Visible Surface Determination

- Painter's Algorithm
- Binary Space Partitioning (BSP) Trees
- Z-Buffer
- Ray Tracing

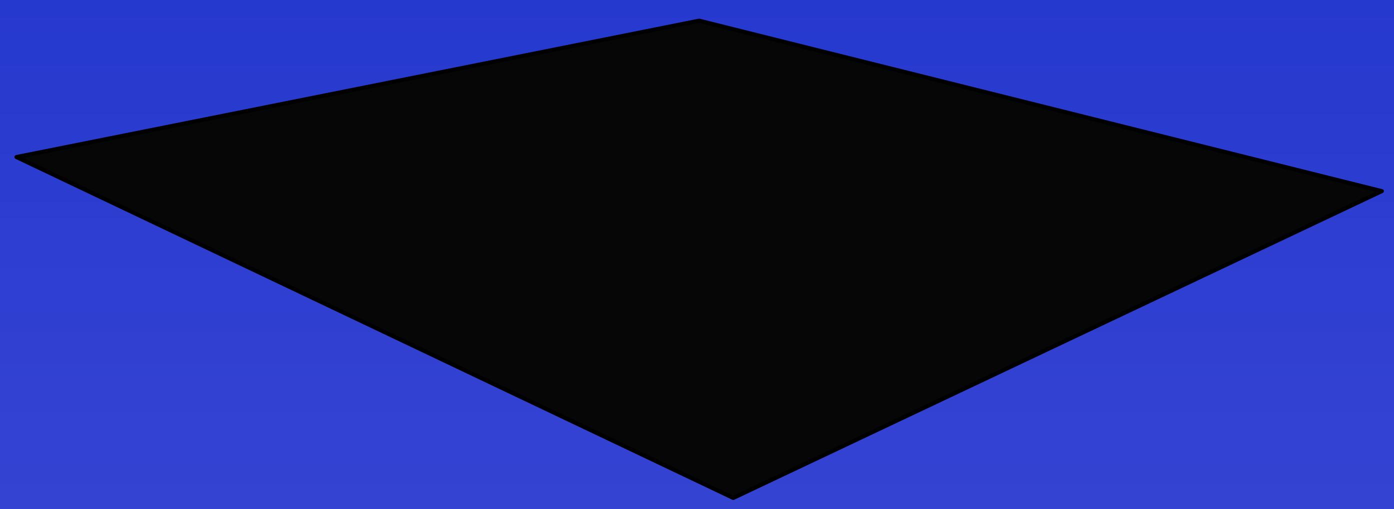
Image or object space

- Ideally an object space method converts the 3D scene into a list of 2D areas to be painted.
- Image space decides for each pixel which surface to paint.

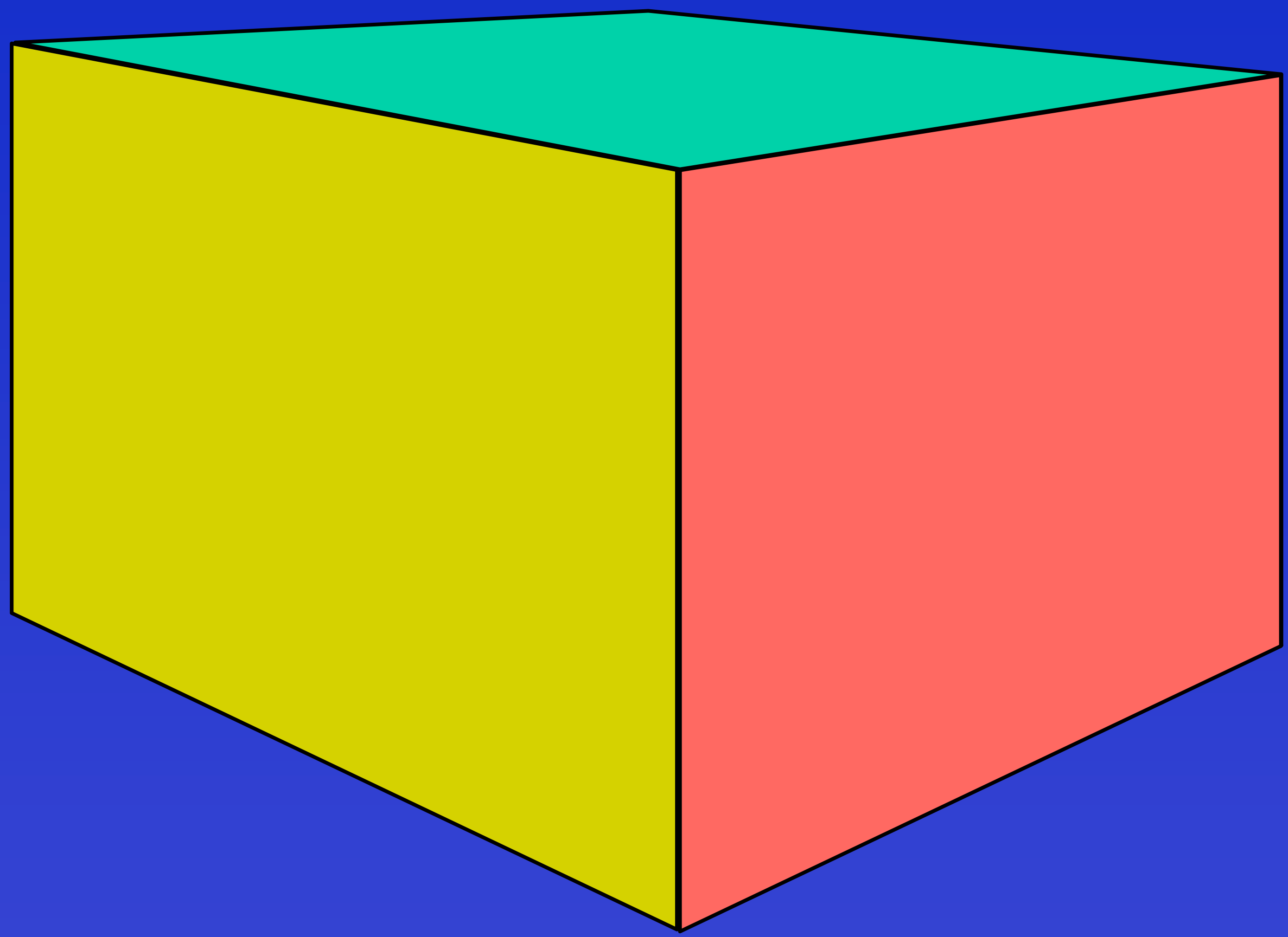
Image or object space

• Painter's Algorithm	Hybrid
• BSP Trees	Hybrid
• Z-Buffer	Image
• Ray Tracing	Object

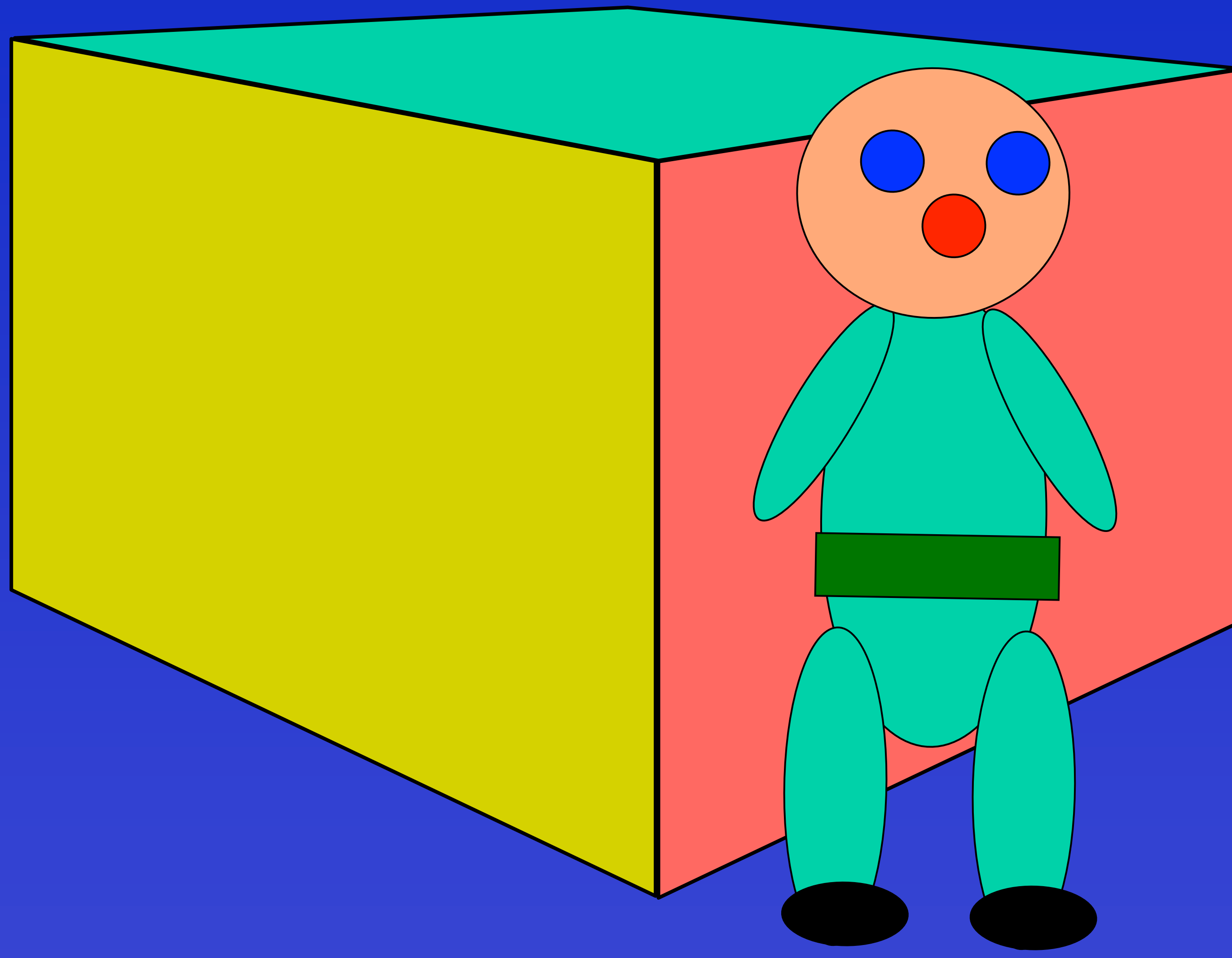
Painter's Algorithm

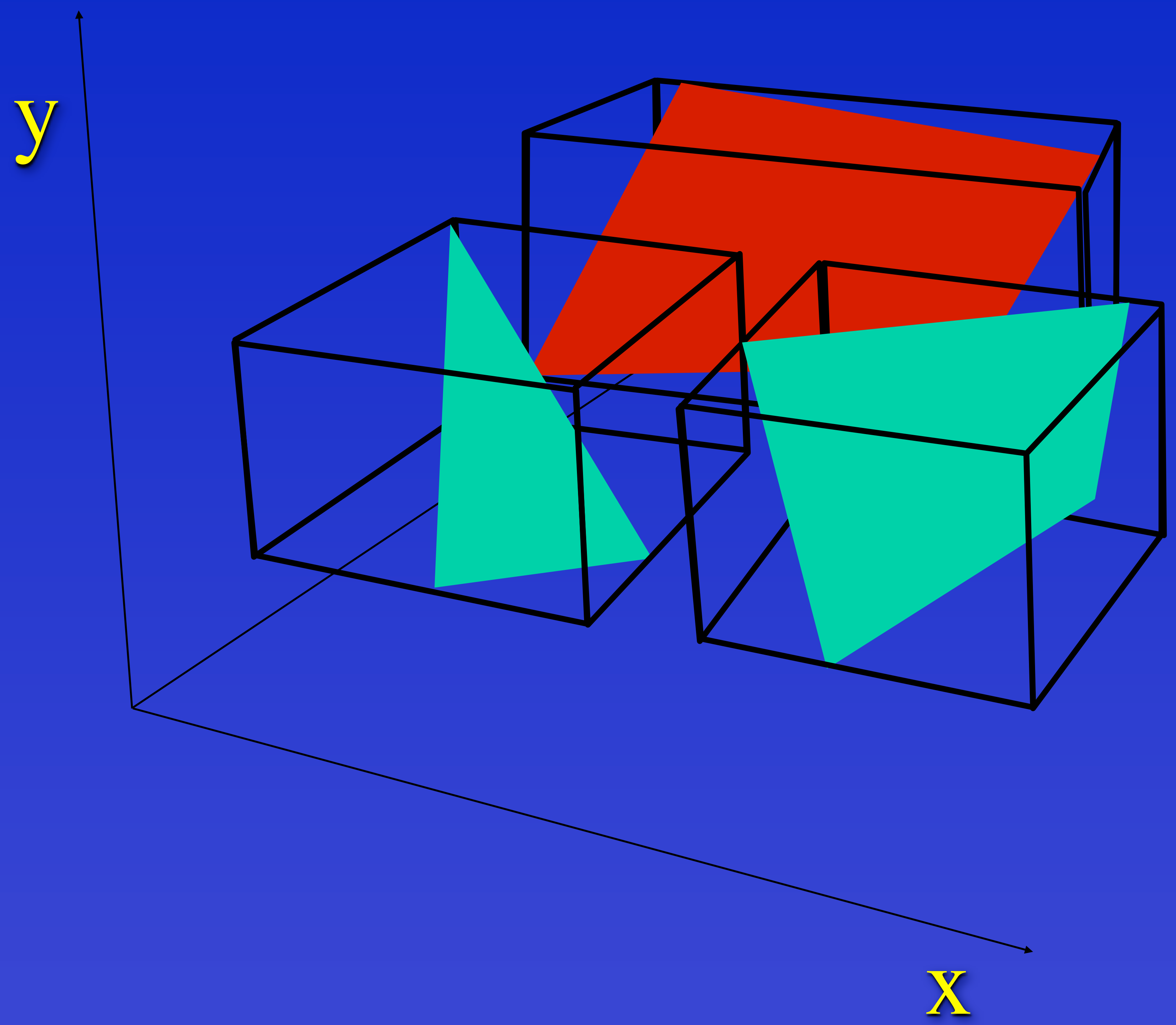


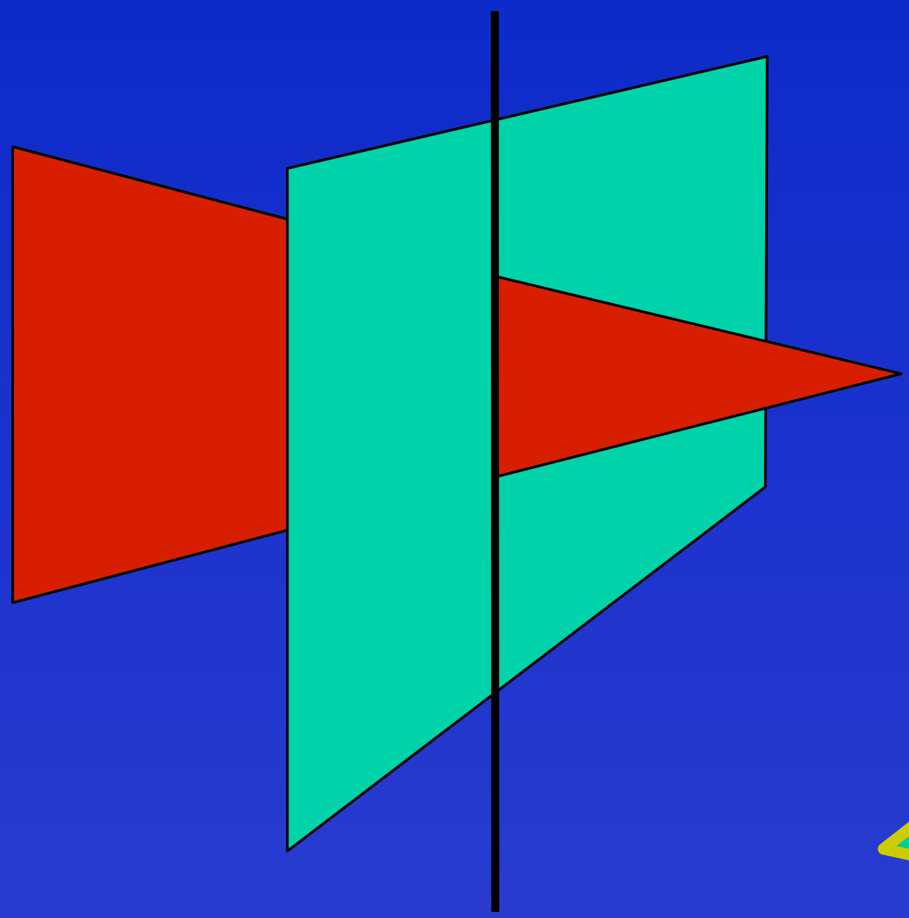
Painter's Algorithm

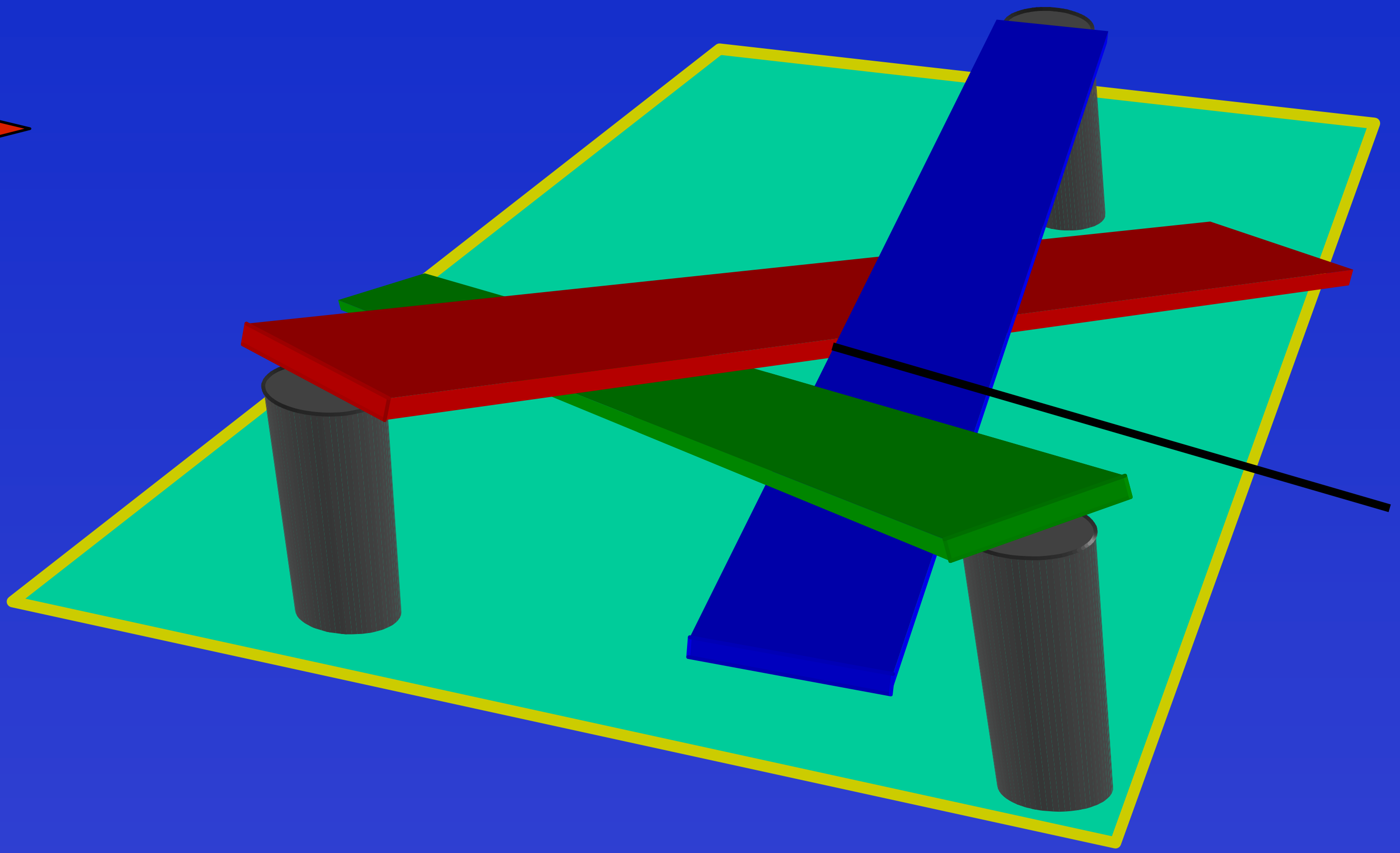


Painter's Algorithm




y
x

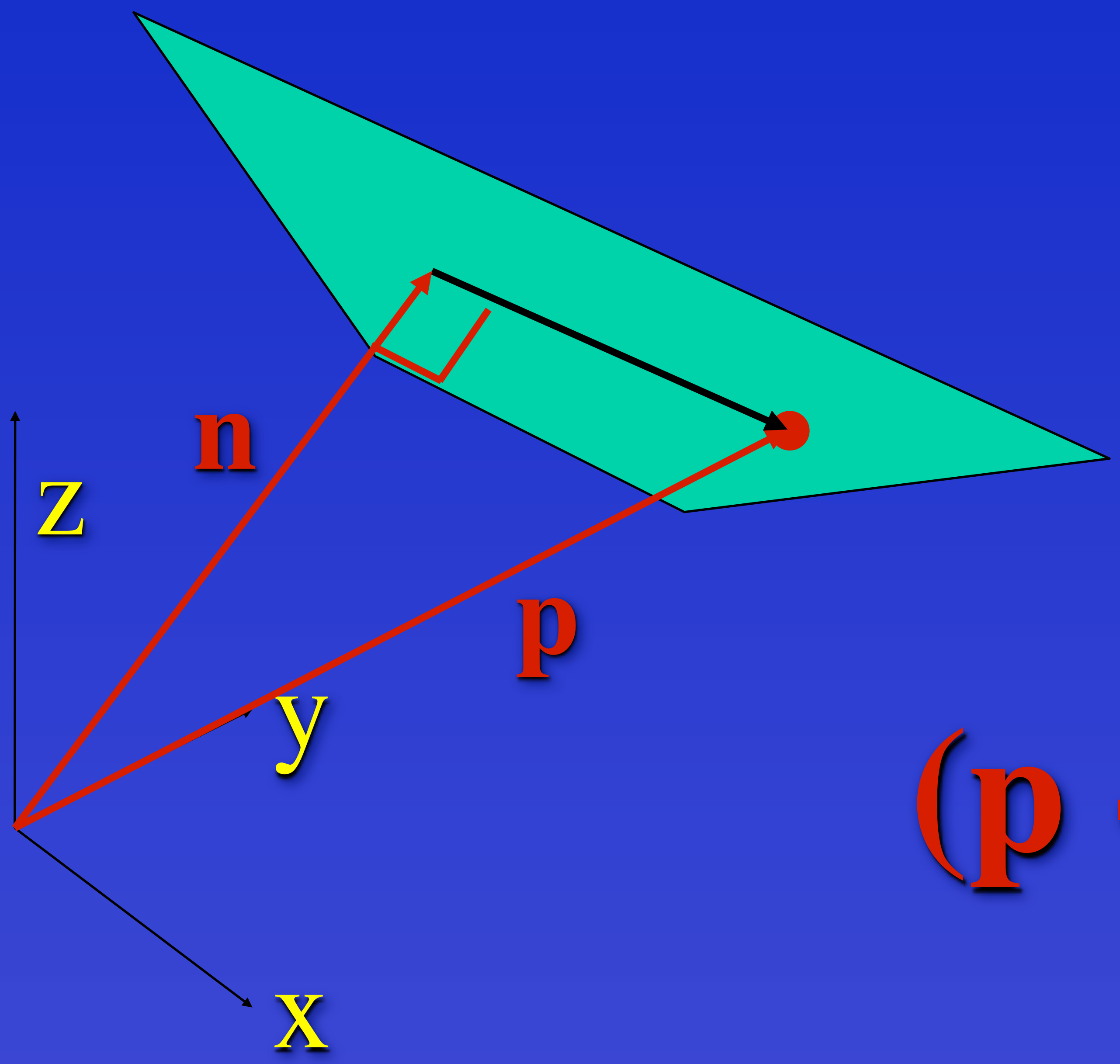




Depth Sorting

- Completely in front–put in front
- Not overlapping in x, y–either
- Intersecting–divide along intersection
- overlapping–divide along plane of one polygon.

Which side of a plane?



$$(p - n).n = 0$$

Plane Equation

$(p - n).n = 0$

$p.n - n.n = 0$

$p = (x, y, z)$

$n = (a, b, c)$

$ax + by + cz - (a^2 + b^2 + c^2) = 0$

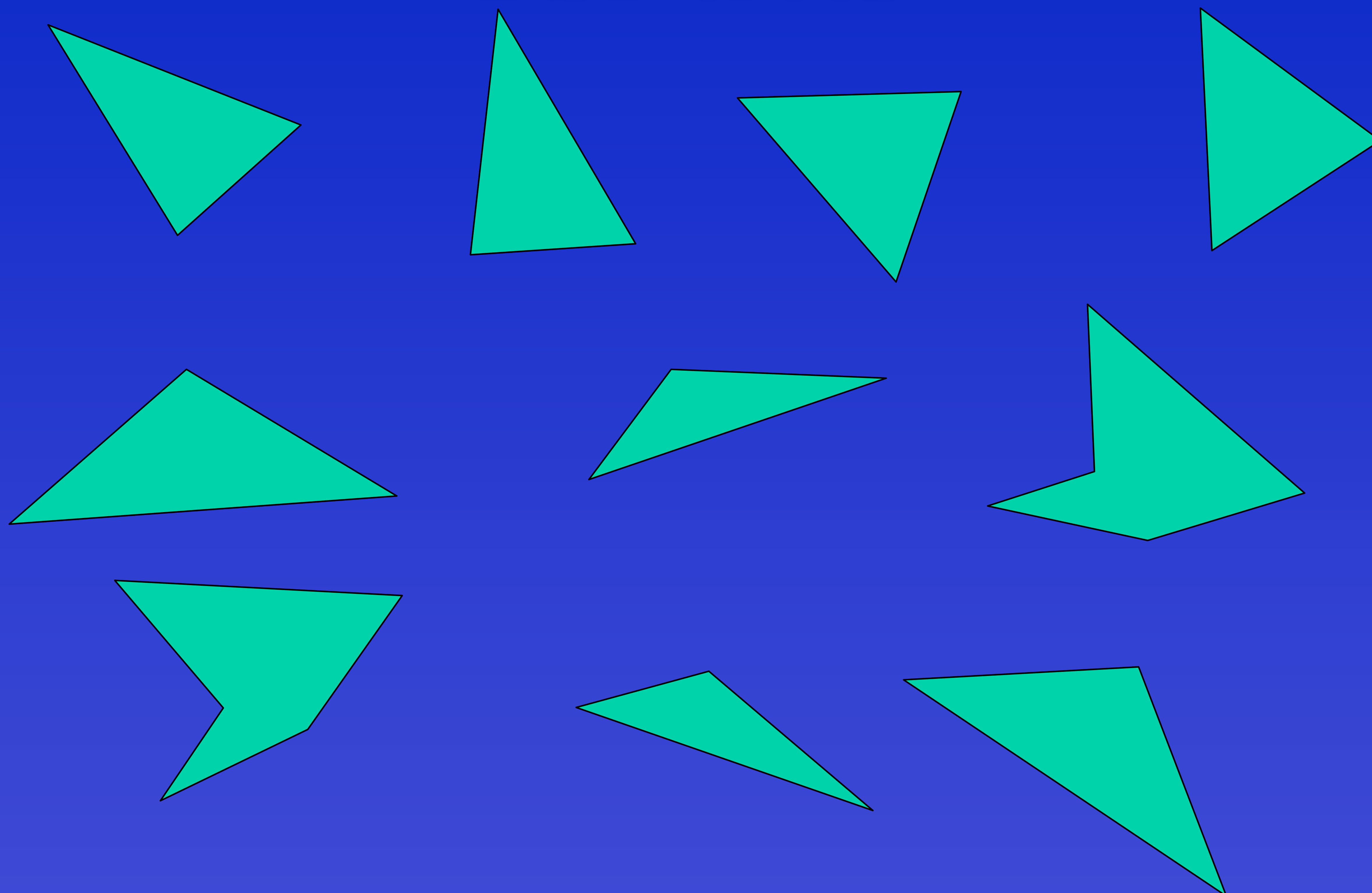
$ax + by + cz + d = 0$

For points p and q

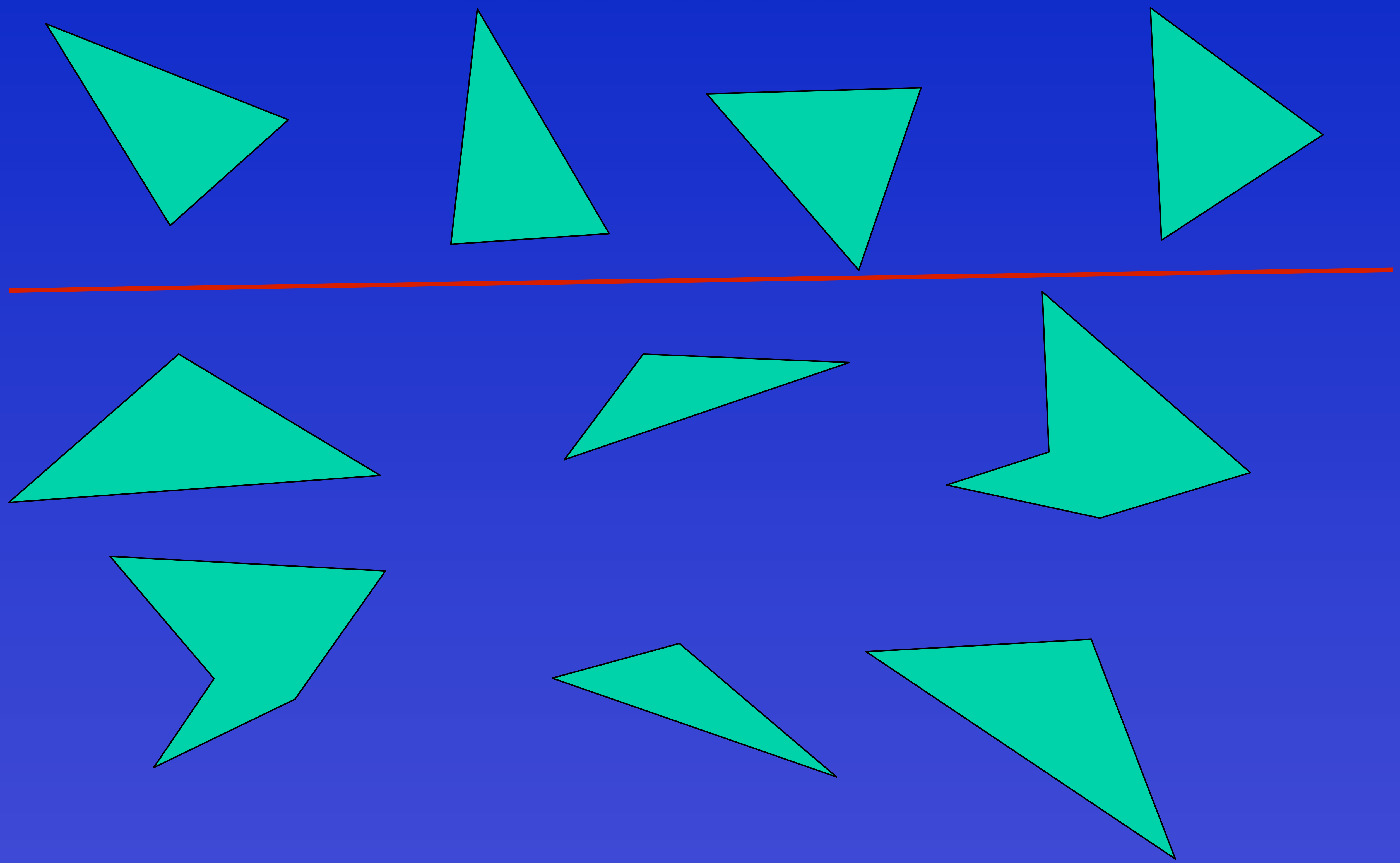
if $(p-n).n > 0$ and $(q-n).n > 0$
or if $(p-n).n < 0$ and $(q-n).n < 0$

p and q are on the same side

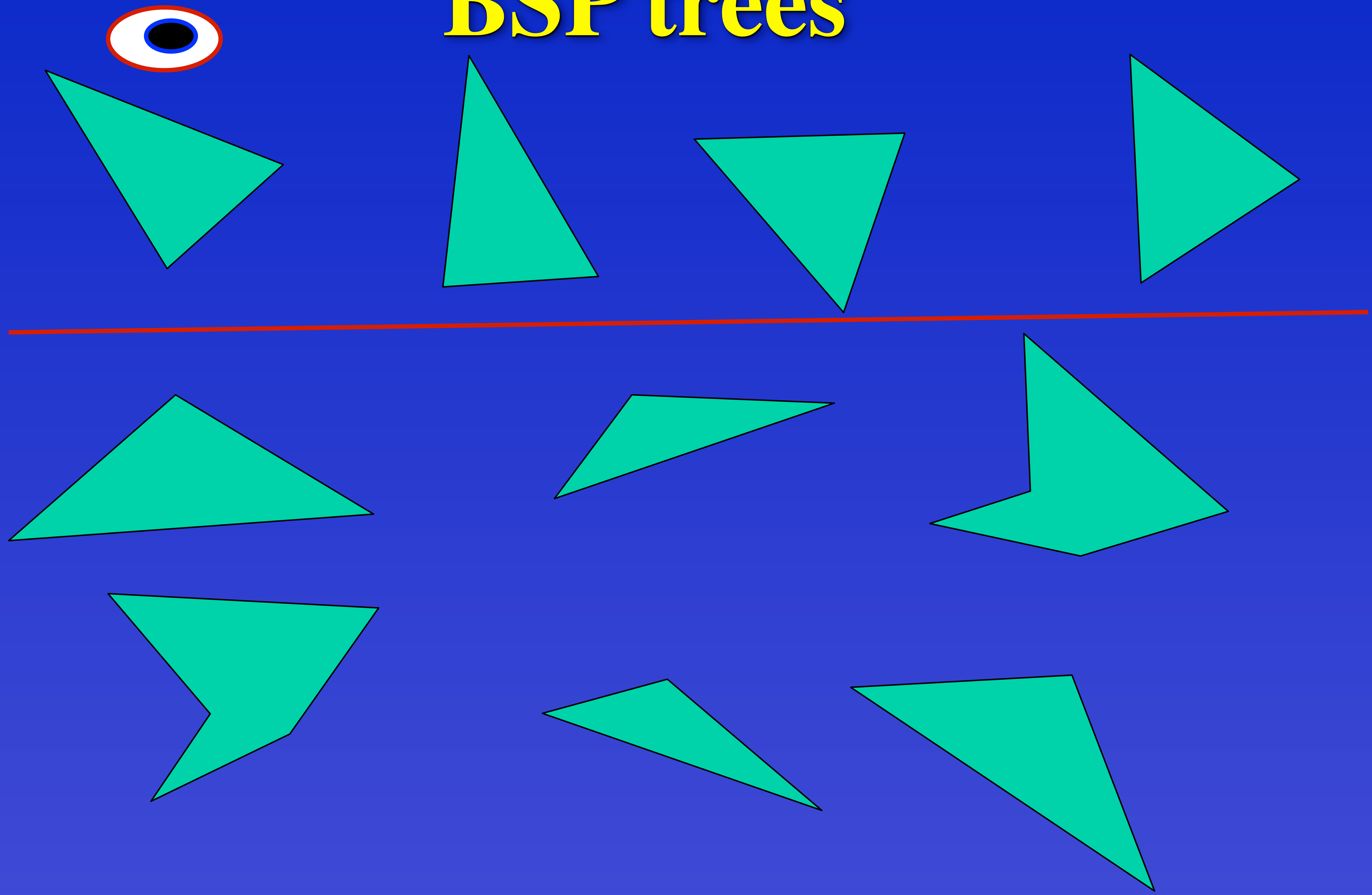
BSP trees



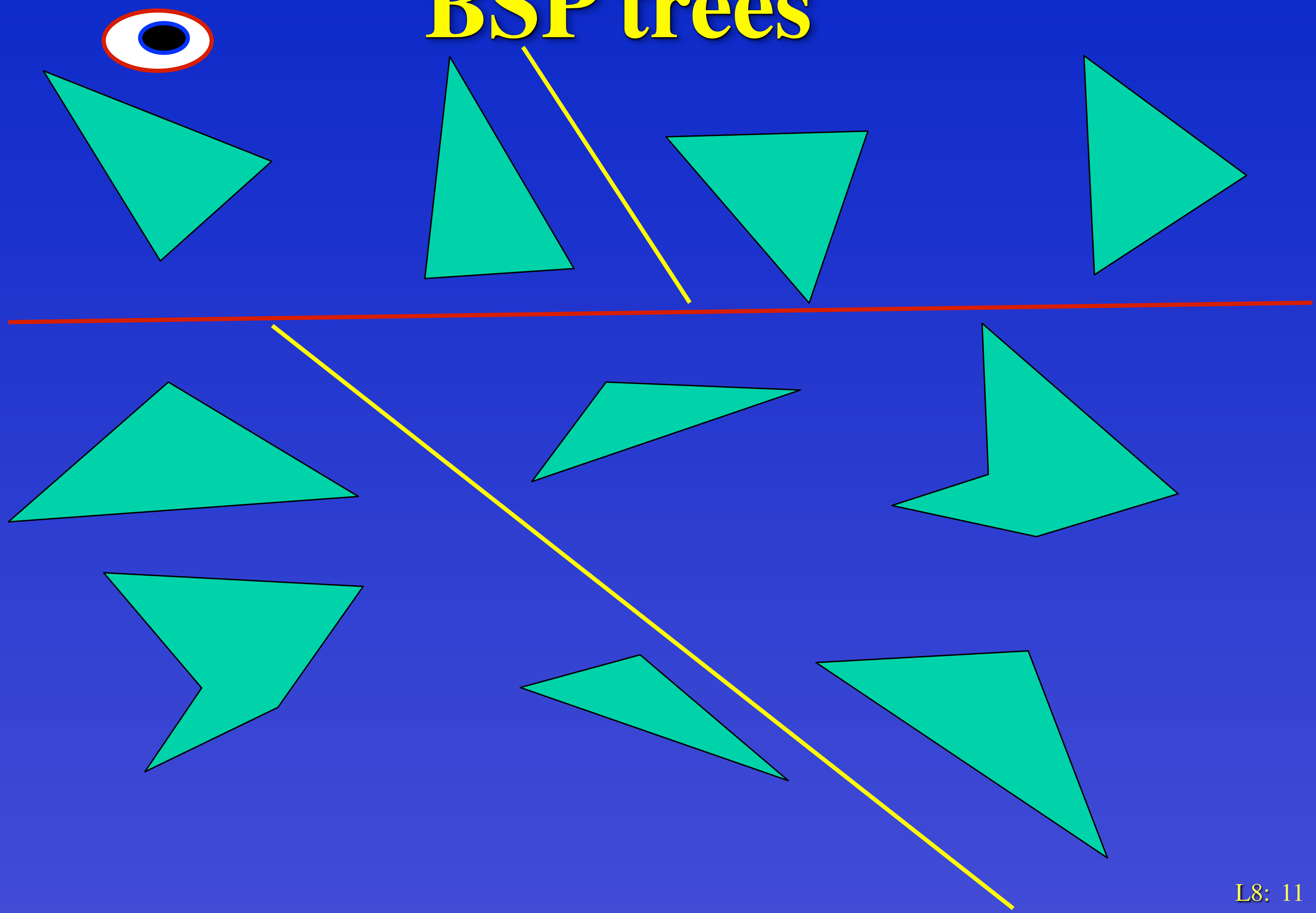
BSP trees



BSP trees



BSP trees



BSP trees

BSP trees

Divide scene with a plane

- Everything on the same side of that plane as the eye is in front of everything else (from that eye's view)

- Divide front and back with more planes
- If necessary split polygons by planes

Efficiency

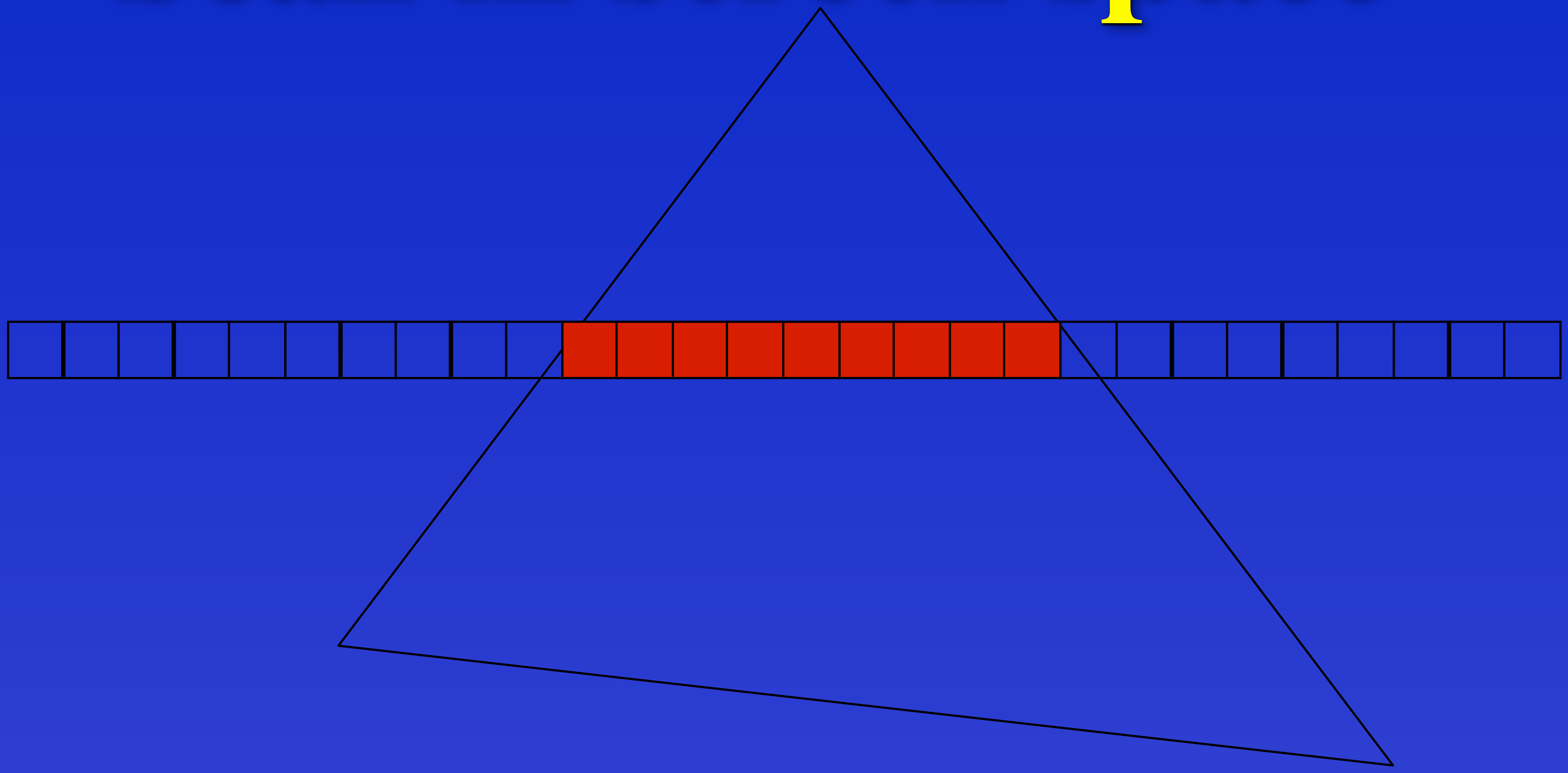
- BSP trees are order n*log(n) in the number of polygons

- They are good for VR ‘walkthroughs’ because you only re-compute traversal when the eye crosses a separating plane

Z-Buffer

- Record r,g,b and z (depth) for each pixel.
- Process each polygon line by line and if closer replace r,g,b,z in the buffer.

Scan in screen space



Finding the depth

- Plane equation is $Ax + By + Cz + D = 0$
  $z = -(Ax + By + D)/C$

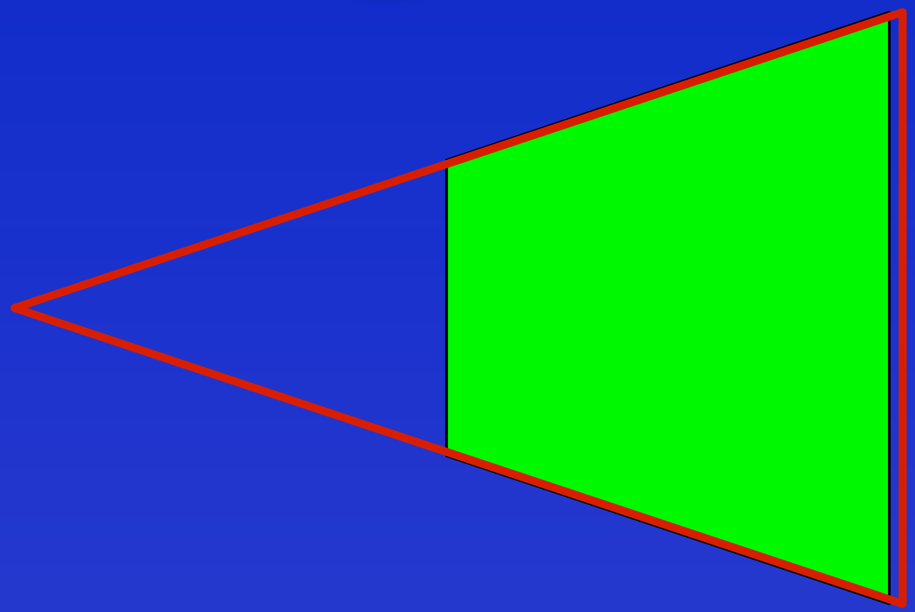
- replace x by x+1
  $z' = -(A(x+1) + By + D)/C$
  $\Delta z = z' - z = -A/C$

- New z is found by adding a constant.

What about the lost z?

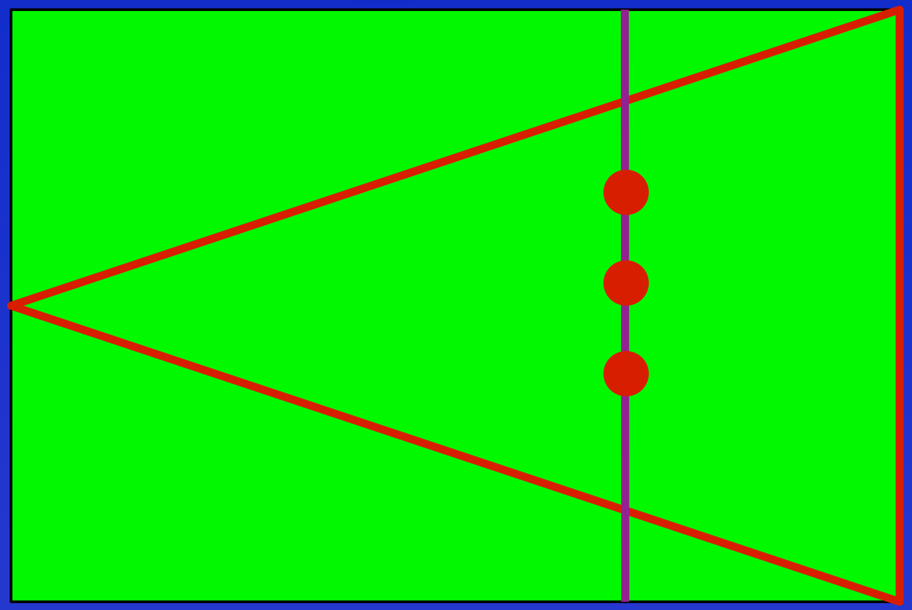
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z \end{bmatrix} \equiv (x/z, y/z, 1)$$

Perspective Transformation



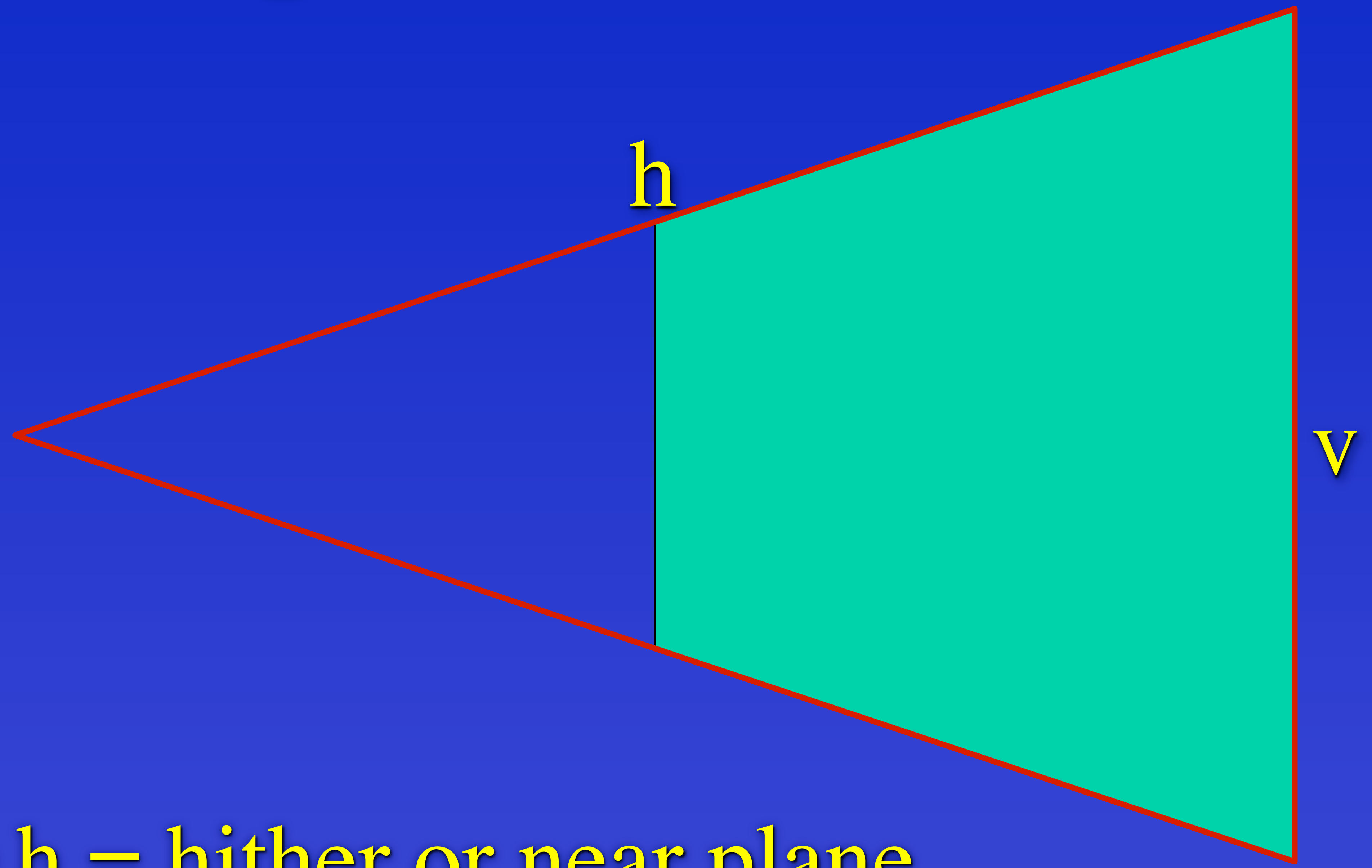
- Preserve x', y'
- Preserve straight lines
- z' independent of x, y

Perspective Transformation



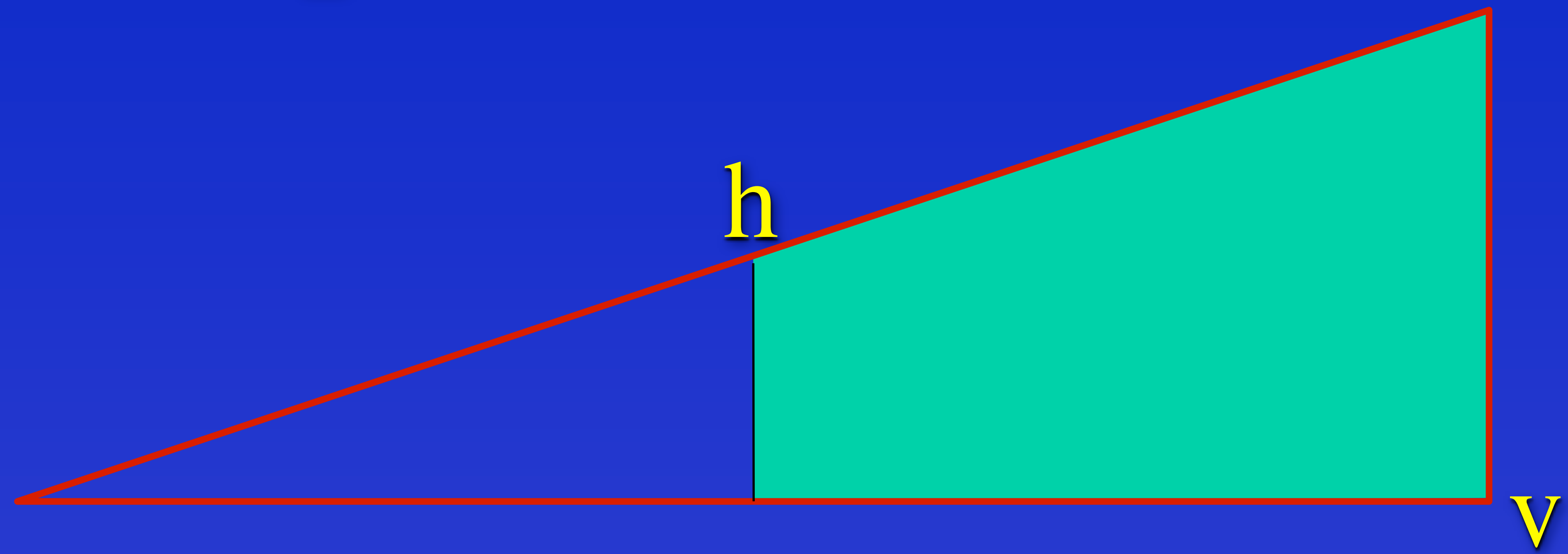
- Preserve x', y'
- Preserve straight lines
- z' independent of x, y

Perspective Transformation

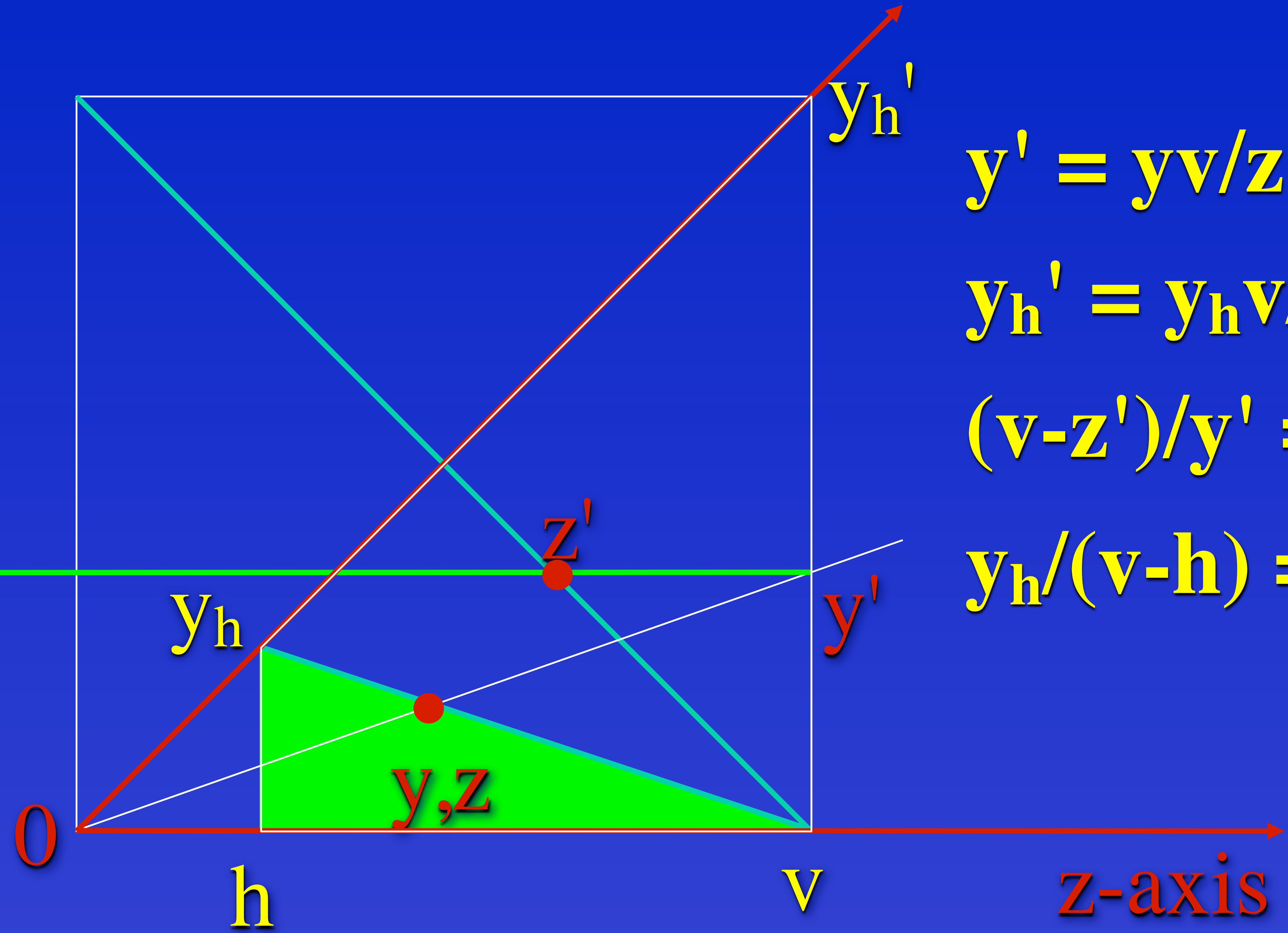


- h = hither or near plane
- v = projection plane

Perspective Transformation



- h = hither or near plane
- v = projection plane



$y' = yv/z$

$y_h' = y_h v/h$

$(v-z')/y' = v/y_h'$

$y_h/(v-h) = y/(v-z)$

1. $y' = yv/z$
2. $y_h' = y_h v/h$
3. $(v-z')/y' = v/y_h'$
4. $y_h/(v-h) = y/(v-z)$

1. $y' = yv/z$
2. $y_h' = y_h v/h$
3. $(v-z')/y' = v/y_h'$
4. $y_h/(v-h) = y/(v-z)$

$(v-z')/(yv/z) = v/(y_h v/h)$

$(v-z')/(yv/z) = v/((y(v-h)/(v-z))v/h)$

$(v-z')/(v/z) = v/(((v-h)/(v-z))v/h)$

$(v-z')z = vh/((v-h)/(v-z))$

$$(v-z')z = vh/((v-h)/(v-z))$$

$$(v-z')z(v-h)/(v-z) = vh$$

$$(v-z')z(v-h) = vh(v-z)$$

$$v-z' = vh(v-z)/z(v-h)$$

$$z' = v - vh(v-z)/z(v-h)$$

$$z' = (vz(v-h) - vh(v-z))/z(v-h)$$

$$z'z = (v^2z - vzh - v^2h + vhz)/(v-h)$$

$$z'z = (v^2z - v^2h)/(v-h)$$

$$z'z = v^2z/(v-h) - v^2h/(v-h)$$

$$z'z = v^2 z/(v-h) - v^2 h/(v-h)$$

In the case where v = 1 (PHIGS GL?)

$$z'z = z/(1-h) - h/(1-h)$$

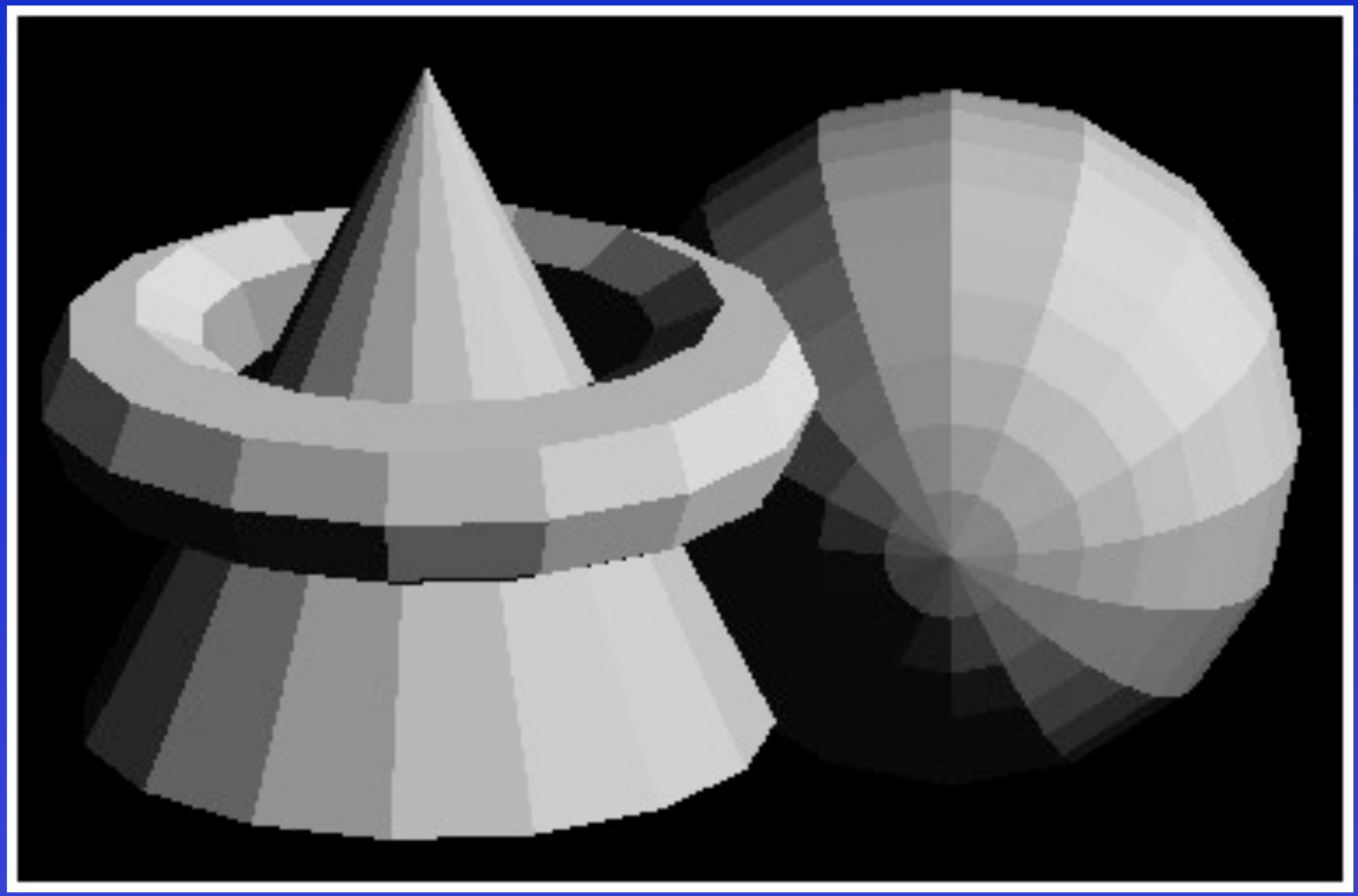
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/(1-h) & -h/(1-h) \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

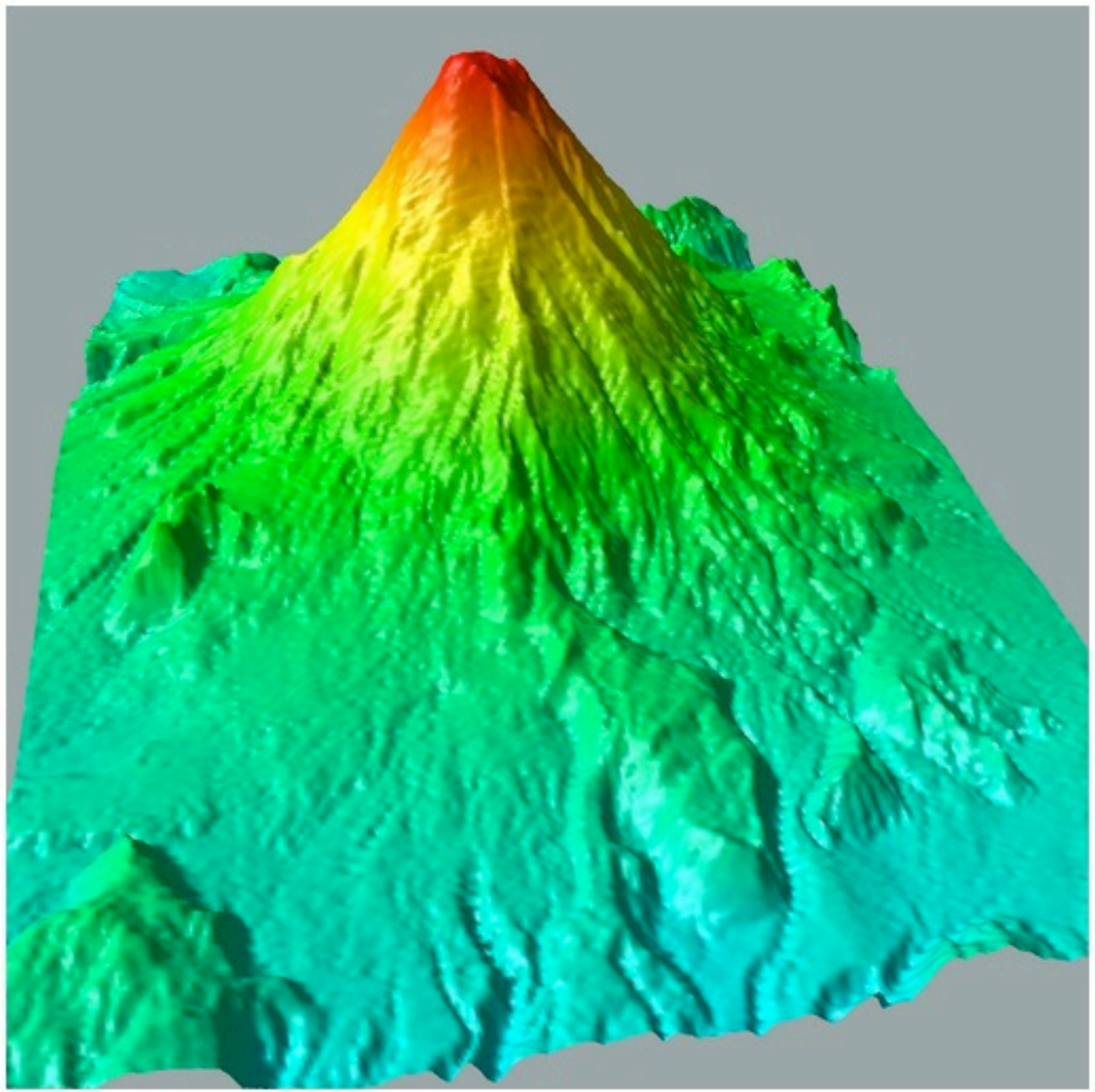
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/(1-h) & -h/(1-h) \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

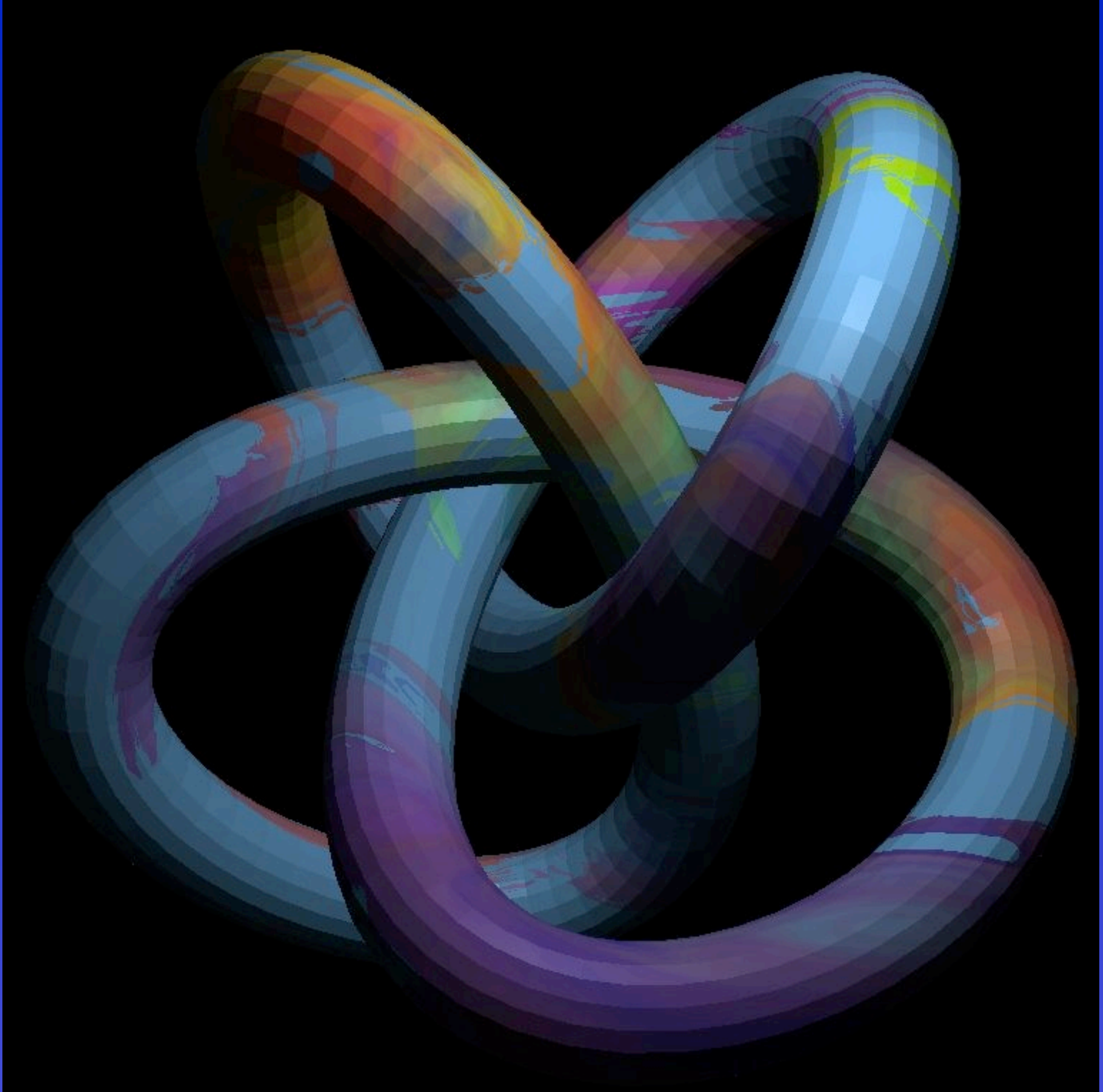
$$= \begin{bmatrix} x \\ y \\ z/(1-h)-h/(1-h) \\ z \end{bmatrix}$$

Many appropriate matrices

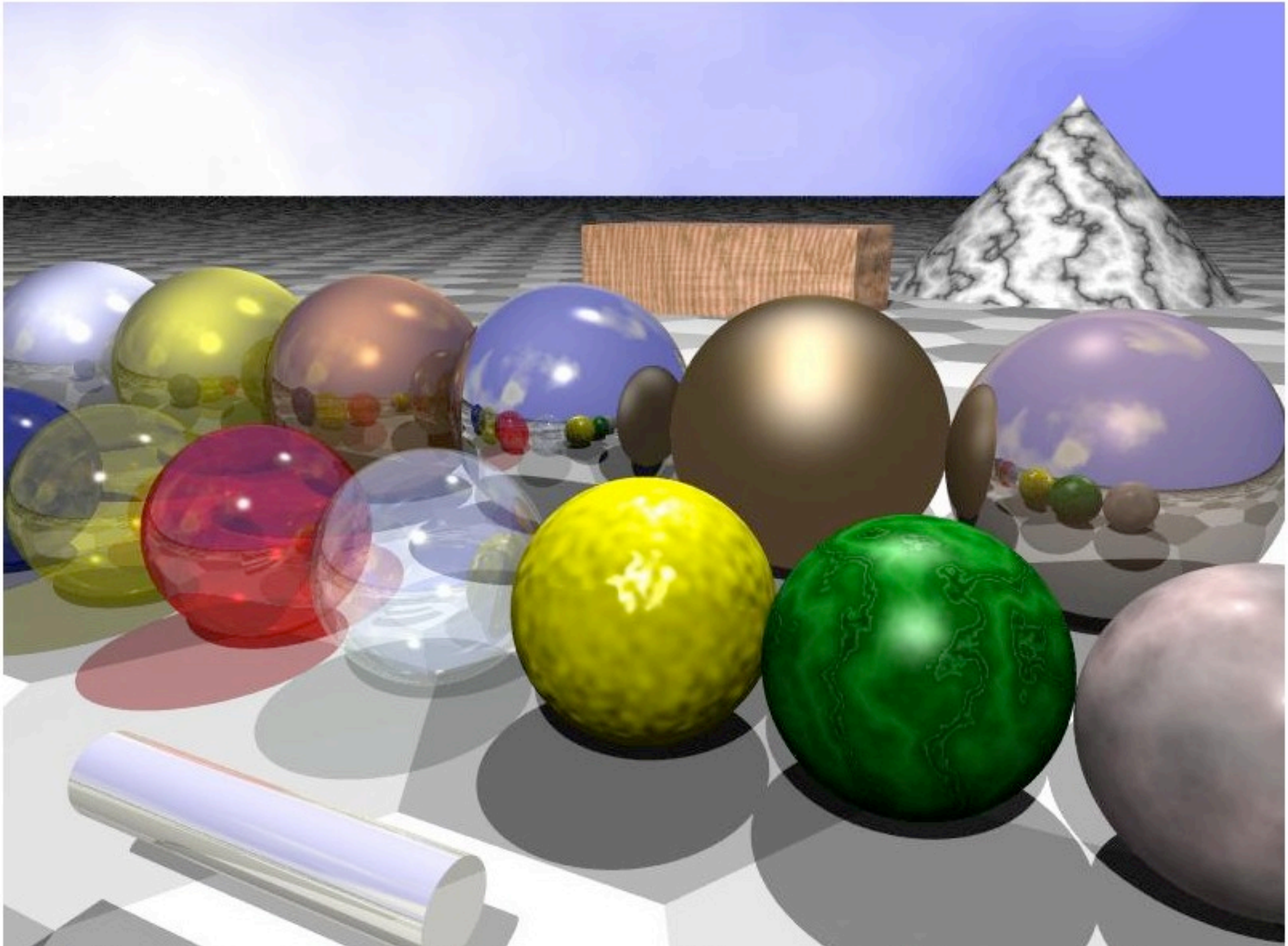
- Similar matrices appear in many different forms
  - Different possible eye position, near plane and view plane configurations
  - some books include an additional transformation to screen coordinates
  - There is not one right answer!








tenui



We move onto Ray Tracing next...