3D Viewing

3D Viewing Transformation Pipeline

MC

Modeling Transformation

WC

Viewing Transformation

VC

Projection Transformation

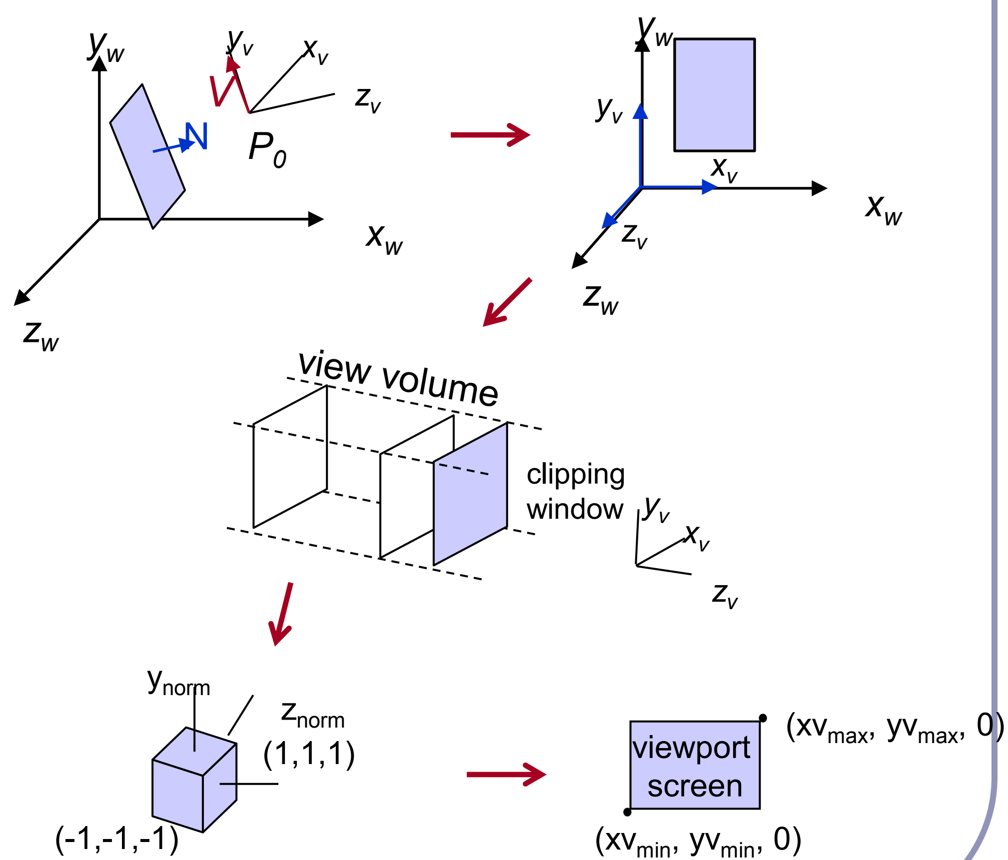
PC

Normalization Transformation and Clipping

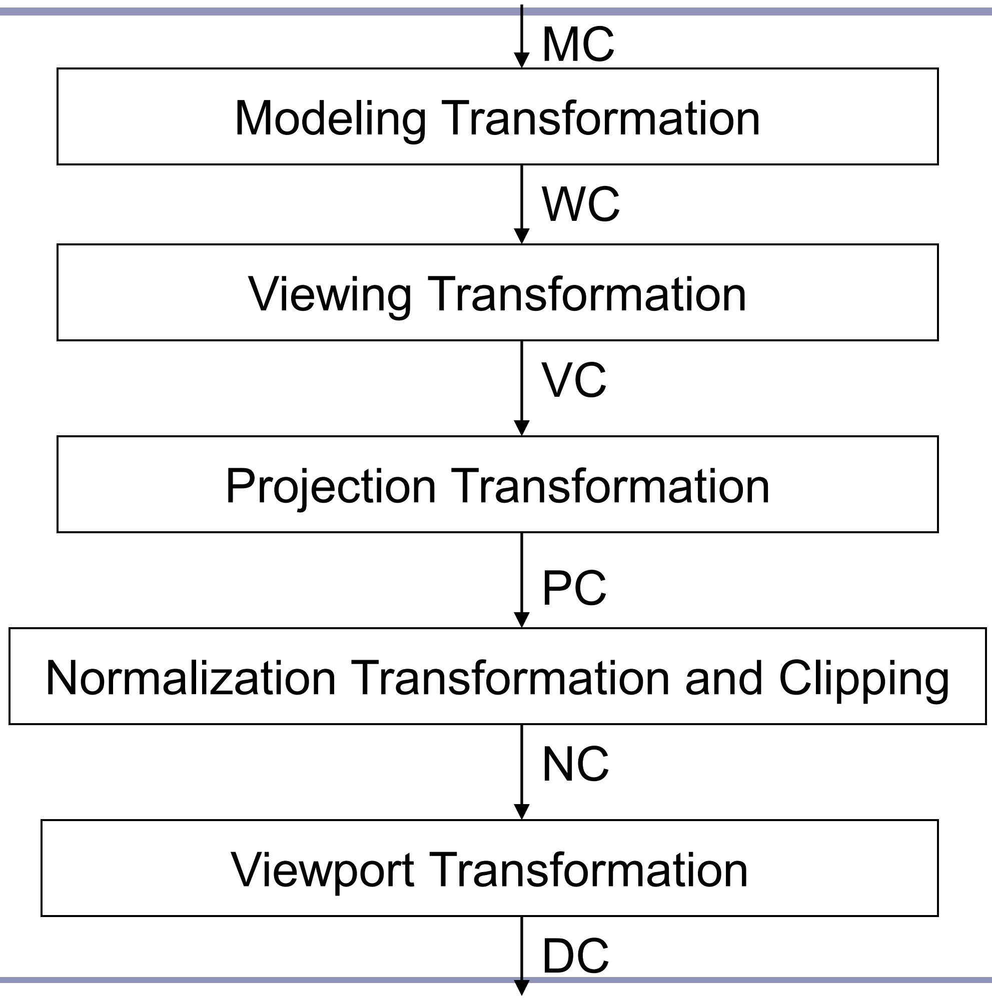
NC

Viewport Transformation

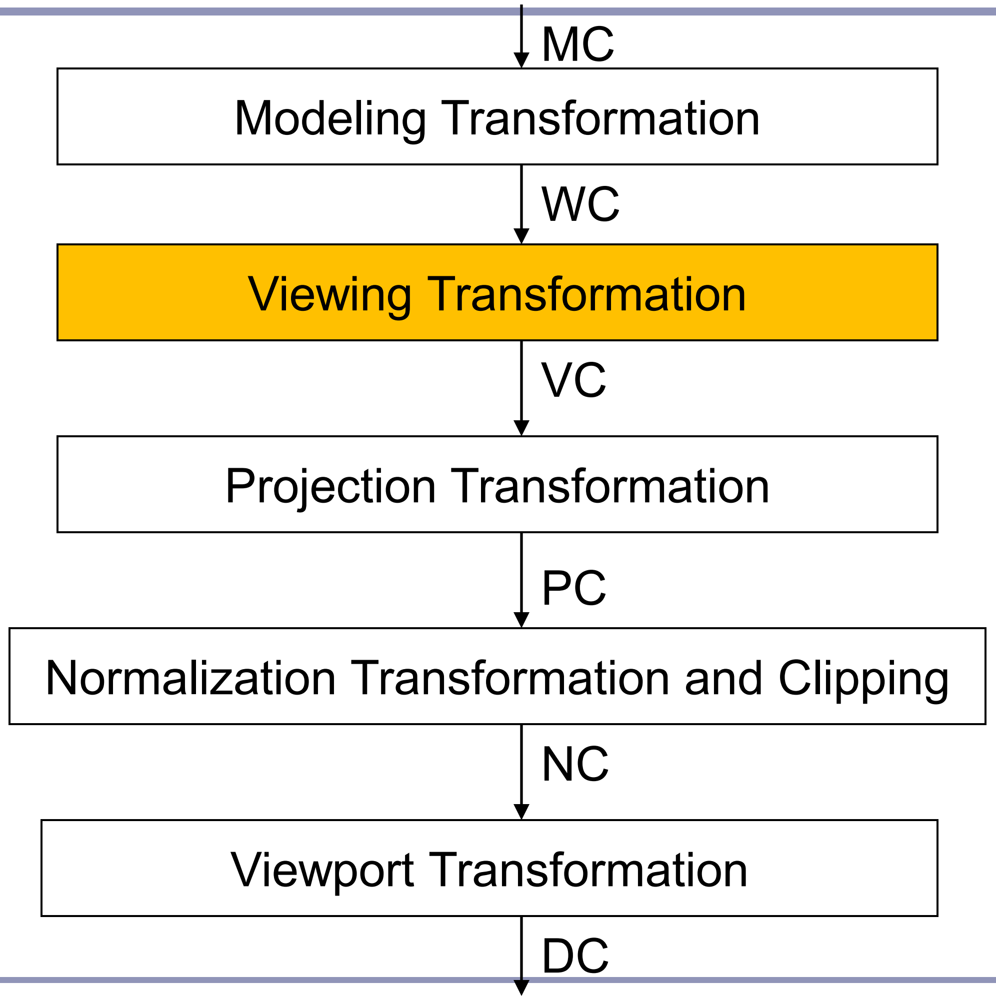
DC



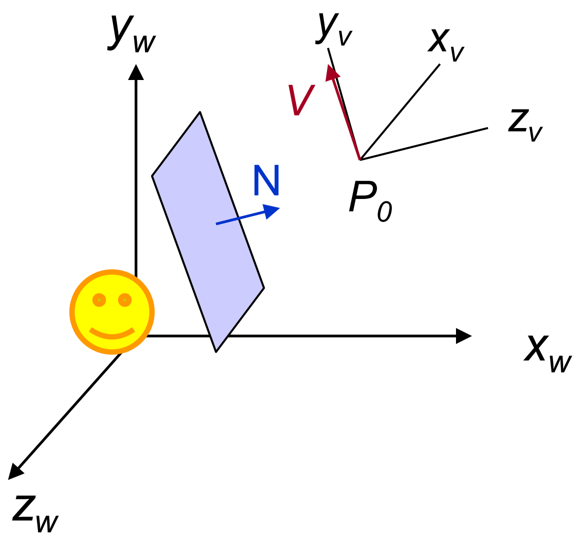
3D Viewing Transformation Pipeline



3D Viewing Transformation Pipeline



3D Viewing



V ***view up vector***

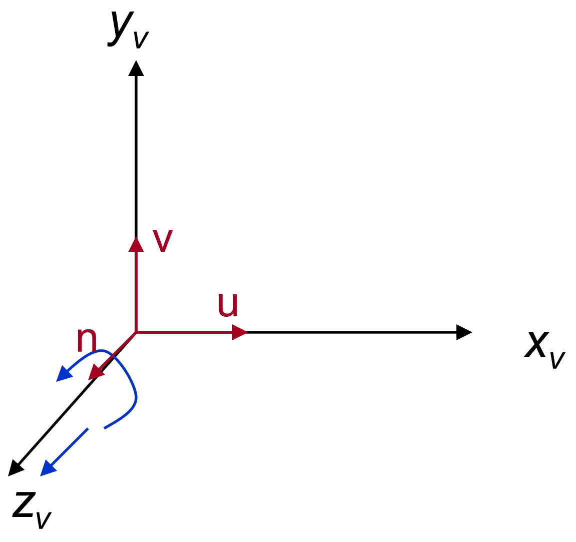
$P_0 = (x_0, y_0, z_0)$ ***view point***

N ***viewplane normal***

Viewplane is at point z_{vp} in negative z_v direction

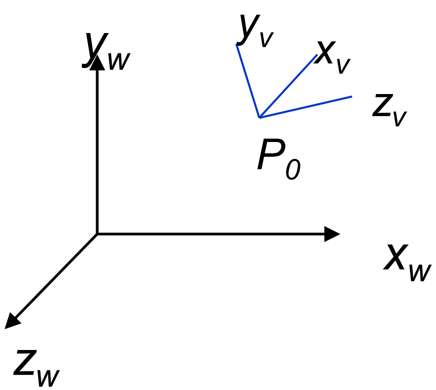
V is perpendicular to N

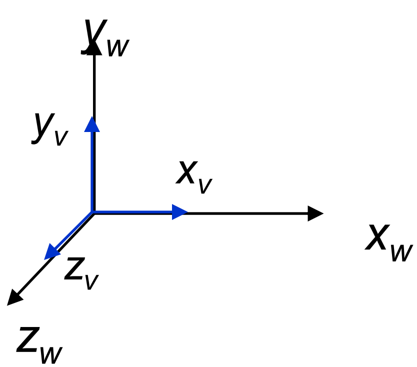
3D Viewing



x_v y_v z_v is a ***right-handed*** **viewing coordinate system**

3D Viewing



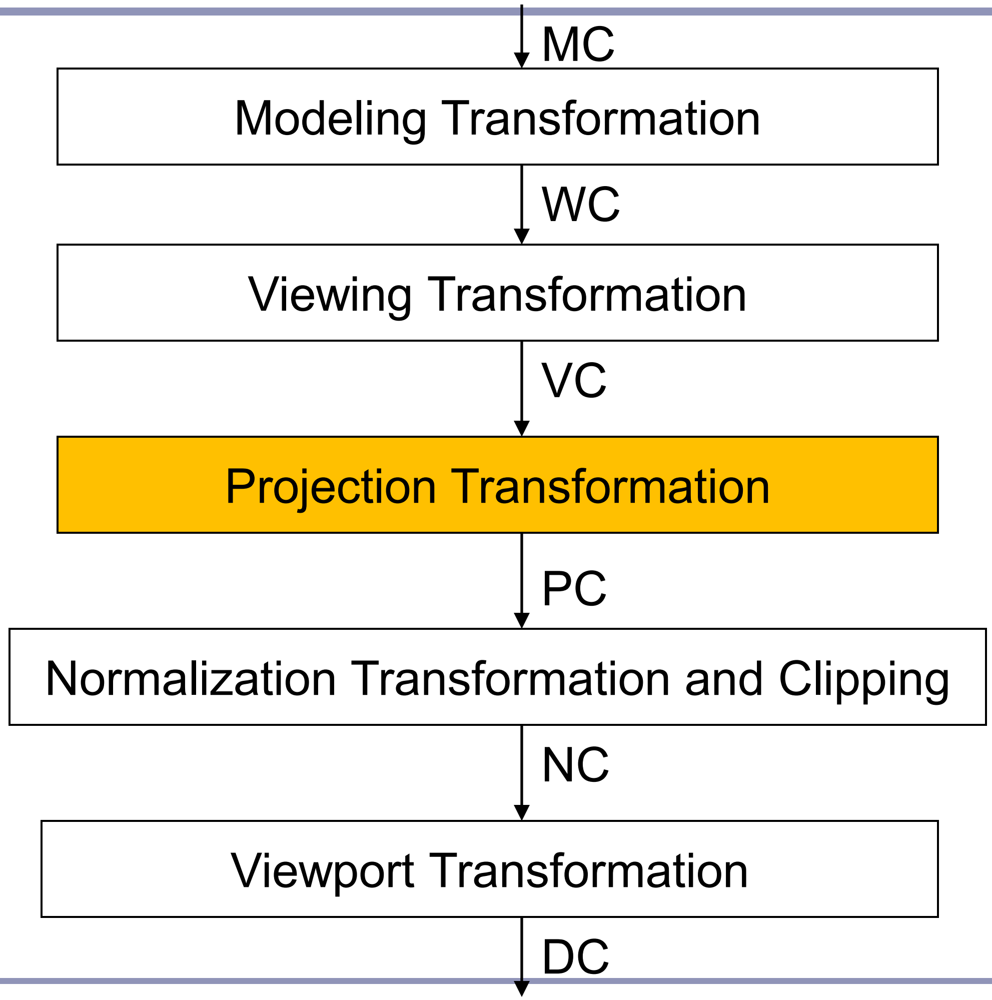


Transformation from World to Viewing Coordinates:

1. ***Translate viewing coordinate origin to the origin of world coordinate system***
2. ***Apply rotations to align x_v y_v z_v axes with x_w y_w z_w axes respectively***

$$M_{WC,VC} = R \cdot T$$

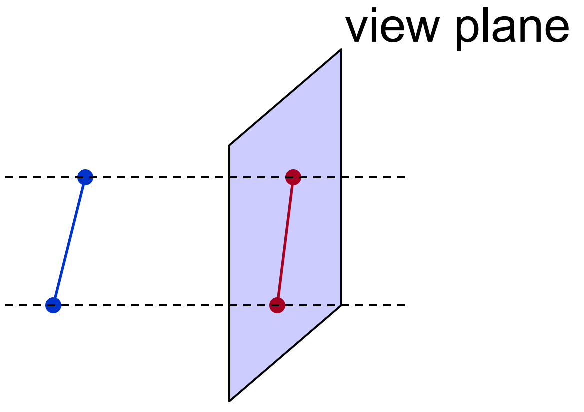
3D Viewing Transformation Pipeline



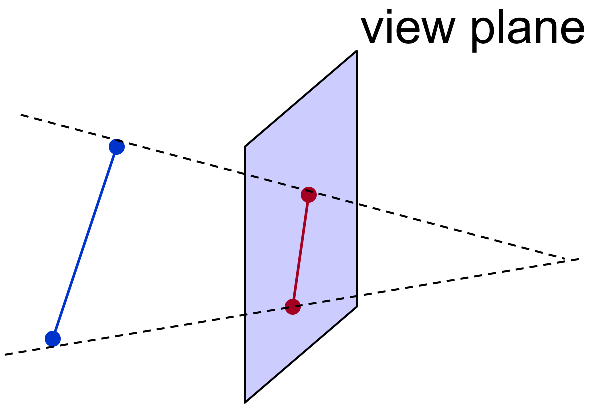
Projections

- Parallel
  - orthogonal
    - axonometric
      - isometric
  - oblique
    - cavalier
    - cabinet
- Perspective

Projections

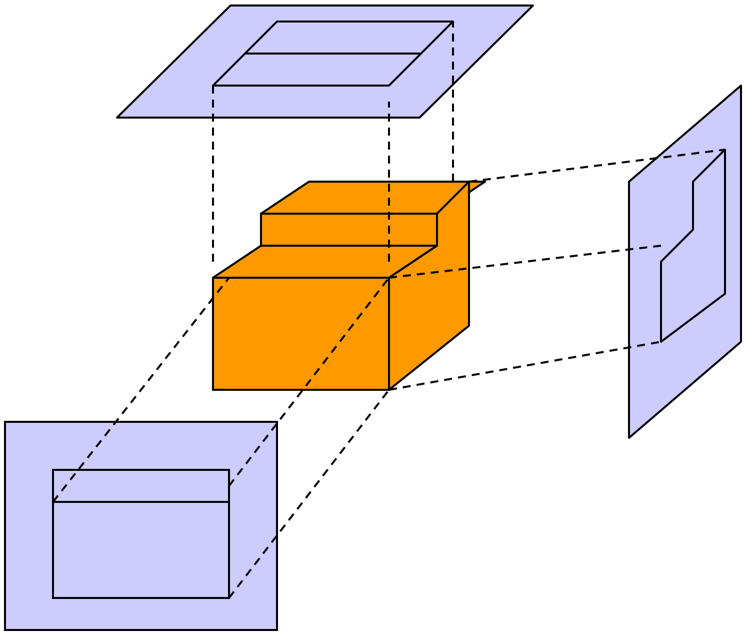


Parallel projection

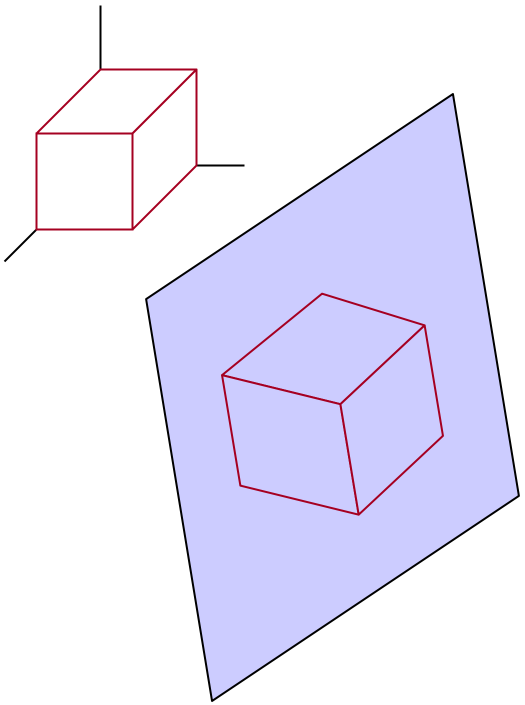


Perspective projection

Orthogonal Projection



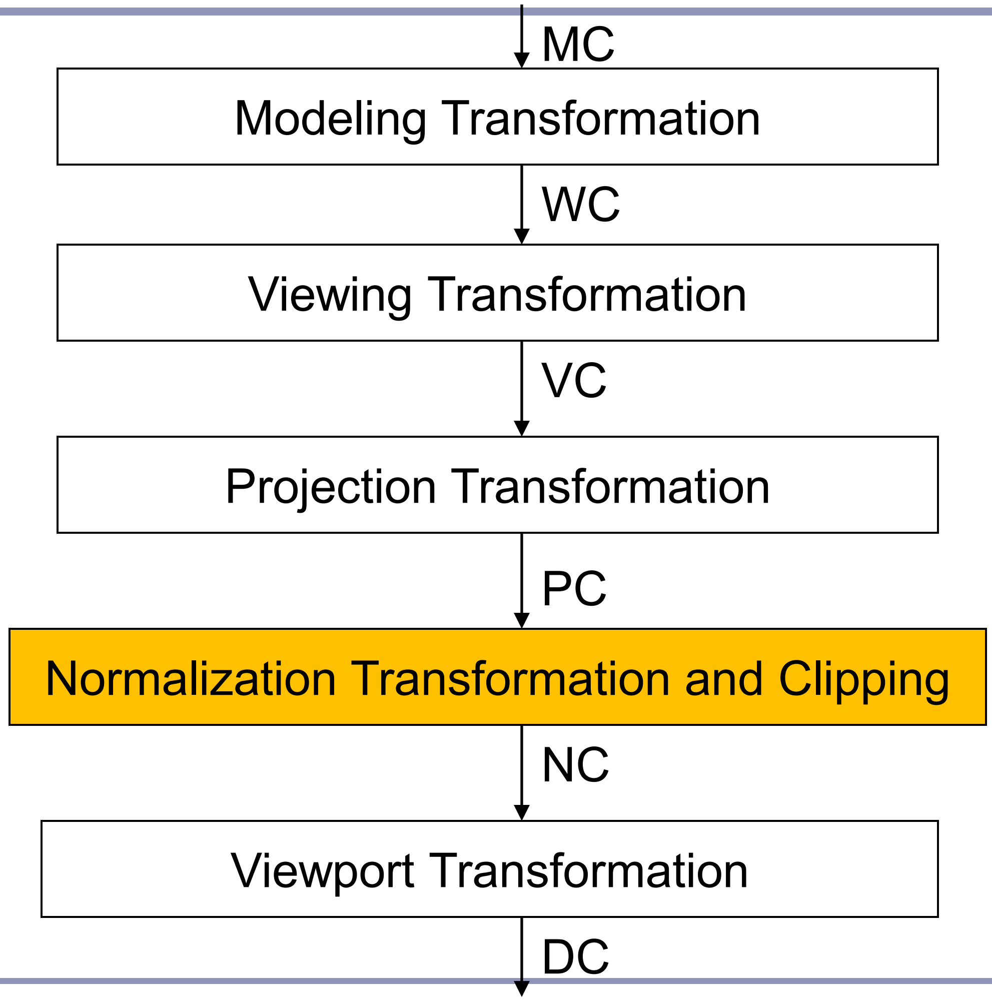
Orthogonal Projection


Isometric

***Axonometric:** displays more than one face of an object*

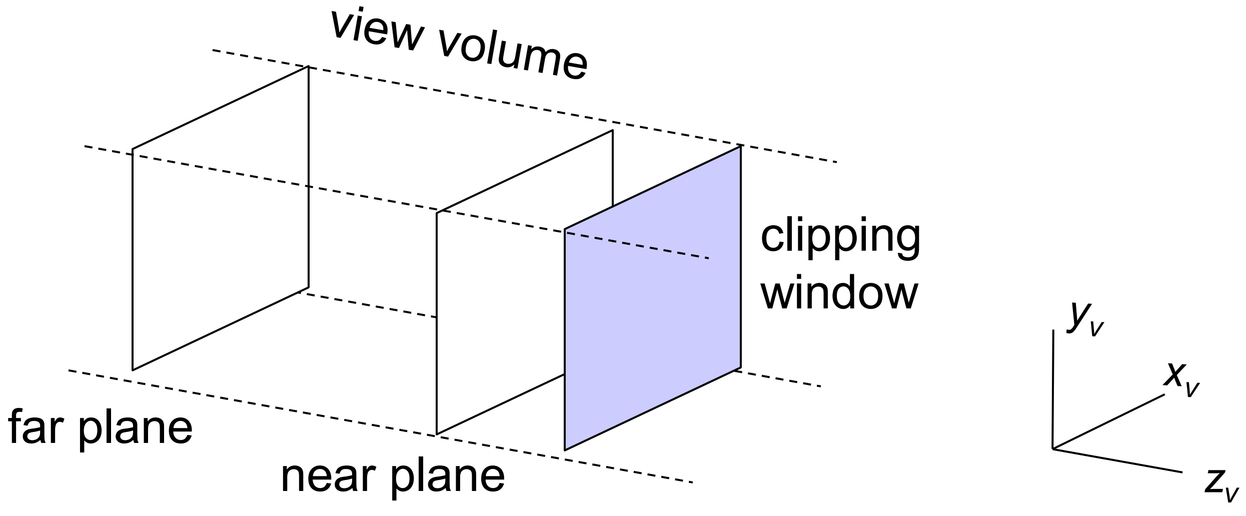
***Isometric:** projection plane intersects each coordinate axis at the sane distance from the origin*

3D Viewing Transformation Pipeline



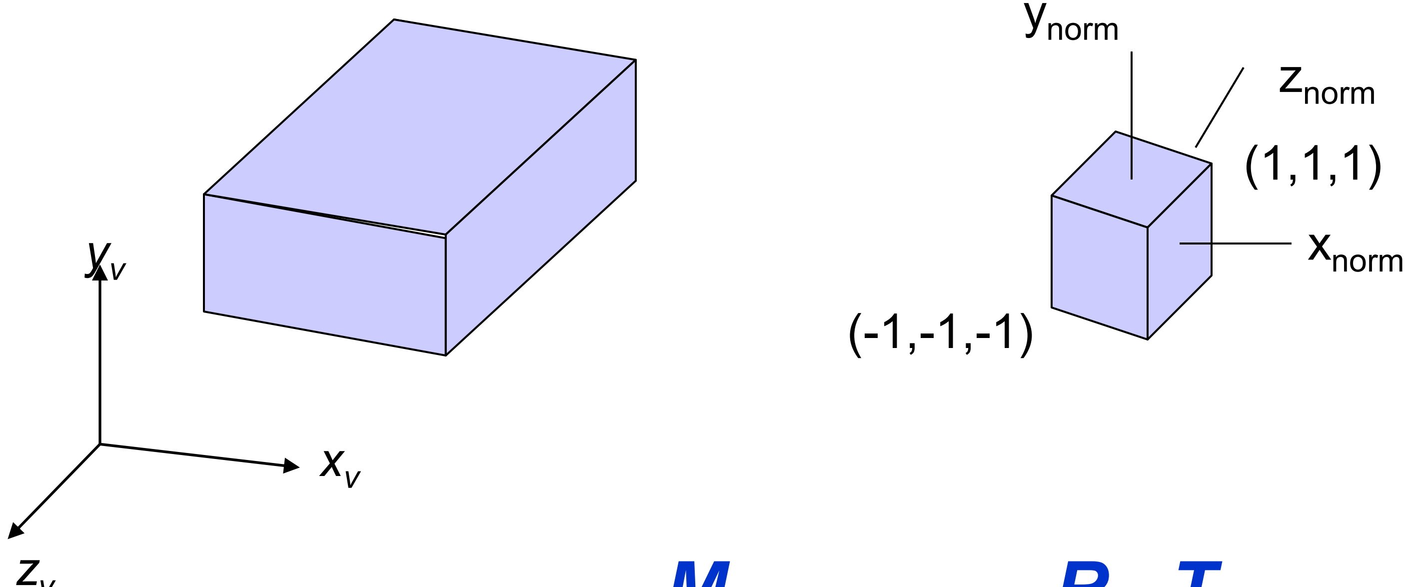
Orthogonal Projection

Clipping



Orthogonal Projection

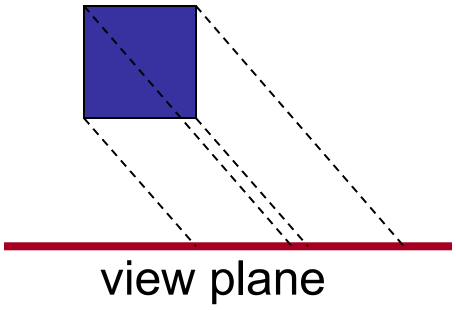
Normalization

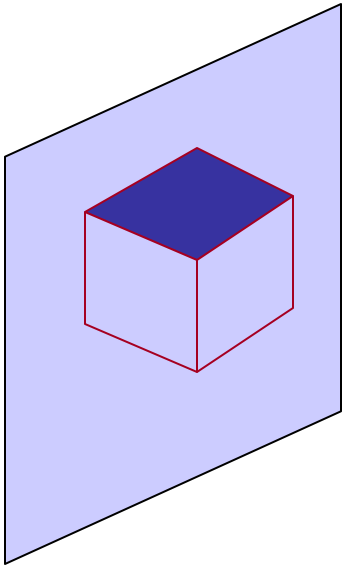


$$M_{ortho,norm} \cdot R \cdot T$$

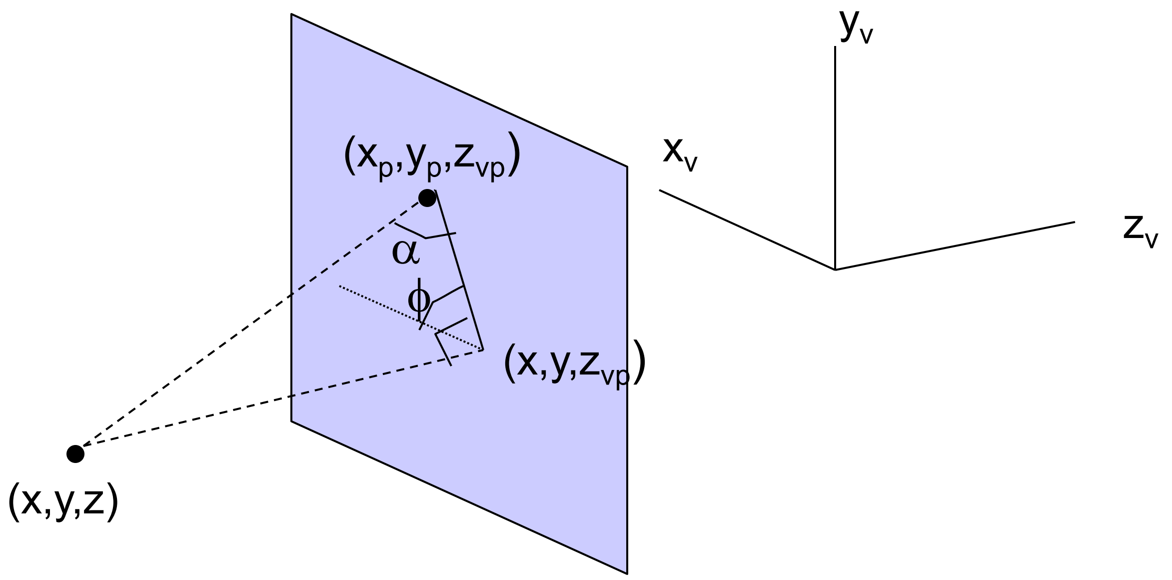
Oblique Projection

Oblique projection: ***projection path is not perpendicular to the view plane***





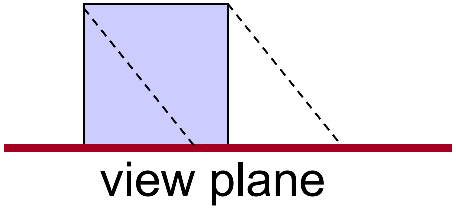
Oblique Projection

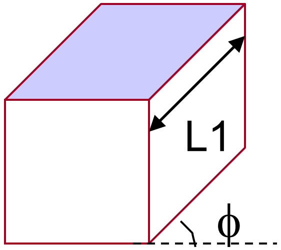


Oblique Projection

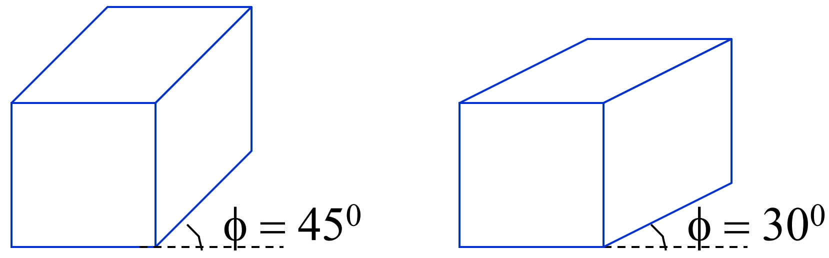
$$x_p = x + L_1 (z_{vp} - z) \cos \phi$$
$$y_p = y + L_1 (z_{vp} - z) \sin \phi$$

shearing transformation

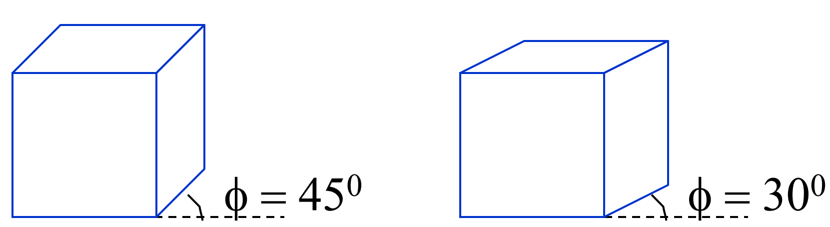




Oblique Projection

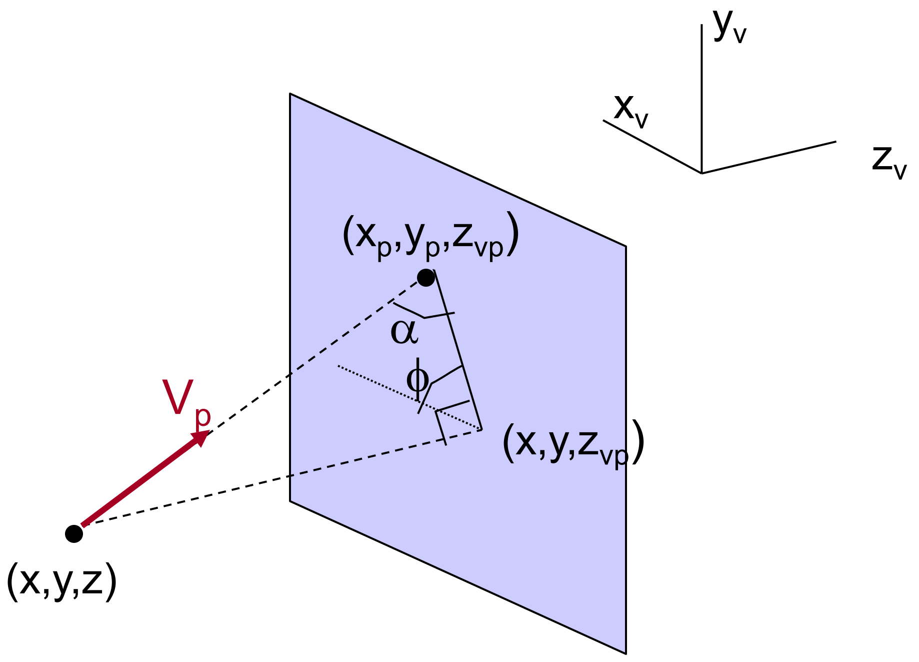


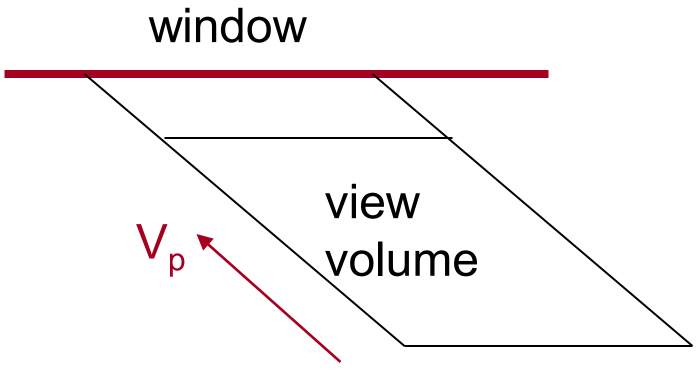
Cavalier Projection



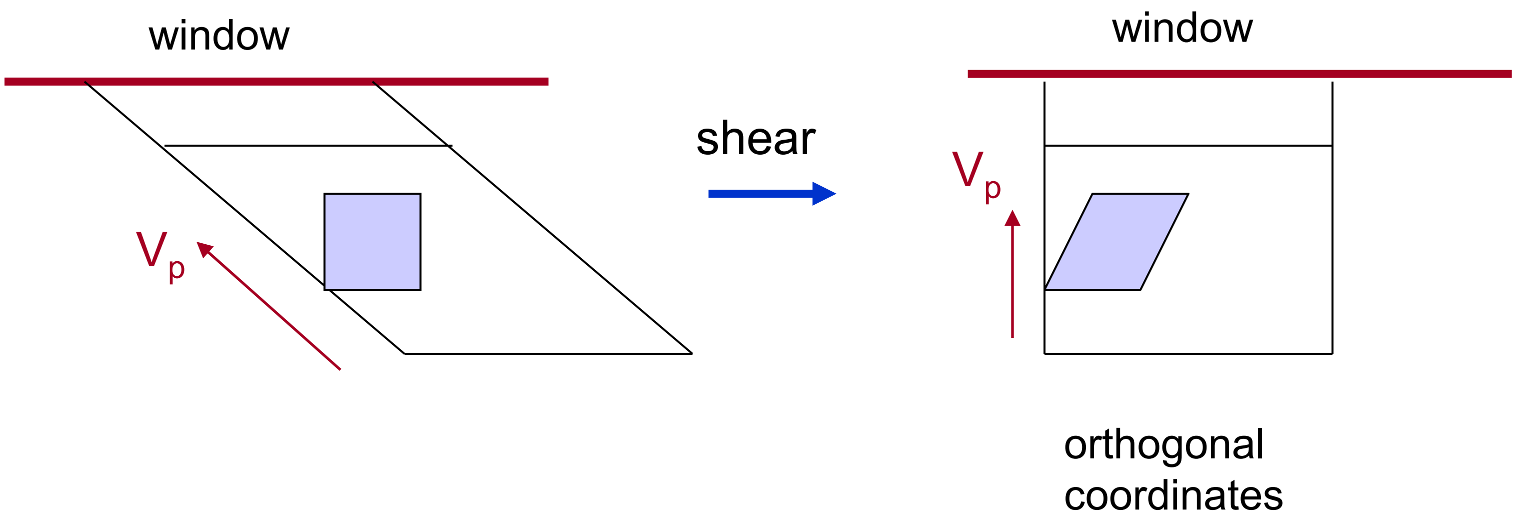
Cabinet Projection

Oblique Projection



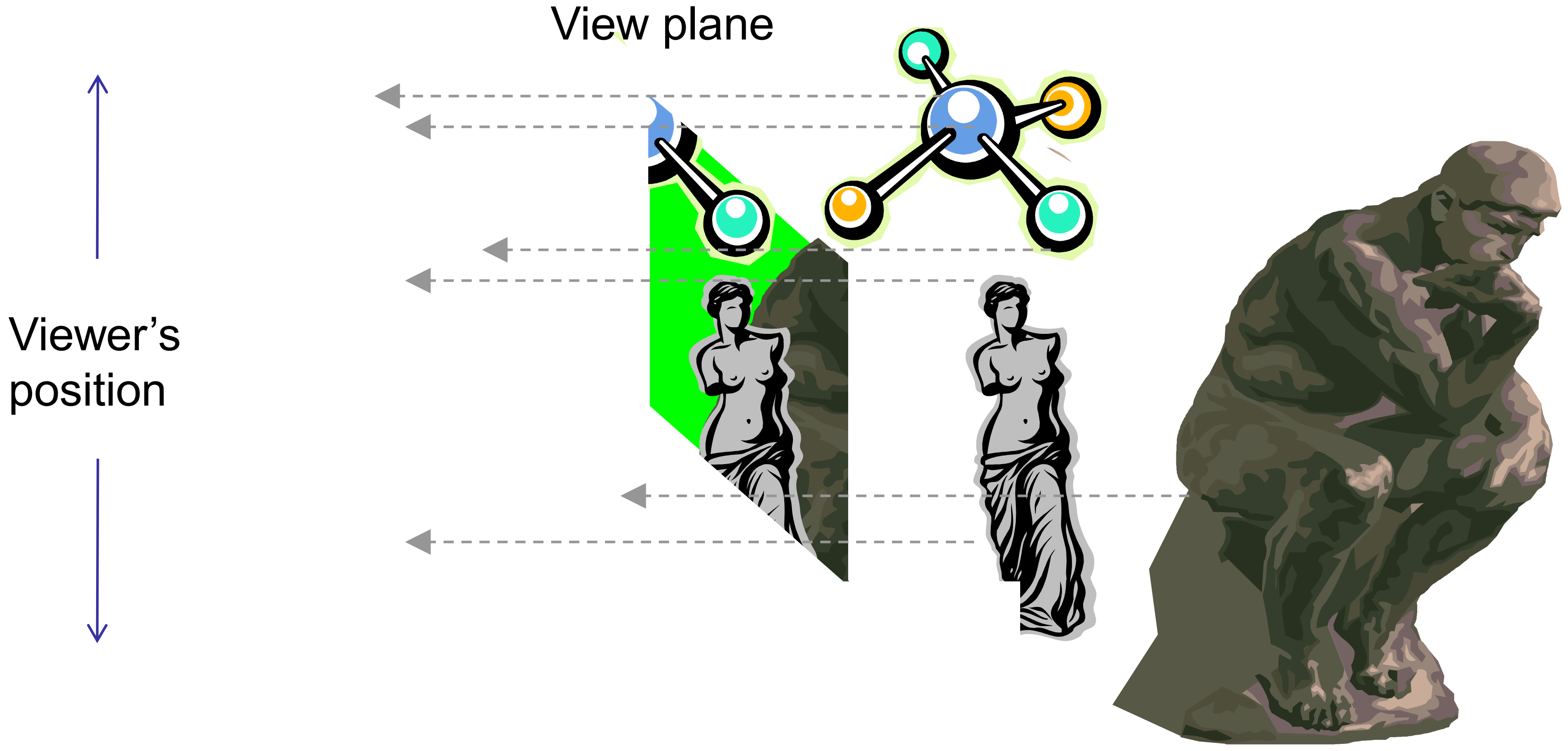


Oblique Projection

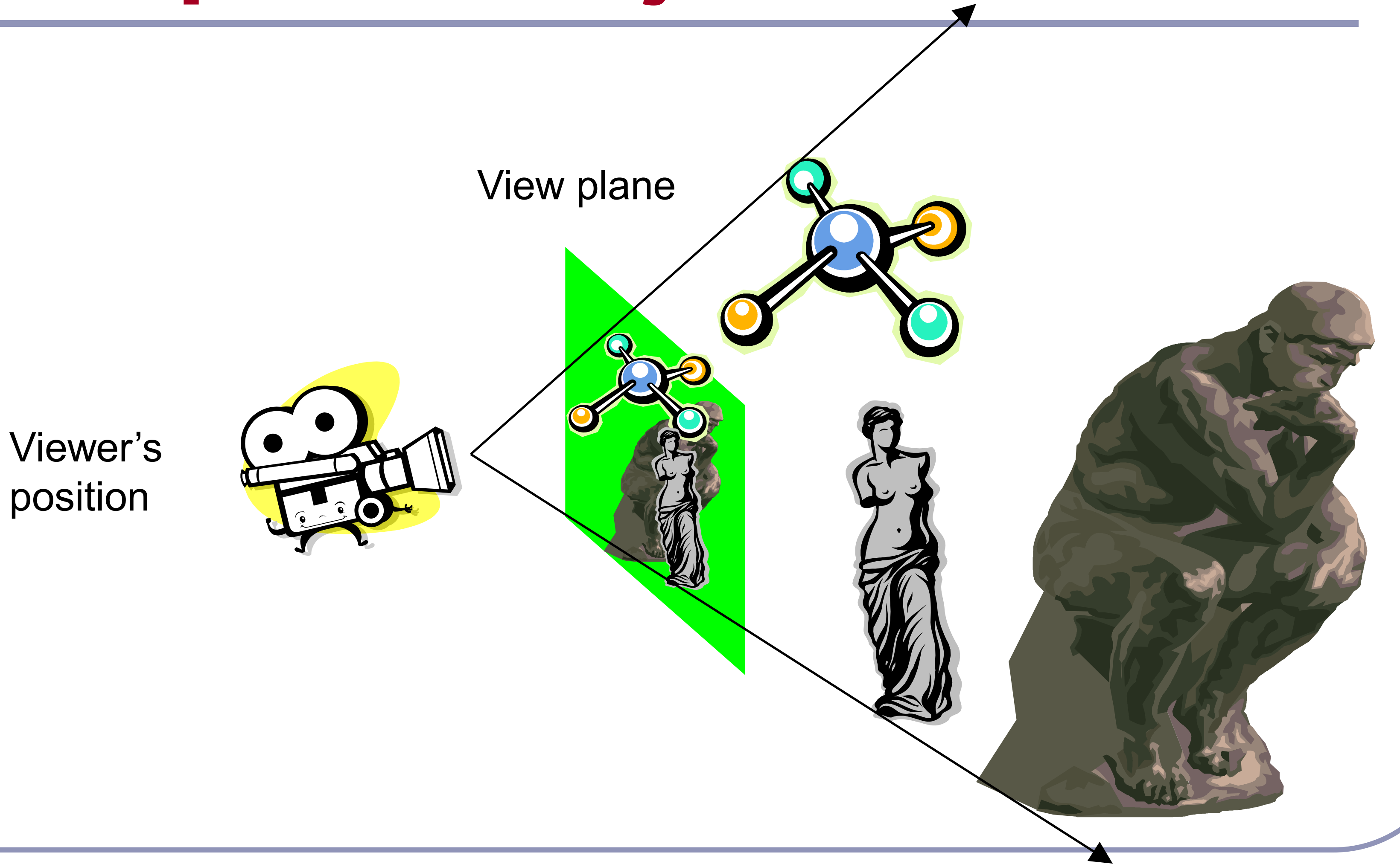


$$\boldsymbol{M_{WC,VC} \cdot \underbrace{M_{orth,norm} \cdot M_{obl}}_{M_{obl,norm}}}$$

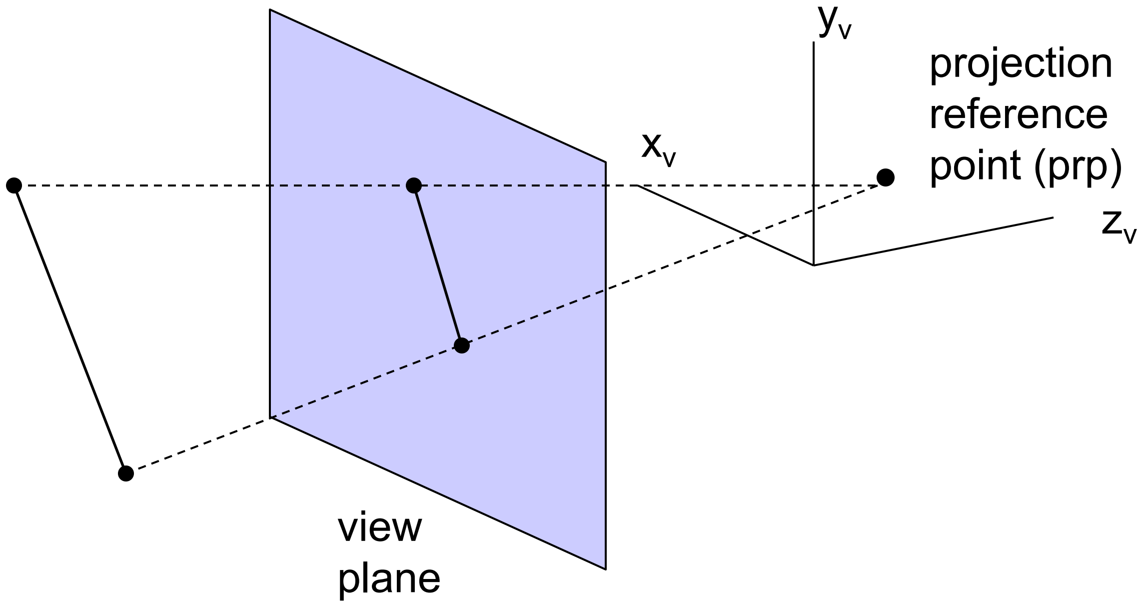
Parallel Projection



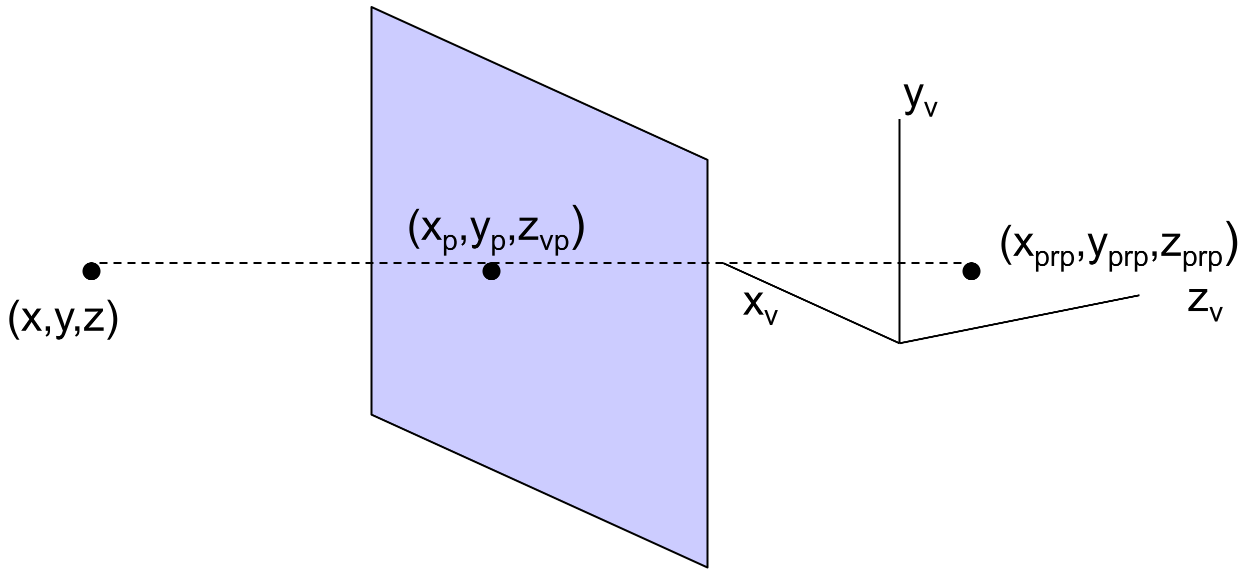
Perspective Projection



Perspective Projection



Perspective Projection



Perspective Projection

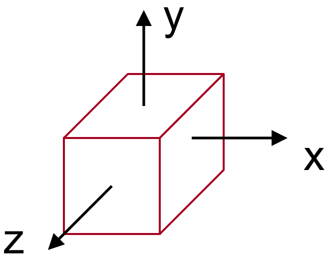
At any point (x_p, y_p, z_p) along the projection line:

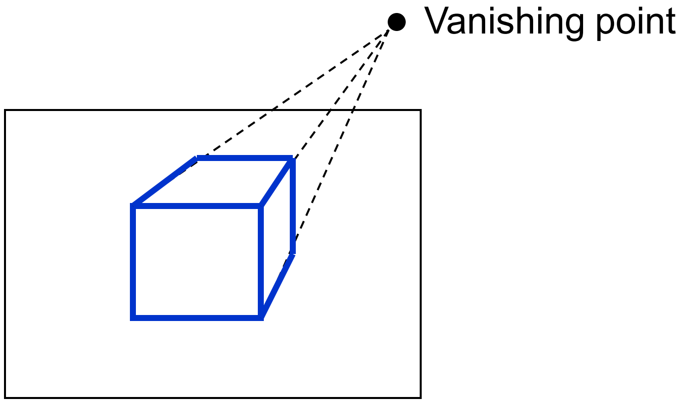
$$z_p = z_{vp}$$

$$x_p = x\,(z_{prp}-z_{vp})\,/\,(z_{prp}-z) + x_{prp}\,(z_{vp}-z)\,/\,(z_{prp}-z)$$

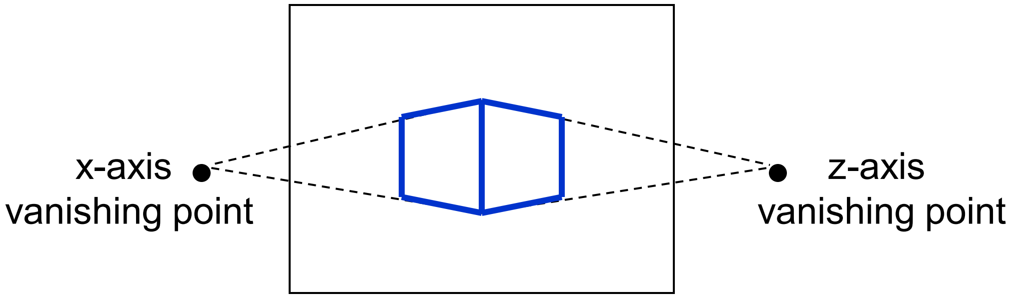
$$y_p = y\,(z_{prp}-z_{vp})\,/\,(z_{prp}-z) + y_{prp}\,(z_{vp}-z)\,/\,(z_{prp}-z)$$

Perspective Projection



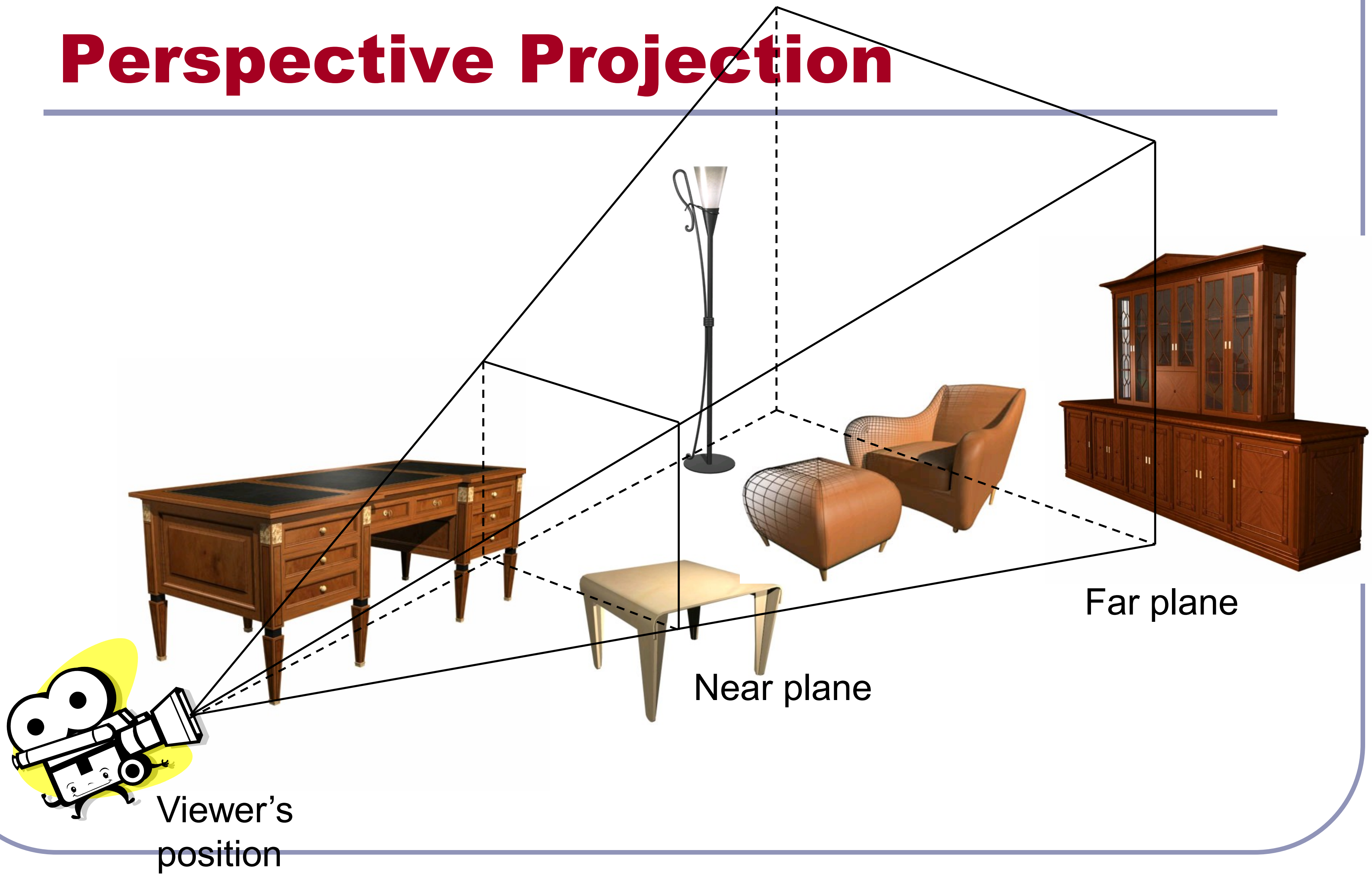


One-point perspective projection

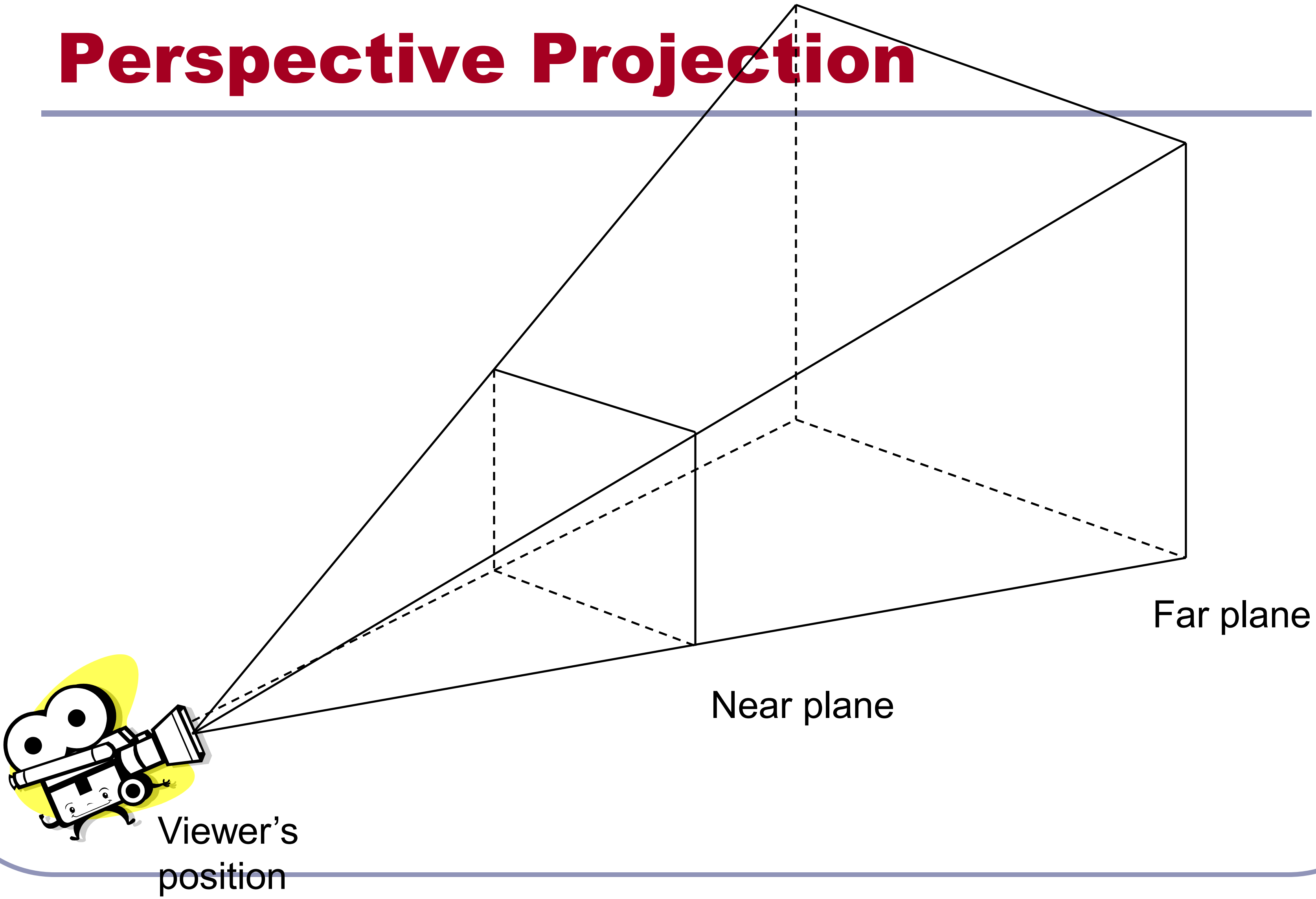


Two-point perspective projection

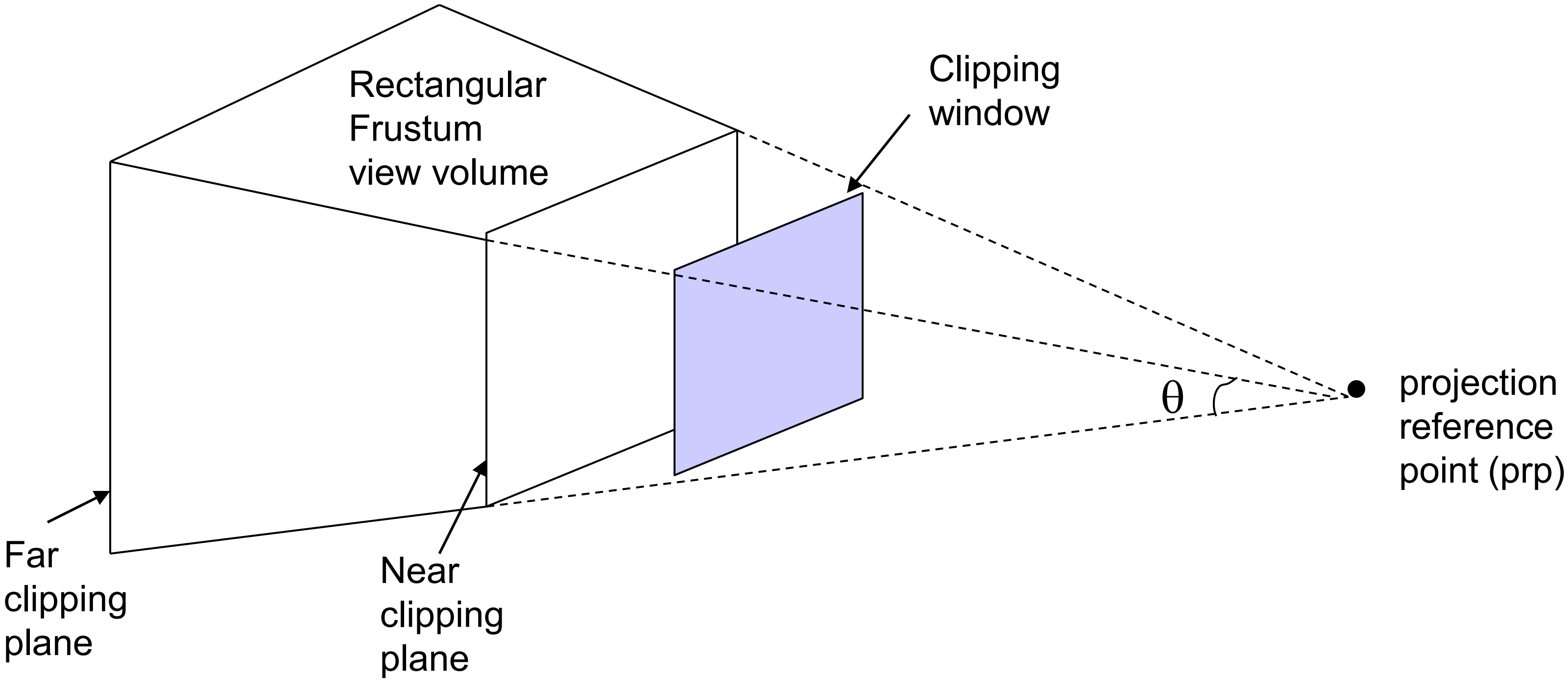
Perspective Projection



Perspective Projection



Perspective Projection



Perspective Projection

$P=(x, y, z, 1)$

$P_h=(x_h, y_h, z_h, h)$

$$x_p = x\,(z_{prp}-z_{vp}) / (z_{prp}-z) + x_{prp}\,(z_{vp}-z) / (z_{prp}-z)$$
$$y_p = y\,(z_{prp}-z_{vp}) / (z_{prp}-z) + y_{prp}\,(z_{vp}-z) / (z_{prp}-z)$$

$h = z_{prp}-z$

$$x_h = x\,(z_{prp}-z_{vp}) + x_{prp}\,(z_{vp}-z)$$
$$y_p = y\,(z_{prp}-z_{vp}) + y_{prp}\,(z_{vp}-z)$$

$x_p = x_h /h$

$y_p = y_h /h$

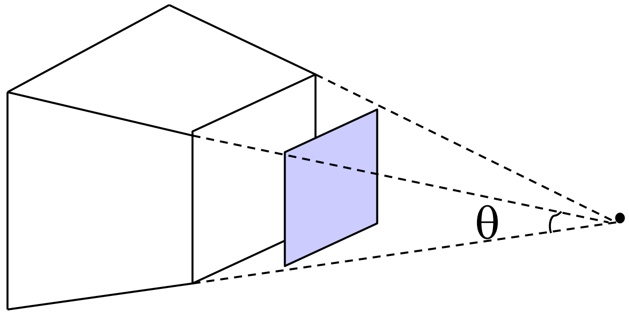
Perspective Projection

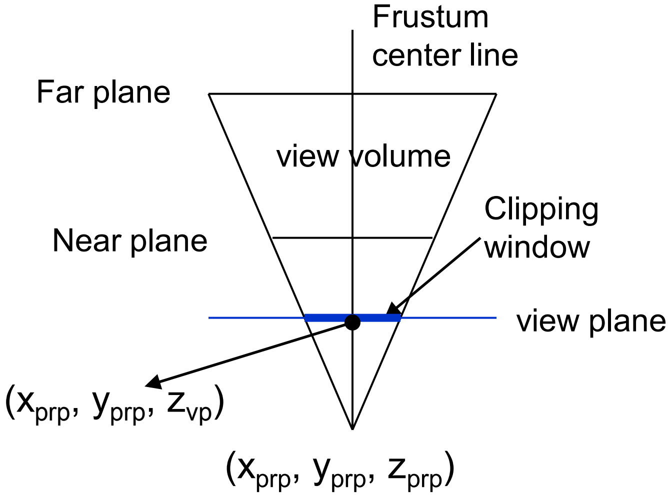
$$P_h = M_{pers} \cdot P$$

$$M_{pers} = \begin{pmatrix} z_{prp}-z_{vp} & 0 & -x_{prp} & x_{prp}z_{prp} \\ 0 & z_{prp}-z_{vp} & -y_{prp} & y_{prp}z_{prp} \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & z_{prp} \end{pmatrix}$$

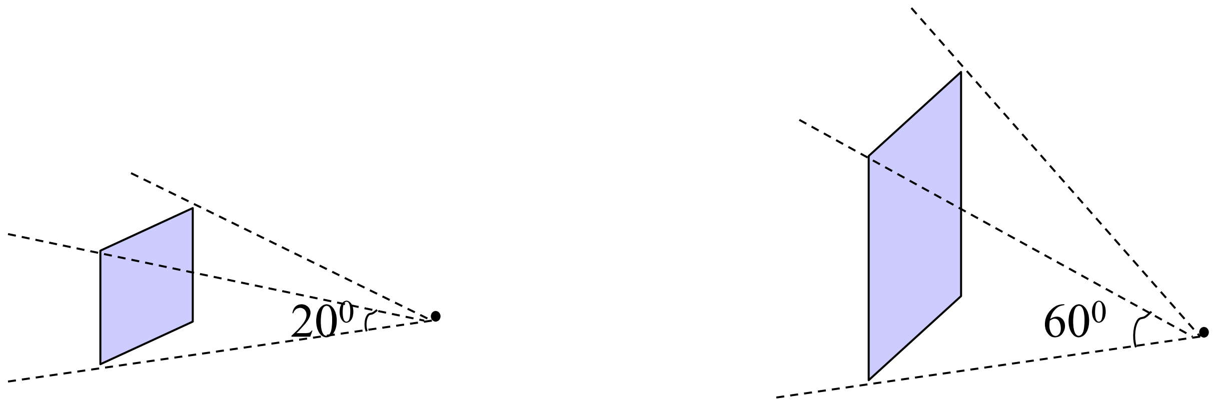
Perspective Projection

Symmetric Frustum





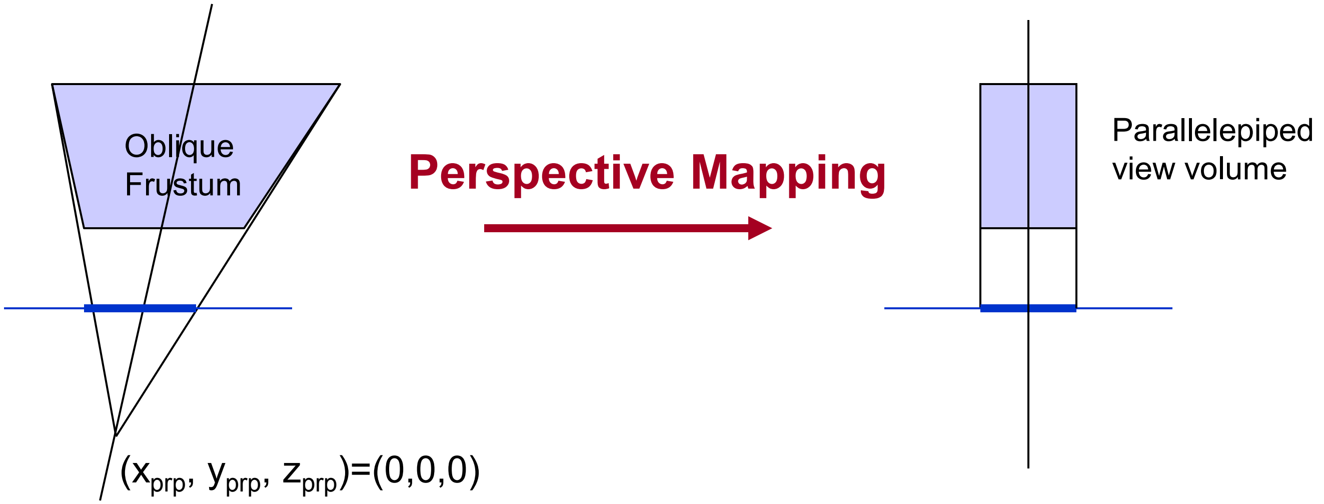
Perspective Projection



Perspective Projection

Then apply normalization transformation

Perspective Projection



1. Transform to asymmetric frustum (z-axis shear)

2. Normalize viewvolume

Perspective Projection

$$M_{shear} = \begin{pmatrix} 1 & 0 & sh_{zx} & 0 \\ 0 & 1 & sh_{zy} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$sh_{zx} = -(xw_{min}+xw_{max})/2.z_{near}$

$sh_{zy} = -(yw_{min}+yw_{max})/2.z_{near}$

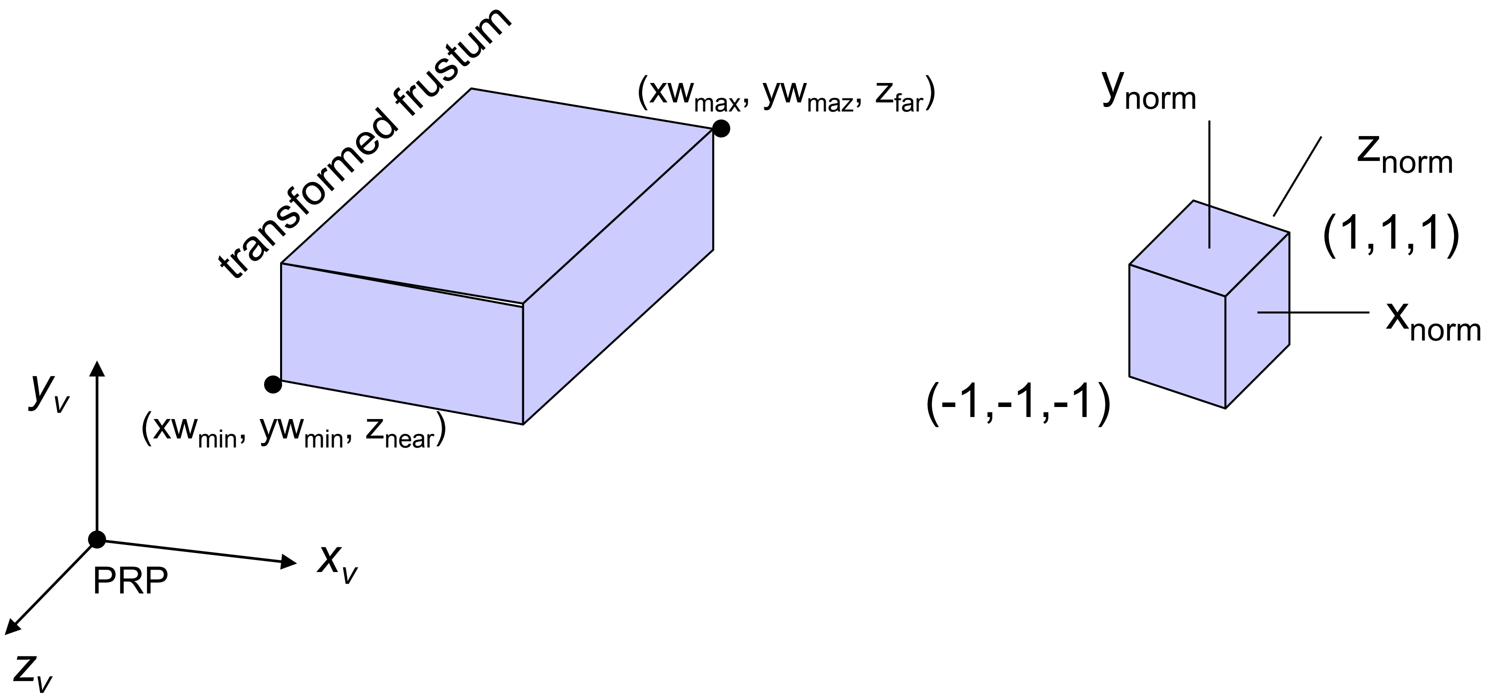
assuming that viewplane is at the position of the near plane

$$M_{pers} = \begin{pmatrix} z_{near} & 0 & 0 & 0 \\ 0 & -z_{near} & 0 & 0 \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

$$M_{oblpers} = M_{pers} \cdot M_{shear}$$

Perspective Projection

Normalization



Perspective Projection

$$M_{normpers} = M_{xyscale} \cdot M_{oblpers}$$

$$= \begin{pmatrix} -z_{near}s_x & 0 & s_x(xw_{min}+xw_{max})/2 & 0 \\ 0 & -z_{near}s_y & s_y(yw_{min}+yw_{max})/2 & 0 \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

$$= \begin{pmatrix} -2z_{near}s_x/(xw_{max}-xw_{min}) & 0 & (xw_{min}+xw_{max})/(xw_{max}-xw_{min}) & 0 \\ 0 & -2z_{near}s_y/(yw_{max}-yw_{min}) & (yw_{min}+yw_{max})/(yw_{max}-yw_{min}) & 0 \\ 0 & 0 & (z_{near}+z_{far})/(z_{near}-z_{far}) & -2.z_{near}.z_{far}/(z_{near}-z_{far}) \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Perspective Projection

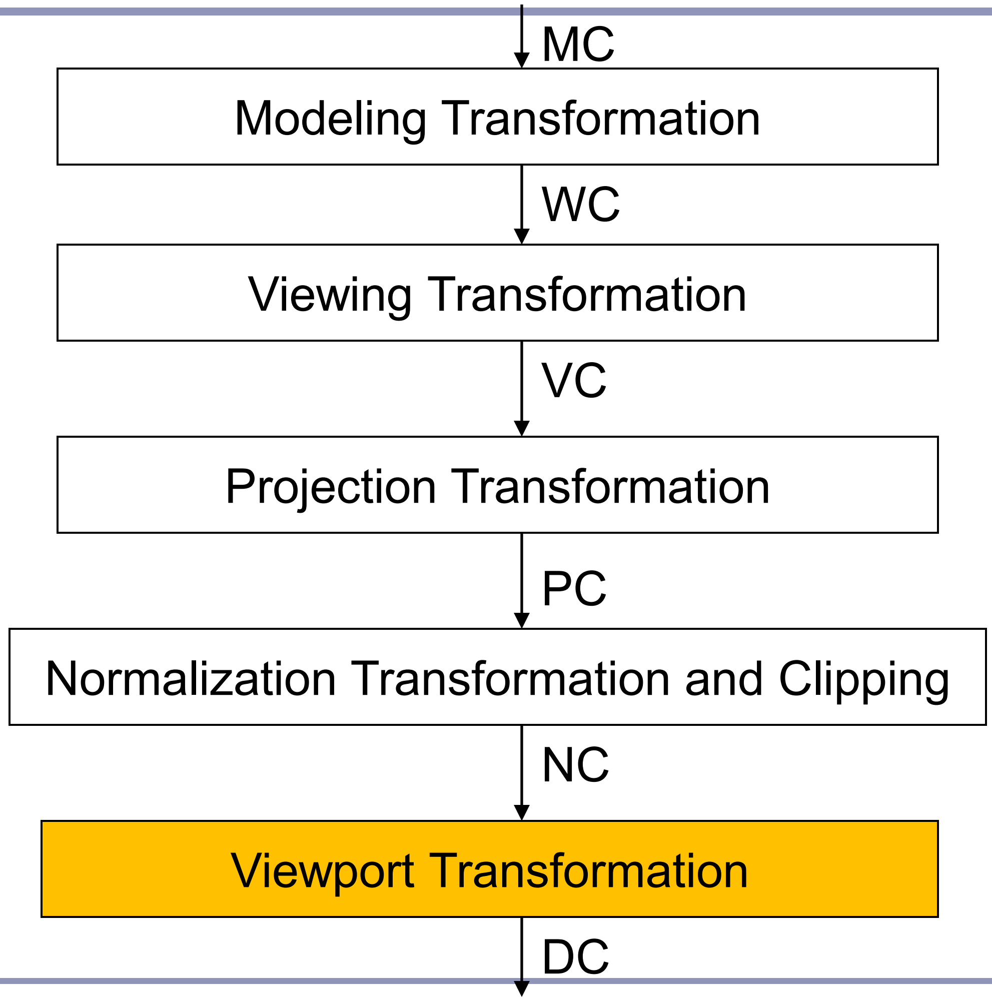
For symmetric frustum with field-of-view angle θ:

$$M_{normsymmpers} =$$

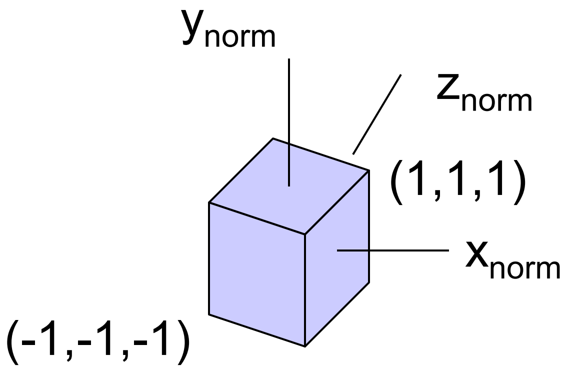
$$\begin{pmatrix} \cot(\theta/2)/aspect & 0 & 0 & 0 \\ 0 & \cot(\theta/2) & 0 & 0 \\ 0 & 0 & (z_{near}+z_{far})/(z_{near}-z_{far}) & -2.z_{near}.z_{far}/(z_{near}-z_{far}) \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

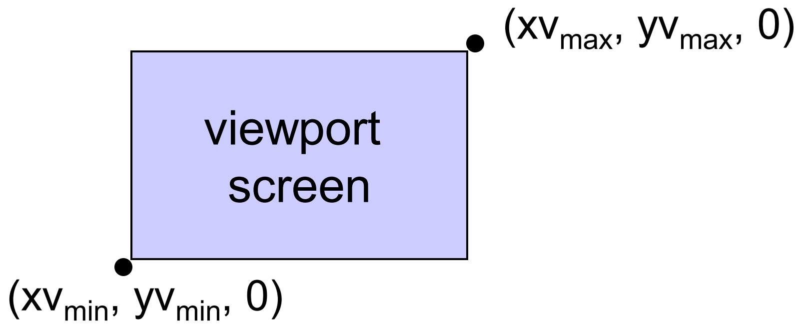
$$M_{normsymmpers} \,.\, \boldsymbol{R} \,.\, \boldsymbol{T}$$

3D Viewing Transformation Pipeline



Viewport Transformation

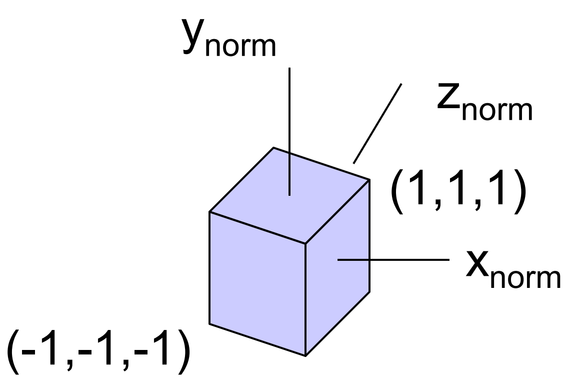




$$M_{normviewvol,3Dscreen} = \begin{pmatrix} (xv_{max}-xv_{min})/2 & 0 & 0 & (xv_{min}+xv_{max})/2 \\ 0 & (yv_{max}-yv_{min})/2 & 0 & (yv_{min}+yv_{max})/2 \\ 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3D Clipping

6	5	4	3	2	1
far	near	top	bottom	right	left



Inside if:

$-h \le x_h \le h$, $-h \le y_h \le h$, $-h \le z_h \le h$

Use sign bits of $h \pm x_h$, $h \pm y_h$, $h \pm z_h$ to set bit1-bit6

Point Clipping

Test sign bits of $h \pm x_h$, $h \pm y_h$, $h \pm z_h$

If region code is 000000 then inside

otherwise eliminate

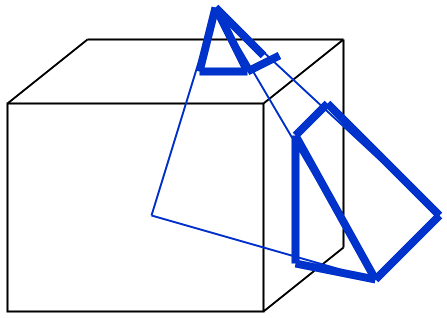
Line Clipping

1. Test region codes

- if 000000 for both endpoints => inside
- if $RC_1 \vee RC_2 = 000000$ => trivially accept
- if $RC_1 \wedge RC_2 \neq 000000$ => trivially reject

2. If a line fails the above tests, use line equation to determine whether there is an intersection

Polygon Clipping



1. Check coordinate limits of the object

- if all limits are inside all boundaries => save the entire object
- if all limits are outside any one of the boundaries => eliminate the entire object

2. Otherwise process the vertices of the polygons

- Apply 2D polygon clipping

  Clip edges to obtain new vertex list.
- Update polygon tables to add new surfaces

If the object consists of triangle polygons, use Sutherland-Hodgman algorithm.

OpenGL

- glMatrixMode (GL_PROJECTION)

- gluOrtho (xwmin, xwmax, ywmin, ywmax, dnear, dfar)

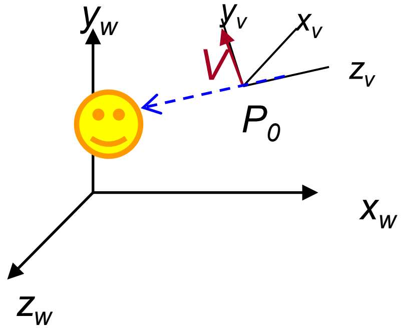
  Orthogonal projection

- gluPerspective (theta, aspect, dnear, dfar)
  - theta: field-of-view angle (0^0 – 180^0)
  - aspect: *aspect ratio* of the clipping window (width/height)
  - dnear, dfar: positions of the near and far planes (must have positive values)

- glFrustum (xwmin, xwmax, ywmin, ywmax, dnear, dfar)
  - if xwmin = -xwmax and ywmin = -ywmax then symmetric view volume

OpenGL

- **glMatrixMode (GL_MODELVIEW)**
- **gluLookAt (x0, y0, z0, xref, yref, zref, Vx, Vy, Vz)**
  Designates the origin of the viewing reference frame as,
  - **the world coordinate position P0=(x0, y0, z0)**
  - **the reference position Pref=(xref, yref, zref) and**
  - **the viewup vector V=(Vx, Vy, Vz)**



OpenGL

```
float eye_x, eye_y, eye_z;

void init (void) {
    glClearColor (1.0, 1.0, 1.0, 0.0);  // Set display-window color to white.

    glMatrixMode (GL_PROJECTION);       // Set projection parameters.
    glLoadIdentity();
    gluPerspective(80.0, 1.0, 50.0, 500.0);       // degree, aspect, near, far

    glMatrixMode (GL_MODELVIEW);
    eye_x = eye_y = eye_z = 60;
    gluLookAt(eye_x, eye_y, eye_z,  0,0,0,  0,1,0);    // eye, look-at, view-up
}

void main (int argc, char** argv) {
    glutInit (&argc, argv);                          // Initialize GLUT.
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);   // Set display mode.
    glutInitWindowPosition (50, 500);   // Set top-left display-window position.
    glutInitWindowSize (400, 300);      // Set display-window width and height.
    glutCreateWindow ("An Example OpenGL Program"); // Create display window.
    glViewport (0, 0, 400, 300);
    init ( );                            // Execute initialization procedure.
    glutDisplayFunc (my_objects);       // Send graphics to display window.
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keybrd);
    glutMainLoop ( );                    // Display everything and wait.
}
```