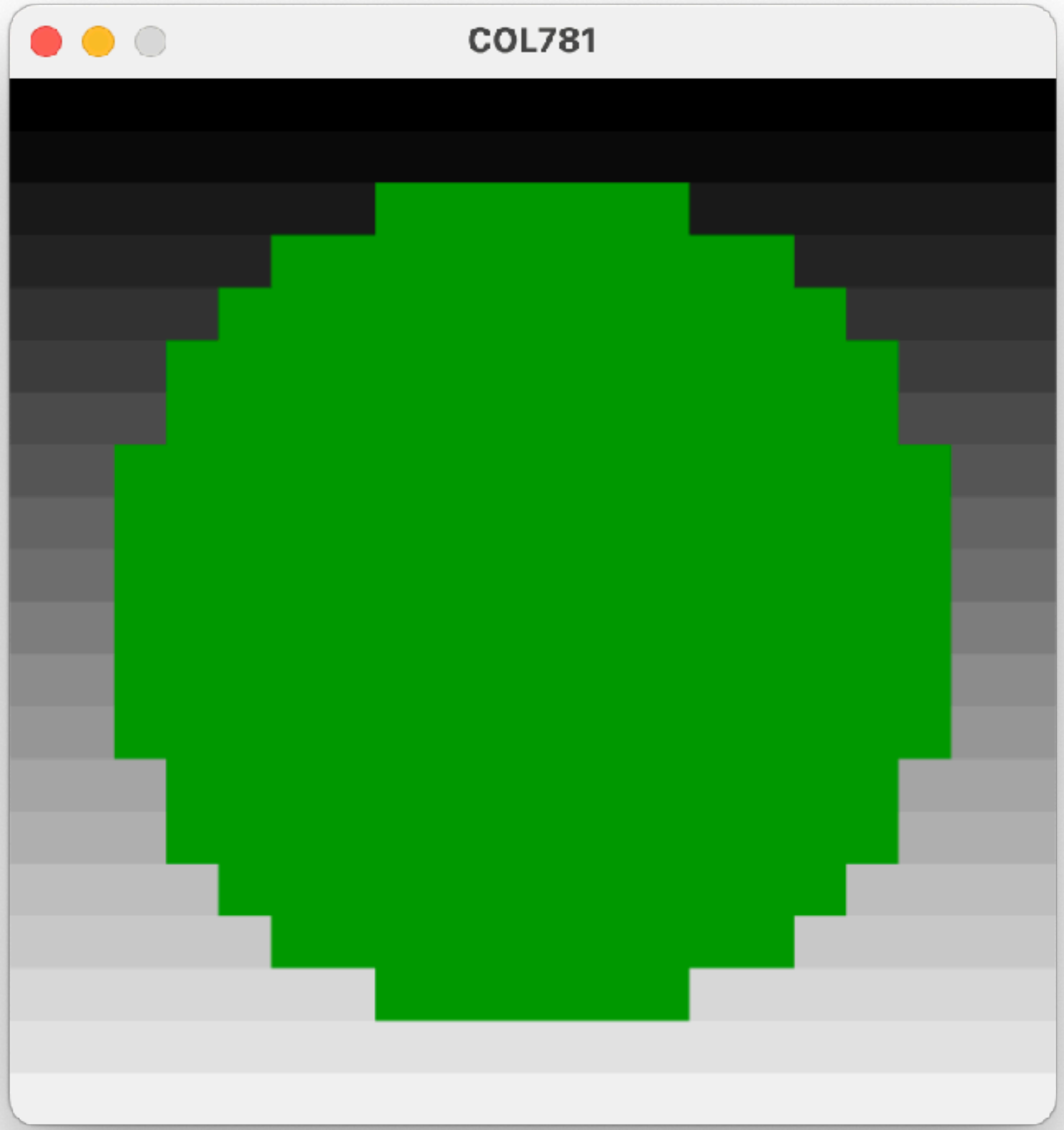




songho.ca

Different surfaces can map to same (rounded) depth: "z-fighting"

Rasterization starter code



Modify it to draw a triangle!

Warm-up for Assignment 1 (end of this week)