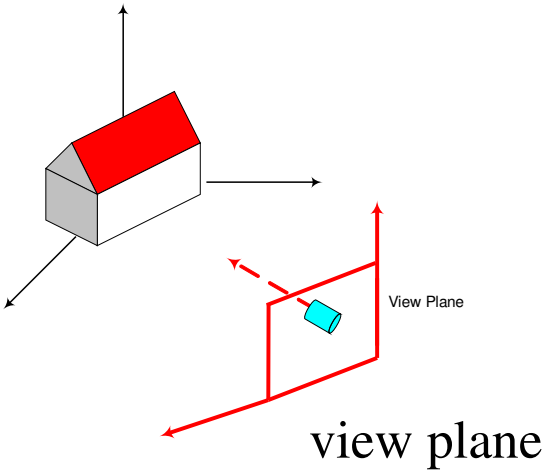
3 DIMENSIONAL VIEWING

Ceng 477
Introduction to Computer Graphics

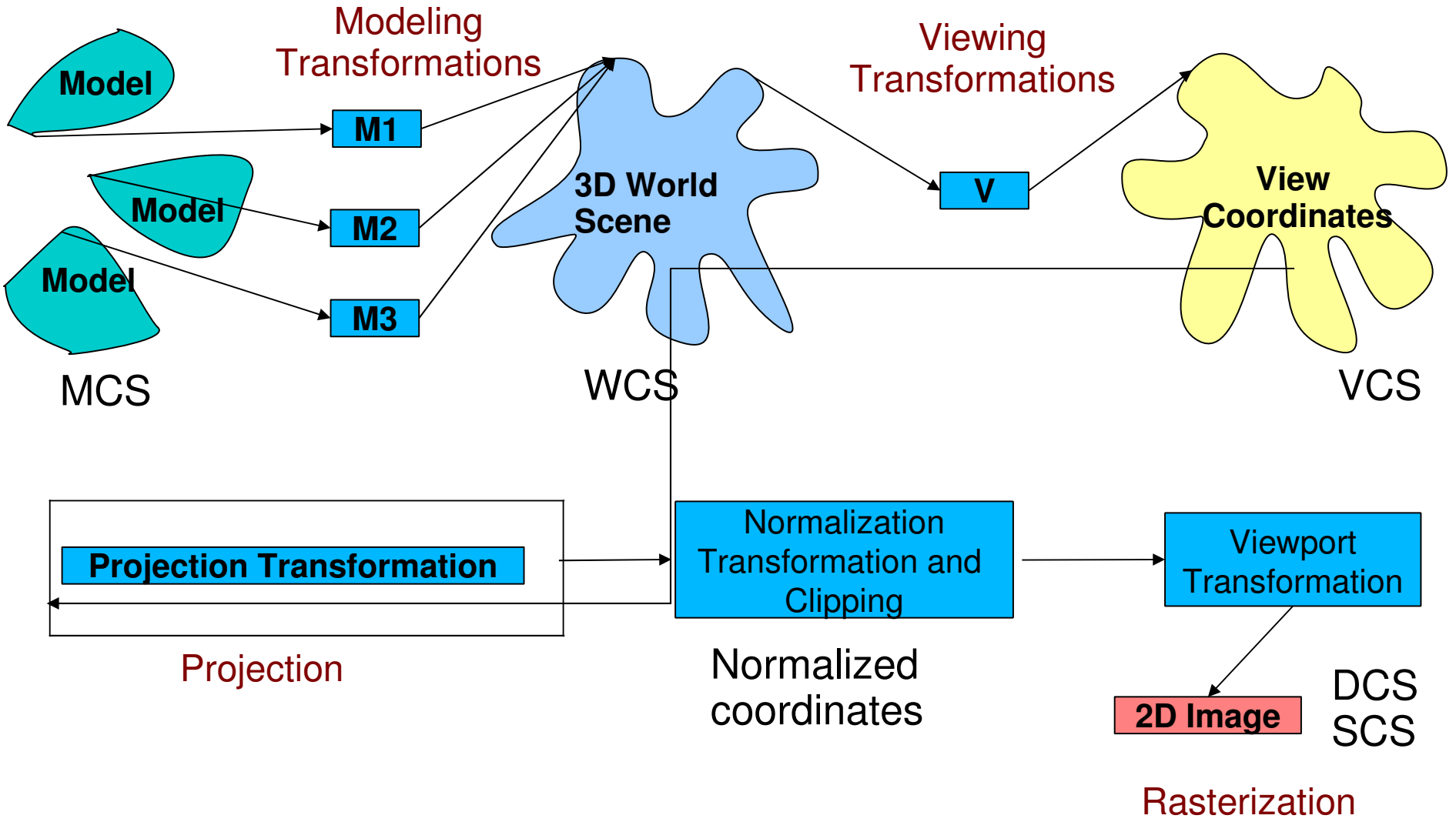
Computer Engineering
METU

3D Viewing Concepts

- Involves some task that are not present in 2D viewing
  - 3D viewing reference frame, i.e., camera
  - Projection: Transfer the scene to view on a planar surface
  - Visible part identification
  - Lighting
  - Surface characteristics (opaque, transparent)

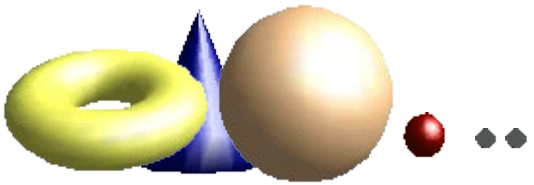


3D Viewing Pipeline

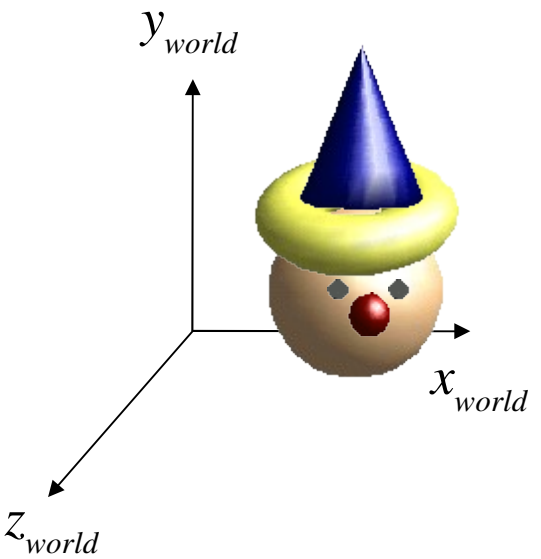


Modeling Transformation

- Model coordinates to World coordinates:



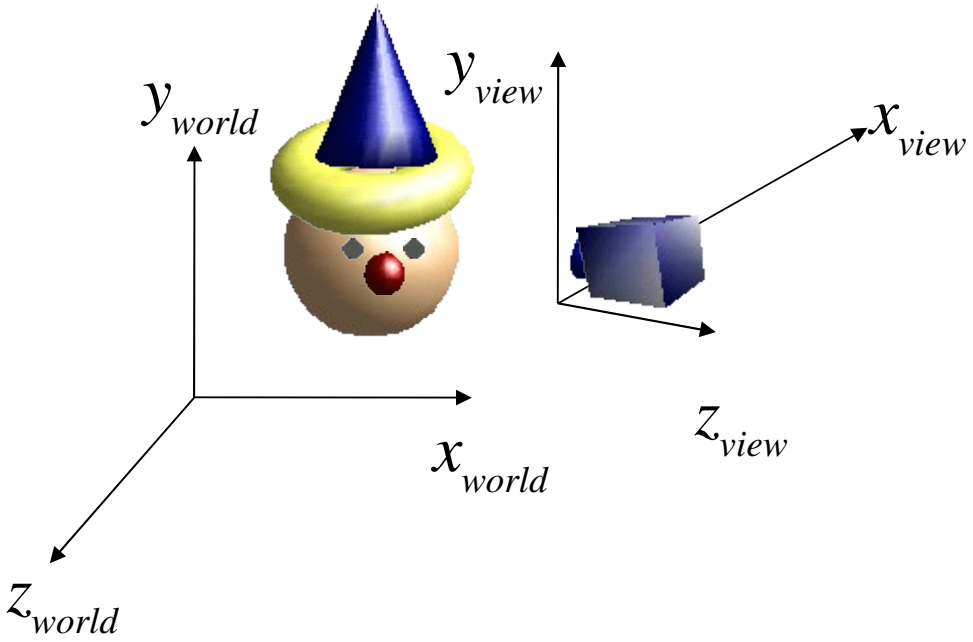
World coordinates:
All shapes with their
absolute coordinates and sizes.



3D Viewing-Coordinate Parameters

- World coordinates to Viewing coordinates: Viewing transformations

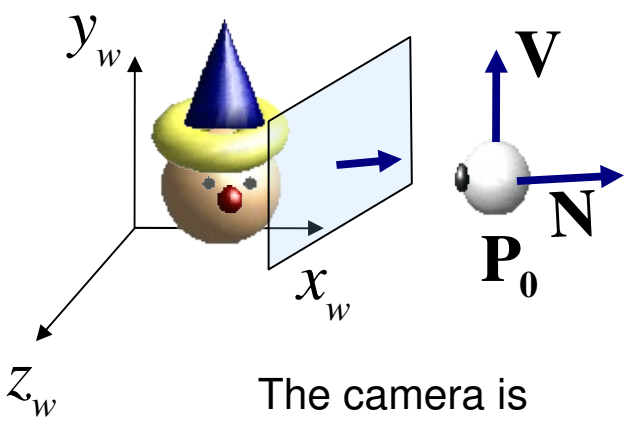
World coordinates



Viewing coordinates:
Viewers (Camera) position and viewing plane.

Viewing Parameters

- How to define the viewing coordinate system (or view reference coordinate system):
  - Position of the viewer: $\mathbf{P}_0$
    - view point or viewing position
  - Orientation of the viewer:
    - View up vector: $\mathbf{V}$
    - Viewing direction: $\mathbf{N}$ (view plane normal)
    - $\mathbf{N}$ and $\mathbf{V}$ should be orthogonal
      - if it is not, $\mathbf{V}$ can be adjusted to be orthogonal to $\mathbf{N}$



The camera is looking at opposite direction of $\mathbf{N}$

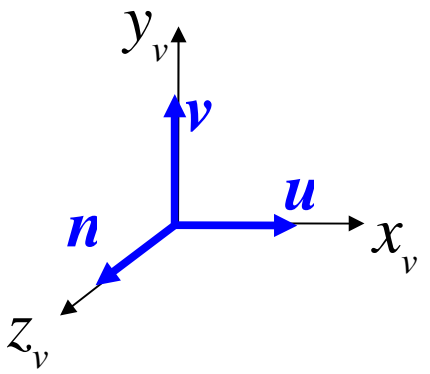
The uvn Viewing-Coordinate Reference Frame

- A set of unit vectors that define the viewing coordinate system is obtained by:

$$n = \frac{N}{|N|} = (n_x, n_y, n_z)$$

$$u = \frac{V \times n}{|V|} = (u_x, u_y, u_z)$$

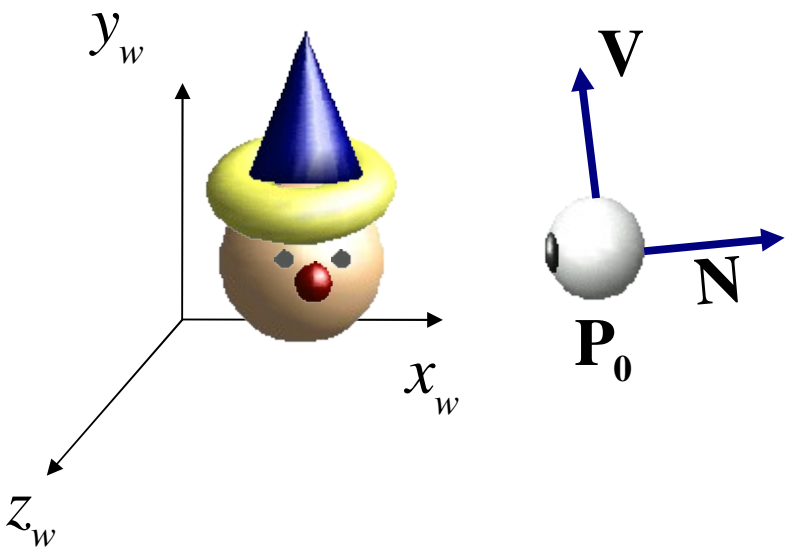
$$v = n \times u = (v_x, v_y, v_z)$$



By changing some of the viewing parameters (e.g., **N**, viewing position), we can generate 3d viewing effects like rotating around an object or flying over a scene.

Transformation Between Coordinate Systems

- Given the objects in world coordinates, find the transformation matrix to transform them into viewing coordinate system. **n, v, u** : unit vectors defining the viewing coordinate system.



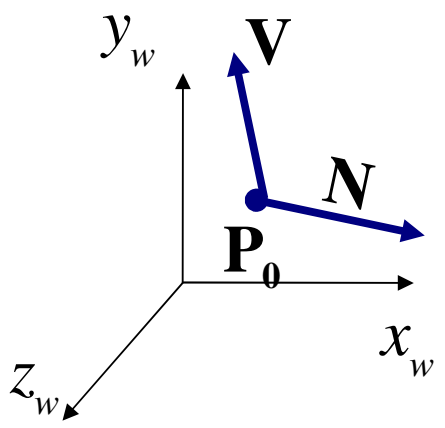
World coordinate system can be aligned with the viewing coordinate system in two steps: (1) Translate $\mathbf{P}_0$ to the origin of the world coordinate system, (2) Rotate to align the axes.

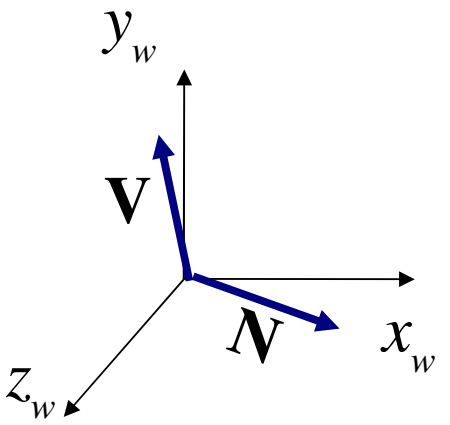
Step 1

- Translation: Move view reference point to origin.

$$P_0 = (x_0, y_0, z_0)$$

$$T = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

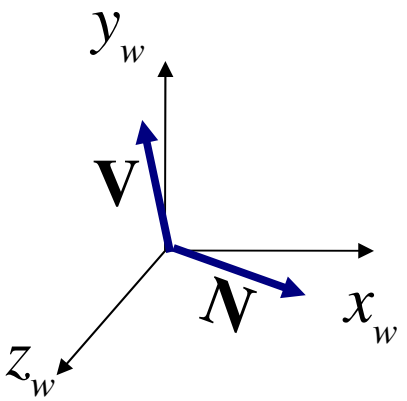


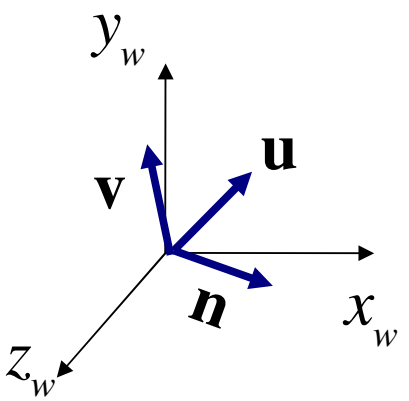


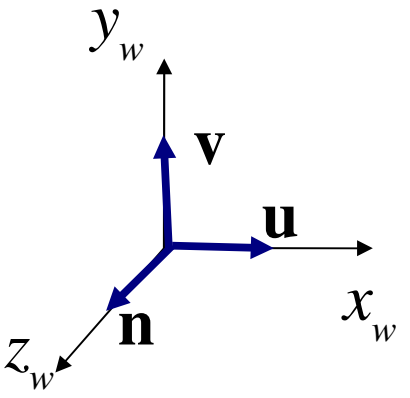
Step 2

- Rotation: After we find $\mathbf{n}$, $\mathbf{v}$ and $\mathbf{u}$, we can use them to define the rotation matrix for aligning the axes.

$$R = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$







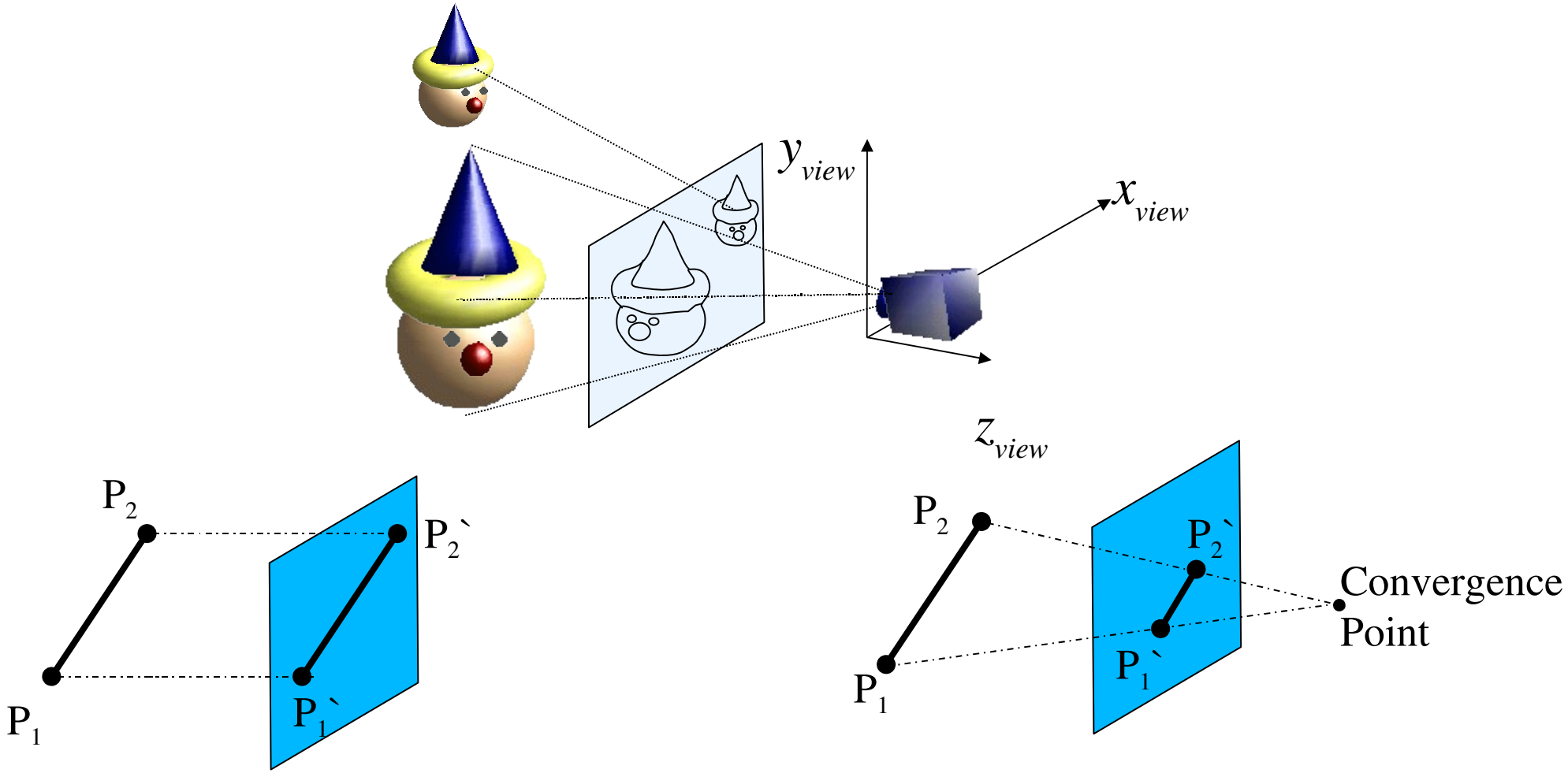
WC to VC transformation matrix

- The transformation matrix from world coordinate to viewing reference frame

$$M_{WC,VC} = R \cdot T = \begin{bmatrix} u_x & u_y & u_z & -x_0 u_x - y_0 u_y - z_0 u_z \\ v_x & v_y & v_z & -x_0 v_x - y_0 v_y - z_0 v_z \\ n_x & n_y & n_z & -x_0 n_x - y_0 n_y - z_0 n_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

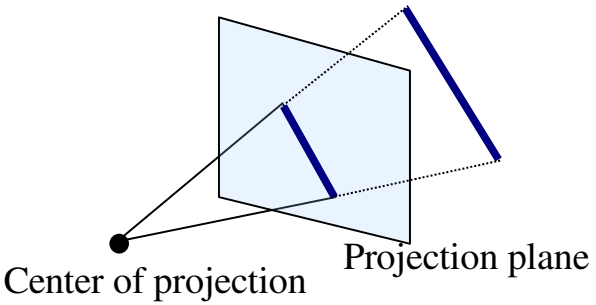
Projection transformations

- Projection: 3D to 2D. Perspective or parallel.

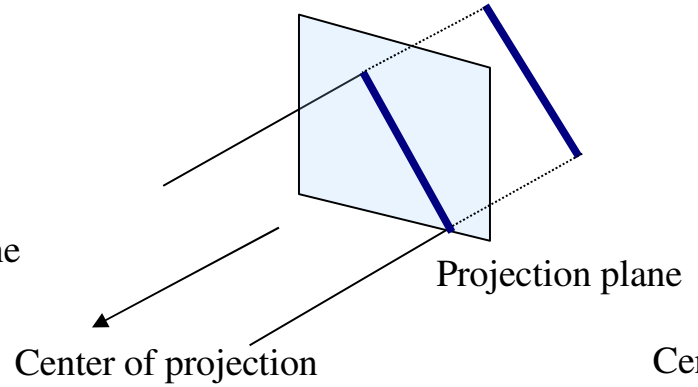


Projections

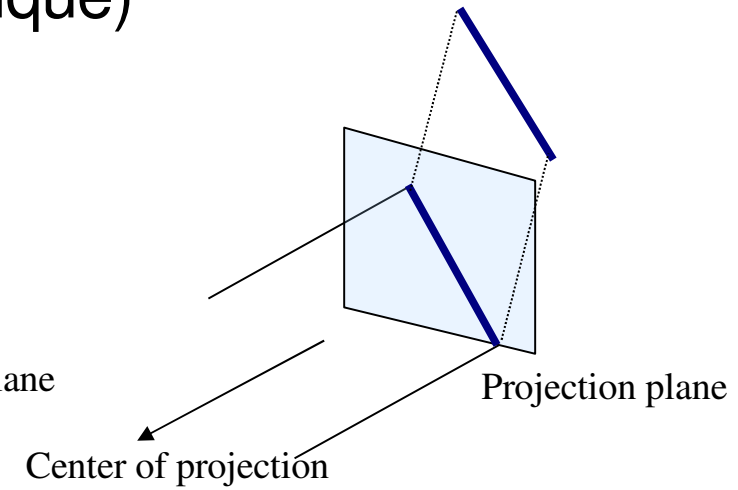
- Classification of projections. Based on:
  - Center of projection: infinity (parallel) or a point (perspective)
  - Projection lines wrt. projection plane: orthogonal (orthographic), another angle (oblique)



Perspective

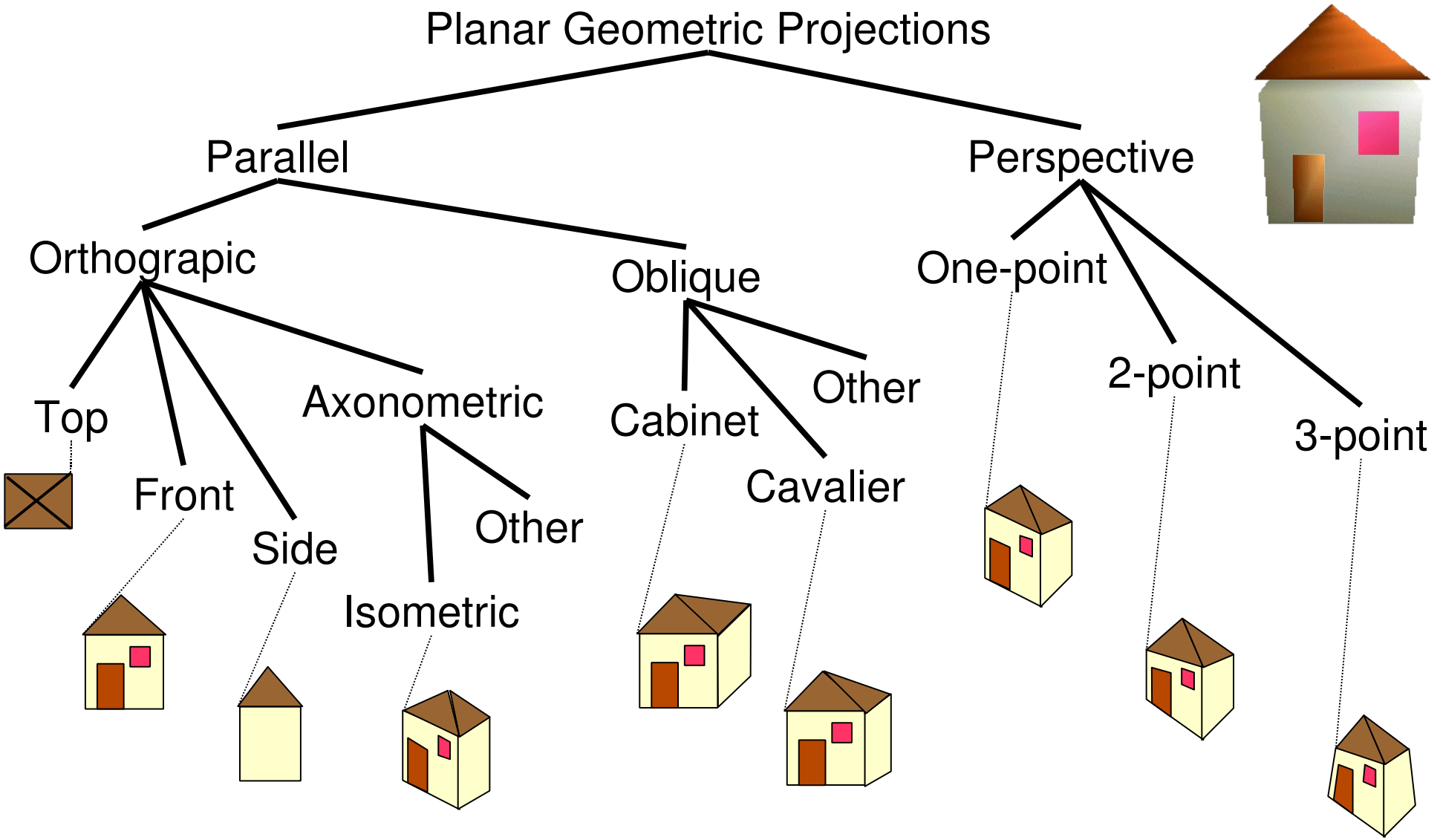


Orthographic



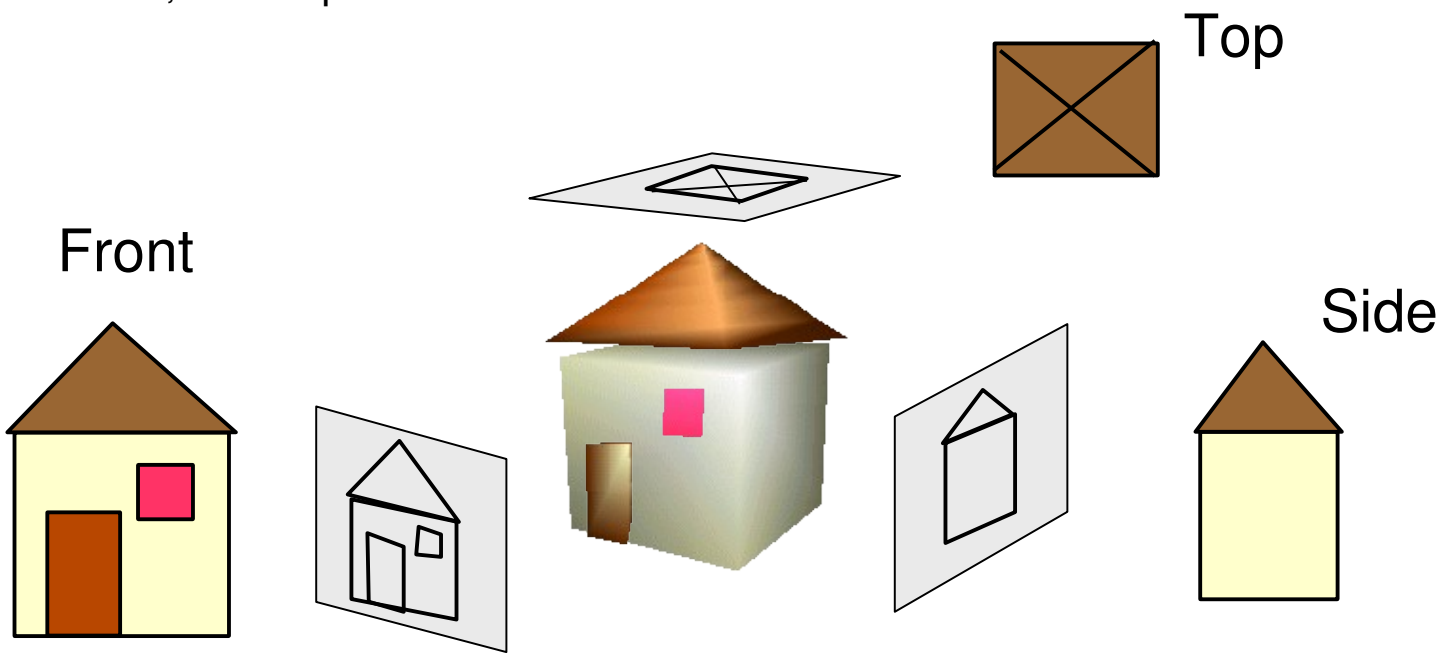
Oblique

Projection types



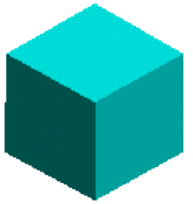
Orthographic projection

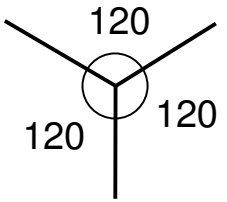
- Multiview Orthographic:
  - Parallel projection to z=0,y=0,x=0 planes
  - Used for engineering drawings, architectural drawing
  - Accurate, scales preserved. Not 3D realistic.



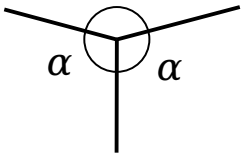
Axonometric projections

- Axonometric Projections to display more than one face of an object
  - Projection plane is not parallel to coordinate planes
  - Isometric: all angles between principal axes are equal

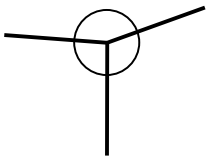




  - Dimetric: angles between two principal axes are equal

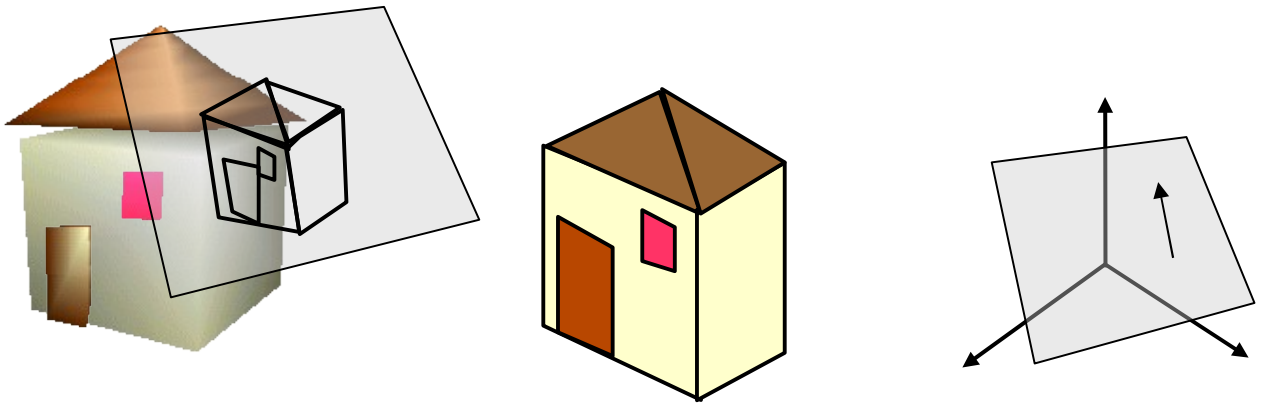


  - Trimetric: all angles different

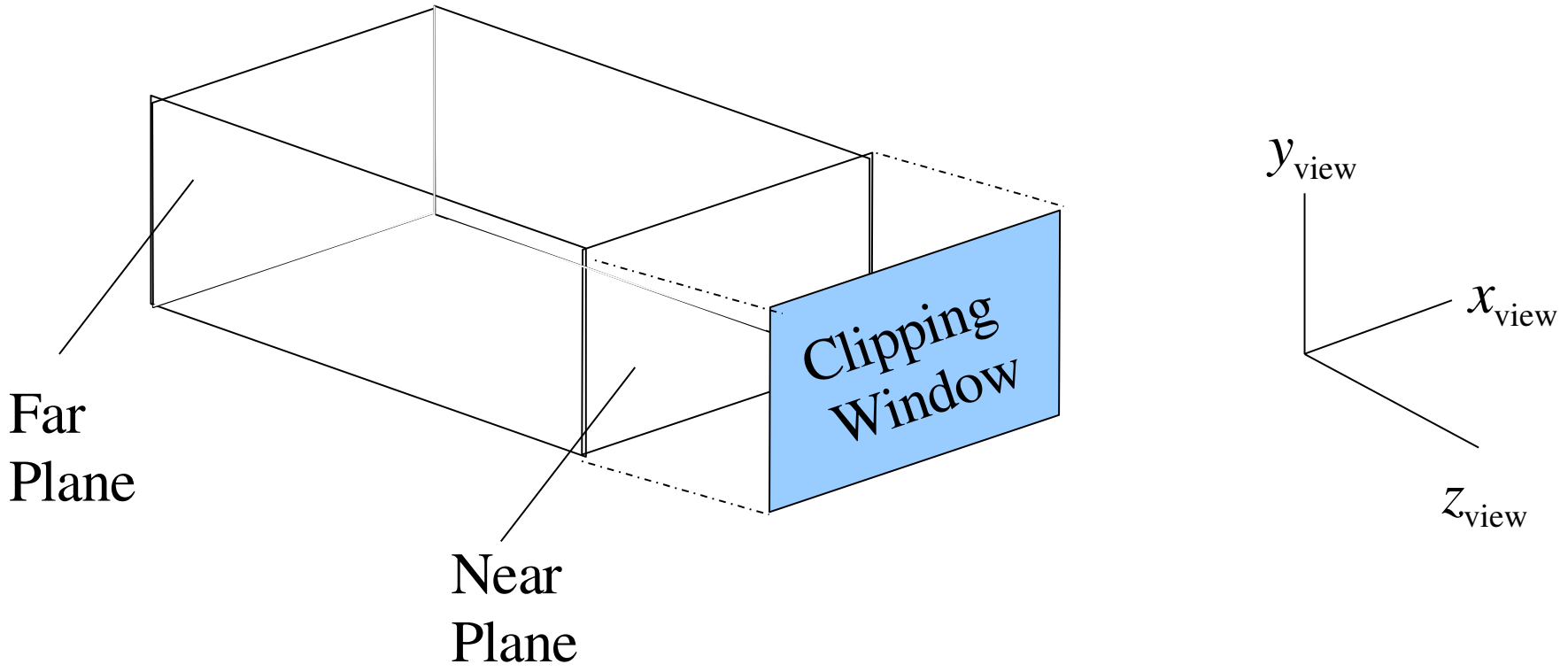


Isometric Projection

- Isometric Projection (i.e. N=[*c,c,c*])
  - Scales preserved along axes, closer to 3D but still not realistic
  - Used in designs and catalogues. Suitable for rectangular bodies.
  - Exercise: Calculate rotation matrix for isometric projection, up vector on $x=z$.



Orthogonal Projection View Volume



Normalization Transformation

- The coordinates within the view volume are normalized to the range [-1,1] (or [0,1])
- The orthographic volume which may be a long rectangular prism is going to be mapped to a unit cube
- A transformation to a left handed coordinate system (i.e., z-axis inversion) may also be needed
- In the orthogonal volume $(x_{\text{min}}, y_{\text{min}}, z_{\text{near}})$ is mapped to (-1,-1,-1) in the normalized volume and $(x_{\text{max}}, y_{\text{max}}, z_{\text{far}})$ is mapped to (1,1,1)

Left-handed coordinate system

Depth (i.e., z values) are increasing away from the viewport

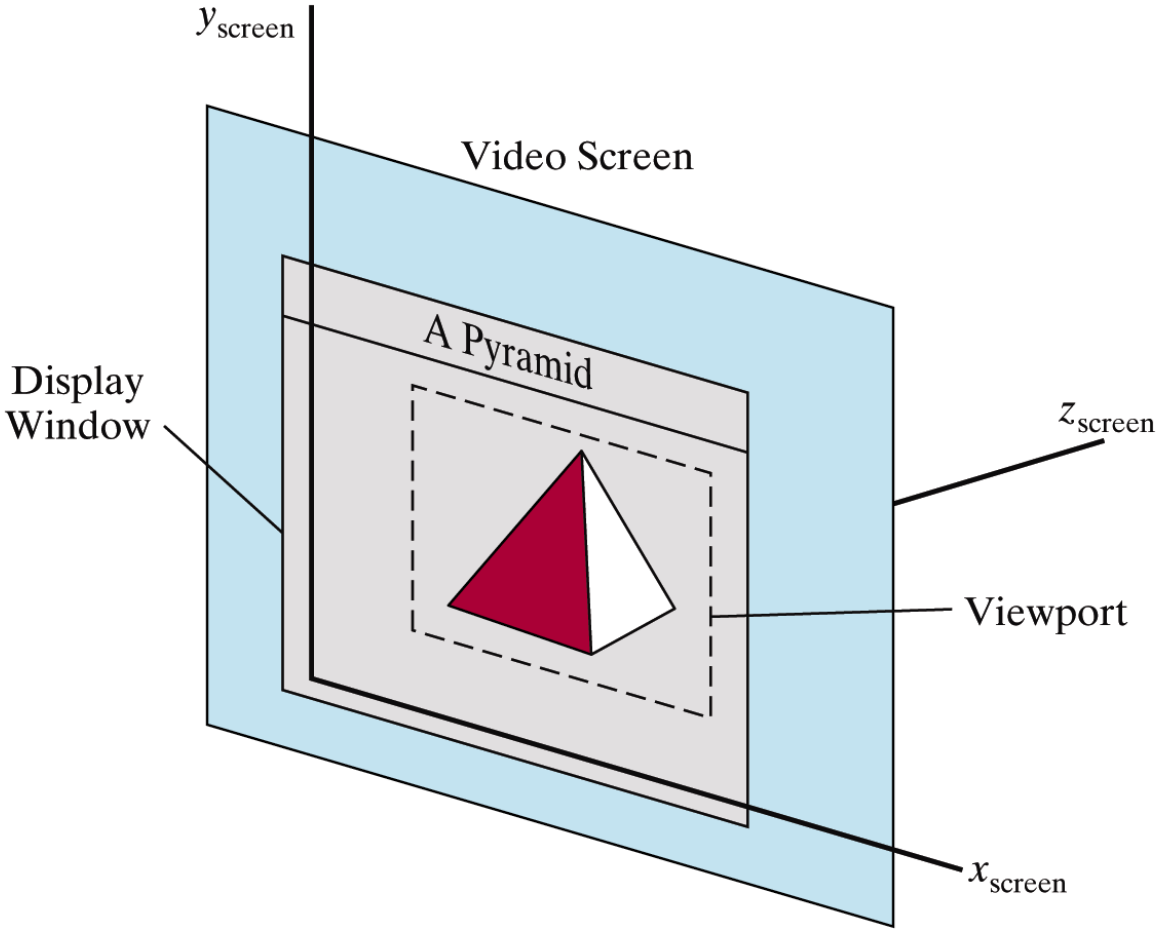
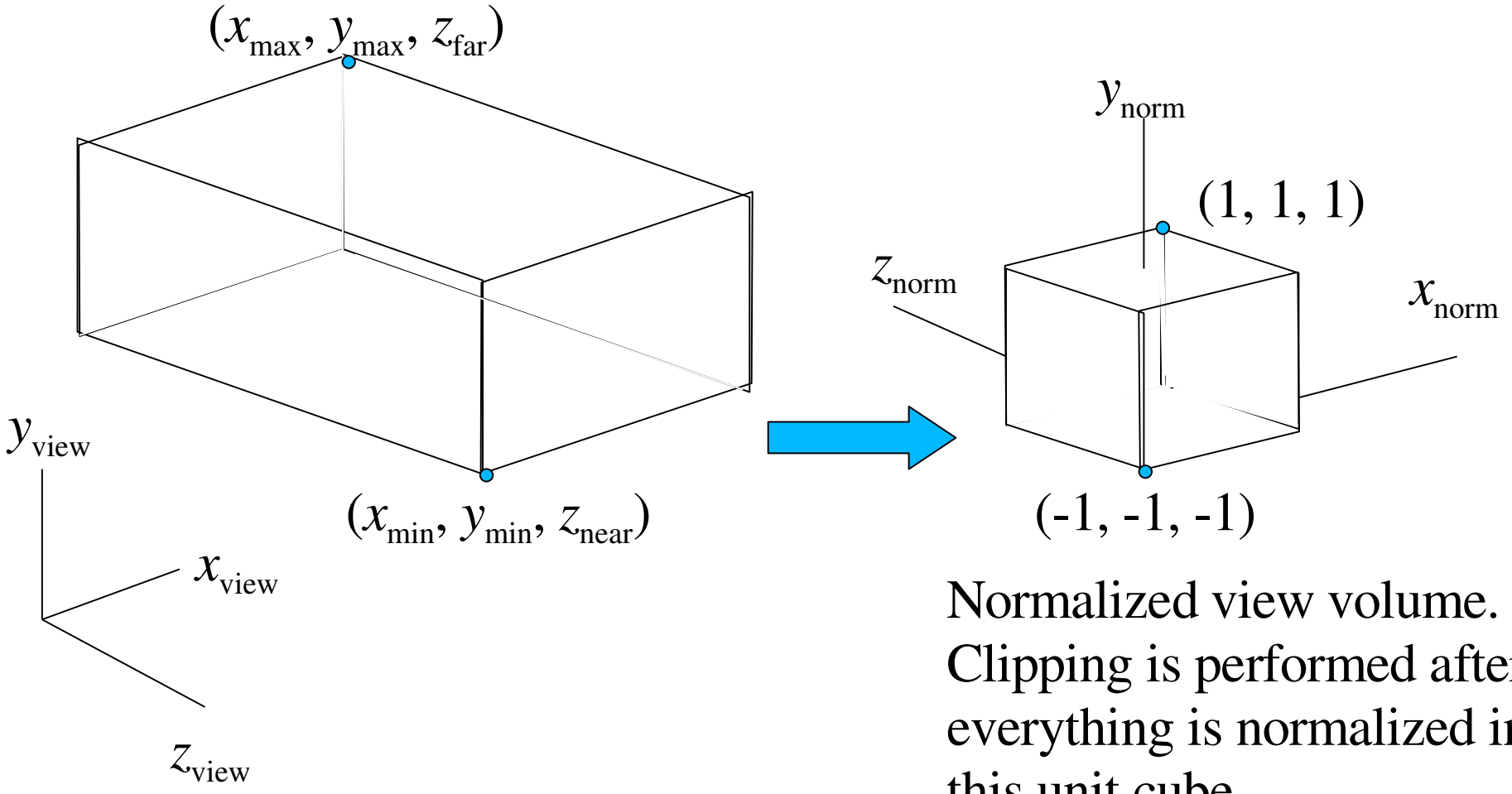


Figure 7-30

A left-handed screen-coordinate reference frame.

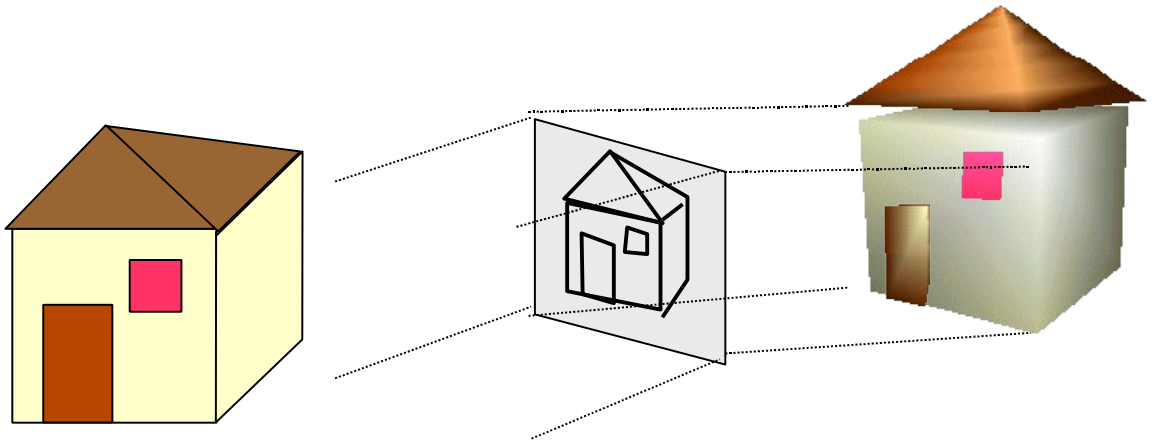
Normalization transformation



Normalized view volume. Clipping is performed after everything is normalized into this unit cube.

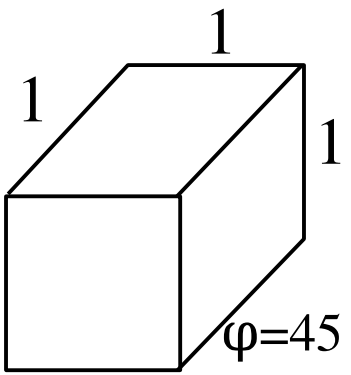
Oblique parallel projections

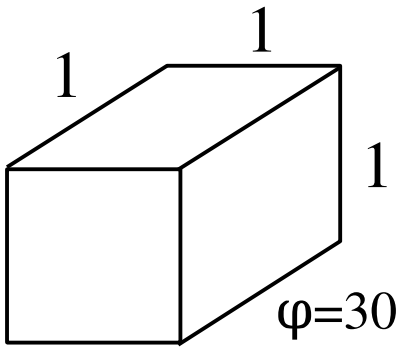
- Oblique projections
  - Projectors have an oblique angle
  - One of the sides have exact dimensions. Others are proportional.
  - Similar to shear transform
  - Using in mechanical viewing



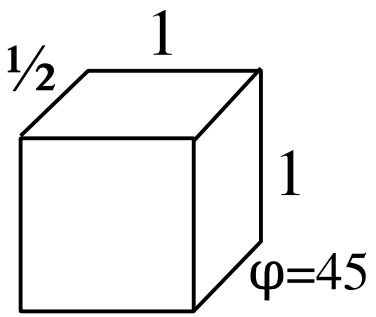
Oblique Projection Types

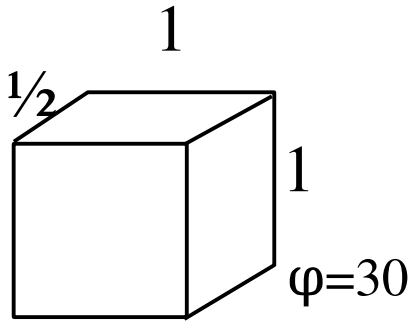
- Based on the angle the projection lines and the projection plane:
  - *Cavalier*: Angle between projectors and projection plane is 45. Depth is projected full scale



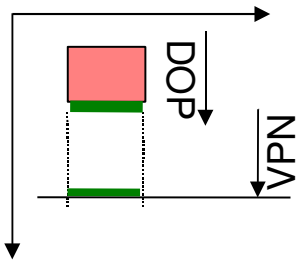


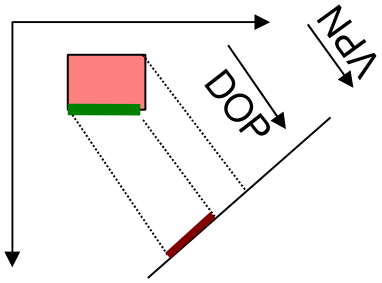
Cabinet: Angle between projectors and proejction plane is arctan(2)=63.4... . Depth is projected ½ scale.

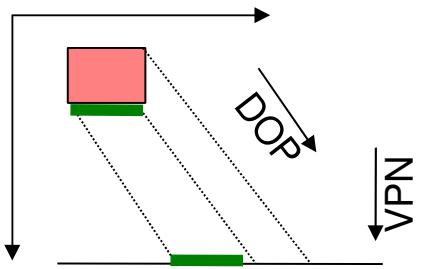




Parallel Projection Summary





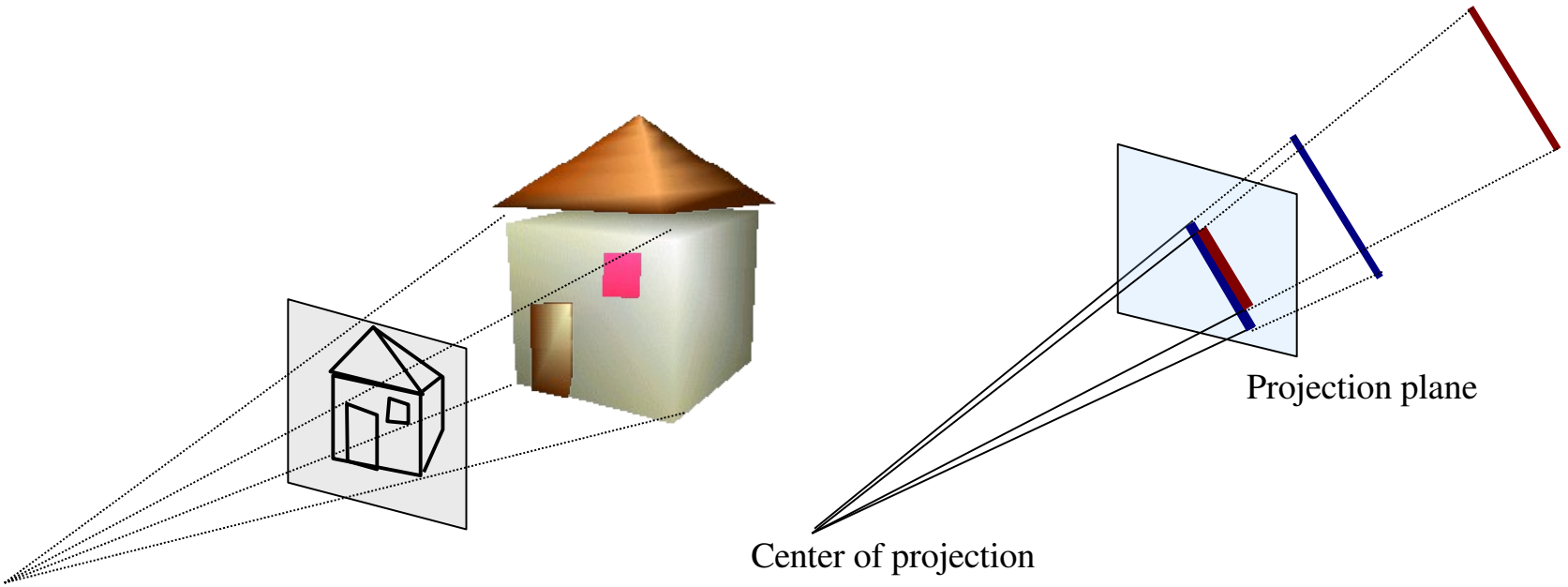


VPN: View Plane Normal
DOP: Direction of Projection

- Multiview orthographic
  - VPN || a principal coordinate axis
  - DOP || VPN
  - Single face, exact dimensions
- Axonometric
  - VPN not || a principal coordinate axis
  - DOP || VPN
  - Not exact dimensions
- Oblique
  - DOP not || VPN
  - One face exact dimension.

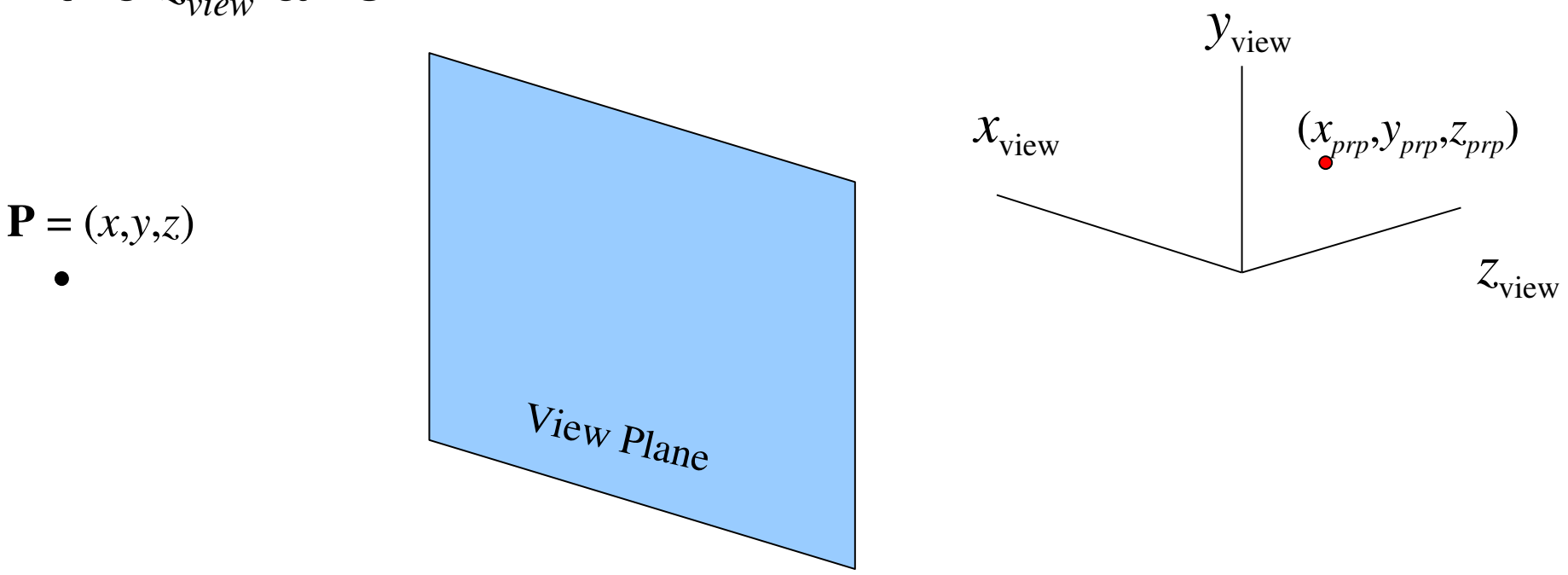
Perspective Projection

- Single point center of projection (i.e. projection lines converge at a point)
- Shapes are projected smaller as their distances to the view plane increase.
- More realistic (human eye is a perspective projector)
- Depending on number of principal axes intersecting the viewing plane: 1, 2 or 3 vanishing points



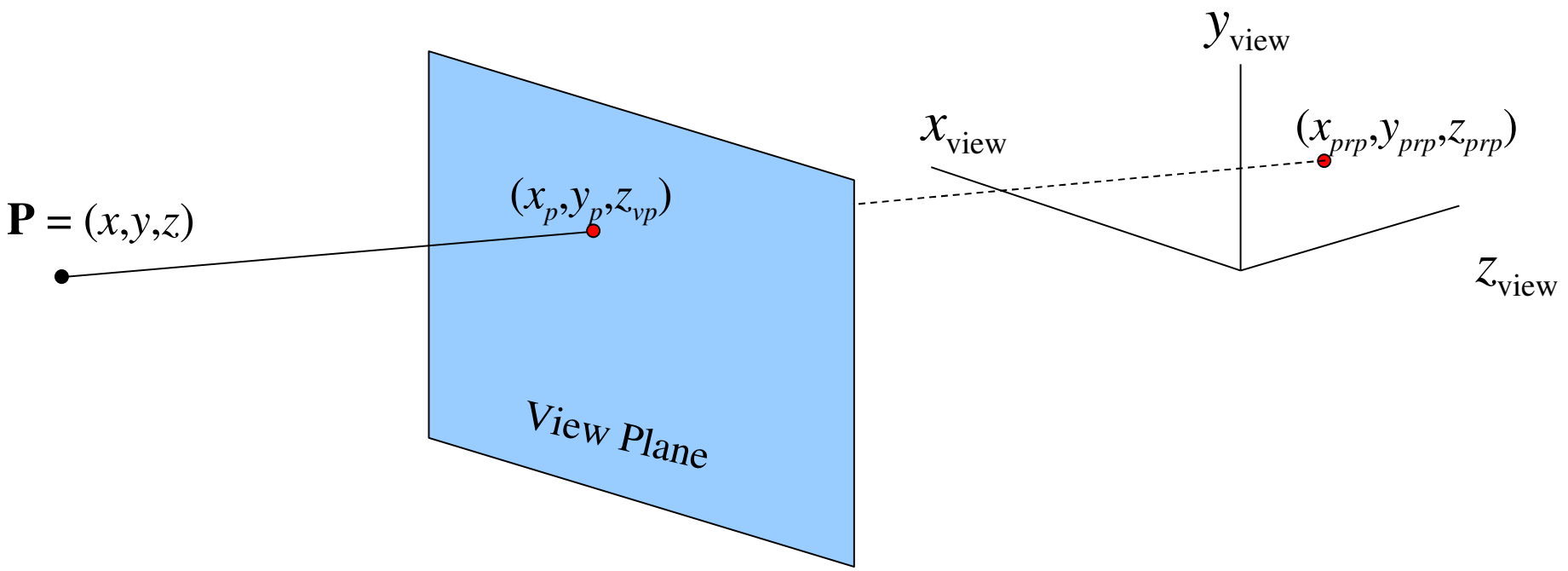
Perspective Projection

- Assume that the projection reference point is selected at an arbitrary position $(x_{prp}, y_{prp}, z_{prp})$ and the view plane is located at a selected position z_{vp} on the z_{view} axis.



Perspective Projection

- Our goal is to find the projection of point P on the view plane which is (x_p, y_p, z_{vp})



Projection equations

- The projected point (x_p, y_p, z_{vp}) can be found using parametric equation of the projection line:

$$x' = x - (x - x_{prp})u$$
$$y' = y - (y - y_{prp})u \qquad 0 \le u \le 1$$
$$z' = z - (z - z_{prp})u$$

We can solve for u at $z' = z_{vp}$ and then plug-in u to find x_p and y_p.

Projection equations

$$u = \frac{z_{vp} - z}{z_{prp} - z}$$

$$x_p = x\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) + x_{prp}\left(\frac{z_{vp} - z}{z_{prp} - z}\right)$$

$$y_p = y\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) + y_{prp}\left(\frac{z_{vp} - z}{z_{prp} - z}\right)$$

The denominators are functions of z, hence we cannot directly derive the matrix form. We will use homogeneous coordinates to do that.

Special cases

- When the projection reference point is on the z_{view} axis.

$$x_{prp} = y_{prp} = 0$$

$$\Rightarrow$$

$$x_p = x\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) \qquad y_p = y\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right)$$

Special cases

- When the projection reference point is at the view coordinate system origin.

$$x_{prp} = y_{prp} = z_{prp} = 0$$

$$\Rightarrow$$

$$x_p = x\left(\frac{z_{vp}}{z}\right) \qquad y_p = y\left(\frac{z_{vp}}{z}\right)$$

Special cases

- When the view plane is the uv plane and the projection reference point is on the z_{view} axis.

$$x_{prp} = y_{prp} = z_{vp} = 0$$

$$\Rightarrow$$

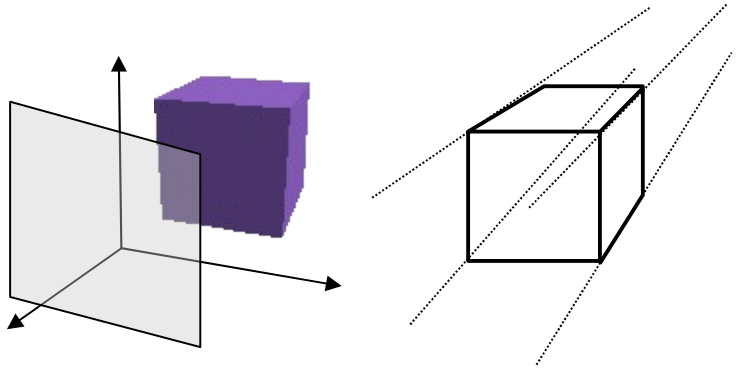
$$x_p = x\left(\frac{z_{prp}}{z_{prp} - z}\right) \qquad y_p = y\left(\frac{z_{prp}}{z_{prp} - z}\right)$$

Vanishing points

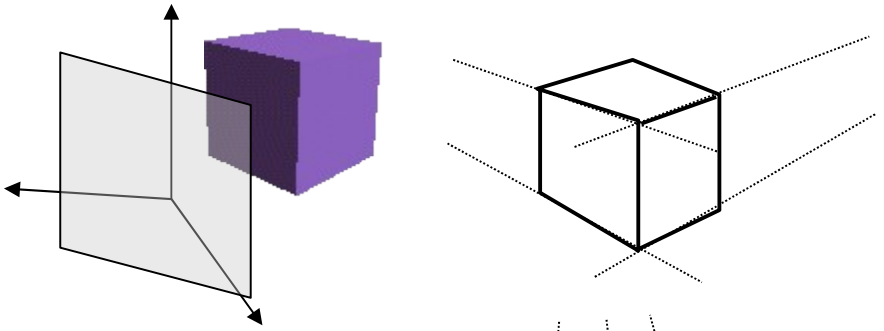
- When a scene is projected, lines that are parallel to the view plane are projected also as parallel lines.
- What about parallel lines that are not parallel to the view plane?
  - They converge at a vanishing point
- For lines that are parallel to one of the principle axes of an object the vanishing point is called the *principle vanishing point*.
- The number of *principle vanishing points* is equal to the number of principle axes that intersect with the view plane.

Perspective projection types wrt principle vanishing points

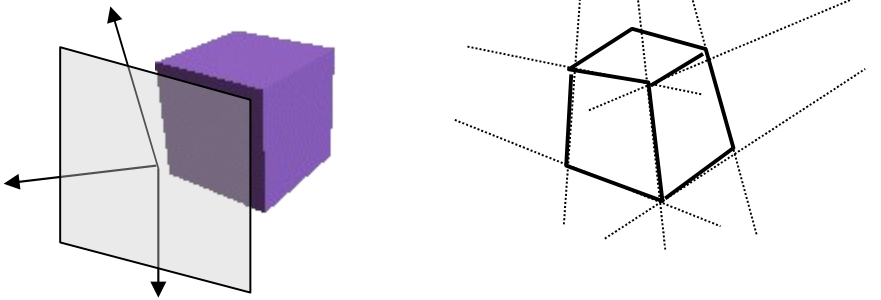
- One point perspective
  only *z* axis intersects
  single vanishing point



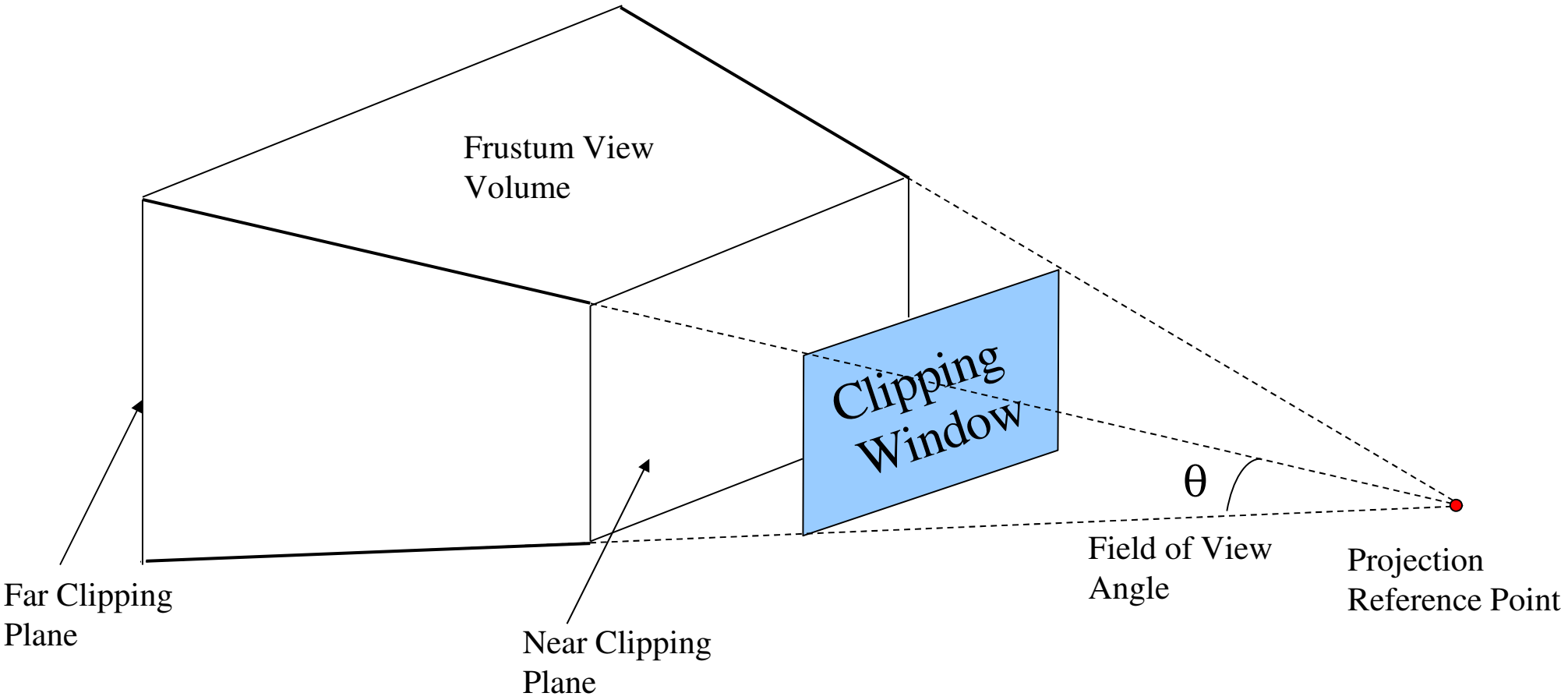
- Two-point perspective
  x and *z* axes intersect
  two vanishing points



- Three-point perspective
  all axes intersect
  three vanishing points



Perspective Projection View Volume



Perspective-Projection Transformation Matrix

- We can use homogeneous coordinates to express the perspective projection equations:

$$x_p = \frac{x_h}{h}, \qquad y_p = \frac{y_h}{h}$$

$$h = z_{prp} - z$$

- We compute homogeneous coordinates in the perspective projection equations

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

Perspective-Projection Transformation Matrix

- We can set up a perspective projection matrix to convert 3D coordinates to homogenous coordinates, then we can divide the homogeneous coordinates by h to obtain the true positions

$$(1) \qquad P_h = M_{pers} \cdot P$$

$$(2) \qquad P_p = \begin{bmatrix} 1/h & 0 & 0 & 0 \\ 0 & 1/h & 0 & 0 \\ 0 & 0 & 1/h & 0 \\ 0 & 0 & 0 & 1/h \end{bmatrix} \cdot P_h$$

Finding $\mathbf{M}_{\mathbf{pers}}$

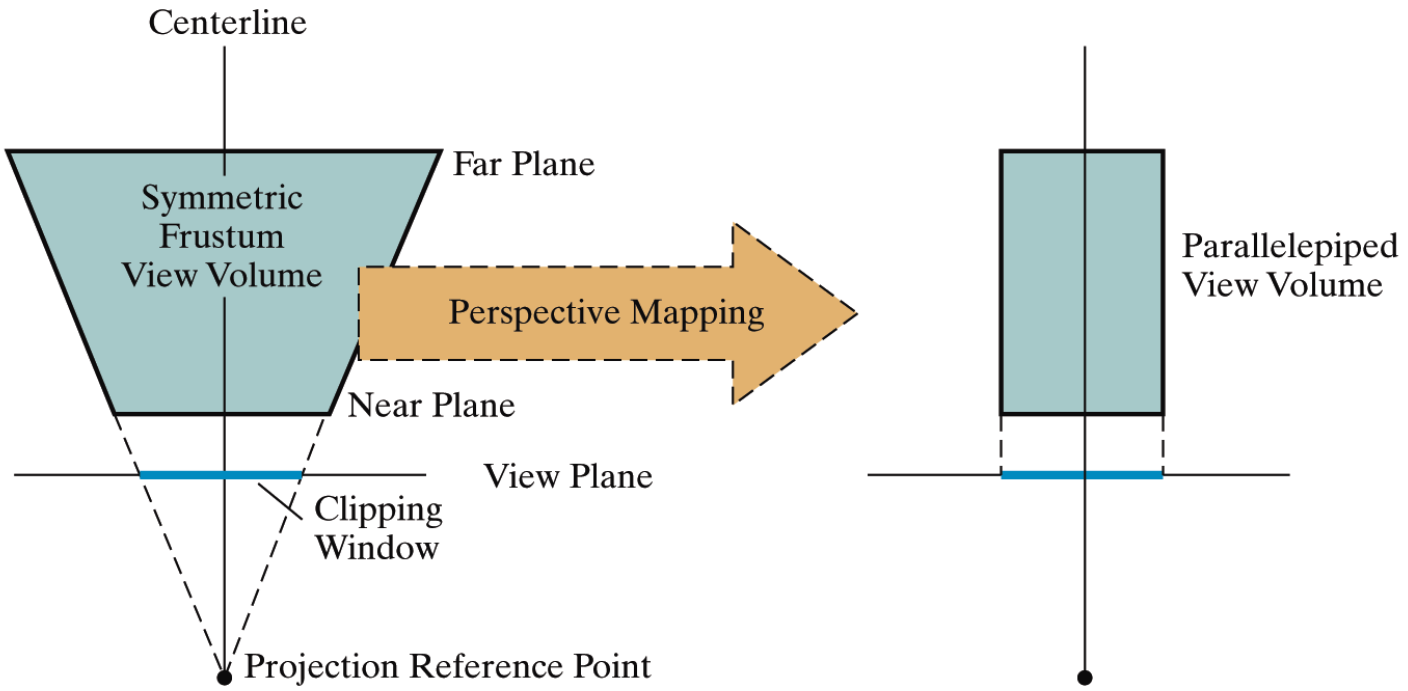
- We need to be careful when finding z_h. We need to preserve the depth information, we cannot simply use the coordinate z_{vp} for all the projected points. We set up the $\mathrm{M}_{\mathrm{pers}}$ matrix so that the z coordinate of a point is converted to a normalized z_p coordinate.

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

s_z and t_z are the scaling and translation parameters for normalizing projected z coordinates.

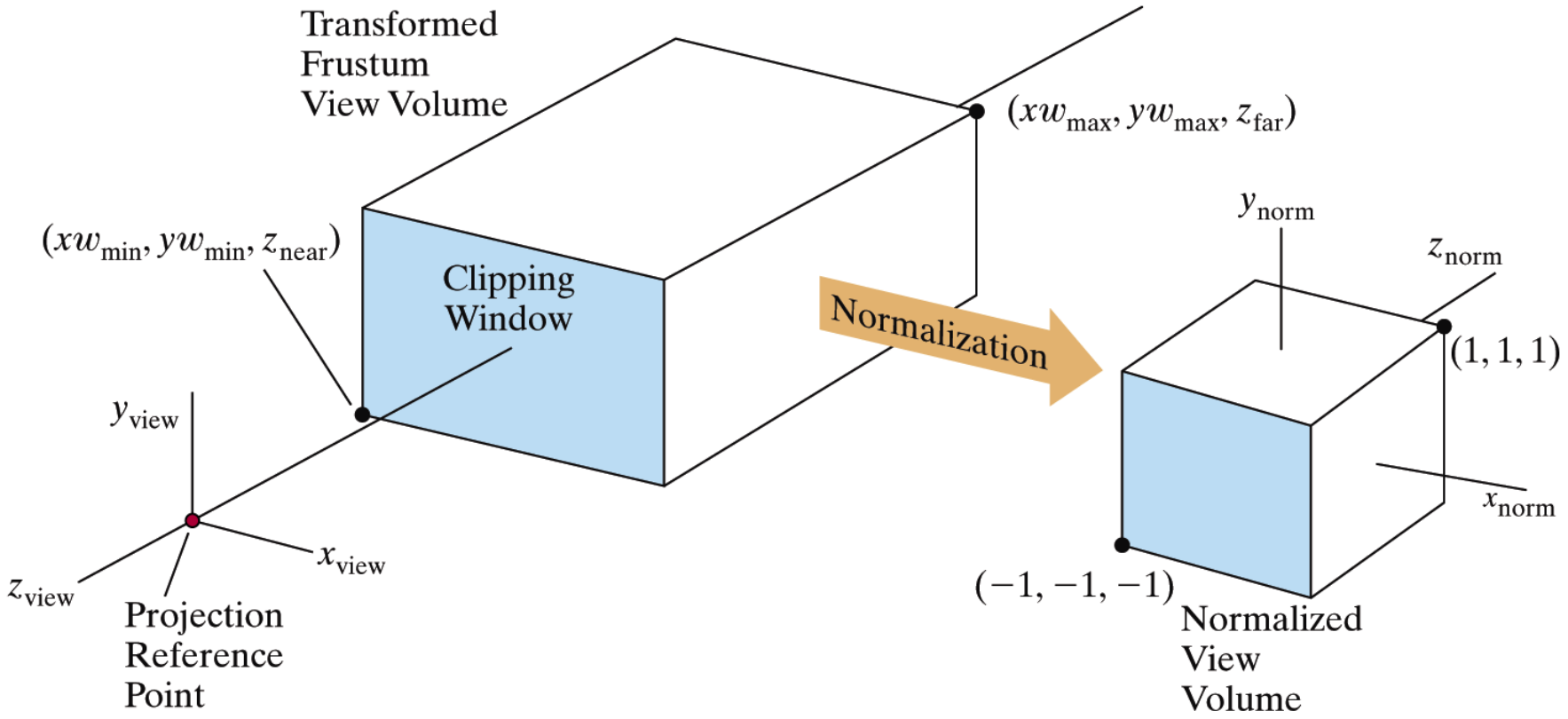
Perspective Transformation

- The symmetric perspective transformation will map the objects into a parallelpiped view volume



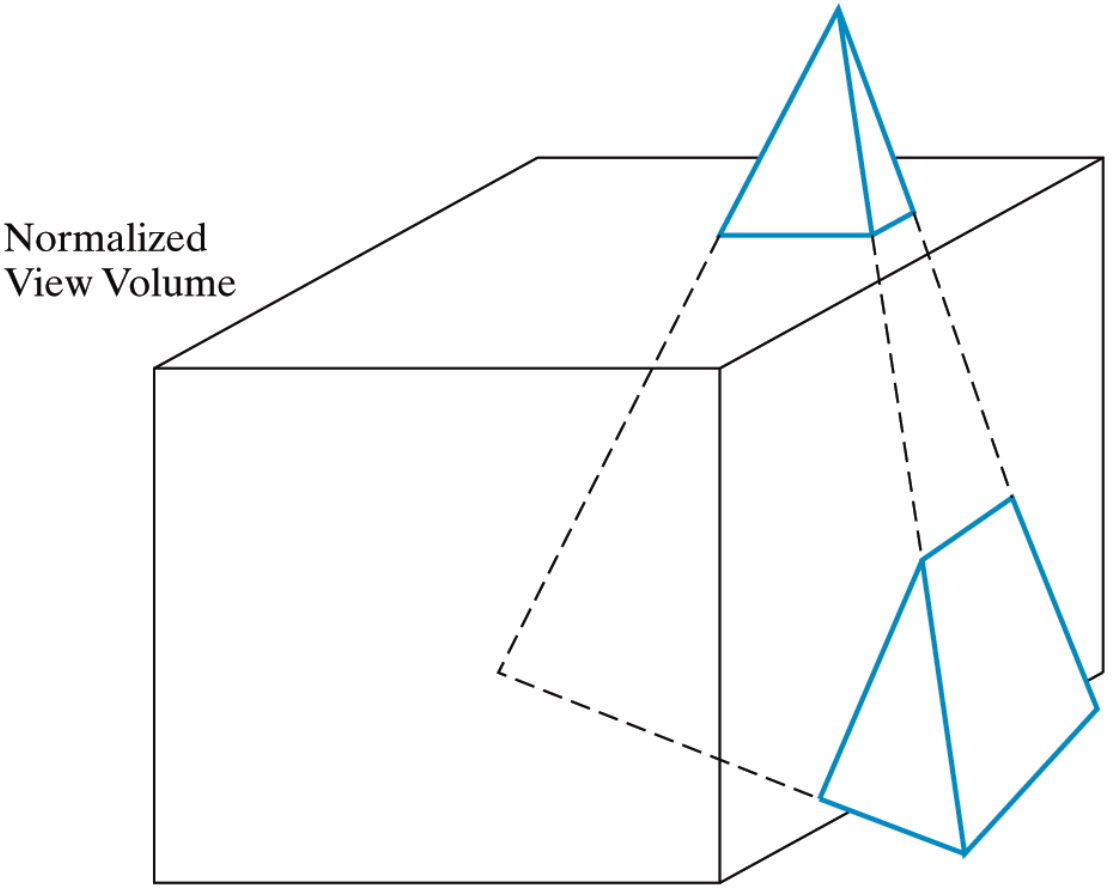
Normalization

- Normalization is then similar to normalization of an orthographic projection



Clipping

- Clipping: Finding parts of the objects in the viewing volume. Algorithms from 2D clipping can easily be applied to 3D and used to clip objects against faces of the normalized view volume.



OpenGL 3D Viewing Functions

- The viewing parameters (camera position, view-up vector, view-plane normal) are specified as part of modeling transformations. A matrix is formed and concatenated with the current modelview matrix. So, to set up camera parameters:

```
glMatrixMode (GL_MODELVIEW);
gluLookAt (x0, y0, z0, xref, yref, zref, Vx, Vy, Vz);
```

$$\mathbf{N} = \mathbf{P}_0 - \mathbf{P}_{ref}$$

Viewing direction is along the $-z_{view}$ axis

gluLookAt

- If we do not invoke the gluLookAt function. The default camera parameters are:
  - $\mathbf{P_0 = (0,0,0)}$
  - $\mathbf{P_{ref} = (0,0,-1)}$
  - $\mathbf{V = (0,1,0)}$

OpenGL Orthogonal-Projection Function

- Projection parameters are set in the OpenGL projection mode.

```
glMatrixMode (GL_PROJECTION);
glOrtho(xwmin,xwmax,ywmin,ywmax,dnear,dfar);
```

- By default we have:

```
glOrtho (-1.0,1.0,-1.0,1.0,-1.0,1.0);
```

OpenGL Symmetric Perspective-Projection Function

- gluPerspective (theta, aspect, dnear, dfar)
- The parameters theta and aspect determines the position and the size of the clipping window on the near plane. *theta* is the field of view angle, i.e., the angle between top and bottom clipping planes. *aspect* is the aspect ratio of the clipping window (i.e., width/height) . *dnear* and *dfar* are assigned positive values with *dnear<dfar.* The actual positions of the near and far planes are z_{near} = -dnear and z_{far} = -dfar

General Perspective-Projection Function

glFrustum (xwmin, xwmax, ywmin, ywmax, dnear, dfar)

Projection reference point is the viewing (camera position) and the view plane normal is the z_{view} axis, viewing direction is $-z_{\text{view}}$.

If xwmin != -xwmax or ywmin!=-ywmax we can obtain an oblique perspective-projection frustum. Otherwise, we will have a symmetric perspective projection.

Additional clipping planes in OpenGL

- In addition to the 6 clipping planes, we may specify new clipping planes and enable clipping of objects with respect to these planes.

  glClipPlane (id, planeparameters);

  glEnable (id); // to enable clipping

  glDisable (id); // to disable clipping

  id is one of GL_CLIP_PLANE0, GL_CLIP_PLANE1,…

  the plane parameters are the for constants, A, B, C, D, in the plane equation Ax+By+Cz+D=0 (any object that satisfies Ax+By+Cz+D<0 is clipped).

3D Dimensional Picking in OpenGL

- In an interactive OpenGL application, you may want to identify the object(s) which are below the clicked mouse position.
- OpenGL API provides a mechanism for this purpose.
- But it is not straightforward.

3D Picking in OpenGL

- Picking involves the following steps:
  - Get the window coordinates of the mouse
  - Enter selection mode
  - Redefine the viewing volume so that only a small area of the window around the cursor is rendered
  - Render the scene, either using all primitives or only those relevant to the picking operation
  - Exit selection mode and identify the objects which were rendered on that small part of the screen.

OpenGL actually uses clipping (in a very small window around the mouse click) to identify clicked objects!

The Name Stack

- In order to identify which objects are clicked, you have to name them. OpenGL provides a Name Stack for this purpose.
  - It is actually a stack of numbers, but those numbers are used as IDs of the objects
- Name Stack functions:
  - void glInitNames(void);
    - This function creates an empty name stack. You are required to call this function to initialize the stack prior to pushing names.

Name Stack Functions

- Name Stack functions:
  - void glPushName(GLuint name);
    - This function pushes an integer ID to stack. Any OpenGL primitives (i.e., vertices that define the geometry of the object) will be associated with the current contents of the name stack
  - void glPopName();
    - Removes the name from top of the stack.
  - void glLoadName(GLunit name);
    - This function replaces the top of the stack with name. Same as popping the stack and then pushing the name.

Name Stack Functions

- Restrictions:
  - Cannot place name stack functions between glBegin() and glEnd()
  - Therefore, if you want to identify components of your objects, you have to write them between separate glBegin()/glEnd()s.
- Name Stack functions are ignored in normal rendering mode. They are only considered in selection rendering mode.

Example

```
#define BODY 1
#define HEAD 2
...
void renderInSelectionMode() {
                glInitNames();

                glPushName(BODY);
                drawBody();
                glPopName();

                glPushName(HEAD);
                drawHead();
                drawEyes();
                glPopName();

                drawGround();
}
```

Remarks

- You can have different rendering functions, i.e., displayFuncNormal(), displayFuncSelection(), for different modes of rendering.
  - The main purpose of such a separation would be for efficiency. For example, if you have a very large scene and you only want to pick some part of the scene. Then in the displayFuncSelection() function you can draw only the primitives related to the objects that you want to pick.

Entering the *selection* mode

```
#define BUFSIZE 512
GLuint selectBuf[BUFSIZE]
...
void startPicking(int cursorX, int cursorY ) {

     GLint viewport[4];

     glSelectBuffer(BUFSIZE,selectBuf);
     glRenderMode(GL_SELECT);

    glMatrixMode(GL_PROJECTION);
    glPushMatrix();
    glLoadIdentity();
    glGetIntegerv(GL_VIEWPORT,viewport);
      gluPickMatrix(cursorX,viewport[3]-cursorY, 5,5,viewport);
      gluPerspective(45,ratio,0.1,1000);
      glMatrixMode(GL_MODELVIEW);
      glInitNames();

    /* Define names and your objects in the scene */
    /* Exit selection mode */
    /* Process the hit objects */

}
```

Processing the hits in the selection buffer

- To end the selection mode:

```
void stopPicking() {
        int hits;

        // restoring the original projection matrix
    glMatrixMode(GL_PROJECTION);
    glPopMatrix();
    glMatrixMode(GL_MODELVIEW);
        glFlush();

        // returning to normal rendering mode
    hits = glRenderMode(GL_RENDER);
        // if there are hits process them
        if (hits != 0) processHits(hits,selectBuf);
    }
```

Format of the hit (i.e., selection) buffer

Number of names for hit 1
Minimum z for hit 1
Maximum z for hit 1
Name 1 of hit 1
…..
Name n of hit 1
Number of names for hit 2
….
….

Example selection buffer

1
4.2822e+009
4.28436e+009
6
0
4.2732e+009
4.27334e+009
3
4.27138e+009
4.27155e+009
….

Processing Selection Buffer

- The object with minimum z in the selection buffer will be what the user selected so you may use the names corresponding to minimum z to identify the object and do further application dependent processing.

Example

>./picking

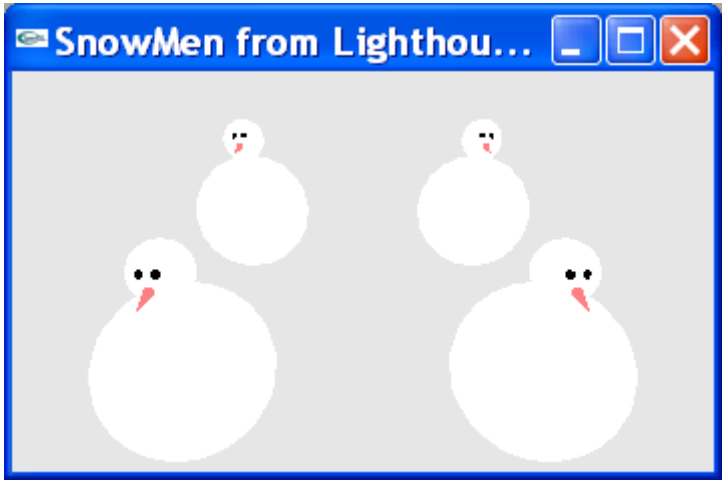
You picked snowman 2

You picked snowman 0

You picked snowman 1

You picked snowman 3

You didn't click a snowman!



Download, try, and examine the source code at:
http://www.ceng.metu.edu.tr/~tcan/misc/picking.c

References

- The textbook: pages 690-696
- OpenGL Lighthouse 3D – Picking Tutorial
  - http://www.lighthouse3d.com/opengl/picking/

Exercise #1

- Given a line segment P_AP_B in the world coordinate system with:
  - P_A= (2,4,1) and P_B= (1,-2,3)

  and the following viewing parameters specified with the gluLookAt function:
  - gluLookAt(1, 2, 3, 0, 2, 3, 0, -1, 0);

  What will be the coordinates of the line when it is transformed into the viewing coordinate system?

Solution steps

- Using the camera parameters, identify the **u, v, n** vectors that define the viewing coordinate system.
- Construct the viewing transformation matrix as a composition of a translation and rotation
- Apply the viewing transformation to the points of the line

Finding u, v, n vectors

$$n = \frac{N}{|N|} = (n_x, n_y, n_z)$$

N = P0-Pref = (1,2,3) - (0,2,3) = (1,0,0) = n

$$u = \frac{V \times n}{|V|} = (u_x, u_y, u_z)$$

u = (0,0,1)

$$v = n \times u = (v_x, v_y, v_z)$$

v = (0,-1,0)

Viewing Transformation Matrix

$$M_{WC,VC} = R \cdot T = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -2 \\ 0 & 0 & 1 & -3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_{WC,VC} = R \cdot T = \begin{bmatrix} 0 & 0 & 1 & -3 \\ 0 & -1 & 0 & 2 \\ 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Apply Viewing Transformation to $P_A P_B$

$$P'_A = M_{WC,VC} \cdot P_A = \begin{bmatrix} 0 & 0 & 1 & -3 \\ 0 & -1 & 0 & 2 \\ 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 4 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -2 \\ -2 \\ 1 \\ 1 \end{bmatrix}$$

$$P'_B = M_{WC,VC} \cdot P_B = \begin{bmatrix} 0 & 0 & 1 & -3 \\ 0 & -1 & 0 & 2 \\ 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ -2 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 4 \\ 0 \\ 1 \end{bmatrix}$$

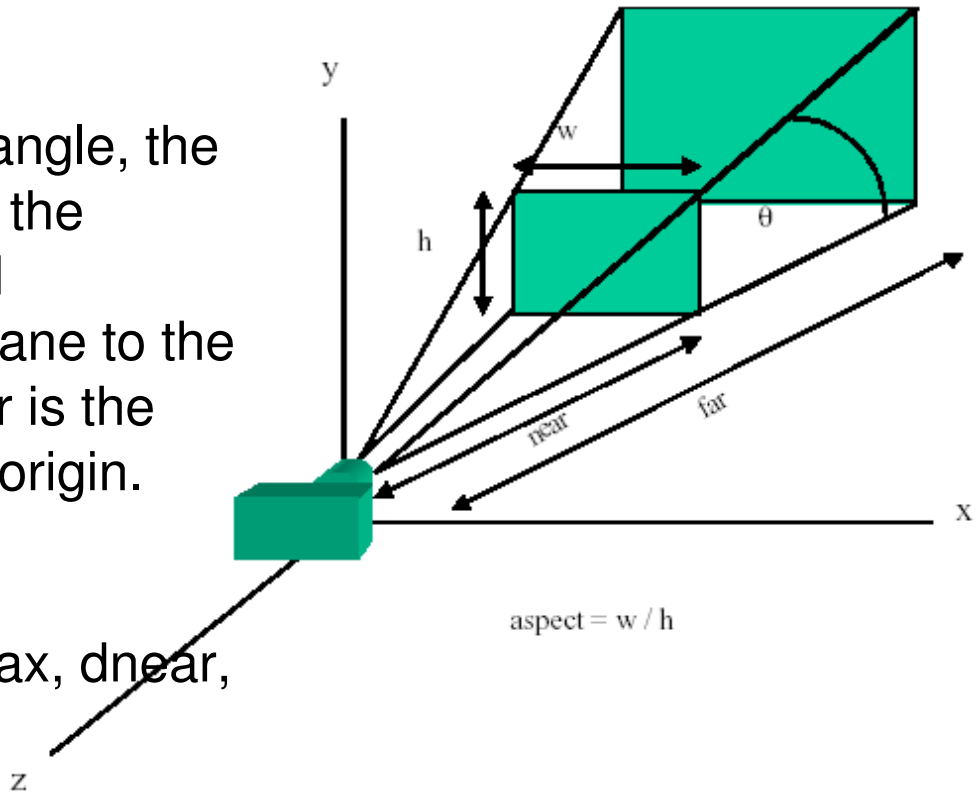
Exercise #2

- Given the following function call to gluPerspective:
  - gluPerspective(60.0,0.8,4.0,100.0);

  where the first parameter is the viewing angle, the second parameter is the aspect ratio of the clipping window (width/height), the third parameter is the distance of the near plane to the viewing origin, and the fourth parameter is the distance of the far plane to the viewing origin.

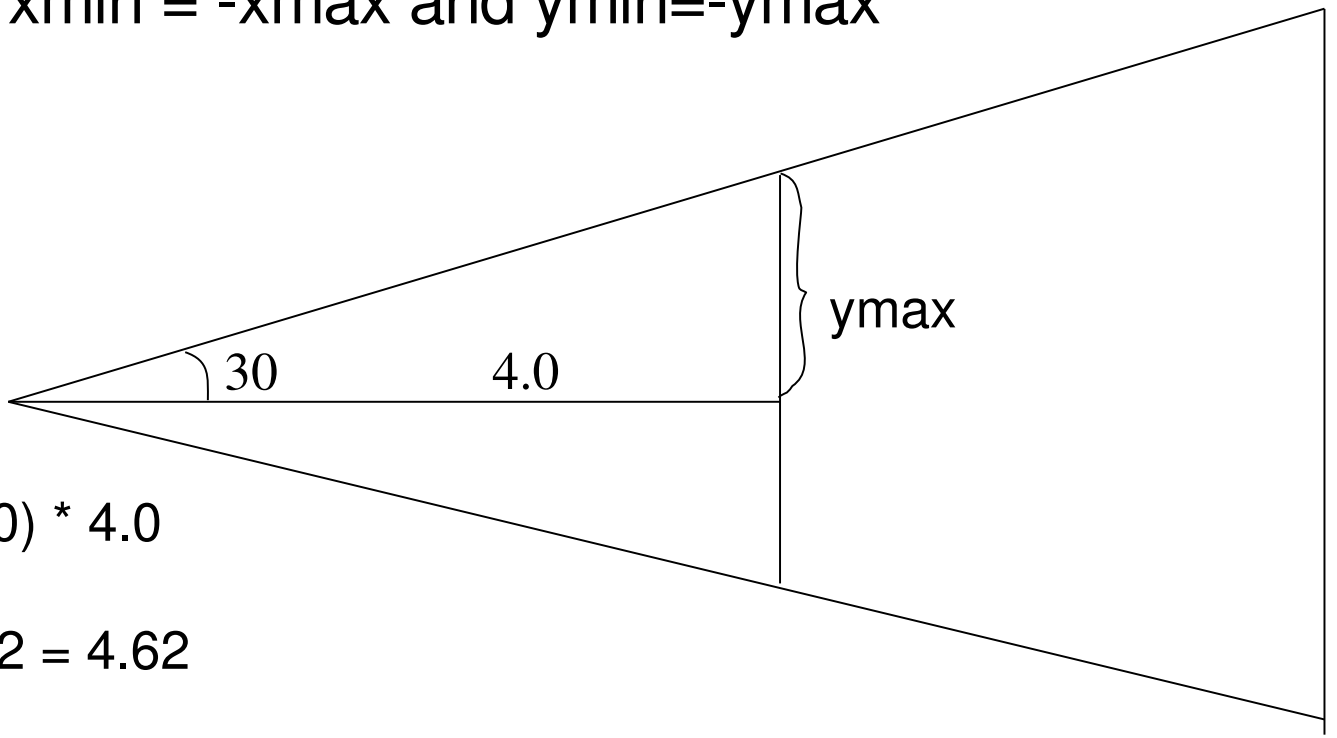
  Find the glFrustum parameters:

  glFrustum (xwmin, xwmax, ywmin, ywmax, dnear, dfar)



Solution

gluPerspective defines a ***symmetric*** perspective projection
Therefore, xmin = -xmax and ymin=-ymax



ymax = tan (30) * 4.0
= 2.31
height = 2.31*2 = 4.62

width = aspect * height
= 0.8 * 4.62
= 3.7

xmax = width/2 = 1.85

glFrustum(-1.85,1.85,-2.31,2.31,4.0,100.0)