MODULE-4

## 3D Viewing and Visible Surface Detection

3D Viewing

Q.What is 3D Viewing? With the help of a block diagram explain 3D viewing pipeline architecture

- When we model a three-dimensional scene, each object in the scene is typically defined with a set of surfaces that form a closed boundary around the object interior.

- In addition to procedures that generate views of the surface features of an object, graphics packages sometimes provide routines for displaying internal components or cross sectional views of a solid object.

- Many processes in three-dimensional viewing, such as the clipping routines, are similar to those in the two-dimensional viewing pipeline.

- But three-dimensional viewing involves some tasks that are not present in two dimensional Viewing

Viewing a Three-Dimensional Scene

- To obtain a display of a three-dimensional world-coordinate scene, we first set up a coordinate reference for the viewing, or "camera," parameters.

- This coordinate reference defines the position and orientation for a view plane (or projection plane) that corresponds to a camera film plane
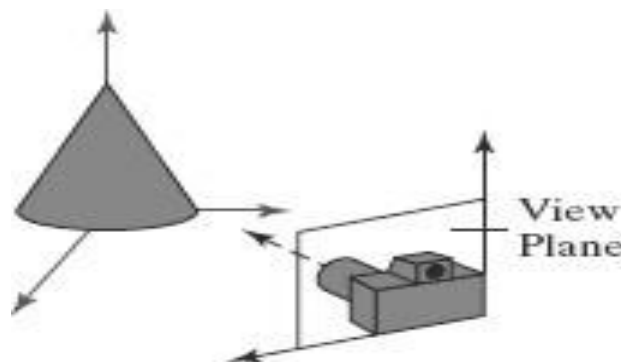
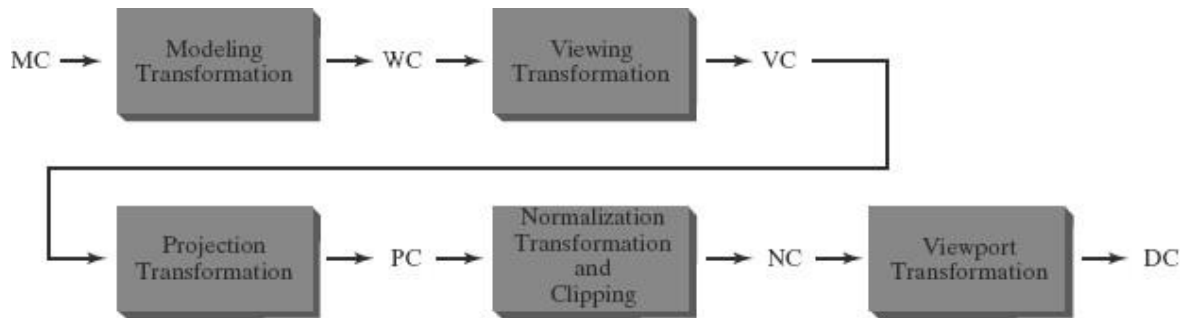## The Three-Dimensional Viewing Pipeline

Figure above shows the general processing steps for creating and transforming a three dimensional scene to device coordinates.

- Construct the shape of individual objects in a scene within modeling coordinate, and place the objects into appropriate positions within the scene (world coordinate).

- World coordinate positions are converted to viewing coordinates.

- Convert the viewing coordinate description of the scene to coordinate positions on the projection plane.

- A two-dimensional clipping window, corresponding to a selected camera lens, is defined on the projection plane, and a three-dimensional clipping region is established. This clipping region is called the view volume.

- Objects are mapped to normalized coordinates, and all parts of the scene outside the view volume are clipped off.

- The clipping operations can be applied after all device-independent coordinate transformations. Then viewport is specified in device coordinates and that normalized coordinates are transferred to viewport coordinates, following the clipping operations.

- The final step is to map viewport coordinates to device coordinates within a selected display window

# Three-Dimensional Viewing-Coordinate Parameters

- Select a world-coordinate position $P_0 = (x_0, y_0, z_0)$ for the viewing origin, which is called the view point or viewing position and we specify a view-up vector V, which defines the y_{view} direction.
- Figure below illustrates the positioning of a three-dimensional viewing-coordinate frame within a world system.
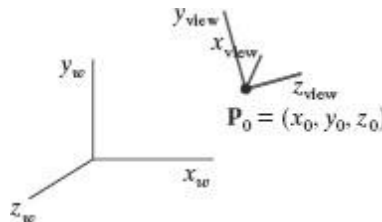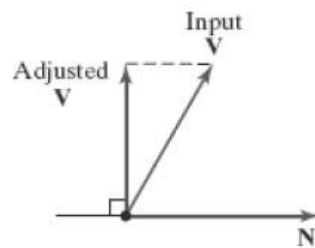
The View-Plane Normal Vector

Because the viewing direction is usually along the z_{view} axis, the view plane, also called the projection plane, is normally assumed to be perpendicular to this axis.

Thus, the orientation of the view plane, as well as the direction for the positive z_{view} axis, can be definedwith a view-plane normal vector N,
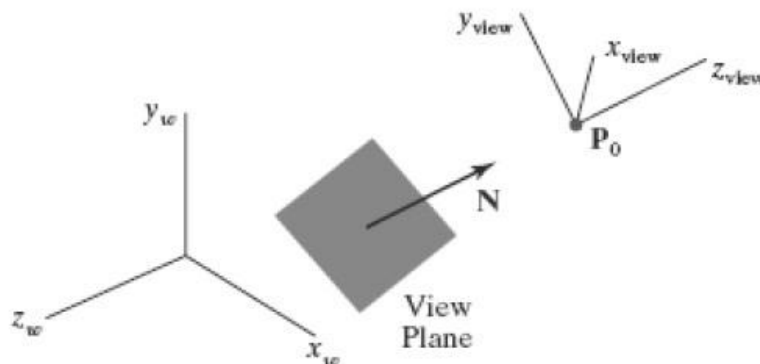
View-Up Vector

- Once we have chosen a view-plane normal vector N, we can set the direction for the view-up vector V.

- This vector is used to establish the positive direction for the y_{view} axis.

- Usually, V is defined by selecting a position relative to the world-coordinate origin, so that the direction for the view-up vector is from the world origin to this selected position

- Because the view-plane normal vector N defines the direction for the z_{view} axis, vector V should be perpendicular to N.

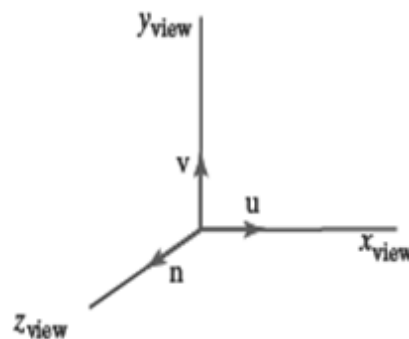- But, in general, it can be difficult to determine a direction for V that is precisely perpendicular to N.

The uvn Viewing-Coordinate Reference Frame

- The coordinate system formed with these unit vectors is often described as a uvn viewing-coordinate reference frame

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

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

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

Transformation from World to Viewing Coordinates

Q.Design the transformation matrix from world to viewing coordinate system with matrix representation

In the three-dimensional viewing pipeline, the first step after a scene has been constructed is to transfer object descriptions to the viewing-coordinate reference frame.

This conversion of object descriptions is equivalent to a sequence of transformations that superimposes the viewing reference frame onto the world frame

(a)   (b)   (c)
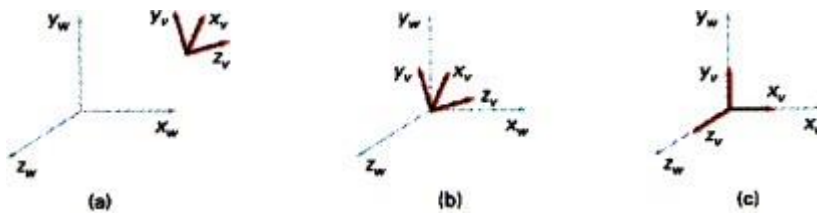
Transformation sequences

1. Translate the view reference point to the origin of the WC system (Figure b)

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

2. Apply rotations to align the x_v, y_v, and z_v axes with the world x_w, y_w, and z_w axes, respectively.

— rotate around the world x_w axis to bring z_v into the $x_w z_w$ plane
— rotate around the world y_w axis to align the z_w and z_v axis
— final rotation is about the z_w axis to align the y_w and y_v axis
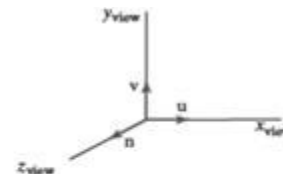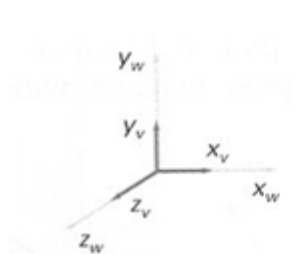
- **Rotation by *uvn* system**
  - Calculate unit *uvn* vectors
    - N : view-plane normal vector
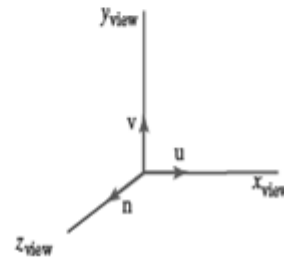   - V : view-up vector
    - U : perpendicular to both N and V

$$\mathbf{n} = \frac{\mathbf{N}}{|\mathbf{N}|} = (n_1, n_2, n_3)$$

$$\mathbf{u} = \frac{\mathbf{V} \times \mathbf{N}}{|\mathbf{V} \times \mathbf{N}|} = (u_1, u_2, u_3)$$

$$\mathbf{v} = \mathbf{n} \times \mathbf{u} = (v_1, v_2, v_3)$$

$$\mathbf{R} = \begin{bmatrix} u_1 & u_2 & u_3 & 0 \\ v_1 & v_2 & v_3 & 0 \\ n_1 & n_2 & n_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The coordinate transformation matrix is then obtained as the product of the preceding translation and rotation matrices:

$$\mathbf{R} = \begin{bmatrix} u_1 & u_2 & u_3 & 0 \\ v_1 & v_2 & v_3 & 0 \\ n_1 & n_2 & n_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{M}_{WC, VC} = \mathbf{R} \cdot \mathbf{T}$$

$$= \begin{bmatrix} u_x & u_y & u_z & -\mathbf{u} \cdot \mathbf{P}_0 \\ v_x & v_y & v_z & -\mathbf{v} \cdot \mathbf{P}_0 \\ n_x & n_y & n_z & -\mathbf{n} \cdot \mathbf{P}_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

These matrix elements are evaluated as

$$-\mathbf{u} \cdot \mathbf{P}_0 = -x_0 u_x - y_0 u_y - z_0 u_z$$
$$-\mathbf{v} \cdot \mathbf{P}_0 = -x_0 v_x - y_0 v_y - z_0 v_z$$
$$-\mathbf{n} \cdot \mathbf{P}_0 = -x_0 n_x - y_0 n_y - z_0 n_z$$

Projection Transformations

Graphics packages generally support both parallel and perspective projections.

Parallel Projection

- Parallel Projection transforms object positions to the view plane along parallel lines.

- A parallel projection preserves relative proportions of objects, and this is the method used in computer aided drafting and design to produce scale drawings of three-dimensional objects.

- All parallel lines in a scene are displayed as parallel when viewed with a parallel projection.
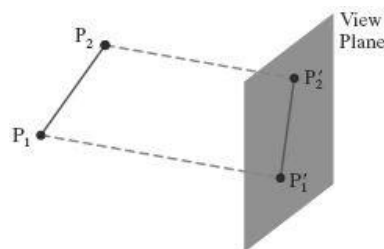
- Parallel Projection Classification: Orthographic Parallel Projection and Oblique Projection
    - Orthographic parallel projections are done by projecting points along parallel lines that are perpendicular to the projection plane.
    - Oblique projections are obtained by projecting along parallel lines that are NOT perpendicular to the projection plane.
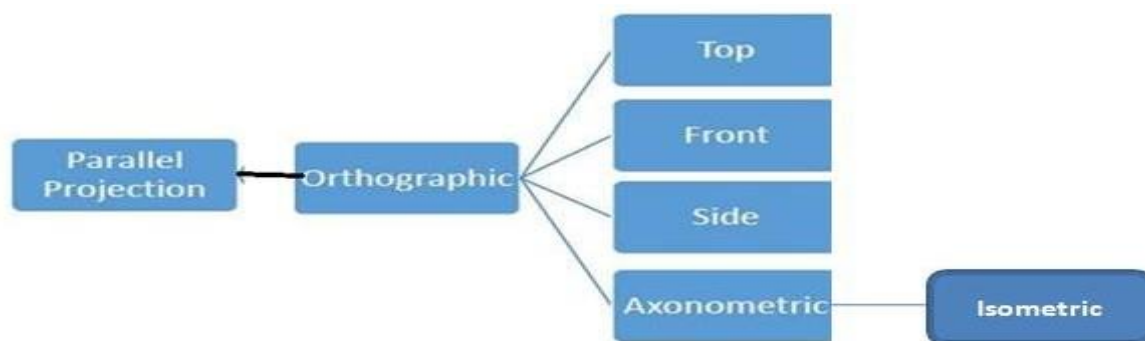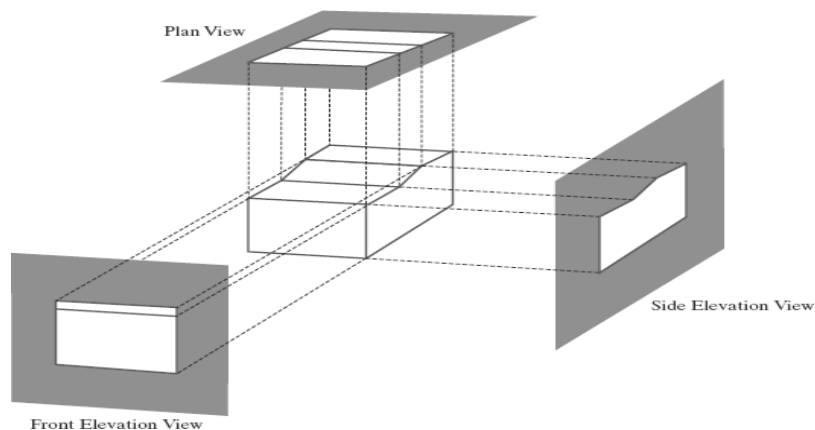
Orthogonal Projections

Q. Explain Orthogonal Projection in details

A transformation of object descriptions to a view plane along lines that are all parallel to the view-plane normal vector N is called an orthogonal projection also termed as orthographic projection

Classification of Orthographic Projection

- Different types of orthographic projections
  - Front Projection
  - Top Projection
  - Side Projection
  - Axonometric Projection – Isometric

- Front, side, and rear orthogonal projections of an object are called elevations
- Top orthogonal projection is called a plan view

---

- Axonometric orthogonal projections
  - We can also form orthogonal projections that display more than one face of an object, such views are called Axonometric projection

  - The most commonly used axonometric projection is the isometric projection, which is generated by aligning the projection plane (or the object) so that the plane intersects each coordinate axis in which the object is defined, called the principal axes, at the same distance from the origin
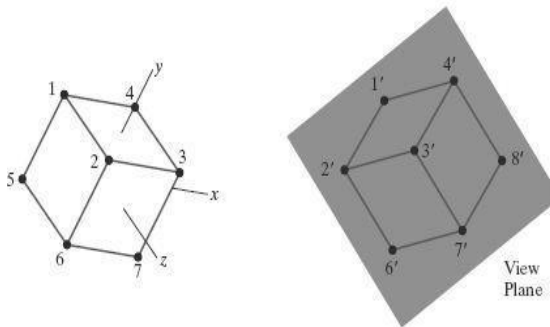
Orthogonal Projection Coordinates

With the projection direction parallel to the z_{view} axis, the transformation equations for an orthogonal projection are trivial. For any position (x, y, z) in viewing coordinates, as in Figure below, the projection coordinates are $x_p = x$, $y_p = y$
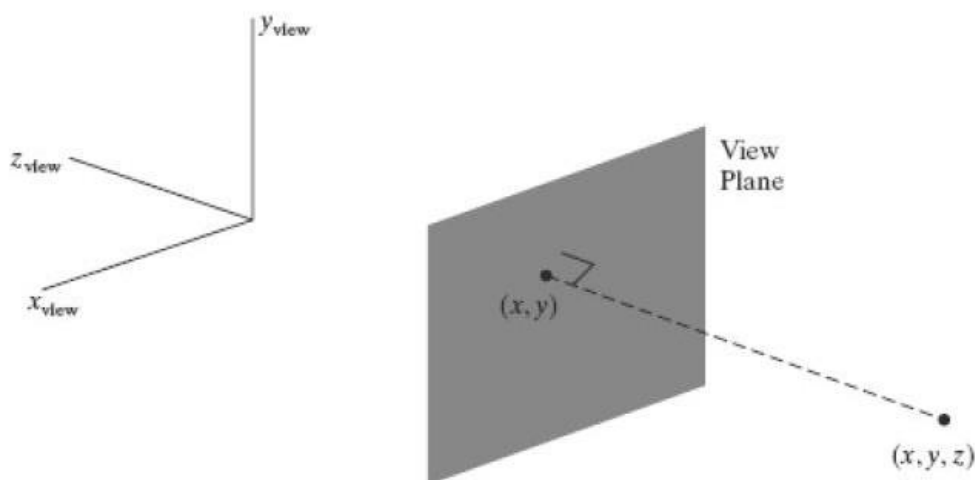
Clipping Window and Orthogonal-Projection View Volume

For three-dimensional viewing, the clipping window is positioned on the view plane with its edges parallel to the x_{view} and y_{view} axes, as shown in Figure below . If we want to use some other shape or orientation for the clipping window, we must develop our own viewing procedures
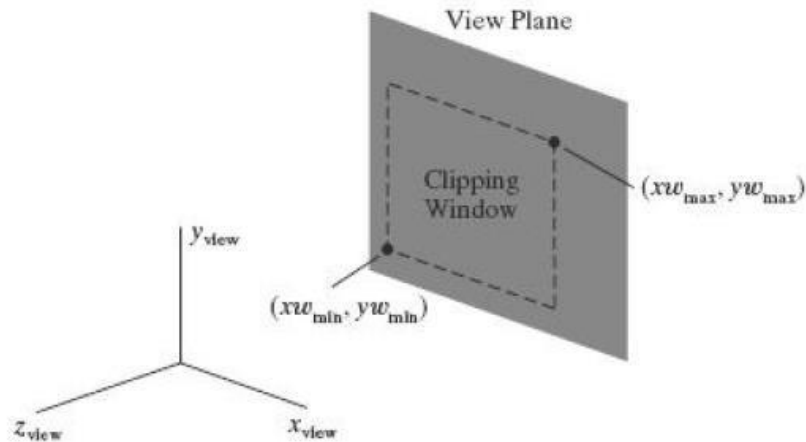
Because projection lines are perpendicular to the view plane, these four boundaries are planes that are also perpendicular to the view plane and that pass through the edges of the clipping window to form an infinite clipping region, as in Figure below.
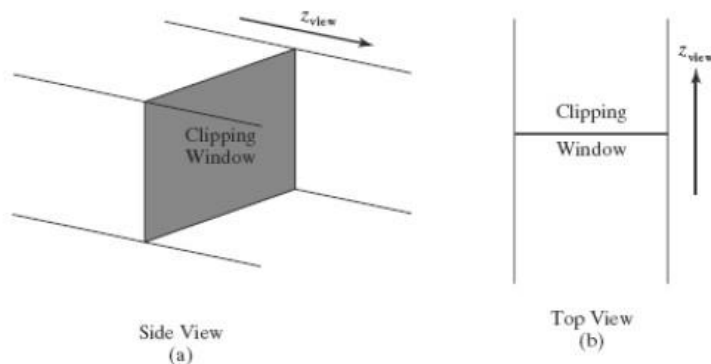
These two planes are called the near-far clipping planes, or the front-back clipping planes.
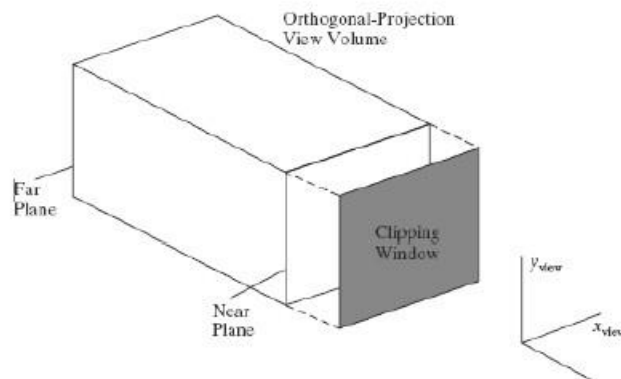
The near and far planes allow us to exclude objects that are in front of or behind the part of the scene that we want to display.

When the near and far planes are specified, we obtain a finite orthogonal view volume that is a *rectangular parallelepiped*, as shown in Figure below along with one possible placement for the view plane

Normalization Transformation for an Orthogonal Projection
- Once we have established the limits for the view volume, coordinate descriptions inside this rectangular parallelepiped are the projection coordinates, and they can be mapped into a normalized view volume without any further projection processing.
- Some graphics packages use a unit cube for this normalized view volume, with each of the x, y, and z coordinates normalized in the range from 0 to 1.
- Another normalization-transformation approach is to use a symmetric cube, with coordinates in the range from −1 to 1
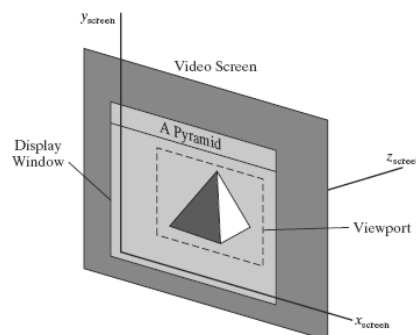
To illustrate the normalization transformation, we assume that the orthogonal-projection view volume is to be mapped into the symmetric normalization cube within a left-handed reference frame.

Also, z-coordinate positions for the near and far planes are denoted as znear and zfar, respectively. Figure below illustrates this normalization transformation
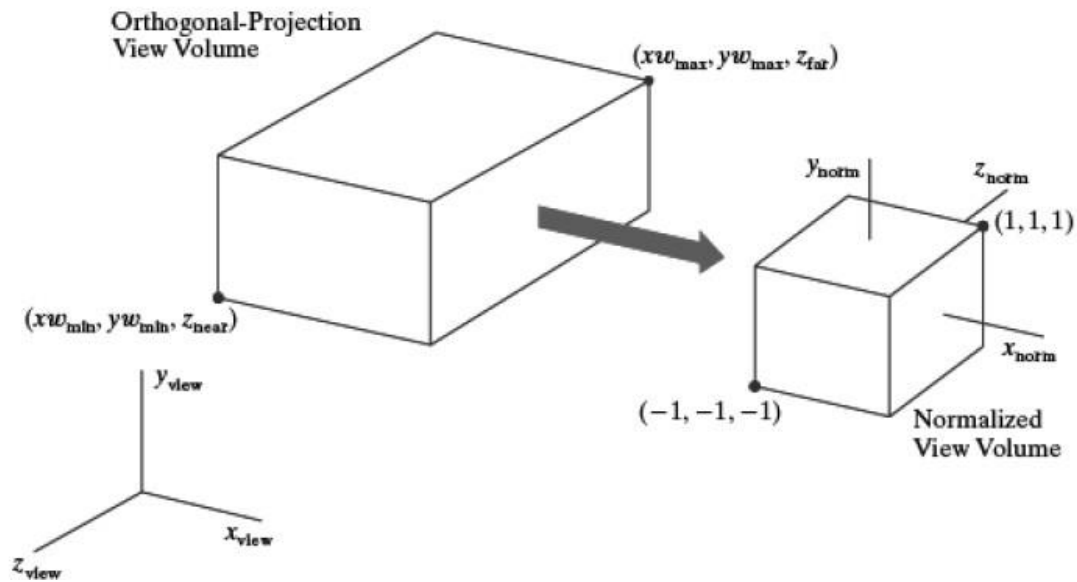
The normalization transformation for the orthogonal view volume is

$$
M_{ortho,norm} = \begin{bmatrix} \dfrac{2}{xw_{max} - xw_{min}} & 0 & 0 & -\dfrac{xw_{max} + xw_{min}}{xw_{max} - xw_{min}} \\[3mm] 0 & \dfrac{2}{yw_{max} - yw_{min}} & 0 & -\dfrac{yw_{max} + yw_{min}}{yw_{max} - yw_{min}} \\[3mm] 0 & 0 & \dfrac{-2}{z_{near} - z_{far}} & \dfrac{z_{near} + z_{far}}{z_{near} - z_{far}} \\[3mm] 0 & 0 & 0 & 1 \end{bmatrix}
$$

Perspective Projection

- Perspective Projection transforms object positions to the view plane while converging to a center point of projection.
- Perspective projection produces realistic views but does not preserve relative proportions.
- Projections of distant objects are smaller than the projections of objects of the same size that are closer to the projection plane.
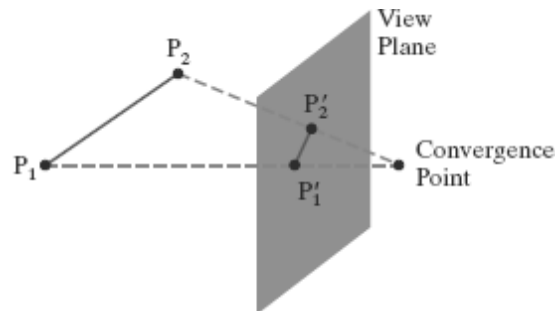
Perspective-Projection Transformation Coordinates

Q. Explain in detail perspective projection transformation coordinates

Figure below shows the projection path of a spatial position (x, y, z) to a general projection reference point at $(x_{prp}, y_{prp}, z_{prp})$.
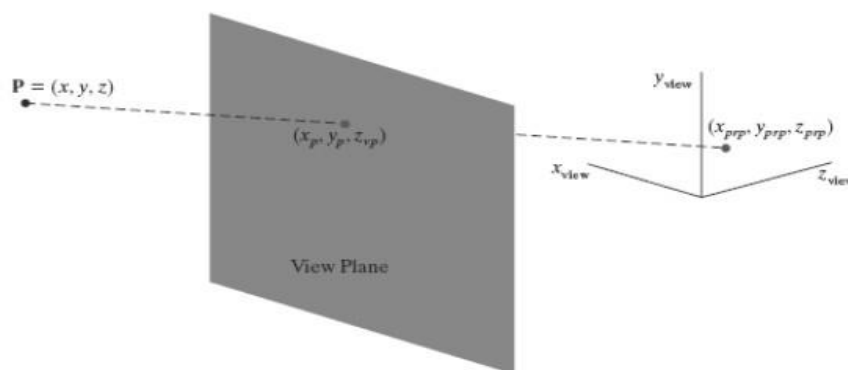
The projection line intersects the view plane at the coordinate position (x_p, y_p, z_{vp}), where z_{vp} is some selected position for the view plane on the z_{view} axis.

We can write equations describing coordinate positions along this perspective-projection line in parametric form as

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

On the view plane, $z' = z_{vp}$ and we can solve the z' equation for parameter u at this position along the projection line:

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

Substituting this value of u into the equations for x' and y', we obtain the general perspective-transformation equations

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

Perspective-Projection Equations: Special Cases

Case 1:

To simplify the perspective calculations, the projection reference point could be limited to positions along the $z$view axis, then

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

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

Case 2:

Sometimes the projection reference point is fixed at the coordinate origin, and

$(x_{prp}, y_{prp}, z_{prp}) = (0, 0, 0)$:

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

Case 3:

If the view plane is the uv plane and there are no restrictions on the placement of the projection reference point, then we have

$z_{vp} = 0$:

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

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

Case 4:

With the uv plane as the view plane and the projection reference point on the z_{view} axis, the perspective equations are

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

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

Vanishing Points for Perspective Projections

Q.Explain the perspective projection with reference and   vanishing point with neat diagram

- The point at which a set of projected parallel lines appears to converge is called a vanishing point.
- Each set of projected parallel lines has a separate vanishing point.

- Based on the number of vanishing points, the perspective projection is of three types
    - One point perspective projection(Figure b)
    - Two point perspective projection(Figure c)

- One point - When the cube is projected to a view plane that intersects only the z axis, a single vanishing point in the z direction

- Two point -When the cube is projected to a view plane that intersects both the z and x axes, two vanishing points are produced.

(a) Principle Axes for Cube

(b) One-Point Perspective Projection

(c) Two-Point Perspective Projection
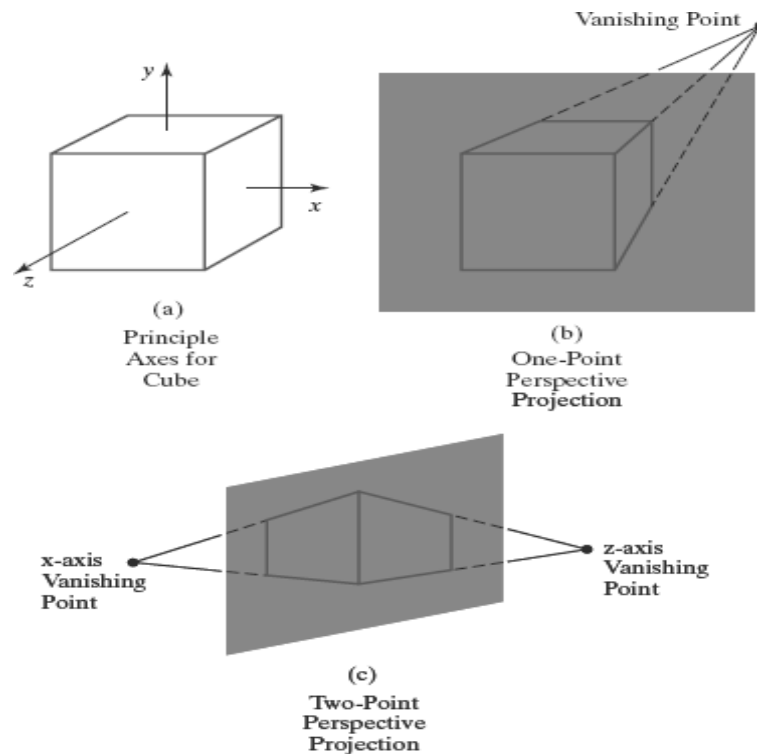
Perspective-Projection Transformation Matrix

Q.Design a Transformation matrix for perspective projection

- Three-dimensional, homogeneous-coordinate representation to express the perspective-projection equations in the form

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

where the homogeneous parameter has the value

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

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

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

- The perspective-projection transformation of a viewing-coordinate position is then accomplished in two steps.

First, calculate the homogeneous coordinates using the perspective-transformation matrix:

$$\mathbf{P}_h = \mathbf{M}_{pers} \cdot \mathbf{P}$$

Where , P_h is the column-matrix representation of the homogeneous point (x_h, y_h, z_h, h)

P is the column-matrix representation of the coordinate position (x, y, z, 1).

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

Second after other processes have been applied, such as the normalization transformation and clipping routines, homogeneous coordinates are divided by parameter *h* to obtain the true transformation-coordinate positions.

Symmetric Perspective-Projection Frustum

Q.Explain in detail symmetric perspective projection Frustum

- The line from the projection reference point through the center of the clipping window and on through the view volume is the centerline for a perspective projection frustum.

- If this centerline is perpendicular to the view plane, we have a symmetric frustum (with respect to its centerline)
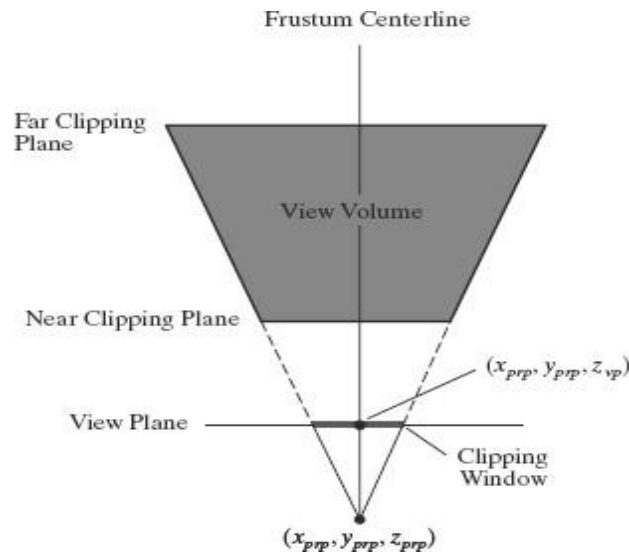
Because the frustum centerline intersects the view plane at the coordinate location $(x_{prp}, y_{prp}, z_{vp})$, we can express the corner positions for the clipping window in terms of the window dimensions:

$$xw_{min} = x_{prp} - \frac{width}{2}, \qquad xw_{max} = x_{prp} + \frac{width}{2}$$

$$yw_{min} = y_{prp} - \frac{height}{2}, \qquad yw_{max} = y_{prp} + \frac{height}{2}$$

- Another way to specify a symmetric perspective projection is to use parameters that approximate the properties of a camera lens.

  In computer graphics, the cone of vision is approximated with a symmetric frustum, and we can use a field-of-view angle to specify an angular size for the frustum.
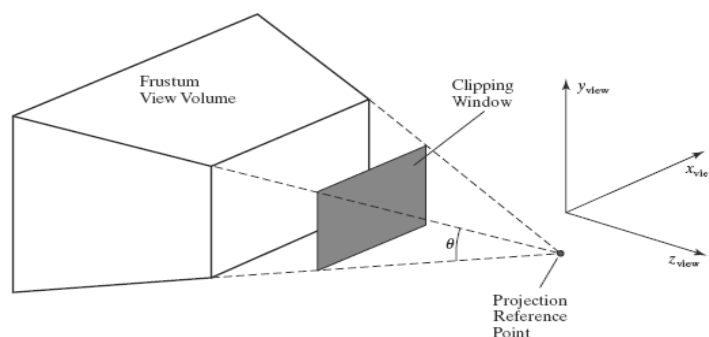
For a given projection reference point and view-plane position, the field-of view angle determines the height of the clipping window from the right triangles in the diagram of Figure below,
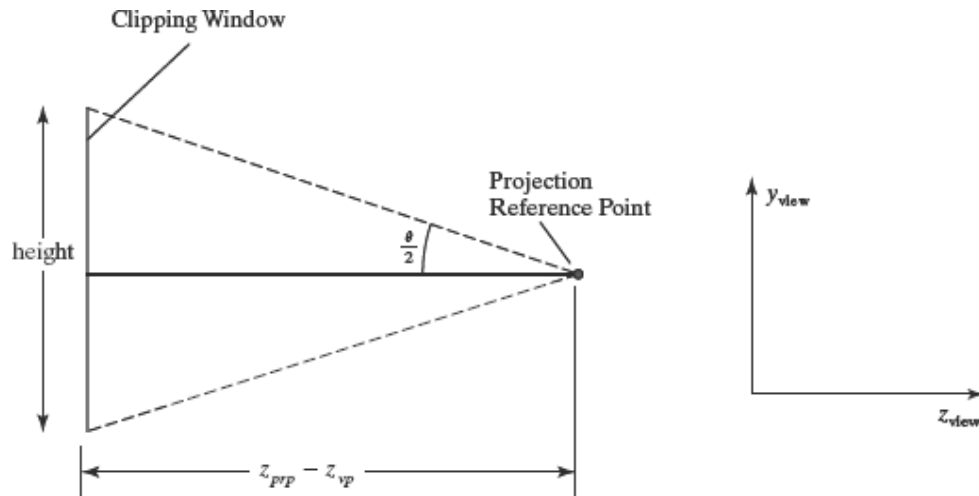
clipping-window height can be calculated as

$$\text{height} = 2(z_{prp} - z_{vp}) \tan\left(\frac{\theta}{2}\right)$$

Therefore, the diagonal elements with the value $z_{prp} - z_{vp}$ could be replaced by either of the following two expressions

$$z_{prp} - z_{vp} = \frac{\text{height}}{2} \cot\left(\frac{\theta}{2}\right)$$
$$= \frac{\text{width} \cdot \cot(\theta/2)}{2 \cdot \text{aspect}}$$

Oblique Perspective-Projection Frustum

Q.Explain in detail oblique perspective projection Frustum

- If the centerline of a perspective-projection view volume is not perpendicular to the view plane, we have an oblique frustum
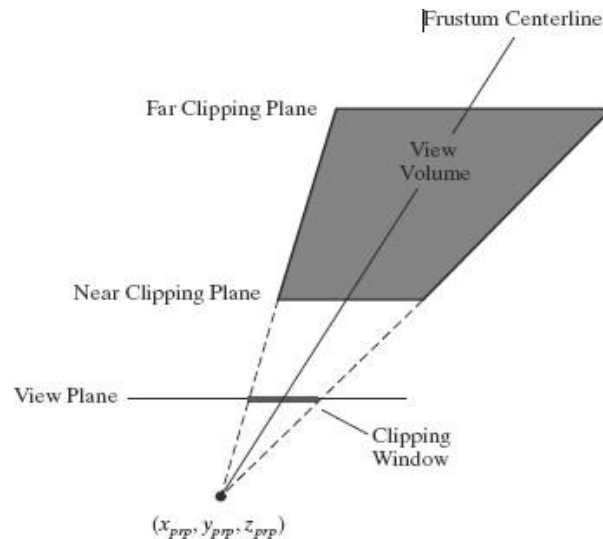
In this case, first transform the view volume to a symmetric frustum and then to a normalized view volume.

- An oblique perspective-projection view volume can be converted to a symmetric frustum by applying a *z*-axis shearing-transformation matrix.
- Taking the projection reference point as *(xprp, yprp, zprp) = (0, 0, 0)*, we obtain the elements of the required shearing matrix as

$$M_{z\,shear} = \begin{bmatrix} 1 & 0 & sh_{zx} & 0 \\ 0 & 1 & sh_{zy} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad sh_{zx} = -\frac{xw_{min} + xw_{max}}{2\,z_{near}}$$

$$sh_{zy} = -\frac{yw_{min} + yw_{max}}{2\,z_{near}}$$

- Similarly, with the projection reference point at the viewing-coordinate origin and with the near clipping plane as the view plane, the perspective-projection matrix is simplified to

$$\mathbf{M}_{pers} = \begin{bmatrix} -z_{near} & 0 & 0 & 0 \\ 0 & -z_{near} & 0 & 0 \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

- Concatenating the simplified perspective-projection matrix with the shear matrix we have

$$\mathbf{M}_{obliquepers} = \mathbf{M}_{pers} \cdot \mathbf{M}_{z\,shear}$$

$$= \begin{bmatrix} -z_{near} & 0 & \dfrac{xw_{min} + xw_{max}}{2} & 0 \\ 0 & -z_{near} & \dfrac{yw_{min} + yw_{max}}{2} & 0 \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Normalized Perspective-Projection Transformation Coordinates

- The final step in the perspective transformation process is to map this parallelepiped to a normalized view volume.
- The transformed frustum view volume, which is a rectangular parallelepiped, is mapped to a symmetric normalized cube within a left-handed reference frame
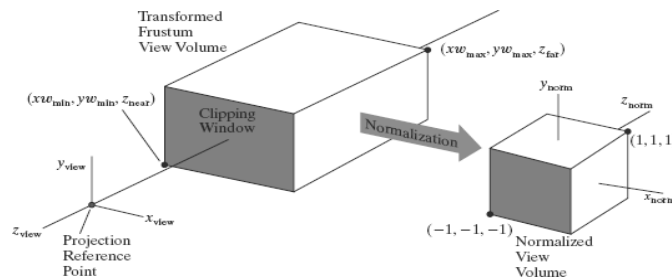
Because the centerline of the rectangular parallelepiped view volume is now the z_{view} axis, no translation is needed in the x and y normalization transformations: We require only the x and y scaling parameters relative to the coordinate origin.

The scaling matrix for accomplishing the xy normalization is

$$
M_{xy\,scale} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Concatenating the xy-scalingmatrix produces the following normalization matrix for a perspective-projection transformation.

$$
M_{normpers} = M_{xy\,scale} \cdot M_{obliquepers}
$$

$$
\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} -z_{near} & 0 & \frac{xw_{min}+xw_{max}}{2} & 0 \\ 0 & -z_{near} & \frac{yw_{min}+yw_{max}}{2} & 0 \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & 0 \end{bmatrix}
$$

$$
M_{normpers} = \begin{bmatrix} -z_{near}s_x & 0 & s_x\frac{xw_{min}+xw_{max}}{2} & 0 \\ 0 & -z_{near}s_y & s_y\frac{yw_{min}+yw_{max}}{2} & 0 \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & 0 \end{bmatrix}
$$

The Viewport Transformation and Three-Dimensional ScreenCoordinates

- Once we have completed the transformation to normalized projection coordinates, clipping can be applied efficiently to the symmetric cube then the contents of the normalized view volume can be transferred to screen coordinates.
- Positions throughout the three-dimensional view volume also have a depth (z coordinate), and we need to retain this depth information for the visibility testing and surface rendering algorithms

If we include this z renormalization, the transformation from the normalized view volume to three dimensional screen coordinates is

$$
M_{\text{normviewvol,3D screen}} = \begin{bmatrix} \dfrac{xv_{\max} - xv_{\min}}{2} & 0 & 0 & \dfrac{xv_{\max} + xv_{\min}}{2} \\ 0 & \dfrac{yv_{\max} - yv_{\min}}{2} & 0 & \dfrac{yv_{\max} + yv_{\min}}{2} \\ 0 & 0 & \dfrac{1}{2} & \dfrac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

In normalized coordinates, the $z_{\text{norm}} = -1$ face of the symmetric cube corresponds to the clipping-window area. And this face of the normalized cube is mapped to the rectangular viewport, which is now referenced at $z$screen = 0.

Thus, the lower-left corner of the viewport screen area is at position $(xv_{\min}, yv_{\min}, 0)$ and the upper-right corner is at position $(xv_{\max}, yv_{\max}, 0)$.

OpenGL Three-Dimensional Viewing Functions

Q. Explain OpenGL 3D Viewing functions

- **OpenGL Viewing-Transformation Function**
- **OpenGL Orthogonal-Projection Function**
- **OpenGL General Perspective-Projection Function**
- **OpenGL Viewports and Display Windows**

OpenGL Viewing-Transformation Function

glMatrixMode (GL_MODELVIEW);

➔ a matrix is formed and concatenated with the current modelview matrix, We set the modelview mode with the statement above

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

Viewing parameters are specified with the above GLU function.

This function designates the origin of the viewing reference frame as the world-coordinate position $P_0 = (x_0, y_0, z_0)$, the reference position as $P_{ref} = (x_{ref}, y_{ref}, z_{ref})$, and the view-up vector as $V = (V_x, V_y, V_z)$.

If we do not invoke the gluLookAt function, the default OpenGL viewing parameters are

$$P_0 = (0, 0, 0)$$
$$P_{ref} = (0, 0, -1)$$
$$V = (0, 1, 0)$$

OpenGL Orthogonal-Projection Function

glMatrixMode (GL_PROJECTION);

set up a projection-transformation matrix.

Then, when we issue any transformation command, the resulting matrix will be concatenated with the current projection matrix.

glOrtho (xwmin, xwmax, ywmin, ywmax, dnear, dfar);

Orthogonal-projection parameters are chosen with the function

All parameter values in this function are to be assigned double-precision, floating point Numbers

Function glOrtho generates a parallel projection that is perpendicular to the view plane

Parameters d_{near} and d_{far} denote distances in the negative z_{view} direction from the viewing-coordinate origin
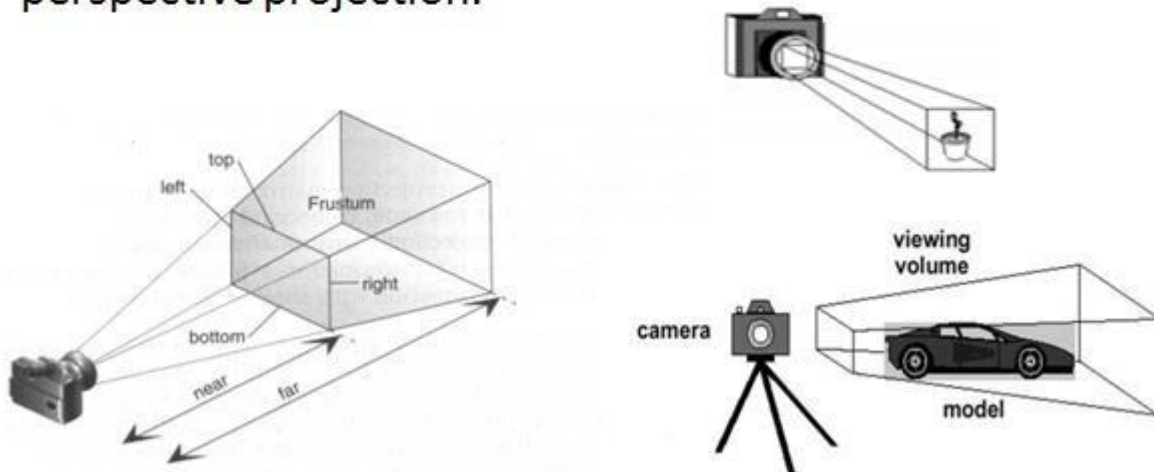
We can assign any values (positive, negative, or zero) to these parameters, so long as $d_{near} < d_{far}$.

Exa: glOrtho (-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
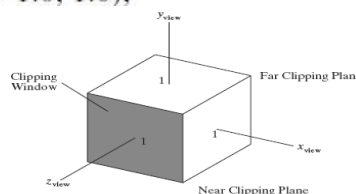
OpenGL General Perspective-Projection Function

glFrustum (xwmin, xwmax, ywmin, ywmax, dnear, dfar);
glFrustum describes a perspective matrix that produces a perspective projection.

OpenGL Viewports and Display Windows

glViewport (xvmin, yvmin, vpWidth, vpHeight);

A rectangular viewport is defined.

The first two parameters in this function specify the integer screen position of the lower-left corner of the viewport relative to the lower-left corner of the display window.

And the last two parameters give the integer width and height of the viewport.

To maintain the proportions of objects in a scene, we set the aspect ratio of the viewport equal to the aspect ratio of the clipping window.

Display windows are created and managed with GLUT routines. The default viewport in OpenGL is the size and position of the current display window

Visible Surface Detection Methods

- Classification of visible surface Detection algorithms,
- back face detection
- Depth buffer method
- OpenGL visibility detection functions.

Classification of Visible Surface Detection Algorithm

Q Give the general Classification of visible detection algorithm and explain any one algorithm in detail (6M)

Visible Surface Detection algorithm are classified into
- Object Space Method
- Image Space Method

Object Space Method
- It compares objects and parts of object to each other within the scene definition to determine which surface is visible.
- Line display algorithms use this method to identify visible line in wire frame display.

Image Space Method
- Here visibility is decided point by Point at each pixel position on the Projection Plane.
- Visible surface algorithm use this method for Visible line detection.

Back-Face Detection

- This method is used to remove back face of any object.
- This method is like a preprocessor that removes back face in all methods
- A fast and simple object-space method for identifying the back faces of a object is based on the "inside-outside" tests.

First Approach INSIDE and OUTSIDE TEST
- A point (x, y, z) is "inside" a polygon surface with plane parameters A, B, C, and D, if When an inside point is along the line of sight to the surface, the polygon must be a back face

- If the polygon surface equation:
$$Ax + By + Cz + D < 0$$

- Then the point is inside polygon surface, So it is back of the front face, can be eliminated

Second approach
- Consider the normal vector N to a polygon surface, which has Cartesian components (A, B, C).
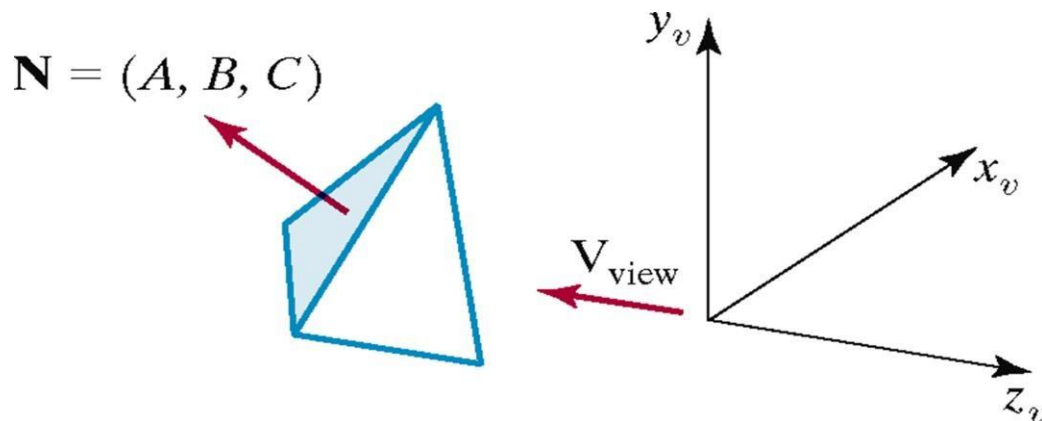
- V_{view} is a vector in the viewing direction from the eye ("camera") position, a polygon surface is a back face if:

$$V_{view} \cdot N > 0$$

$$V_{view} = (0,0, V_z) \text{ and } N = Ax + By + Cz$$

$$V_{view} \cdot N = (0,0, V_z)(Ax + By + Cz)$$

$$= V_z C \ (\text{If } V_z = 1)$$

$$V_{view} \cdot N \quad = C$$

$$Sign(C) \geq 0 \rightarrow \text{Surface is back(invisible)}$$
$$Sign(C) < 0 \rightarrow \text{Surface if front (visible)}$$

Depth Buffer Algorithm (Z Buffer Method)

Q. Write and explain Depth Buffer algorithm(8m)

- Depth buffer method uses image-space approach for detecting visible surfaces.

- It compares surface depth values throughout a scene for each pixel position on the projection plane.

- Each surface of a scene is processed separately, one pixel position at a time, across the surface.

- Consider three surfaces at varying distances along the orthographic projection line from position (x, y) on a view plane.
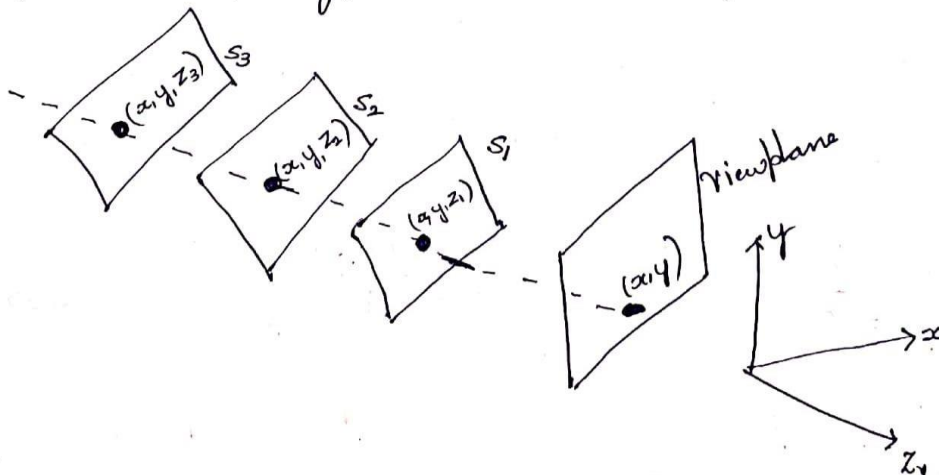
- Basically 2 buffers are used
  1) Depth Buffer store Z values (Depth values)
  2) Refresh Buffer store Surface intensity value
     (Frame Buffer)

- Consider 2 cases

Case 1
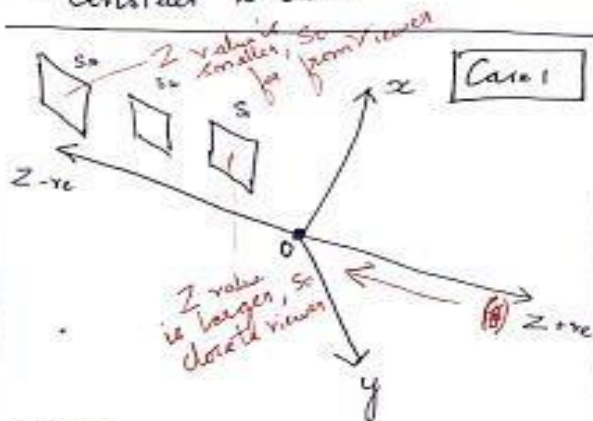
Viewer
Sitting at Z +ve & looking at
Origin (ie Viewing along Z -ve)

↳ Surface with
  ↳ Larger Z value is closer to viewer
  ↳ Smaller Z value is farther to viewer

Depth buffer will be initialised
with $\boxed{d(x,y) = 0}$

$Z > depthBuff(x,y)$

Case 2

Viewer Sitting at Z -ve &
looking at Origin (viewing Z +ve)

↳ Surface with
  ↳ Smaller Z value is closer
  ↳ Larger Z value is far.

Depth buffer will be intialized
with $\boxed{d(x,y) = Z_{max}}$

$Z < depthBuff(x,y)$

Algorithm steps

1) Initialize the Depth buffer and frame buffer so that for all buffer position (x, y)

$$\text{depth Buff}(x, y) = 1.0$$
$$\text{frame Buff}(x, y) = I_{background}$$

2) Process each polygon in a scene, one at a time, as follows:

- For each projected (x, y) pixel position of a polygon, calculate the depth z

- If $z < \text{depth Buff}(x, y)$
  Compute the surface color at that position and set

$$\text{dept Buff}(x, y) = z$$
$$\text{frame Buff}(x, y) = \text{Surf Color}(x, y)$$

After all Surfaces have been processed, the depth buffer contains depth value for Visible Surface & Frame buffer contains color value for those Surface.

Calculate **Z** (depth at any point on plane containing Polygon)
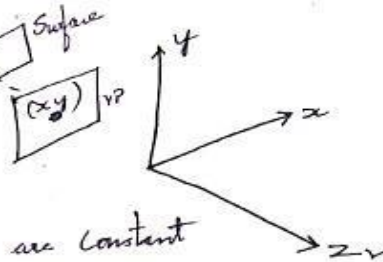
• Plane Equation is $\boxed{Ax + By + Cz + D = 0}$

• At surface position (x,y), the depth is calculated by

$$\boxed{Z = \dfrac{-Ax - By - D}{C}}$$

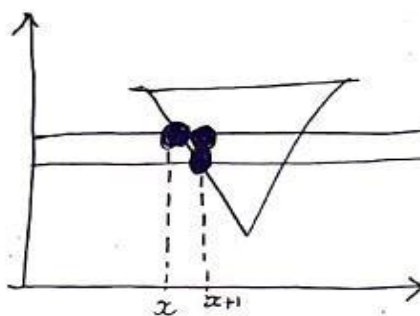$\longrightarrow \textcircled{1}$

Z = ?

Surface

(x,y) VP

→ where A, B, C, D are Constant

Pixel (x,y)

| For Every Pixel (x,y) if we apply this method to calculate Z, it is Time Consuming |

To Solve this problem, another approach is Scanline

If Z value at (x,y) is found then we can find Z' value at $(x+1,y)$
⌐ Z' value at $(x+1, y-1)$

y
y-1

x   x+1

To Find z' value at $(x+1, y)$

We Know $\quad Z = \dfrac{-Ax - By - D}{C}$

$$z' = \dfrac{-A(x+1) - By - D}{C}$$

$$= \dfrac{-Ax - By - D - A}{C}$$

$$\boxed{z' = Z - \dfrac{A}{C}} \longrightarrow ②$$

where $-A/c$ is constant

So succeeding depth value across a scanline are obtained from preceding values with a single addition
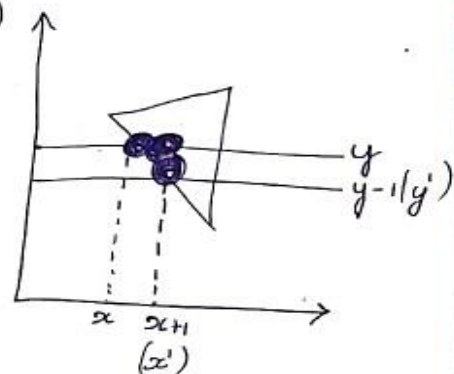
Now to Find z' value at $(x+1, y-1)$

$$m = \dfrac{y - y'}{x - x'}$$

$$= \dfrac{y - (y-1)}{x - x'} \implies \dfrac{y - y + 1}{x - x'}$$

$$m = \dfrac{1}{x - x'} \implies x - x' = \dfrac{1}{m}$$

$$\boxed{x - \dfrac{1}{m} = x'}$$

(x,y) $(x' y')$

We Know $\quad Z = \dfrac{-Ax - By - D}{C} \Rightarrow \dfrac{-Ax' - By' - D}{C}$

$$Z' = \dfrac{-A\left(x - \dfrac{1}{m}\right) - B(y-1) + 0 - D}{C}$$

$$= \dfrac{-Ax + \dfrac{A}{m} - By + B \, 0 - D}{C}$$

$$= \dfrac{-Ax - By - D}{C} + \dfrac{A/m + B}{C}$$

$$\boxed{Z' = Z + \dfrac{A/m + B}{C}}$$

If we are processing down a vertical edge,
Slope m is infinite, So

$$\boxed{Z' = Z + \dfrac{B}{C}} \longrightarrow ③ \qquad \dfrac{A}{m} \text{ is infinite}$$

Equation ①, ② & ③ can be used to Calculate Z

— ✗ —

Drawback :
 - Its time consuming process
 - 2 Buffers are need to be maintained , So Costly
 - Requires large Memory

Advantage

 - It is efficient
 - Easy to Implement

OpenGL Visibility Detection Function

Q Explain OpenGL Visibility Detection Function (8M)

OpenGL Polygon - Culling Functions

glCullFace (mode);

- This function is used to remove Back face, Front face or both front and back faces of object.

- Parameter mode → assigned value OpenGL symbolic Constant
  $\begin{cases} GL\_BACK \\ GL\_FRONT \\ GL\_FRONT\_AND\_BACK \end{cases}$

glEnable (GL_CULL_FACE); → Activate Culling function
glDisable (GL_CULL_FACE); → Deactivate Culling function

OpenGL Depth - Buffer Functions

glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH)

- This initialization function will request for depth buffer and refresh buffer.

glClear (GL_DEPTH_BUFFER_BIT); → Depth buffer values are initialized to max Value 1.0 by default

→ It can be Normalized in range 0.0 to 1.0

gl Enable (GL_DEPTH_TEST) ⟶ activate depth buffer visibility
detection routine
gl Disable (GL_DEPTH_TEST) ⟶ Deactivate depth buffer visibility
detection routine

gl ClearDepth (max Depth) ⟶ Depth buffer value can
be chosen b/w
gl Clear (GL_DEPTH_BUFFER_BIT) 0.0 & 1.0
⟶ load the new value to
Depth buffer

glDepth Range (nearNormDepth , farNormDepth)

This function will Normalize depth buffer with
parameter nearNormDepth = 0.0 &
far NormDepth = 1.0

OpenGL WireFrame Surface - Visibility Method

glPolygonMode (GL_FRONT_AND_BACK , GL_LINE);

This function is used for wire frame display
of object., But display both visible and
hidden edges

OpenGL Depth - Cueing Function

gl Fogi (GL_FOG_MODE , GL_LINEAR) → Start or End mode

- This function is used to vary the brightness of an object.
- It applies linear depth function to object colors using $d_{min} = 0.0$ & $d_{max} = 1.0$ by default.

gl Fogf (GL_FOG_START , minDepth) → d_{min} value is set
gl Fogf (GL_FOG_END , maxDepth) → d_{max} value is set

gl Enable (GL_FOG) → Activate Fog function