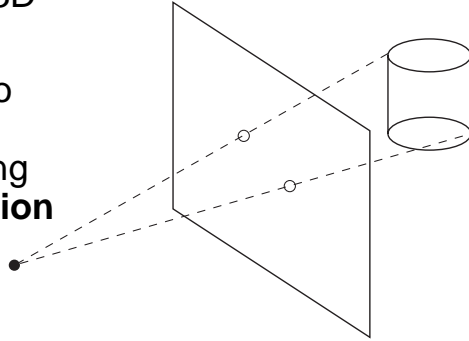


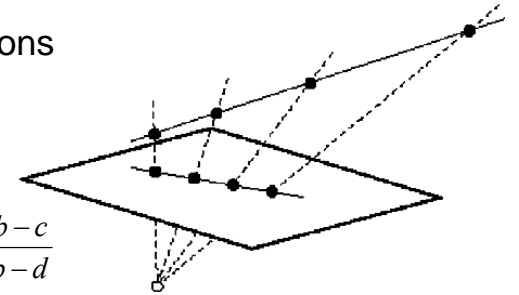
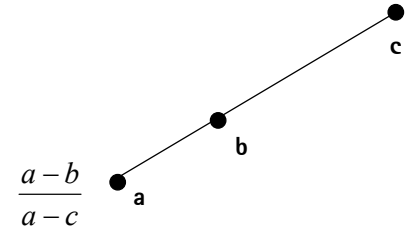
# Projections

- **Projections** transform points in  $n$ -space to  $m$ -space, where  $m < n$
- In graphics, we map from 3D to 2D
- Map points from 3-space to the **projection plane (PP)** along **projectors** emanating from the **center of projection (COP)**



# Properties: projections vs. Euclidean

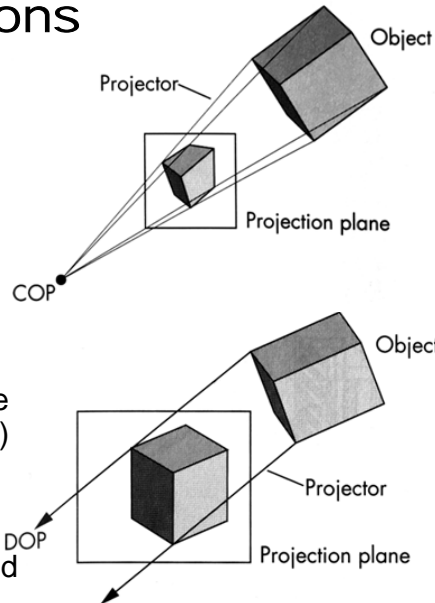
- Preserved with Euclidean transformations but *not* with projections
  - Distance
  - Angle
  - Parallelism
  - Ratios
- Preserved with projections
  - Lines map to lines
  - Intersections
  - Tangency
  - Cross-ratios



$$\{a, b; c, d\} = \frac{a-c}{a-d} : \frac{b-c}{b-d}$$

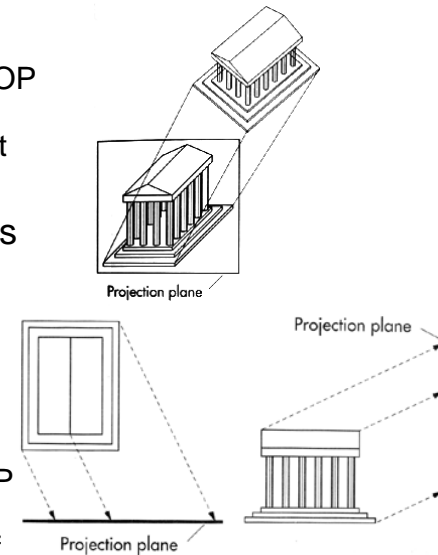
# Perspective vs. parallel projections

- Perspective projections
  - Distance from COP to PP finite
  - + Size varies inversely with distance - looks realistic
  - Distance and angles are not (in general) preserved
  - Parallel lines do not (in general) remain parallel
- Parallel projection
  - Distance from COP to PP infinite (use direction of projection DOP)
  - Less realistic looking
  - + Good for exact measurements
  - + Parallel lines remain parallel
  - Angles not (in general) preserved



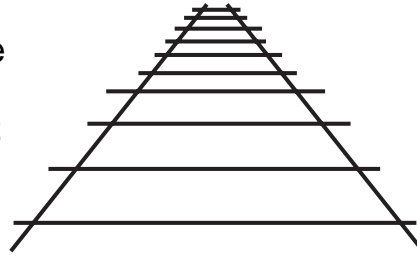
# Parallel projections

- Two types:
  - **Orthographic projection** — DOP perpendicular to PP
  - **Oblique projection** — DOP not perpendicular to PP
- Two especially useful obliques
- **Cavalier projection**
  - DOP makes  $45^\circ$  angle with PP
  - Does not foreshorten lines perpendicular to PP
- **Cabinet projection**
  - DOP makes  $63.4^\circ$  angle with PP
  - Foreshortens lines perpendicular to PP by one-half



# Vanishing points

- Under perspective projections, any set of parallel lines that are not parallel to the PP will converge to a **vanishing point**



- Vanishing points of lines parallel to a principal axis  $x$ ,  $y$ , or  $z$  are called **principal vanishing points**

- Q:
  - How many of these can there be? **3 (as many as there are principal axes)**
  - How many regular vanishing points? **infinite number, as many as pairs of parallel lines in 3D**

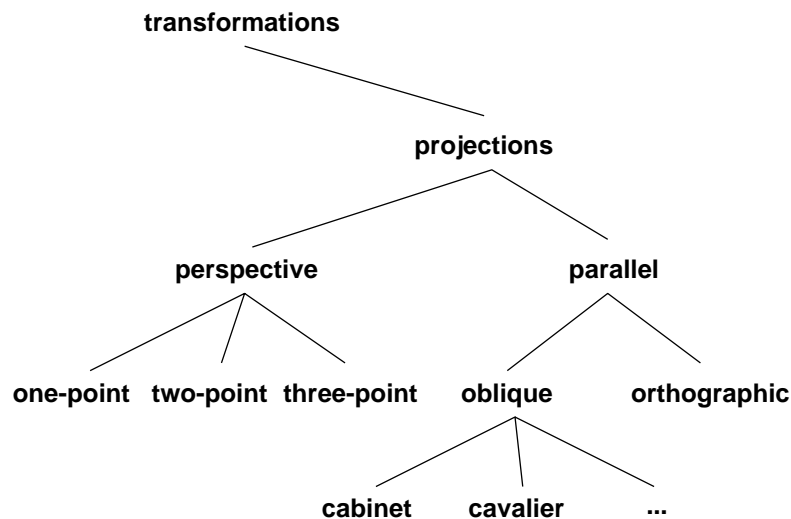
# Types of perspective drawing

- Perspective drawings are often classified by the number of principal vanishing points
  - One-point perspective — simplest to draw
  - Two-point perspective — gives better impression of depth
  - Three-point perspective — most difficult to draw



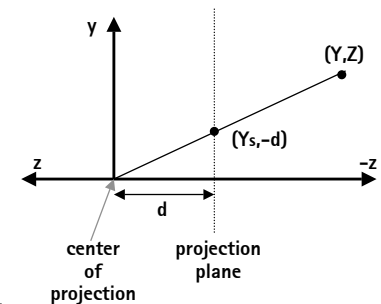
- All three types are equally simple with computer graphics

# Projection taxonomy



# Simple perspective

- Let's analyze central projection
  - center of projection (COP) at origin
  - projection plane perpendicular to  $z$ -axis, distance  $d$  away (on negative side)
- Project point  $(Y,Z)$ 
  - draw a straight line through COP
  - intersect the line with projection plane
  - projects to  $(Y_s, -d)$
- Calculate  $Y_s$  using similar triangles
  - $\frac{Y}{Z} = \frac{Y_s}{-d} \Leftrightarrow Y_s = \frac{-d}{Z} Y$
  - projection means multiplication with  $-d/Z$
  - $-d$  is constant,  $Z$  depends on the point
  - nonlinear because of the division with  $Z$

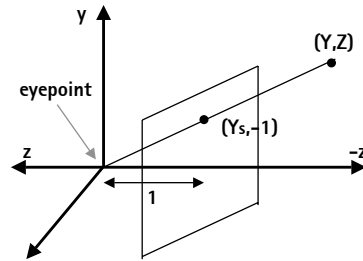


# A typical eye space

- **Eye**
  - Acts as the COP
  - Placed at the origin
  - Looks down the negative z-axis
- **Perspective projection**
  - $[x\ y\ z]' \rightarrow [-1/z\ x,\ -1/z\ y,\ -1]'$
  - Switch to homogen. coords
  - $[-1/z\ x,\ -1/z\ y,\ -1,\ 1]'$
  - multiply with  $-z$  (same homog. point!)
  - $[x,\ y,\ z,\ -z]'$

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- **Screen**
  - Lies in the PP
  - Perpendicular to z-axis
  - At distance 1 from the eye

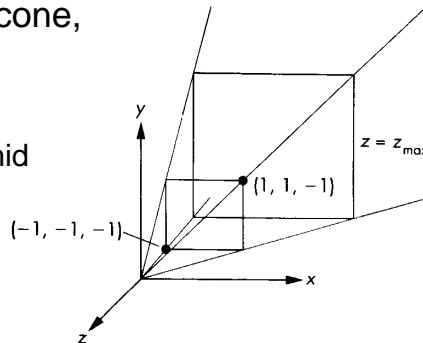


# What about Z?

- Now all points project to  $Z=-1$ 
  - a mapping from “eye space” to image plane
- What do we really want?
  - project X and Y coordinates as stated
  - keep Z-information around for figuring out relative depths
- **Solution**
  - choose which part of the world should be visible
  - map the z-values of the visible part between  $-1$  (near) and  $1$  (far)
  - introduce a mapping from “eye space” to “projection space” or “normalized device coordinates”
- **Invertible projection matrix**
  - as a side product, the projection matrix becomes invertible
  - allows to project the points back to original “eye space”

# Viewing frustum

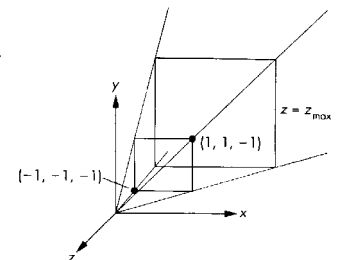
- **View cone**
  - starts from the view point, goes through the view port
  - for now, let the sides of the cone be planes
  - $Z = x, Z = -x, Z = y, Z = -y$
  - so we get 90 degree vertical and horizontal opening angles
  - view through a window at  $-1$  in z, from  $-1$  to  $1$  in x and y
- **View frustum** is the view cone, cut by two planes
  - near (hither)
  - far (yonder)
  - frustum = a truncated pyramid
- Only things within view frustum are visible



# Perspective normalization

- Map the view frustum into the unit box  $[-1,1]x[-1,1]x[-1,1]$
- With the previous page’s view cone x and y are ok
- Need to map z of *near* ( $-n$ ) to  $-1$  and z of *far* ( $-f$ ) to  $1$ 
  - don’t let x or y influence z
  - calculate what the matrix must be!

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{bmatrix} \quad z' = (\alpha z + \beta) / -z = -\alpha - \frac{\beta}{z} \Rightarrow \begin{cases} -1 = -\alpha + \frac{\beta}{n} \\ 1 = -\alpha + \frac{\beta}{f} \end{cases} \Rightarrow \begin{cases} \alpha = -\frac{f+n}{f-n} \\ \beta = -\frac{2fn}{f-n} \end{cases}$$



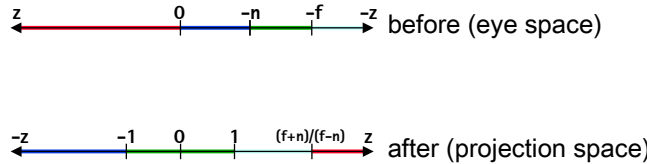
**2 constraints:**  
for the near  
and far planes

# What does it mean?

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

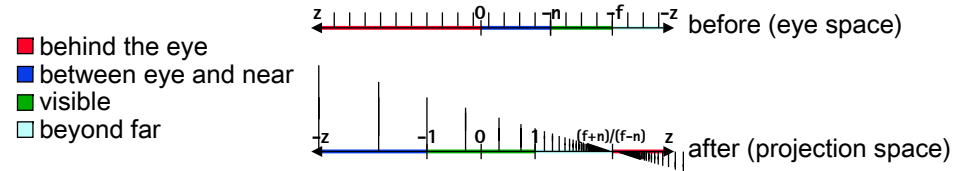
- Let's transform some points
  - $P [x, y, -n, 1]' = [x, y, (fn+nn-2fn)/(f-n), n]' = [x, y, -n, n]' = [xn, yn, -1, 1]'$   
point in near plane moves to -1
  - $P [x, y, -f, 1]' = [x, y, (ff+fn-2fn)/(f-n), f]' = [x, y, f, f]' = [xf, yf, 1, 1]'$   
point in far plane moves to 1
  - $P [0, 0, 0, 1]' = [0, 0, -2fn/(f-n), 0]'$   
eyepoint moves to infinity
  - $P [0, 0, -1, 0]' = [0, 0, (f+n)/(f-n), 1]'$   
point infinitely far becomes local
- Mapping of depths

- behind the eye
- between eye and near
- visible
- beyond far



# More illustrations

- What about a line of telephone poles?
  - equidistant, equal heights in metric space
  - decreasing distances, shortening, and flipping in projective space



- Perspective foreshortening is visible
  - lines get shorter (the y component)
  - but also z gets "denser" with distance
  - what's the effect in z resolution?

# Z resolution

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \begin{matrix} \alpha = -\frac{f+n}{f-n} \\ \beta = -\frac{2fn}{f-n} \end{matrix}$$

- Assume all objects are between 4 and 5 units from camera
- z-buffer resolution to cover z in [-1,1]
  - 24 bits: resolution in z is  $1.2 \cdot 10^{-7}$  (after projection)
  - 16 bits: resolution in z is  $3.1 \cdot 10^{-5}$  (after projection)
- Set  $near = 0.001$ ,  $far = 1000$ 
  - $\alpha = -1$ ,  $\beta = -0.002$
  - 4 maps to 0.9995, 5 maps to 0.9996  $(-1 \cdot 4 - 0.002 \cdot 1) / -4 \sim 0.9995$
  - need over 14 bits  $\lceil \log_2(2 / (0.9996 - 0.9995)) \rceil = 14.3$  to locate the visible part in z, i.e., wasting over 14 bits, leaving < 10 useful bits with 24 bit, < 2 bits with 16 bit buffer!
- Set  $near = 0.001$ ,  $far = 10$ 
  - $\alpha = -1$ ,  $\beta = -0.02$
  - 4 maps to 0.9997, 5 maps to 0.9998
  - same waste of z resolution!

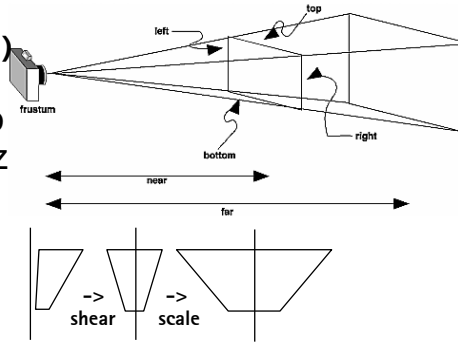
# Z resolution, cont.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \begin{matrix} \alpha = -\frac{f+n}{f-n} \\ \beta = -\frac{2fn}{f-n} \end{matrix}$$

- Set  $near = 1$ ,  $far = 10$ 
  - $\alpha = -1.2222$ ,  $\beta = -2.2222$
  - 4 maps to 0.66667, 5 maps to 0.77778
  - now waste only a bit more than 4 bits
- Set  $near = 1$ ,  $far = 1000$ 
  - $\alpha = -1.0020$ ,  $\beta = -2.0020$
  - 4 maps to 0.50150, 5 maps to 0.60160
  - almost the same situation!
- Lesson:
  - beginners set  $near$  close to camera and  $far$  really far  $\Rightarrow$  too little effective z resolution  $\Rightarrow$  objects' relative depth ordering becomes random
  - set  $near$  as far from camera as you can!
  - $far$  doesn't matter much at all

# General view frustum

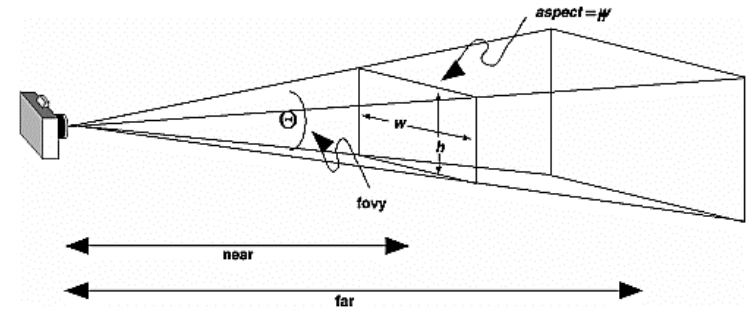
- `glFrustum(l, r, b, t, n, f)`
- Shear asymmetric frustum to become symmetric around -z
  - map  $[(r+l)/2, (t+b)/2, -n, 1]^T$  to  $[0, 0, -n, 1]^T$
- Scale sides to  $x = \pm z, y = \pm z$ 
  - map  $[(r-l)/2, (t-b)/2, -n, 1]^T$  to  $[n, n, -n, 1]^T$



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

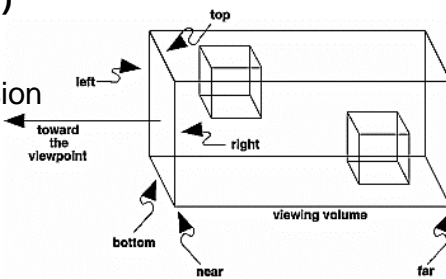
# The easy way in OpenGL

- `gluPerspective(fovy, aspect, n, f)`



# General orthographic matrix

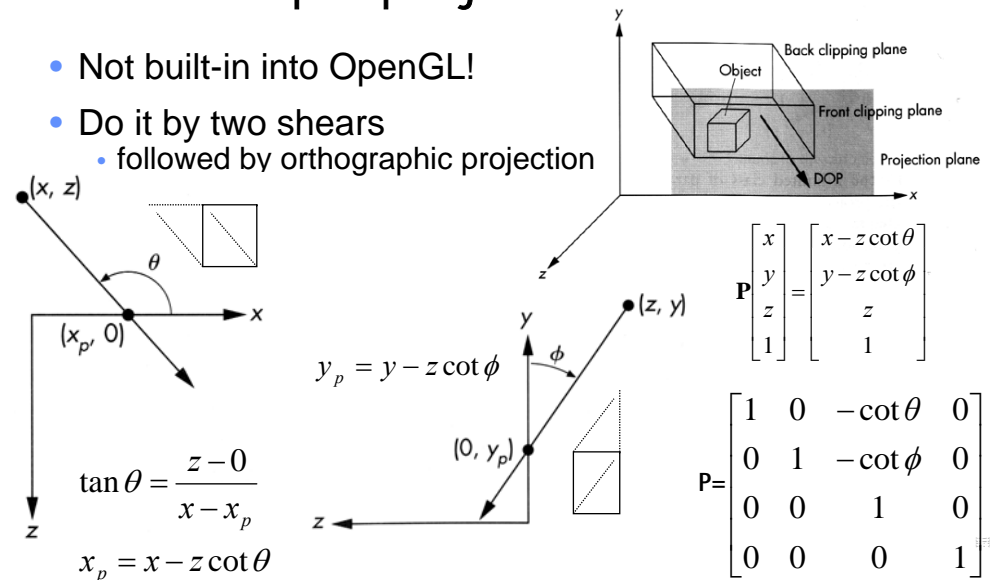
- `glOrtho(l, r, b, t, n, f)`
- View frustum already a box
  - no need for the projective division
- Translate center to origin
  - map  $[(r+l)/2, (t+b)/2, -(f+n)/2]^T$  to  $[0,0,0]^T$
- Scale to a  $[-1,1]$  box
  - map  $[(r-l)/2, (t-b)/2, -(f-n)/2]^T$  to  $[1,1,1]^T$



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

# Oblique projection matrix

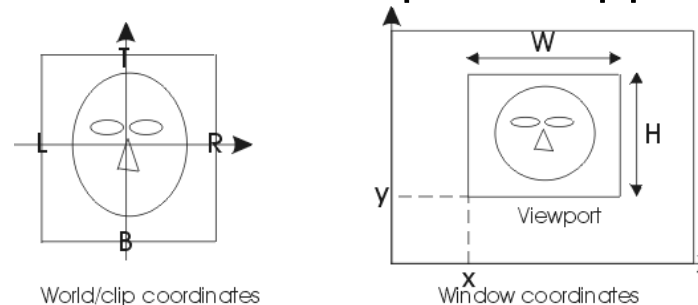
- Not built-in into OpenGL!
- Do it by two shears
  - followed by orthographic projection



# Perspective transformations in OpenGL

- `glMatrixMode(GL_PROJECTION)`
- `glLoadIdentity()`
- `glOrtho(left, right, bottom, top, near, far)`
- `glFrustum(left, right, bottom, top, near, far)`
- **get / set:**
  - `glLoadMatrix(mat)`
  - `mat = glGetDoublev(GL_PROJECTION_MATRIX)`
  - `mat` is a `float[16]`, in column-major order (like Fortran)!
- `glMultMatrixf(mat)`
- `glPushMatrix(), glPopMatrix()`
- `gluPerspective(fovy, aspect, near, far)`
- `gluOrtho2D(left, right, bottom, top)`

# Window-to-viewport mapping



- Want to do a linear mapping

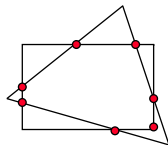
$$\begin{aligned} \text{Match origins} \quad & \frac{x - L}{R - L} = \frac{sx - x}{W} \\ \text{Match widths} \quad & \frac{y - B}{T - B} = \frac{sy - y}{H} \end{aligned}$$

Check Chapter 3 of Red Book, section Viewport Transformation

# Clipping

- Dropping out things that are not visible
  - points are stay or go
  - lines may get shorter
  - parts of polygons can be clipped off
  - if new vertices introduced (lines, polygons) properties (color, etc.) will have to be interpolated
- Order of processing in OpenGL
  - modeling transformations ( $\Rightarrow$  camera coordinates)
  - user-defined clip planes
  - lighting and shading
  - apply projection matrix ( $\Rightarrow$  clip coordinates)
  - clipping
  - projective divide (division by w) ( $\Rightarrow$  normalized device coordinates)
  - rasterization
  - scissor test (if enabled, modifies only pixels within scissor box)

How many new vertices possible?



Each clip can grow number of vertices by 1

# Cohen-Sutherland clipping

- How to make clipping cheap?
  - avoid it!
- Detect whether need any clipping
- Calculate **outcodes** for line end points
  - in 2D: 4 bits
  - how many in 3D?  $3 \times 2 = 6$
- Easy to do in hardware

1010	1000	1001
0010	0000	0001
0110	0100	0101

Meaning of bits?

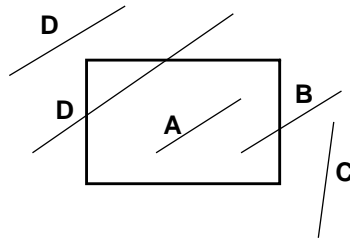
1XXX  
X1XX  
XX1X  
XXX1

above viewport  
below viewport  
left of viewport  
right of viewport

# Cohen-Sutherland cases

- Compare (bitwise) outcodes o1 and o2 of the end points of an edge
- Four cases
  - A:** o1 = o2 = 0
    - accept: the line is completely inside
  - B:** o1 != 0, o2 = 0 (or vice versa)
    - clip: one end inside, the other outside
  - C:** o1 & o2 != 0
    - reject: line segment fully outside one edge
  - D:** o1 & o2 = 0
    - can't tell: both ends outside, but not of the same edge!

1010	1000	1001
0010	0000	0001
0110	0100	0101



# Parametric line representation

- Represent a line as an affine combination of two points

$$p(t) = (1-t) p_1 + t p_2$$

- Coordinate version

$$x(t) = (1-t) x_1 + t x_2$$

$$y(t) = (1-t) y_1 + t y_2$$

- What if
  - t = 0 ? *at p1*
  - t = 1 ? *at p2*
  - t between 0, 1 ? *between p1 and p2*
  - otherwise ? *on the line of p1 and p2, but elsewhere*

# Intersect line with window

- Example: clip with line y = 0
- Solve t from the equation for y:
 
$$y(t) = (1-t) y_1 + t y_2$$

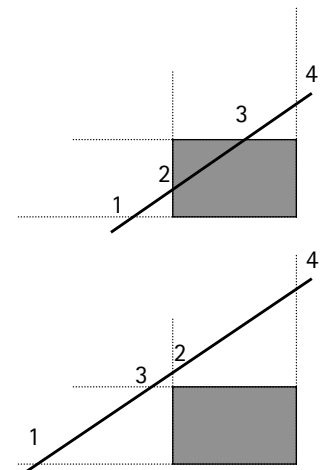
$$\Rightarrow 0 = (1-t) y_1 + t y_2$$

$$\Rightarrow 0 = y_1 + t (y_2 - y_1)$$

$$\Rightarrow t = y_1 / (y_1 - y_2)$$
- get x(t)
 
$$x(t) = (1-t) x_1 + t x_2$$

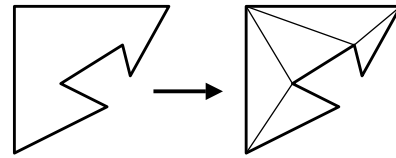
# Liang-Barsky clipping

- Solve t1, t2, t3, t4 in order
- If t < 0 or t > 1, ignore
- In the first example the order is 0 < t1 < t2 < t3 < t4 < 1
  - in this case we only need to solve the intersection points for y2 and x3
- In the second example the order is 0 < t1 < t3 < t2 < t4 < 1
  - see from the order that the line does not hit the window and can be rejected

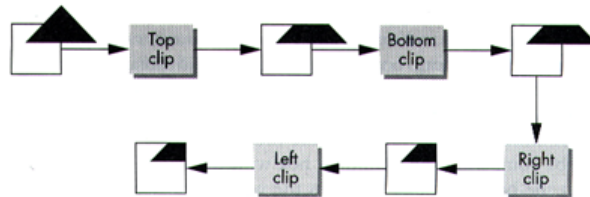


# Clipping polygons

- Ignore concave polygons
  - split them into convex parts



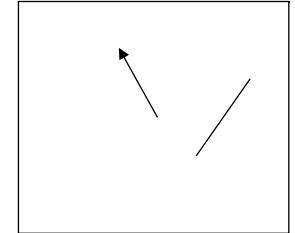
- Clip against one window edge at a time



# Plane definition

- Plane
  - is the set of points whose difference (a vector) is perpendicular to the normal vector

- Assume we have
  - normal vector  $\mathbf{n} = (a,b,c)$
  - point  $\mathbf{p}_0$  on the surface
  - an arbitrary point  $\mathbf{p} = (x,y,z)$



- Let's write this out
  - vector on the surface:  $\mathbf{p} - \mathbf{p}_0$
  - that's perpendicular to  $\mathbf{n}$

$$0 = \mathbf{n} \cdot (\mathbf{p} - \mathbf{p}_0) = \mathbf{n} \cdot \mathbf{p} - \mathbf{n} \cdot \mathbf{p}_0 = ax + by + cz + d,$$

where  $d = -\mathbf{n} \cdot \mathbf{p}_0$

# Extra clipping planes

- OpenGL allows to define extra clipping planes
- glClipPlane(plane, equation)
  - plane = `GL_CLIP_PLANEi`
  - equation = `[a,b,c,d]`
  - test for being on the positive side:  $ax + by + cz + d \geq 0$
  - clipping done in camera coordinates
  - as usual, need to enable

- The way it's done: Clipping a line with a plane:

• line:  $\mathbf{p}(t) = (1-t) \mathbf{p}_1 + t \mathbf{p}_2$

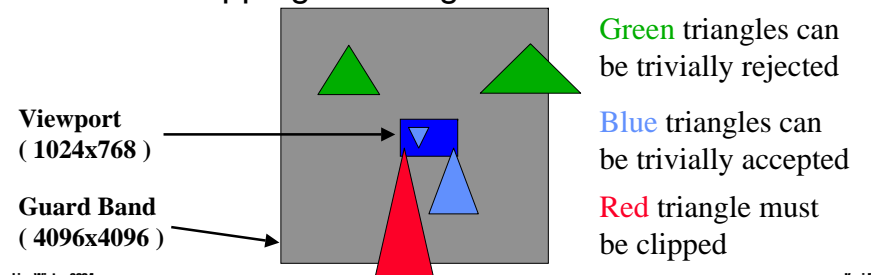
• plane:  $\mathbf{n} \cdot (\mathbf{p}(t) - \mathbf{p}_0) = 0$

• insert line equation and solve for parameter t:

$$\begin{aligned} \mathbf{n} \cdot (t(\mathbf{p}_2 - \mathbf{p}_1) + \mathbf{p}_1 - \mathbf{p}_0) &= 0 \\ \Rightarrow t \mathbf{n} \cdot (\mathbf{p}_2 - \mathbf{p}_1) &= \mathbf{n} \cdot (\mathbf{p}_0 - \mathbf{p}_1) \\ \Rightarrow t &= \frac{\mathbf{n} \cdot (\mathbf{p}_0 - \mathbf{p}_1)}{\mathbf{n} \cdot (\mathbf{p}_2 - \mathbf{p}_1)} \end{aligned}$$

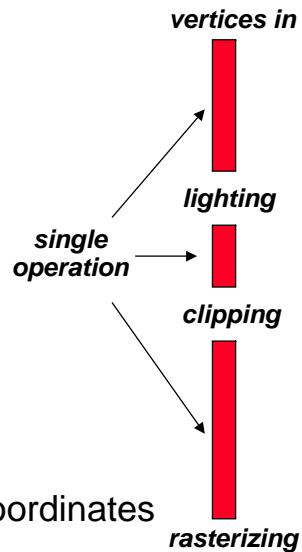
# Guard band clipping

- Clipping is slow
  - new vertices are introduced
  - must interpolate values (colors, alpha, texture coordinates, fog, ...)
  - can destroy coherency of triangle strips, etc.
- Scissoring can be faster
  - just skip the pixels outside the viewport
- Near and far clipping unchanged



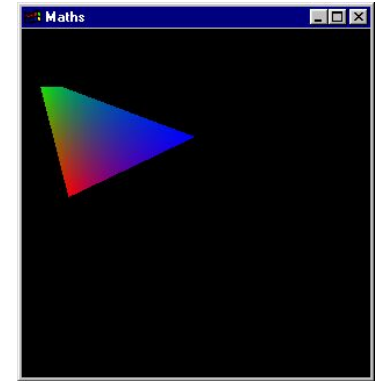
# Transformation pipeline

- Object coordinates  
 ↓ (Modelview matrix)
- World coordinates  
 ↓ (Modelview matrix)
- Camera/Eye coordinates  
 ↓ (Projection matrix)
- Clipping coordinates  
 ↓ (Homogeneous division)
- Normalized device coordinates  
 ↓ (Viewport transformation)
- Window/Screen/Image/Viewport coordinates



# Example: Analyze the maths

```
glViewport(10, 50, 150, 200)
glClearColor(GL_COLOR_BUFFER_BIT)
glMatrixMode(GL_PROJECTION)
glLoadIdentity()
glFrustum(0, 2, -1, .5, 1, 10)
glMatrixMode(GL_MODELVIEW)
glLoadIdentity()
gluLookAt(2, 7, 2, 6, 6, 6, 0, 1, 0)
glTranslatef(4, 5, 7)
glRotate(60, 0, 0, 1)
glBegin(GL_TRIANGLES)
glColor3f(1, 0, 0)
glVertex3f(0, 0, 0)
glColor3f(0, 1, 0)
glVertex3f(5, 0, 0)
glColor3f(0, 0, 1)
glVertex3f(0, 4, 0)
glEnd()
```



Window is 300x300

# Projection matrix

```
glFrustum(0, 2, -1, .5, 1, 10)
```

- Projection from eye to screen
- Map -1 to -1 and -10 to 1

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{bmatrix} z' = (\alpha z + \beta) / -z = -\alpha - \frac{\beta}{z} \Rightarrow \begin{cases} -1 = -\alpha + \beta/1 \\ 1 = -\alpha + \beta/10 \end{cases} \Rightarrow \begin{cases} 2 = -\beta 9/10 \Rightarrow \beta = -20/9 \\ \alpha = 1 + \beta = -11/9 \end{cases} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -11 & -20 \\ 0 & 0 & 9 & 9 \end{bmatrix}$$

- Shear the center of near plane onto z axis ([1, -1/4, -1] -> [0, 0, -1])
- Resize: (2-0, 0.5- -1)/2=(1, 3/4) to (1, 1)

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & -1/4 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 4/3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Projection...

- Multiply out
  - scale & shear
- and project

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 4/3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & -1/4 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 4/3 & -1/3 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -11 & -20 \\ 0 & 0 & 9 & 9 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 4/3 & -1/3 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 4/3 & -1/3 & 0 \\ 0 & 0 & -11 & -20 \\ 0 & 0 & 9 & 9 \end{bmatrix}$$

# Camera transformation

gluLookAt(2, 7, 2, 6, 6, 6, 0, 1, 0)

• Camera position is (2,7,2)

• Z is (2,7,2) - (6,6,6) = (-4,1,-4)

• X is VUP x Z = (0,1,0) x (-4,1,-4) =

$$\begin{bmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ 0 & 1 & 0 \\ -4 & 1 & -4 \end{bmatrix} = (-4, 0, 4)$$

• Y is Z x X = (-4,1,-4) x (-4,0,4) =

$$\begin{bmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ -4 & 1 & -4 \\ -4 & 0 & 4 \end{bmatrix} = (4, 32, 4)$$

• Check: dot(X,Y) = dot(X,Z) = dot(Y,Z) = 0

• Normalize and get cam->world

• X = (-1,0,1) / sqrt(2)

• Y = (1,8,1) / sqrt(66)

• Z = (-4,1,-4) / sqrt(33)

$$\begin{bmatrix} -1/\sqrt{2} & 1/\sqrt{66} & -4/\sqrt{33} & 2 \\ 0 & 8/\sqrt{66} & 1/\sqrt{33} & 7 \\ 1/\sqrt{2} & 1/\sqrt{66} & -4/\sqrt{33} & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Camera transformation

• Invert to get world->cam:

• transpose the rotation part

• multiply that with neg. translation

$$\mathbf{M}\mathbf{M}^{-1} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R}' & -\mathbf{R}'\mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}$$

$$\begin{bmatrix} -1/\sqrt{2} & 1/\sqrt{66} & -4/\sqrt{33} & 2 \\ 0 & 8/\sqrt{66} & 1/\sqrt{33} & 7 \\ 1/\sqrt{2} & 1/\sqrt{66} & -4/\sqrt{33} & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} -1/\sqrt{2} & 0 & 1/\sqrt{2} & 0 \\ 1/\sqrt{66} & 8/\sqrt{66} & 1/\sqrt{66} & -60/\sqrt{66} \\ -4/\sqrt{33} & 1/\sqrt{33} & -4/\sqrt{33} & 9/\sqrt{33} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Modeling transformation

glTranslatef(4, 5, 7)

glRotate(60, 0, 0, 1)

• Translation

$$\begin{bmatrix} 1 & 0 & 0 & 4 \\ 0 & 1 & 0 & 5 \\ 0 & 0 & 1 & 7 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

• Rotation

• cos 60 = .5, sin 60 = sqrt(1-.5\*.5) = sqrt(3)/2  
• around z axis

$$\begin{bmatrix} 1/2 & -\sqrt{3}/2 & 0 & 0 \\ \sqrt{3}/2 & 1/2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

• Multiply

$$\begin{bmatrix} 1/2 & -\sqrt{3}/2 & 0 & 4 \\ \sqrt{3}/2 & 1/2 & 0 & 5 \\ 0 & 0 & 1 & 7 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Transform vertices

• First vertex: (0,0,0)

• modeling

$$\begin{bmatrix} 1/2 & -\sqrt{3}/2 & 0 & 4 \\ \sqrt{3}/2 & 1/2 & 0 & 5 \\ 0 & 0 & 1 & 7 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \\ 7 \\ 1 \end{bmatrix}$$

• to camera

$$\begin{bmatrix} -1/\sqrt{2} & 0 & 1/\sqrt{2} & 0 \\ 1/\sqrt{66} & 8/\sqrt{66} & 1/\sqrt{66} & -60/\sqrt{66} \\ -4/\sqrt{33} & 1/\sqrt{33} & -4/\sqrt{33} & 9/\sqrt{33} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 4 \\ 5 \\ 7 \\ 1 \end{bmatrix} = \begin{bmatrix} 3/\sqrt{2} \\ -9/\sqrt{66} \\ -30/\sqrt{33} \\ 1 \end{bmatrix}$$

• and projection

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 4/3 & -1/3 & 0 \\ 0 & 0 & \frac{-11}{9} & \frac{-20}{9} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} 3/\sqrt{2} \\ -9/\sqrt{66} \\ -30/\sqrt{33} \\ 1 \end{bmatrix} \approx \begin{bmatrix} -3.101 \\ 0.264 \\ 4.161 \\ 5.222 \end{bmatrix}$$

## To viewport

glViewport(10, 50, 150, 200)

- Homogenous division to NDC:

- $x: 10 + 150 * (-0.594 - (-1))/2 = 40$

- $y: 50 + 200 * (0.050 - (-1))/2 = 155$

- Second vertex in NDC: (-.943, 1.077, .864)

- needs clipping!

screen coordinates would have been (14, 258)

- calculate t from y:

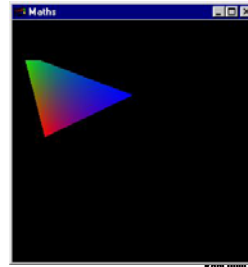
$(1.077 - 0.050) * t = (1.000 - 0.050) \Rightarrow 0.925$

- new screen coordinates

$x: 40 + (14-40)*t = 16$

$y: 155 + (258-155)*t = 250$

- color:  $(1,0,0) + ((0,1,0) - (1,0,0))*t = (0.075, 0.925, 0)$



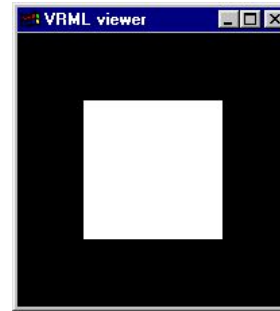
## Virtual trackball

Inspect an object by

- rotating

- panning

- zooming



## Rotating

- Rotate an object around its center

- $p' = R(p - c) + c = T_c R T_{-c} p$

- What's R?

- imagine a unit sphere over the viewport

- when you click a point, project it onto the sphere

- the rotation is the one that rotates the sphere from the first point to the second

- get angle from dot product

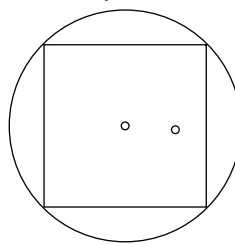
- $\arccos(p_0 \cdot p_1)$

- get axis from cross product

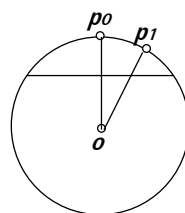
- $p_0 \times p_1$

- use quaternion (or glRotate)

Top view



Side view



## Panning

- Effect:

- the point you click,
- at the distance of rotation center,
- should stay glued to the mouse pointer

- In other words:

- how big a translation of object center corresponds to one pixel?
- if that scale factor is s,
- and you move dx, dy pixels,
- translate by s dx, s dy

- What is s?

- project object center to image
- move by one pixel
- project back to 3D
- calculate distance

```

wln = gluProject( obj_c[0], obj_c[1], obj_c[2] )
obj = gluUnProject( wln[0], wln[1] + 1, wln[2] )
s = dist(obj, obj_c)
    
```

# Zooming

- For zooming
  - modify the distance from camera to object
  - so that the same mouse motion corresponds to the same relative zooming, must change distance by a multiplicative factor
  - e.g., multiply distance by  $2^{(dy / a)}$
  - if  $dy == a$  pixels, distance doubles
  - if  $dy == -a$  pixels, distance halves