Unit III: 3D Object Representations, Transformations, Viewing & Clipping

Three-dimensional Geometric Transformations

- Methods for geometric transformations in three dimensions are extended from two-dimensional methods by including considerations for the z coordinate.
- When we discussed 2D rotations in the xy plane, we needed to consider only rotations about axes that were perpendicular to the xy plane.
- In 3D space, we can now select any spatial orientation for the rotation axis.
- Some graphics packages handle 3D rotation as a composite of three rotations, one for each of the three Cartesian axes.

- A three-dimensional position, expressed in homogeneous coordinates, is represented as a four-element column vector.

- Thus, each geometric transformation operator is now 4x4 matrix, which premultiplies a coordinate column vector.

- Three-dimensional translation
- Three-dimensional rotation
- Three-dimensional scaling

1. Three-dimensional translation

- A position **P** = *(x, y, z)* in three-dimensional space is translated to a location **P** = *(x, y, z)* by adding translation distances *tx*, *ty*, and *tz* to the Cartesian coordinates of **P**:

$$x' = x + t_x, \qquad y' = y + t_y, \qquad z' = z + t_z$$

- We can express these three-dimensional translation operations in matrix form. But now the coordinate positions, **P** and **P** , are represented in homogeneous coordinates with four-element column matrices, and the translation operator **T** is a 4 × 4 matrix:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\mathbf{P}' = \mathbf{T} \cdot \mathbf{P}$$

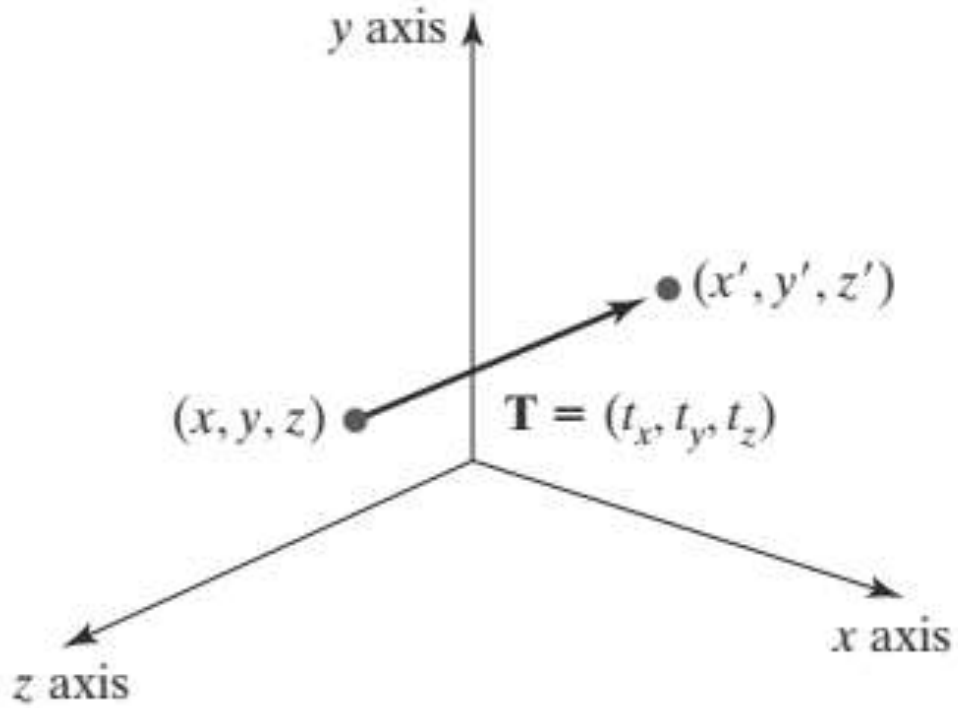


FIGURE 1
Moving a coordinate position with translation vector $\mathbf{T} = (t_x, t_y, t_z)$.

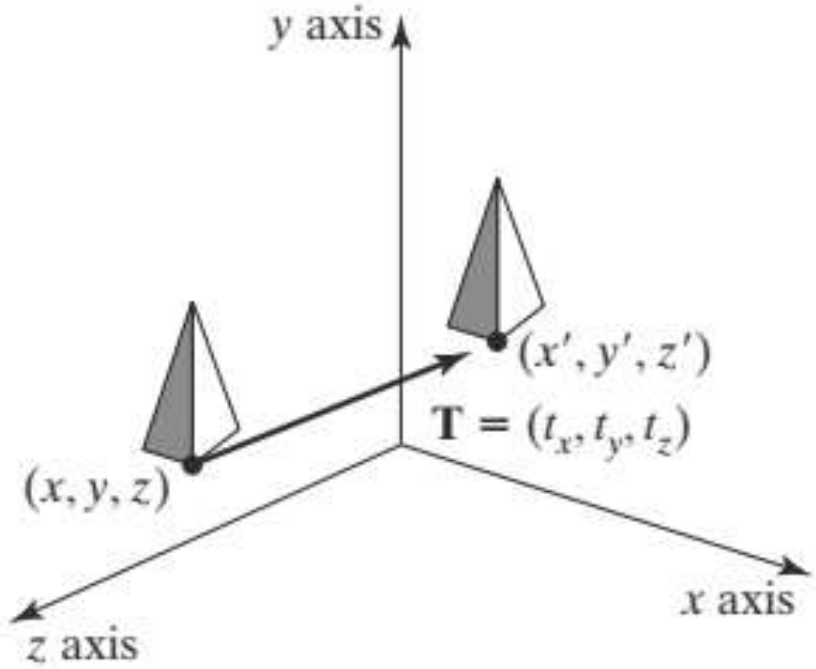


FIGURE 2
Shifting the position of a three-dimensional object using translation vector **T**.

- An object is translated in three dimensions by transforming each of the defining coordinate positions for the object, then reconstructing the object at the new location.

- For an object represented as a set of polygon surfaces, we translate each vertex for each surface as shown in Figure 2 and redisplay the polygon facets at the translated positions.

2. Three-dimensional rotation

- We can rotate an object about any axis in space, but the easiest rotation axes to handle are those that are parallel to the Cartesian-coordinate axes.
- Also, we can use combinations of coordinate-axis rotations (along with appropriate translations) to specify a rotation about any other line in space.
- Therefore, we first consider the operations involved in coordinate-axis rotations, then we discuss the calculations needed for other rotation axes.
- By convention, positive rotation angles produce counterclockwise rotations about a coordinate axis, assuming that we are looking in the negative direction along that coordinate axis.

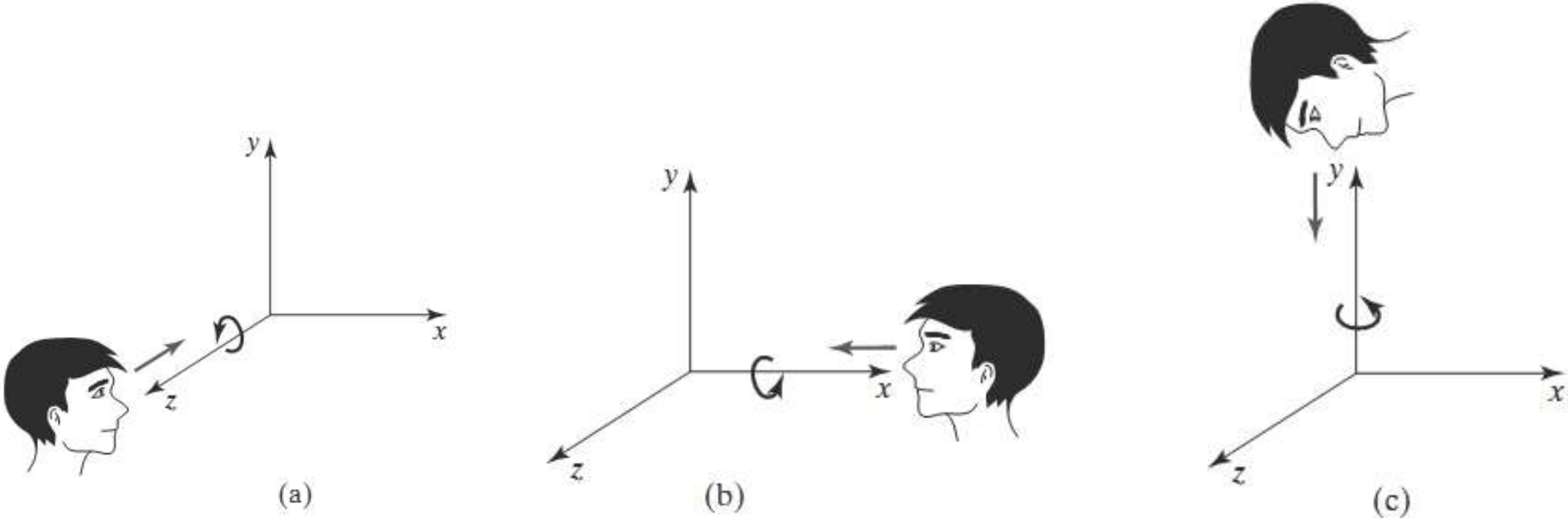


FIGURE 3
Positive rotations about a coordinate axis are counterclockwise, when looking along the positive half of the axis toward the origin.

- The two-dimensional z-axis rotation equations are easily extended to three dimensions, as follows:

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$
$$z' = z$$

- Parameter θ specifies the rotation angle about the z axis, and z-coordinate values are unchanged by this transformation.
- In homogeneous-coordinate form, the three-dimensional z-axis rotation equations are:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- which we can write more compactly as:

$$\mathbf{P}' = \mathbf{R}_z(\theta) \cdot \mathbf{P}$$

- Transformation equations for rotations about the other two coordinate axes can be obtained with a cyclic permutation of the coordinate parameters x, y, and z in Equations:

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$
$$z' = z$$

$$x \rightarrow y \rightarrow z \rightarrow x$$

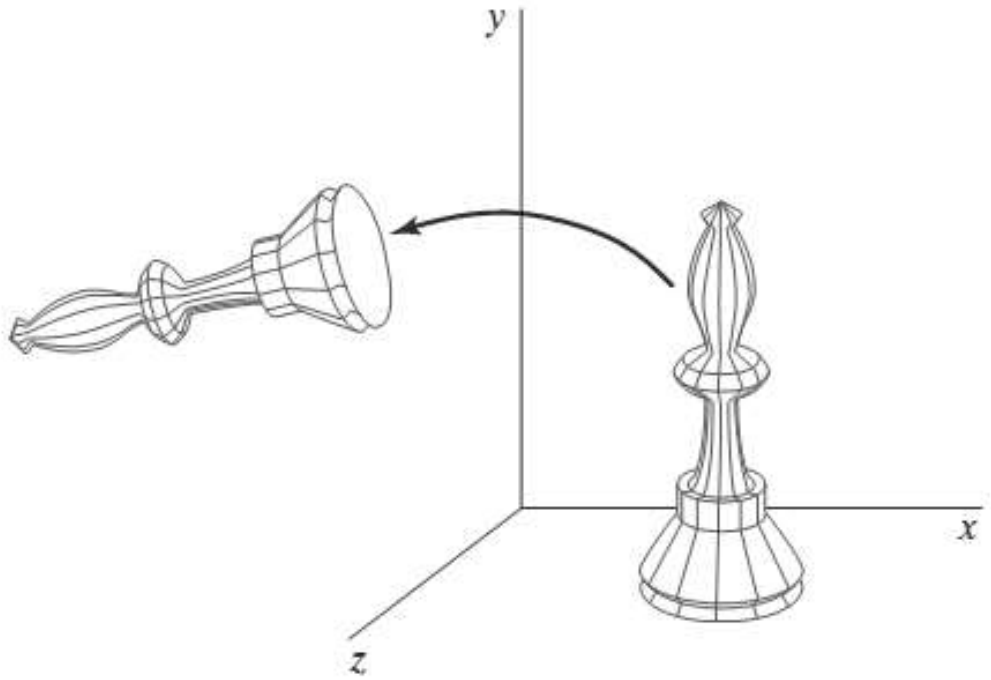


FIGURE 4
Rotation of an object about the *z* axis.

- Thus, to obtain the x-axis and y-axis rotation transformations, we cyclically replace x with y, y with z, and z with x, as illustrated in Figure 5.
- Substituting permutations 7 into Equations 4, we get the equations for an x-axis rotation:

$$y' = y\cos\theta - z\sin\theta$$
$$z' = y\sin\theta + z\cos\theta$$
$$x' = x$$

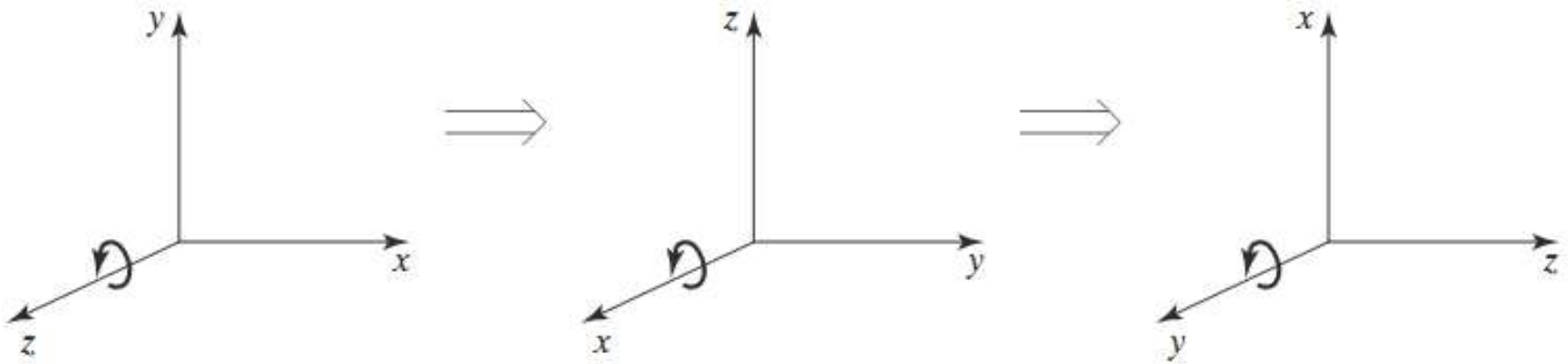


FIGURE 5
Cyclic permutation of the Cartesian-coordinate axes to produce the three sets of coordinate-axis rotation equations.

- Rotation of an object around the x axis is demonstrated in Figure 6.

- A cyclic permutation of coordinates in Equations 8 gives us the transformation equations for a y-axis rotation:

$$z' = z\cos\theta - x\sin\theta$$
$$x' = z\sin\theta + x\cos\theta$$
$$y' = y$$

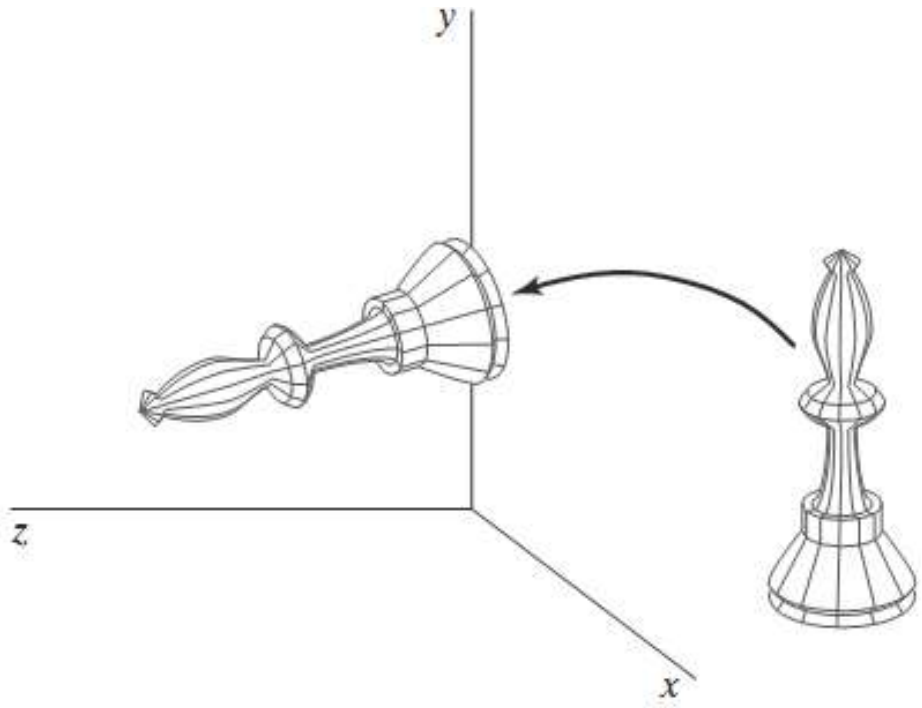


FIGURE 6
Rotation of an object about the *x* axis.

SUBSTITUTION AFTER X-AXIS SUBSTITUTION

- An example of y-axis rotation is shown in Figure 7.
- An inverse three-dimensional rotation matrix is obtained in the same way as the inverse rotations in two dimensions. We just replace the angle θ with −θ.
- Negative values for rotation angles generate rotations in a clockwise direction, and the identity matrix is produced when we multiply any rotation matrix by its inverse.
- Because only the sine function is affected by the change in sign of the rotation angle, the inverse matrix can also be obtained by interchanging rows and columns.
- That is, we can calculate the inverse of any rotation matrix R by forming its transpose ($R^{-1} = R^T$).

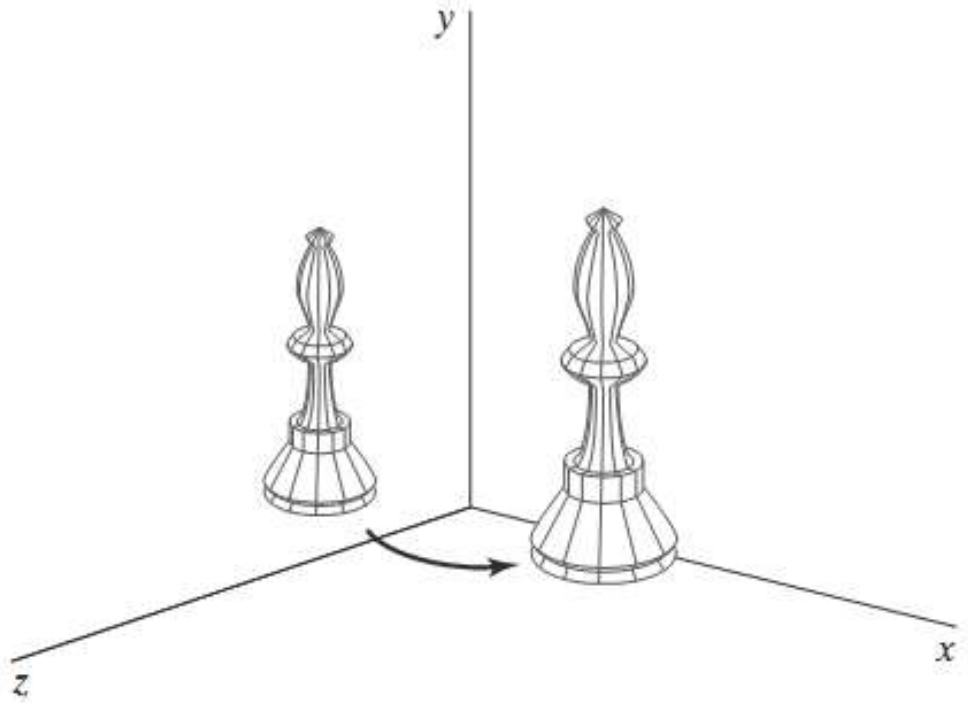


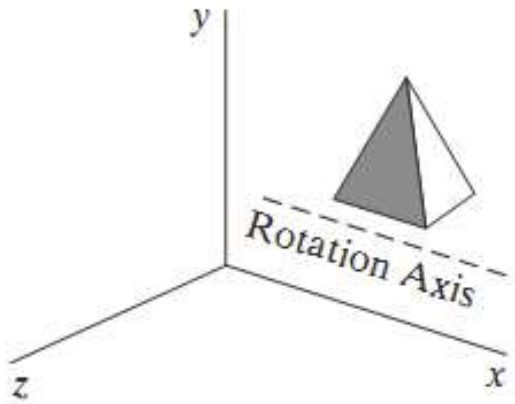
FIGURE 7
Rotation of an object about the *y* axis.

General Three-dimensional rotations:

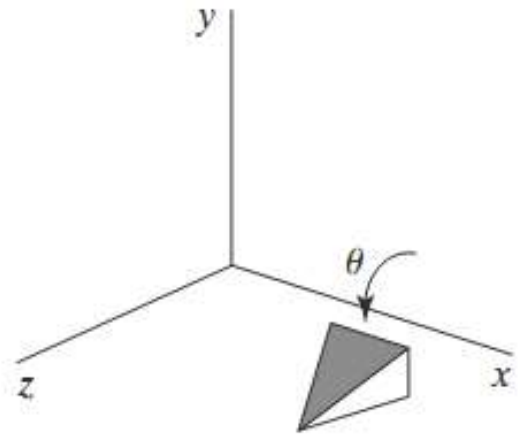
- A rotation matrix for any axis that does not coincide with a coordinate axis can be set up as a composite transformation involving combinations of translations and the coordinate-axis rotations.
- We first move the designated rotation axis onto one of the coordinate axes.
- Then we apply the appropriate rotation matrix for that coordinate axis.
- The last step in the transformation sequence is to return the rotation axis to its original position.

- In the special case where an object is to be rotated about an axis that is parallel to one of the coordinate axes, we attain the desired rotation with the following transformation sequence:

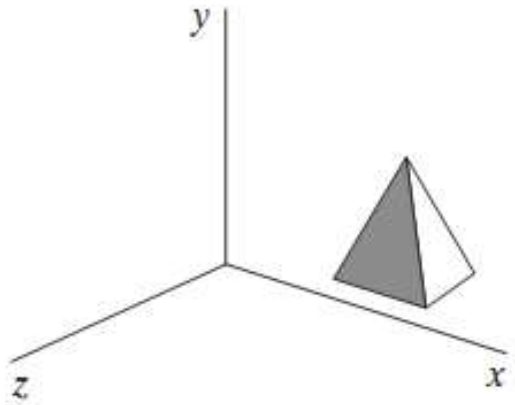
1. Translate the object so that the rotation axis coincides with the parallel coordinate axis.
2. Perform the specified rotation about that axis.
3. Translate the object so that the rotation axis is moved back to its original position.



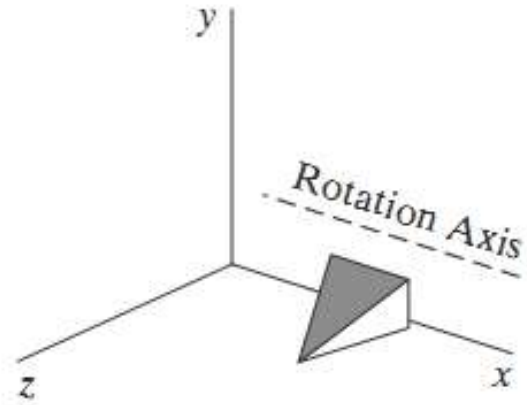
(a)
Original Position of Object



(c)
Rotate Object Through Angle θ



(b)
Translate Rotation Axis onto x Axis



(d)
Translate Rotation
Axis to Original Position

FIGURE 8
Sequence of transformations for rotating an object about an axis that is parallel to the x axis.

- The steps in this sequence are illustrated in Figure 8. A coordinate position P is transformed with the sequence shown in this figure as:

$$\mathbf{P}' = \mathbf{T}^{-1} \cdot \mathbf{R}_x(\theta) \cdot \mathbf{T} \cdot \mathbf{P}$$

- Where the composite rotation matrix for the transformation is:

$$\mathbf{R}(\theta) = \mathbf{T}^{-1} \cdot \mathbf{R}_x(\theta) \cdot \mathbf{T}$$

- This composite matrix is of the same form as the two-dimensional transformation sequence for rotation about an axis that is parallel to the z axis (a pivot point that is not at the coordinate origin).
- When an object is to be rotated about an axis that is not parallel to one of the coordinate axes, we must perform some additional transformations.

- In this case, we also need rotations to align the rotation axis with a selected coordinate axis and then to bring the rotation axis back to its original orientation.
- Given the specifications for the rotation axis and the rotation angle, we can accomplish the required rotation in five steps:

1. Translate the object so that the rotation axis passes through the coordinate origin.
2. Rotate the object so that the axis of rotation coincides with one of the coordinate axes.
3. Perform the specified rotation about the selected coordinate axis.
4. Apply inverse rotations to bring the rotation axis back to its original orientation.
5. Apply the inverse translation to bring the rotation axis back to its original spatial position.

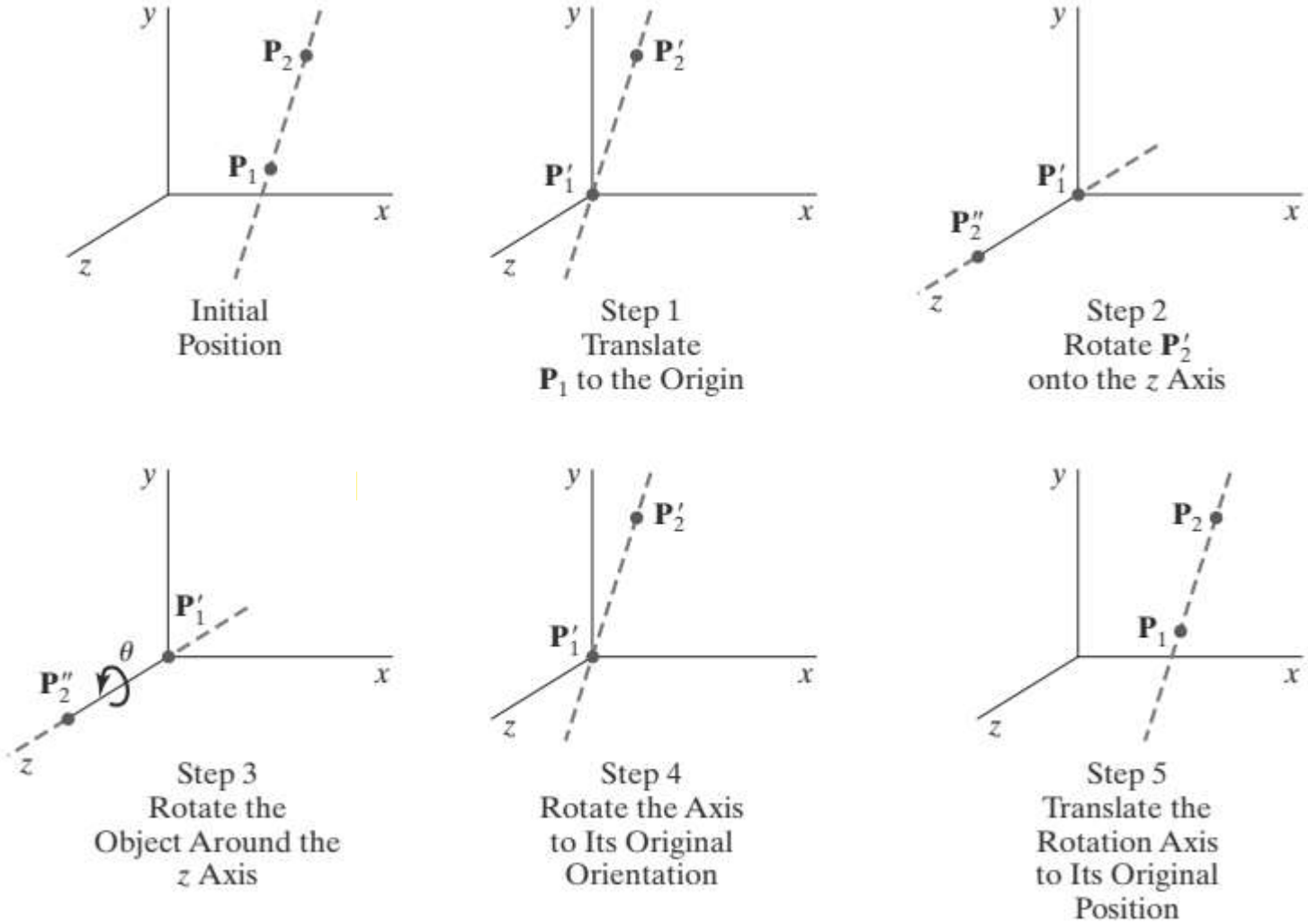


FIGURE 9
Five transformation steps for obtaining a composite matrix for rotation about an arbitrary axis, with the rotation axis projected onto the z axis.

- We can transform the rotation axis onto any one of the three coordinate axes.
- The *z* axis is often a convenient choice, and we next consider a transformation sequence using the *z*-axis rotation matrix.
- A rotation axis can be defined with two coordinate positions, or with one coordinate point and direction angles (or direction cosines) between the rotation axis and two of the coordinate axes.
- We assume that the rotation axis is defined by two points, and that the direction of rotation is to be counterclockwise when looking along the axis from P2 to P1.
- The components of the rotation-axis vector are then computed as:

Start

$$\mathbf{V} = \mathbf{P}_2 - \mathbf{P}_1$$
$$= (x_2 - x_1, y_2 - y_1, z_2 - z_1)$$

$$\mathbf{u} = \frac{\mathbf{V}}{|\mathbf{V}|} = (a, b, c)$$

- where the components a, b, and c are the direction cosines for the rotation axis:

$$a = \frac{x_2 - x_1}{|\mathbf{V}|}, \qquad b = \frac{y_2 - y_1}{|\mathbf{V}|}, \qquad c = \frac{z_2 - z_1}{|\mathbf{V}|}$$

- The first step in the rotation sequence is to set up the translation matrix that repositions the rotation axis so that it passes through the coordinate origin.
- Because we want a counterclockwise rotation when viewing along the axis from **P**2 to **P**1 (Figure 10), we move the point **P**1 to the origin.

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

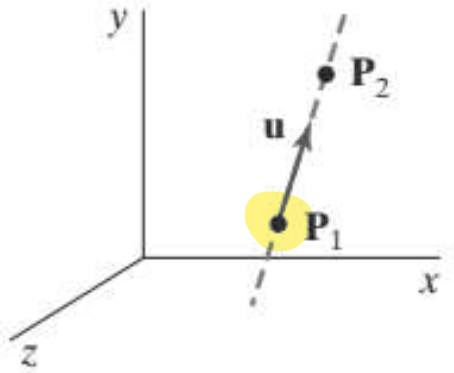


FIGURE 10
An axis of rotation (dashed line) defined with points $\mathbf{P}_1$ and $\mathbf{P}_2$. The direction for the unit axis vector **u** is determined by the specified rotation direction.

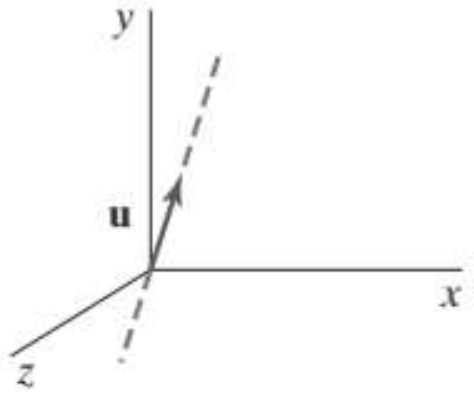


FIGURE 11
Translation of the rotation axis to the coordinate origin.

- Next, we formulate the transformations that will put the rotation axis onto the *z* axis.
- We can use the coordinate-axis rotations to accomplish this alignment in two steps, and there are a number of ways to perform these two steps.
- For this example, we first rotate about the *x* axis, then rotate about the *y* axis.
- The *x*-axis rotation gets vector **u** into the *xz* plane, and the *y*-axis rotation swings **u** around to the *z* axis.

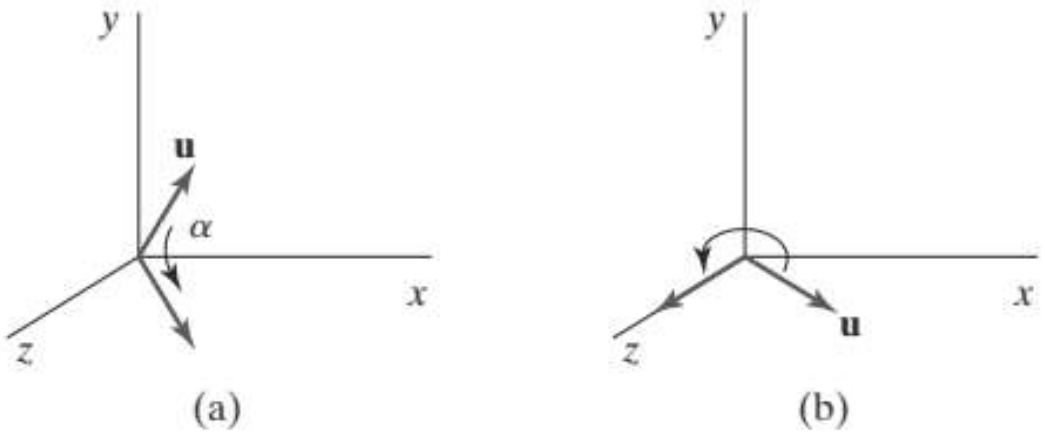


FIGURE 12
Unit vector **u** is rotated about the *x* axis to bring it into the *xz* plane (a), then it is rotated around the *y* axis to align it with the *z* axis (b).

- We establish the transformation matrix for rotation around the *x* axis by determining the values for the sine and cosine of the rotation angle necessary to get **u** into the *xz* plane.
- This rotation angle is the angle between the projection of **u** in the *yz* plane and the positive *z* axis (Figure 13).
- If we represent the projection of **u** in the *yz* plane as the vector **u** = *(0, b, c)*, then the cosine of the rotation angle α can be determined from the dot product of **u** and the unit vector **u***z* along the *z* axis:

$$\cos\alpha = \frac{\mathbf{u}' \cdot \mathbf{u}_z}{|\mathbf{u}'|\,|\mathbf{u}_z|} = \frac{c}{d}$$

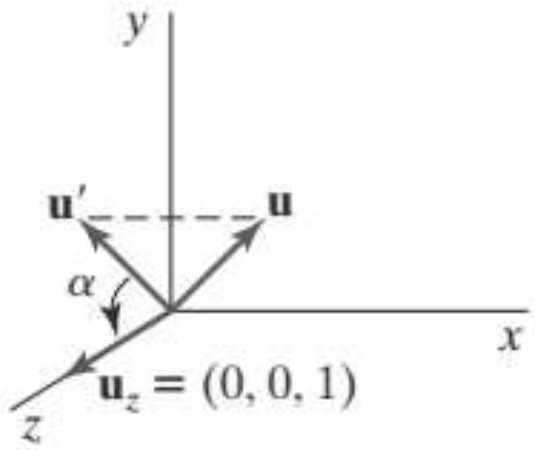


FIGURE 13
Rotation of **u** around the *x* axis into the *xz* plane is accomplished by rotating **u**′ (the projection of **u** in the *yz* plane) through angle α onto the *z* axis.

- Where d is the magnitude of u': $d = \sqrt{b^2 + c^2}$
- Similarly, we can determine the sine of α from the cross-product of **u** and **u**z. The coordinate-independent form of this cross-product is :

$$\mathbf{u}' \times \mathbf{u}_z = \mathbf{u}_x |\mathbf{u}'| |\mathbf{u}_z| \sin\alpha$$

and the Cartesian form for the cross-product gives us

$$\mathbf{u}' \times \mathbf{u}_z = \mathbf{u}_x \cdot b$$

$$d \sin\alpha = b$$

$$\sin\alpha = \frac{b}{d}$$

-

- Now that we have determined the values for cosα and sin α in terms of the components of vector **u**, we can set up the matrix elements for rotation of this vector about the x axis and into the xz plane:

$$\mathbf{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c}{d} & -\frac{b}{d} & 0 \\ 0 & \frac{b}{d} & \frac{c}{d} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

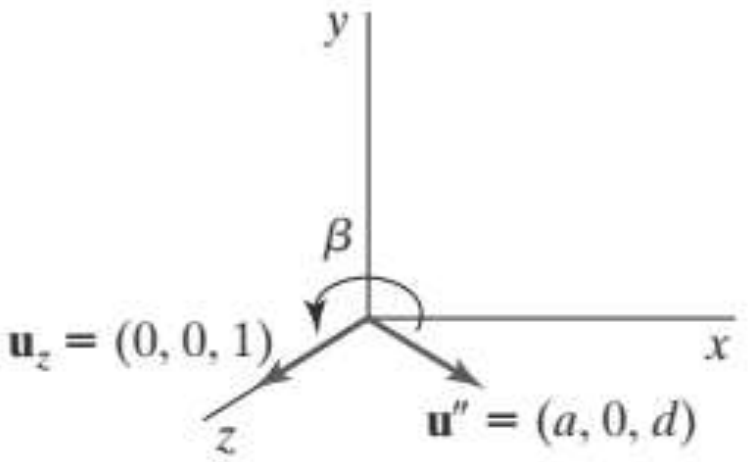


FIGURE 14
Rotation of unit vector **u**″ (vector **u** after rotation into the *xz* plane) about the *y* axis. Positive rotation angle *β* aligns **u**″ with vector $\mathbf{u}_z$.

- The next step in the formulation of the transformation sequence is to determine the matrix that will swing the unit vector in the *xz* plane counterclockwise around the *y* axis onto the positive *z* axis.

$$\cos\beta = \frac{\mathbf{u}'' \cdot \mathbf{u}_z}{|\mathbf{u}''|\,|\mathbf{u}_z|} = d \tag{22}$$

because $|\mathbf{u}_z| = |\mathbf{u}''| = 1$. Comparing the coordinate-independent form of the cross-product

$$\mathbf{u}'' \times \mathbf{u}_z = \mathbf{u}_y\,|\mathbf{u}''|\,|\mathbf{u}_z|\sin\beta \tag{23}$$

with the Cartesian form

$$\mathbf{u}'' \times \mathbf{u}_z = \mathbf{u}_y \cdot (-a) \tag{24}$$

we find that

$$\sin\beta = -a \tag{25}$$

Therefore, the transformation matrix for rotation of $\mathbf{u}''$ about the y axis is

$$\mathbf{R}_y(\beta) = \begin{bmatrix} d & 0 & -a & 0 \\ 0 & 1 & 0 & 0 \\ a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{26}$$

$$\mathbf{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{27}$$

To complete the required rotation about the given axis, we need to transform the rotation axis back to its original position. This is done by applying the inverse of transformations 15, 21, and 26. The transformation matrix for rotation about an arbitrary axis can then be expressed as the composition of these seven individual transformations:

$$\mathbf{R}(\theta) = \mathbf{T}^{-1} \cdot \mathbf{R}_x^{-1}(\alpha) \cdot \mathbf{R}_y^{-1}(\beta) \cdot \mathbf{R}_z(\theta) \cdot \mathbf{R}_y(\beta) \cdot \mathbf{R}_x(\alpha) \cdot \mathbf{T} \tag{28}$$

A somewhat quicker, but perhaps less intuitive, method for obtaining the composite rotation matrix $\mathbf{R}_y(\beta) \cdot \mathbf{R}_x(\alpha)$ is to use the fact that the composite matrix for any sequence of three-dimensional rotations is of the form

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{29}$$

$$\mathbf{R} \cdot \begin{bmatrix} r_{11} \\ r_{12} \\ r_{13} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \qquad \mathbf{R} \cdot \begin{bmatrix} r_{21} \\ r_{22} \\ r_{23} \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}, \qquad \mathbf{R} \cdot \begin{bmatrix} r_{31} \\ r_{32} \\ r_{33} \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \tag{30}$$

$$\begin{aligned} \mathbf{u}'_z &= \mathbf{u} \\ \mathbf{u}'_y &= \frac{\mathbf{u} \times \mathbf{u}_x}{|\mathbf{u} \times \mathbf{u}_x|} \\ \mathbf{u}'_x &= \mathbf{u}'_y \times \mathbf{u}'_z \end{aligned} \tag{31}$$

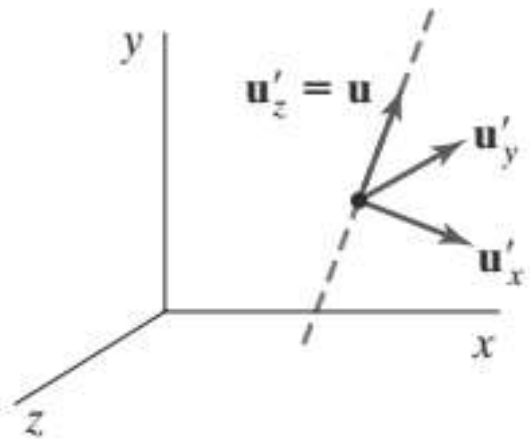


FIGURE 15
Local coordinate system for a rotation axis defined by unit vector **u**.

If we express the elements of the unit local vectors for the rotation axis as

$$\begin{aligned} \mathbf{u}'_x &= (u'_{x1}, u'_{x2}, u'_{x3}) \\ \mathbf{u}'_y &= (u'_{y1}, u'_{y2}, u'_{y3}) \\ \mathbf{u}'_z &= (u'_{z1}, u'_{z2}, u'_{z3}) \end{aligned} \tag{32}$$

then the required composite matrix, which is equal to the product $\mathbf{R}_y(\beta) \cdot \mathbf{R}_x(\alpha)$, is

$$\mathbf{R} = \begin{bmatrix} u'_{x1} & u'_{x2} & u'_{x3} & 0 \\ u'_{y1} & u'_{y2} & u'_{y3} & 0 \\ u'_{z1} & u'_{z2} & u'_{z3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{33}$$

This matrix transforms the unit vectors $\mathbf{u}'_x$, $\mathbf{u}'_y$, and $\mathbf{u}'_z$ onto the x, y, and z axes, respectively. This aligns the rotation axis with the z axis, because $\mathbf{u}'_z = \mathbf{u}$.

Quaternion Methods for 3D Rotations

- Quaternions, which are extensions of two-dimensional complex numbers, are useful in a number of computer-graphics procedures, including the generation of fractal objects.
- They require less storage space than 4 × 4 matrices, and it is simpler to write quaternion procedures for transformation sequences.
- This is particularly important in animations, which often require complicated motion sequences and motion interpolations between two given positions of an object.
- One way to characterize a quaternion is as an ordered pair, consisting of a *scalar part* and a *vector part:*

$$q = (s, \mathbf{v})$$

- A rotation about any axis passing through the coordinate origin is accomplished by first setting up a unit quaternion with the scalar and vector parts as follows:

$$s = \cos\frac{\theta}{2}, \qquad \mathbf{v} = \mathbf{u}\sin\frac{\theta}{2}$$

  where **u** is a unit vector along the selected rotation axis and ϑ is the specified rotation angle about this axis (Figure 16).

- Any point position **P** that is to be rotated by this quaternion can be represented in quaternion notation as :

$$\mathbf{P} = (0, \mathbf{p})$$

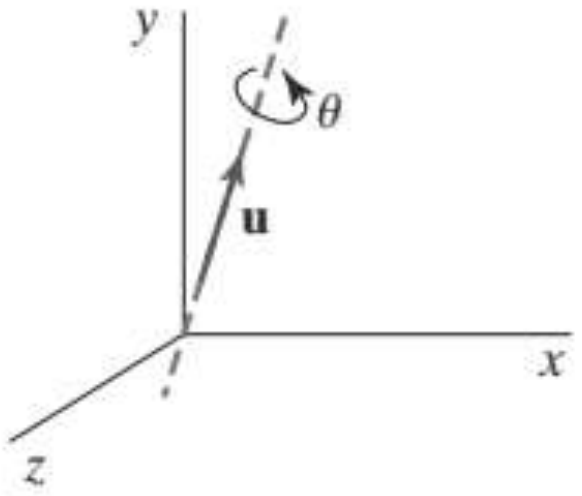


FIGURE 16
Unit quaternion parameters θ and **u** for rotation about a specified axis.

with the coordinates of the point as the vector part $\mathbf{p} = (x, y, z)$. The rotation of the point is then carried out with the quaternion operation

$$\mathbf{P}' = q\mathbf{P}q^{-1} \tag{35}$$

where $q^{-1} = (s, -\mathbf{v})$ is the inverse of the unit quaternion q with the scalar and vector parts given in Equations 34. This transformation produces the following new quaternion:

$$\mathbf{P}' = (0, \mathbf{p}') \tag{36}$$

The second term in this ordered pair is the rotated point position $\mathbf{p}'$, which is evaluated with vector dot and cross-products as

$$\mathbf{p}' = s^2\mathbf{p} + \mathbf{v}(\mathbf{p} \cdot \mathbf{v}) + 2s(\mathbf{v} \times \mathbf{p}) + \mathbf{v} \times (\mathbf{v} \times \mathbf{p}) \tag{37}$$

Values for parameters s and $\mathbf{v}$ are obtained from the expressions in 34. Many computer graphics systems use efficient hardware implementations of these vector calculations to perform rapid three-dimensional object rotations.

We can evaluate the terms in Equation 37 using the definition for quaternion multiplication. Also, designating the components of the vector part of q as $\mathbf{v} = (a, b, c)$, we obtain the elements for the composite rotation matrix $\mathbf{R}_x^{-1}(\alpha) \cdot \mathbf{R}_y^{-1}(\beta) \cdot \mathbf{R}_z(\theta) \cdot \mathbf{R}_y(\beta) \cdot \mathbf{R}_x(\alpha)$ in a 3×3 form as

$$\mathbf{M}_R(\theta) = \begin{bmatrix} 1-2b^2-2c^2 & 2ab-2sc & 2ac+2sb \\ 2ab+2sc & 1-2a^2-2c^2 & 2bc-2sa \\ 2ac-2sb & 2bc+2sa & 1-2a^2-2b^2 \end{bmatrix} \tag{38}$$

The calculations involved in this matrix can be greatly reduced by substituting explicit values for parameters a, b, c, and s, and then using the following trigonometric identities to simplify the terms:

$$\cos^2\frac{\theta}{2} - \sin^2\frac{\theta}{2} = 1 - 2\sin^2\frac{\theta}{2} = \cos\theta, \qquad 2\cos\frac{\theta}{2}\sin\frac{\theta}{2} = \sin\theta$$

Thus, we can rewrite Matrix 38 as

$$\mathbf{M}_R(\theta) = \begin{bmatrix} u_x^2(1-\cos\theta)+\cos\theta & u_xu_y(1-\cos\theta)-u_z\sin\theta & u_xu_z(1-\cos\theta)+u_y\sin\theta \\ u_yu_x(1-\cos\theta)+u_z\sin\theta & u_y^2(1-\cos\theta)+\cos\theta & u_yu_z(1-\cos\theta)-u_x\sin\theta \\ u_zu_x(1-\cos\theta)-u_y\sin\theta & u_zu_y(1-\cos\theta)+u_x\sin\theta & u_z^2(1-\cos\theta)+\cos\theta \end{bmatrix} \tag{39}$$

where u_x, u_y, and u_z are the components of the unit axis vector $\mathbf{u}$.

- To complete the transformation sequence for rotating about an arbitrarily placed rotation axis, we need to include the translations that move the rotation axis to the coordinate axis and return it to its original position.
- Thus, the complete quaternion rotation expression, corresponding to Equation 28, is:

$$\mathbf{R}(\theta) = \mathbf{T}^{-1} \cdot \mathbf{M}_R \cdot \mathbf{T}$$

- For example, we can perform a rotation about the *z* axis by setting rotation axis vector **u** to the unit *z*-axis vector (0, 0, 1). Substituting the components of this vector into Matrix 39, we get the 3×3 version of the *z*-axis rotation matrix **R***z*(*ϑ*).

End

3. Three-dimensional scaling

- The matrix expression for the three-dimensional scaling transformation of a position P = (x, y, z) relative to the coordinate origin is a simple extension of two-dimensional scaling.
- We just include the parameter for z-coordinate scaling in the transformation matrix:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- The three-dimensional scaling transformation for a point position can be represented as:

$$\mathbf{P}' = \mathbf{S} \cdot \mathbf{P}$$

- where scaling parameters sx , sy, and sz are assigned any positive values.
- Explicit expressions for the scaling transformation relative to the origin are:

$$x' = x \cdot s_x, \qquad y' = y \cdot s_y, \qquad z' = z \cdot s_z$$

- A parameter value greater than 1 moves a point farther from the origin in the corresponding coordinate direction.
- Similarly, a parameter value less than 1 moves a point closer to the origin in that coordinate direction.
- Also, if the scaling parameters are not all equal, relative dimensions of a transformed object are changed.
- We preserve the original shape of an object with a uniform scaling: sx = sy = sz.

- Because some graphics packages provide only a routine that scales relative to the coordinate origin, we can always construct a scaling transformation with respect to any selected fixed position (xf , yf , zf) using the following transformation sequence:

1. Translate the fixed point to the origin.
2. Apply the scaling transformation relative to the coordinate origin using Equation for z-coordinate scaling in the transformation matrix.
3. Translate the fixed point back to its original position.

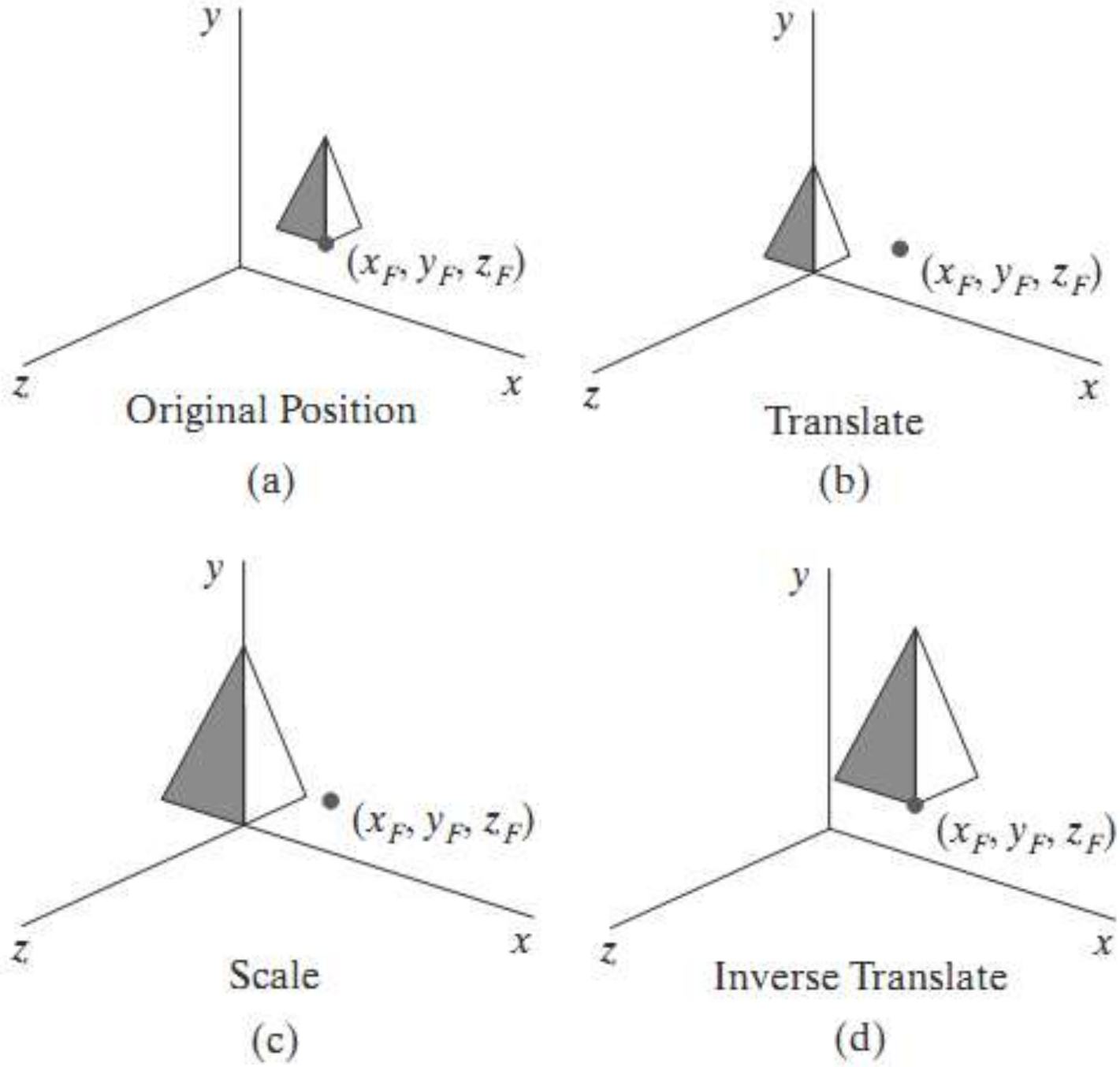


FIGURE 18
A sequence of transformations for scaling an object relative to a selected fixed point, using Equation 41.

- The matrix representation for an arbitrary fixed-point scaling can then be expressed as the concatenation of these translate-scale-translate transformations:

Important

$$\mathbf{T}(x_f, y_f, z_f) \cdot \mathbf{S}(s_x, s_y, s_z) \cdot \mathbf{T}(-x_f, -y_f, -z_f) = \begin{bmatrix} s_x & 0 & 0 & (1-s_x)x_f \\ 0 & s_y & 0 & (1-s_y)y_f \\ 0 & 0 & s_z & (1-s_z)z_f \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- We can set up programming procedures for constructing a three-dimensional scaling matrix using either a translate-scale-translate sequence or a direct incorporation of the fixed-point coordinates.

Composite 3D Transformations:

- As with two-dimensional transformations, we form a composite three dimensional transformation by multiplying the matrix representations for the individual operations in the transformation sequence.

- Any of the two-dimensional transformation sequences, such as scaling in non-coordinate directions, can be carried out in three-dimensional space.

- We can implement a transformation sequence by concatenating the individual matrices from right to left or from left to right, depending on the order in which the matrix representations are specified.

Other Three-dimensional Transformations

- In addition to translation, rotation, and scaling, the other transformations discussed for two-dimensional applications are also useful in many three-dimensional situations. These additional transformations include reflection, shear, and transformations between coordinate-reference frames.

- Three-dimensional Reflections

- Three-dimensional Shear

Three-dimensional Reflections

- A reflection in a three-dimensional space can be performed relative to a selected *reflection axis* or with respect to a *reflection plane*. In general, three-dimensional reflection matrices are set up similarly to those for two dimensions.
- Reflections relative to a given axis are equivalent to 180◦ rotations about that axis.
- Reflections with respect to a plane are similar; when the reflection plane is a coordinate plane (*xy*, *xz*, or *yz*), we can think of the transformation as a 180◦ rotation in four-dimensional space with a conversion between a left-handed frame and a right-handed frame.

- The matrix representation for this reflection relative to the xy plane is:

$$M_{\text{zreflect}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

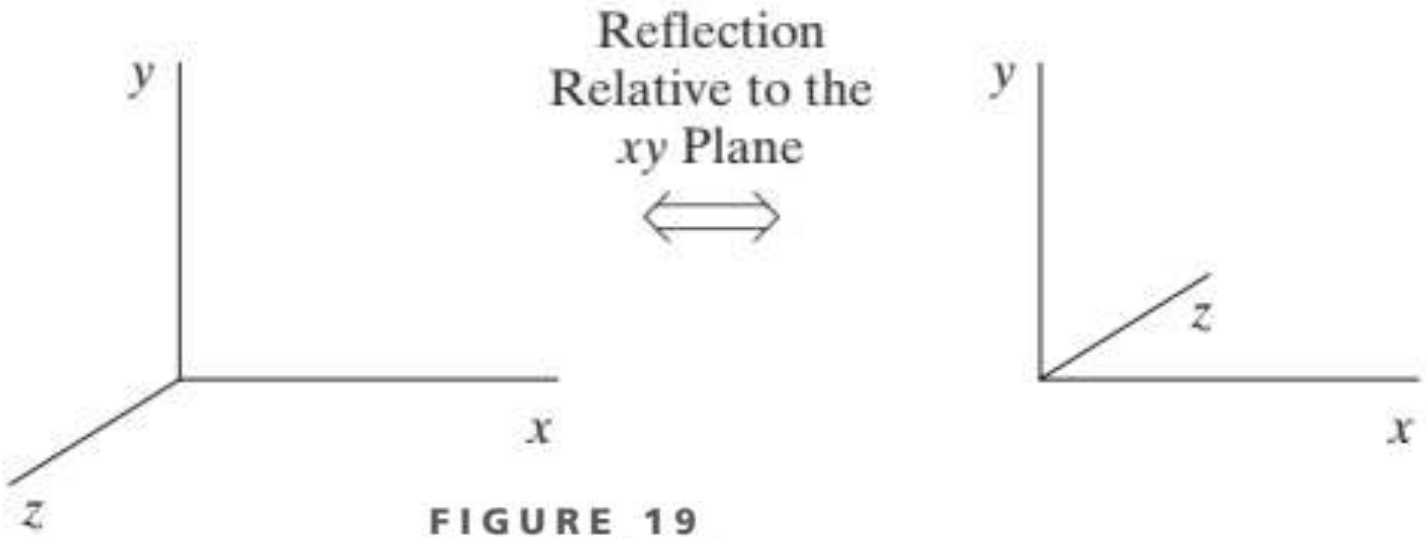


FIGURE 19
Conversion of coordinate specifications between a right-handed and a left-handed system can be carried out with the reflection transformation 45.

Three-dimensional Shear

DOUBT

- These transformations can be used to modify object shapes, just as in two-dimensional applications. They are also applied in three-dimensional viewing transformations for perspective projections.
- A general *z*-axis shearing transformation relative to a selected reference position is produced with the following matrix:

$$M_{\text{zshear}} = \begin{bmatrix} 1 & 0 & \text{sh}_{zx} & -\text{sh}_{zx} \cdot z_{\text{ref}} \\ 0 & 1 & \text{sh}_{zy} & -\text{sh}_{zy} \cdot z_{\text{ref}} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Shearing parameters sh*zx* and sh*zy* can be assigned any real values.

- The effect of this transformation matrix is to alter the values for the *x* and *y* coordinates by an amount that is proportional to the distance from *z*ref, while leaving the *z* coordinate unchanged.

- Plane areas that are perpendicular to the *z* axis are thus shifted by an amount equal to *z*-*z*ref.

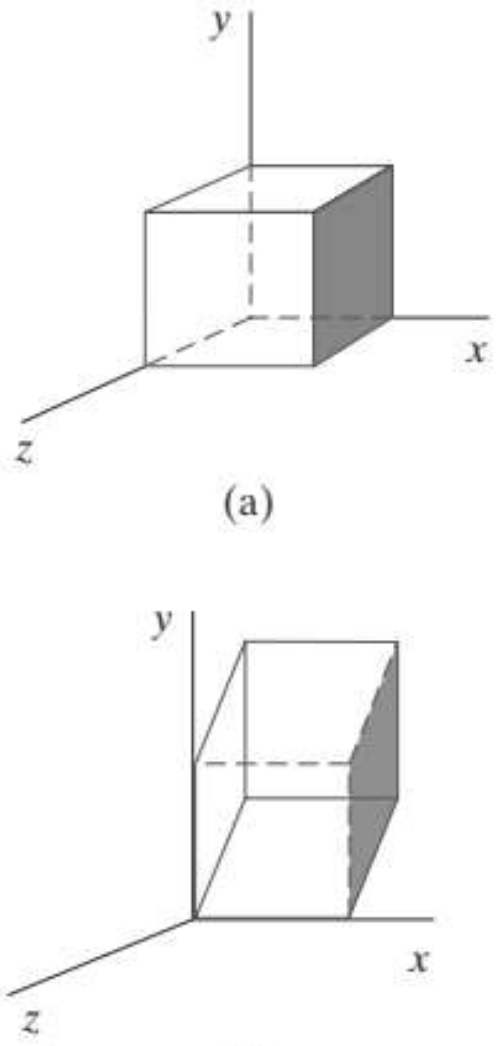


FIGURE 20
A unit cube (a) is sheared relative to the origin (b) by Matrix 46, with $\mathbf{sh}_{zx} = \mathbf{sh}_{zy} = 1$.

Affine Transformations

- A coordinate transformation of the form:

$$x' = a_{xx}x + a_{xy}y + a_{xz}z + b_x$$
$$y' = a_{yx}x + a_{yy}y + a_{yz}z + b_y$$
$$z' = a_{zx}x + a_{zy}y + a_{zz}z + b_z$$

- Each of the transformed coordinates *x*, *y*, and *z* is a linear function of the original coordinates *x*, *y*, and *z*, and parameters *ai j* and *bk* are constants determined by the transformation type.
- Affine transformations (in two dimensions, three dimensions, or higher dimensions) have the general properties that parallel lines are transformed into parallel lines, and finite points map to finite points.

Important

Summary of OpenGL Geometric Transformation Functions

Function	Description
`glTranslate*`	Specifies translation parameters.
`glRotate*`	Specifies parameters for rotation about any axis through the origin.
`glScale*`	Specifies scaling parameters with respect to coordinate origin.
`glMatrixMode`	Specifies current matrix for geometric-viewing transformations, projection transformations, texture transformations, or color transformations.
`glLoadIdentity`	Sets current matrix to identity.
`glLoadMatrix*` (elems);	Sets elements of current matrix.
`glMultMatrix*` (elems);	Postmultiplies the current matrix by the specified matrix.
`glGetIntegerv`	Gets max stack depth or current number of matrices in the stack for the selected matrix mode.
`glPushMatrix`	Copies the top matrix in the stack and store copy in the second stack position.
`glPopMatrix`	Erases the top matrix in the stack and moves the second matrix to the top of the stack.
`glPixelZoom`	Specifies two-dimensional scaling parameters for raster operations.

Smita

Problems

- A triangle is defined by 3 vertices A (0, 2, 1), B (2, 3, 0), C (1, 2, 1). Find the final co-ordinates after it is rotated by 45 degree around a line joining the points (1, 1, 1) and (0, 0, 0).

- A cube is defined by 8 vertices A (0,0,0), B (2,0,0), C (2,2,0), D (0,2,0), E (0,0,2), F (2,0,2), G (2,2,2), and H (0,2,2). Find the final coordinates after it is rotated by 45 degree around a line joining the points (2,0,0), and (0,2,2).

Three-dimensional Viewing

- For two-dimensional graphics applications, viewing operations transfer positions from the world-coordinate plane to pixel positions in the plane of the output device.
- Using the rectangular boundaries for the clipping window and the viewport, a two-dimensional package clips a scene and maps it to device coordinates.
- Three-dimensional viewing operations, however, are more involved, because we now have many more choices as to how we can construct a scene and how we can generate views of the scene on an output device.

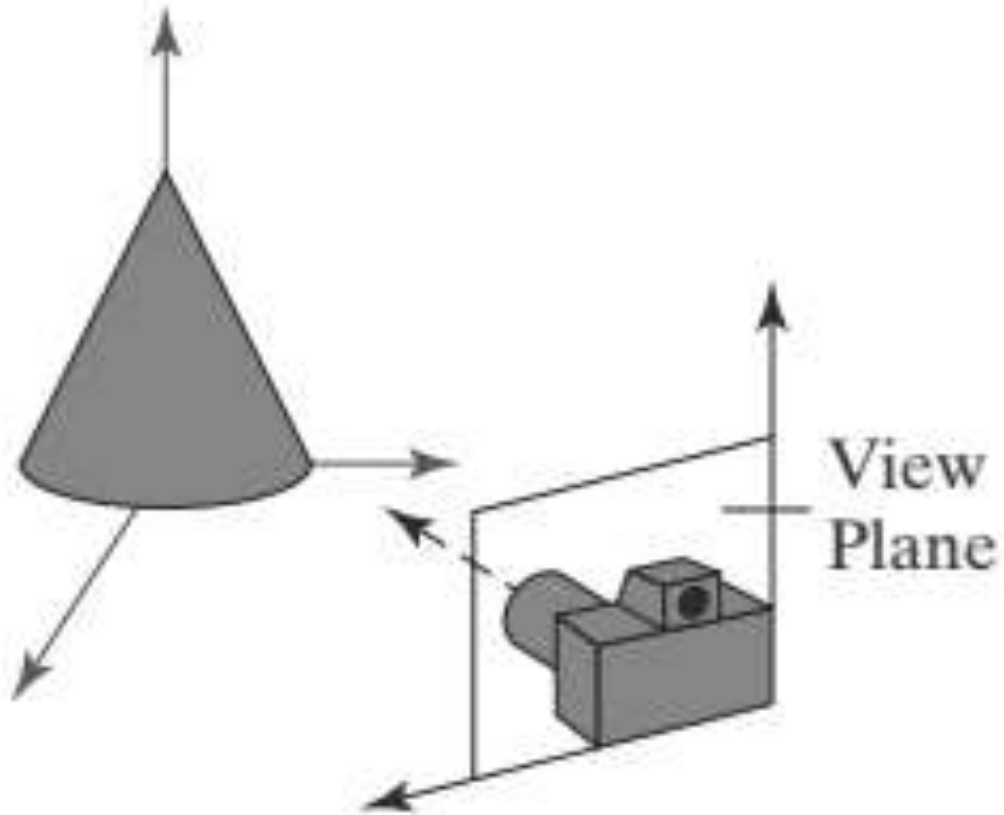


FIGURE 1
Coordinate reference for obtaining a selected view of a three-dimensional scene.

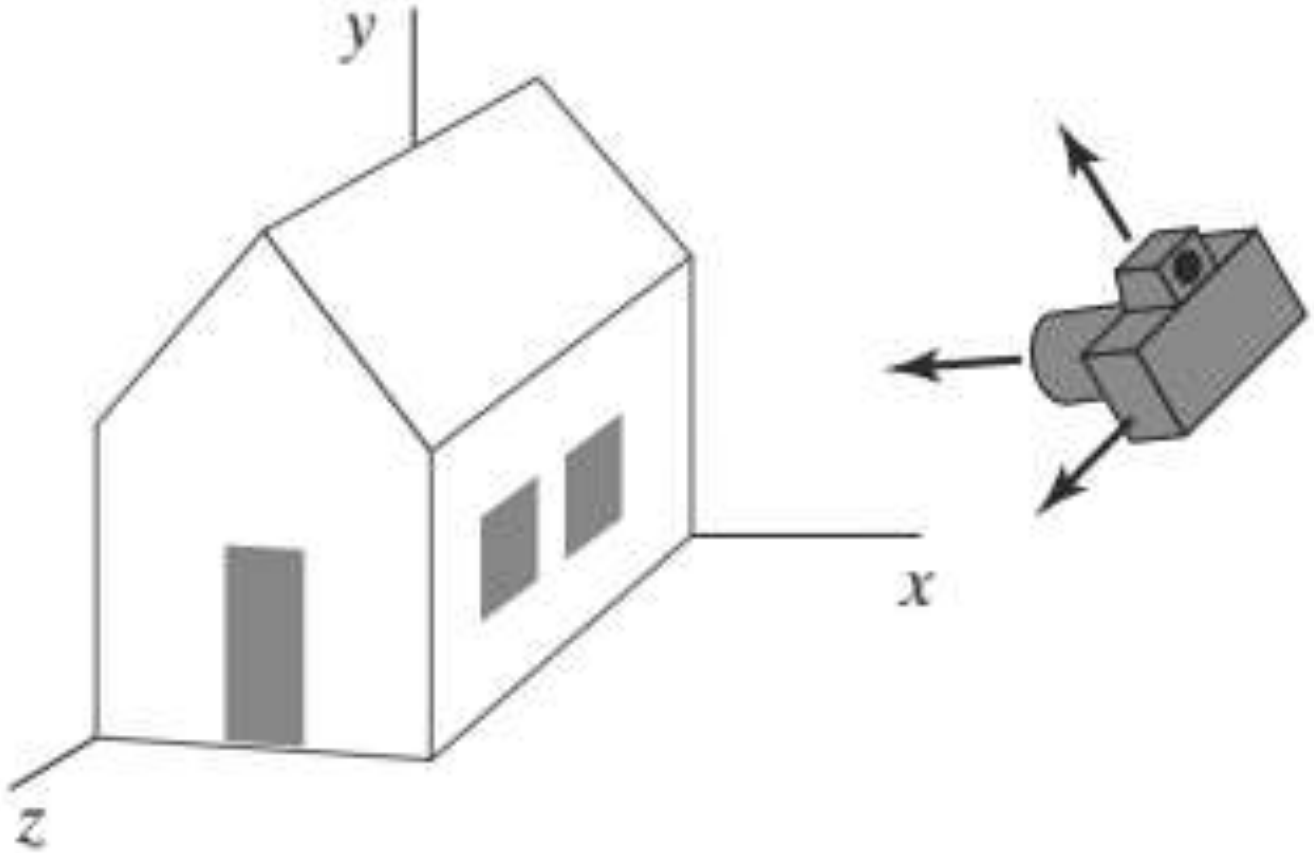


FIGURE 5
Photographing a scene involves selection of the camera position and orientation.

I. 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.
- Object descriptions are then transferred to the viewing reference coordinates and projected onto the view plane.
- We can generate a view of an object on the output device in wireframe (outline) form, or we can apply lighting and surface-rendering techniques to obtain a realistic shading of the visible surfaces.

II. Projections

- Unlike a camera picture, we can choose different methods for projecting a scene onto the view plane.
- One method for getting the description of a solid object onto a view plane is to project points on the object surface along parallel lines.
- This technique, called *parallel projection,* is used in engineering and architectural drawings to represent an object with a set of views that show accurate dimensions of the object.
- Another method for generating a view of a three-dimensional scene is to project points to the view plane along converging paths.
- This process, called a *perspective projection,* causes objects farther from the viewing position to be displayed smaller than objects of the same size that are nearer to the viewing position.

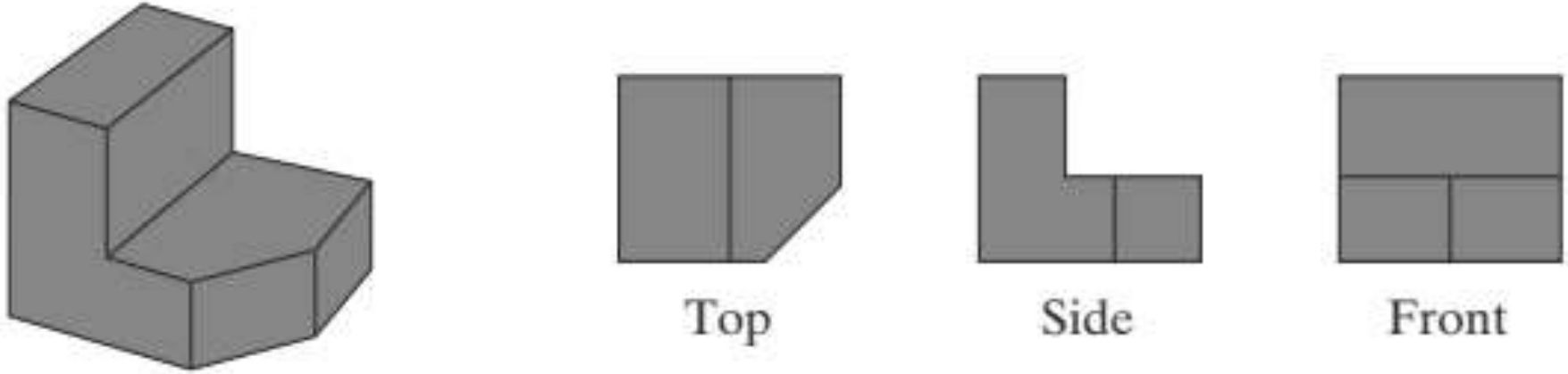
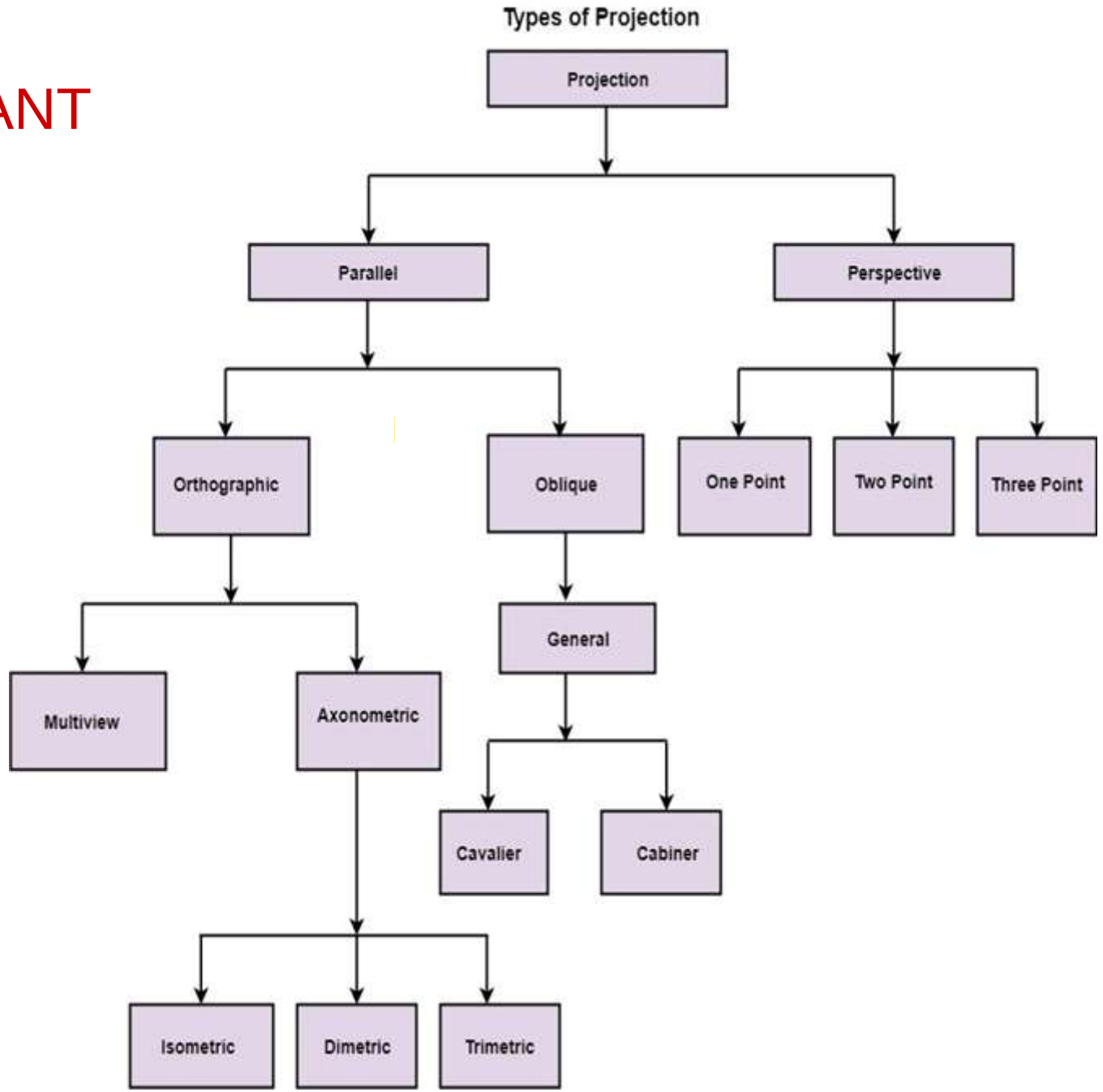


FIGURE 2
Three parallel-projection views of an object, showing relative proportions from different viewing positions.

- A scene that is generated using a perspective projection appears more realistic, because this is the way that our eyes and a camera lens form images.
- Parallel lines along the viewing direction appear to converge to a distant point in the background, and objects in the background appear to be smaller than objects in the foreground.

IMPORTANT


Types of Projection
Projection
Parallel
Perspective
Orthographic
Oblique
One Point
Two Point
Three Point
Multiview
Axonometric
General
Cavalier
Cabiner
Isometric
Dimetric
Trimetric

III. Depth Cueing

- With few exceptions, depth information is important in a three-dimensional scene so that we can easily identify, for a particular viewing direction, which is the front and which is the back of each displayed object.
- A simple method for indicating depth with wire-frame displays is to vary the brightness of line segments according to their distances from the viewing position.
- The lines closest to the viewing position are displayed with the highest intensity, and lines farther away are displayed with decreasing intensities.
- Depth cueing is applied by choosing a maximum and a minimum intensity value and a range of distances over which the intensity is to vary.

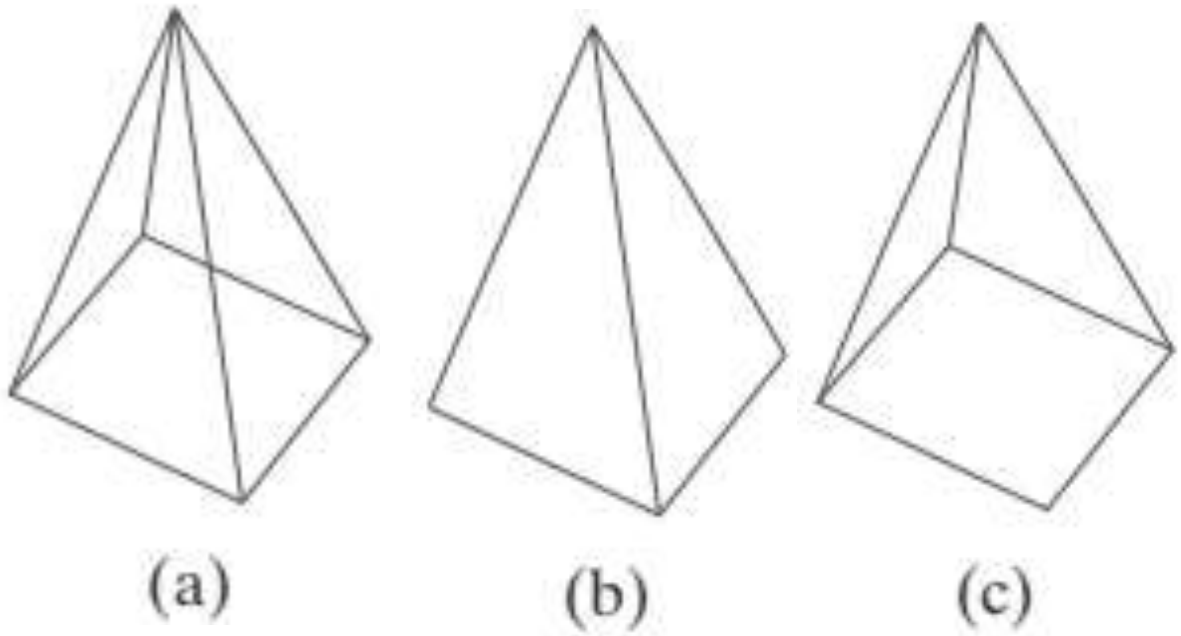


FIGURE 3
The wire-frame representation of the pyramid in (a) contains no depth information to indicate whether the viewing direction is (b) downward from a position above the apex or (c) upward from a position below the base.

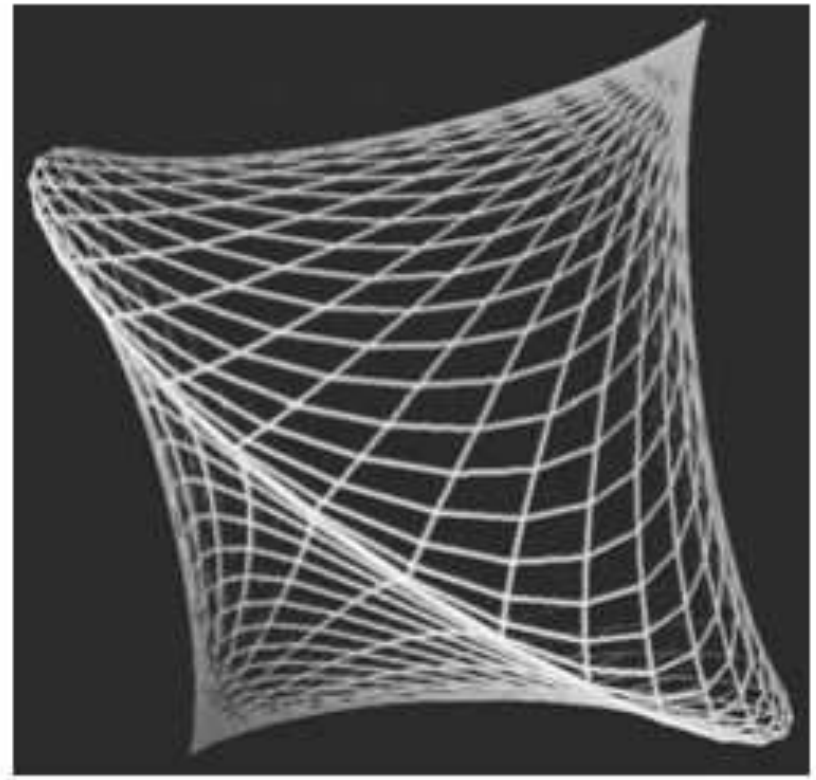


FIGURE 4
A wire-frame object displayed with depth cueing, so that the brightness of lines decreases from the front of the object to the back.

- Another application of depth cuing is modeling the effect of the atmosphere on the perceived intensity of objects.

- More distant objects appear dimmer to us than nearer objects due to light scattering by dust particles, haze, and smoke.

- Some atmospheric effects can even change the perceived color of an object, and we can model these effects with depth cueing.

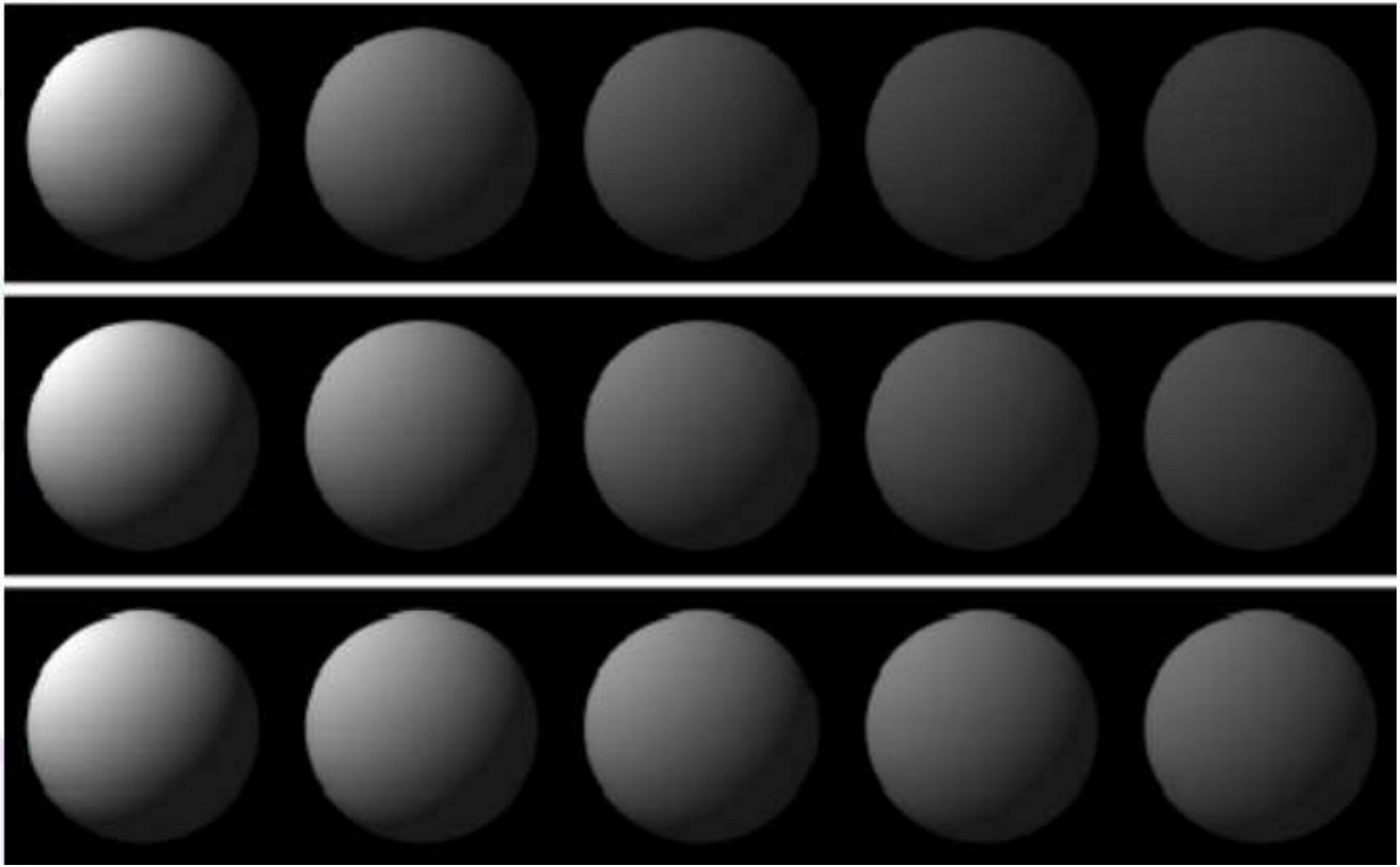
IV. Identifying Visible Lines and Surfaces

- We can also clarify depth relationships in a wire-frame display using techniques other than depth cueing.
- One approach is simply to highlight the visible lines or to display them in a different color.
- Another technique, commonly used for engineering drawings, is to display the non-visible lines as dashed lines.
- Or we could remove the non-visible lines from the display.

- But removing the hidden lines also removes information about the shape of the back surfaces of an object, and wire-frame representations are generally used to get an indication of an object's overall appearance, front and back.
- A wire-frame object displayed with depth cueing, so that the brightness of lines decreases from the front of the object to the back.
- When a realistic view of a scene is to be produced, back parts of the objects are completely eliminated so that only the visible surfaces are displayed.
- In this case, surface-rendering procedures are applied so that screen pixels contain only the color patterns for the front surfaces.

V. Surface Rendering

- Added realism is attained in displays by rendering object surfaces using the lighting conditions in the scene and the assigned surface characteristics.
- We set the lighting conditions by specifying the color and location of the light sources, and we can also set background illumination effects.
- Surface properties of objects include whether a surface is transparent or opaque and whether the surface is smooth or rough.
- We set values for parameters to model surfaces such as glass, plastic, wood-grain patterns, and the bumpy appearance of an orange.



VI. Exploded and Cutaway Views

- Many graphics packages allow objects to be defined as hierarchical structures, so that internal details can be stored.

- Exploded and cutaway views of such objects can then be used to show the internal structure and relationship of the object parts.

- An alternative to exploding an object into its component parts is a cutaway view, which removes part of the visible surfaces to show internal structure.

VII. Three-dimensional and Stereoscopic Viewing

- Three-dimensional views can be obtained by reflecting a raster image from a vibrating, flexible mirror.
- The vibrations of the mirror are synchronized with the display of the scene on the cathode ray tube (CRT).
- As the mirror vibrates, the focal length varies so that each point in the scene is reflected to a spatial position corresponding to its depth.
- Stereoscopic devices present two views of a scene: one for the left eye and the other for the right eye.
- The viewing positions correspond to the eye positions of the viewer.
- These two views are typically displayed on alternate refresh cycles of a raster monitor.

The Three-dimensional Viewing Pipeline

- First of all, we need to choose a viewing position corresponding to where we would place a camera.
- We choose the viewing position according to whether we want to display a front, back, side, top, or bottom view of the scene.
- We could also pick a position in the middle of a group of objects or even inside a single object, such as a building or a molecule.
- Which way do we want to point the camera from the viewing position, and how should we rotate it around the line of sight to set the “up” direction for the picture?
- Some of the viewing operations for a three-dimensional scene are the same as, or similar to, those used in the two-dimensional viewing pipeline.
- A two-dimensional viewport is used to position a projected view of the three-dimensional scene on the output device, and a two-dimensional clipping window is used to select a view that is to be mapped to the viewport.

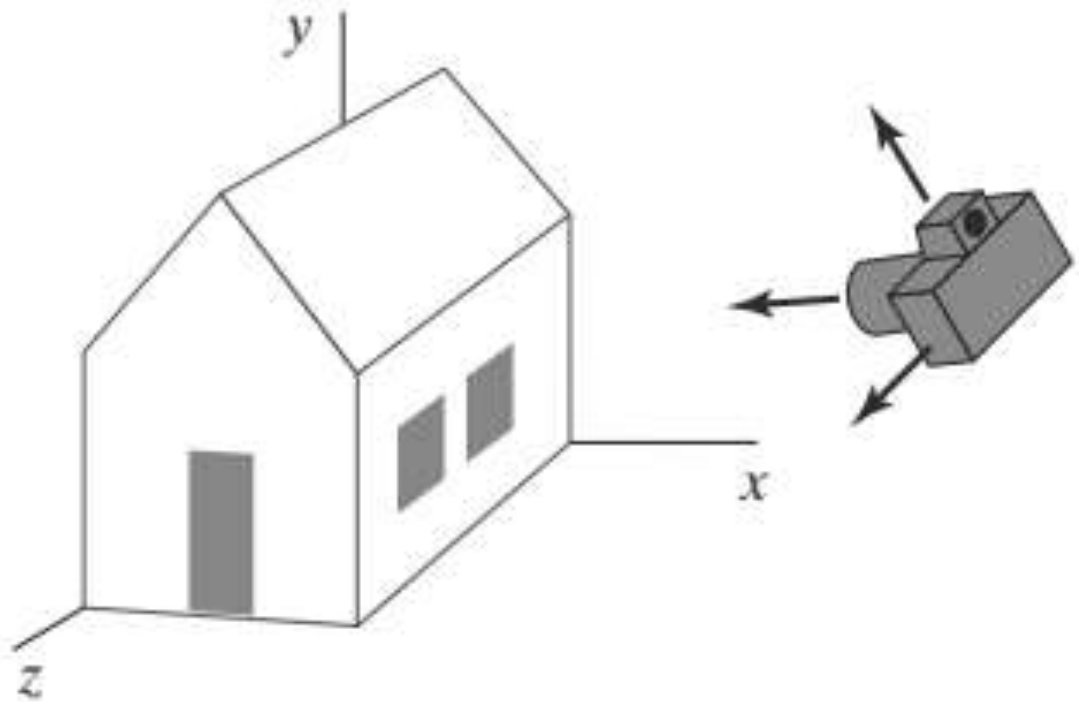
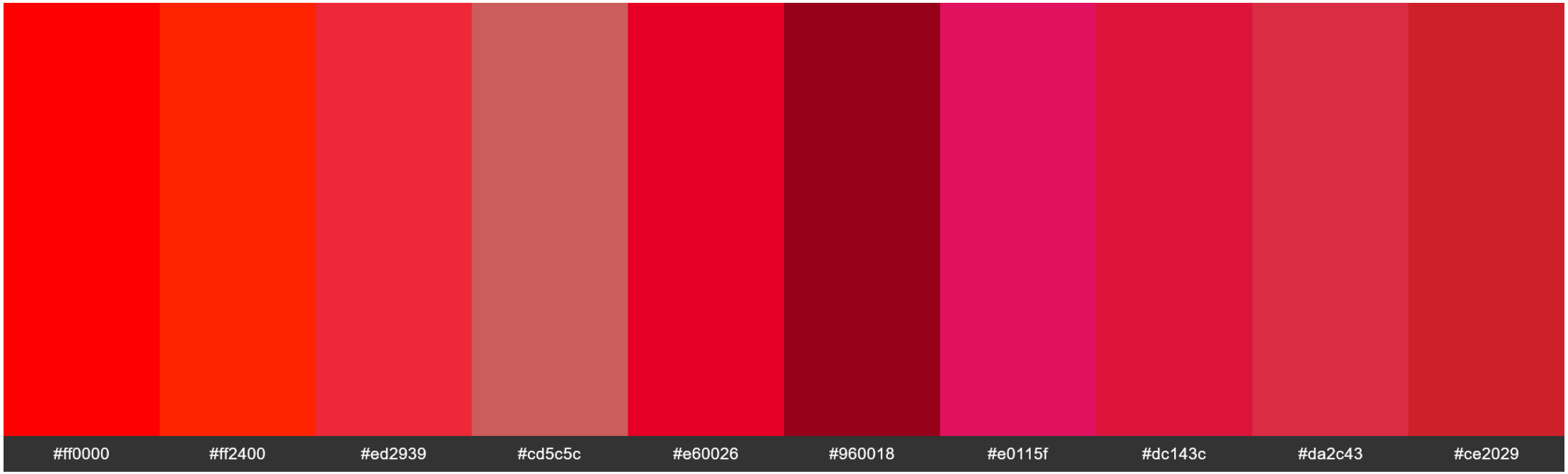
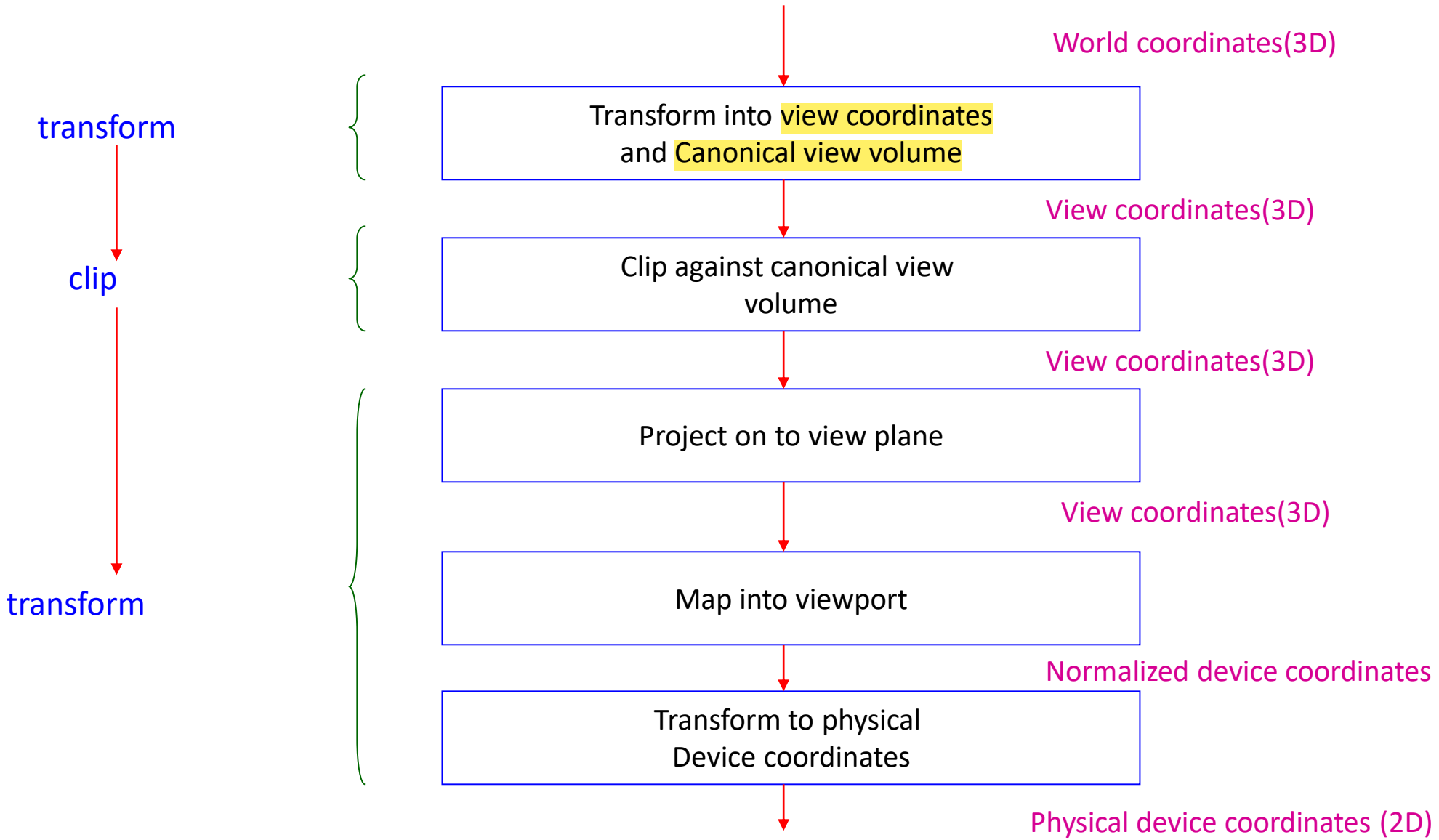


FIGURE 5
Photographing a scene involves selection of the camera position and orientation.


#ff0000
#ff2400
#ed2939
#cd5c5c
#e60026
#960018
#e0115f
#dc143c
#da2c43
#ce2029

Three-dimensional Viewing Pipeline



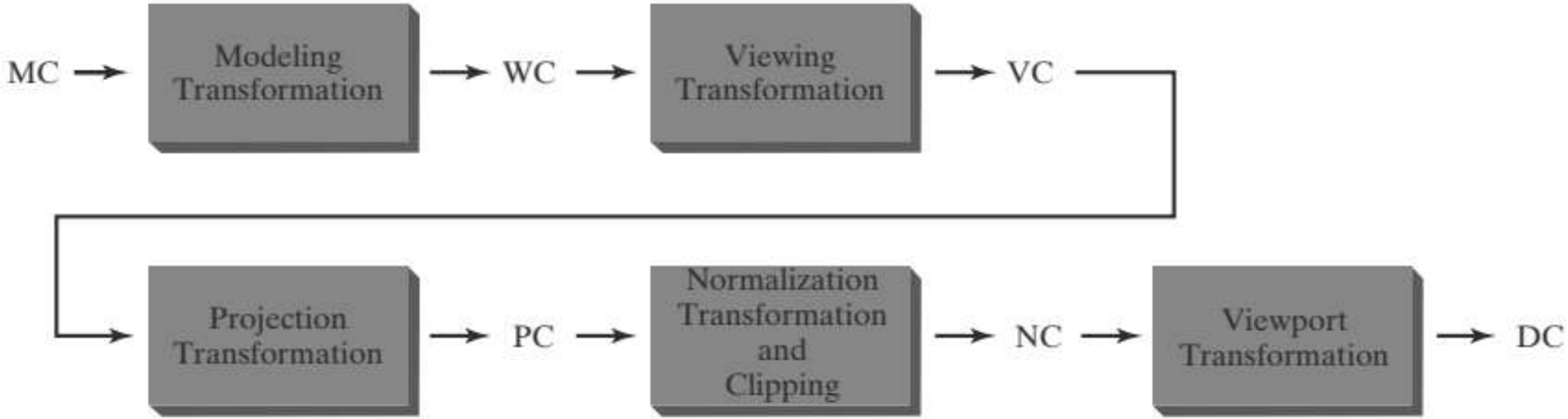


FIGURE 10-6
General three-dimensional transformation pipeline, from modeling coordinates (MC) to world coordinates (WC) to viewing coordinates (VC) to projection coordinates (PC) to normalized coordinates (NC) and, ultimately, to device coordinates (DC).

- Clipping windows, viewports, and display windows are usually specified as rectangles with their edges parallel to the coordinate axes.
- In three-dimensional viewing, however, the clipping window is positioned on a selected view plane, and scenes are clipped against an enclosing volume of space, which is defined by a set of *clipping planes*.
- The viewing position, view plane, clipping window, and clipping planes are all specified within the viewing-coordinate reference frame.
- 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,** and its shape and size depends on the dimensions of the clipping window, the type of projection we choose, and the selected limiting positions along the viewing direction.

- Projection operations are performed to convert the viewing-coordinate description of the scene to coordinate positions on the projection plane.
- 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 (from world coordinates to normalized coordinates) are completed.
- In this way, the coordinate transformations can be concatenated for maximum efficiency.
- Scene descriptions in device coordinates are sometimes expressed in a left-handed reference frame so that positive distances from the display screen can be used to measure depth values in the scene.

Three-dimensional Viewing-Coordinate Parameters

- Establishing a three-dimensional viewing reference frame is similar to setting up the two-dimensional viewing reference frame.

- We first select a world-coordinate position **P**0 =(x0, y0, z0) for the viewing origin, which is called the **view point** or **viewing position**. (Sometimes the view point is also referred to as the *eye position* or the *camera position*.)

- And we specify a **view-up vector V,** which defines the *y*view direction.

- For three-dimensional space, we also need to assign a direction for one of the remaining two coordinate axes.

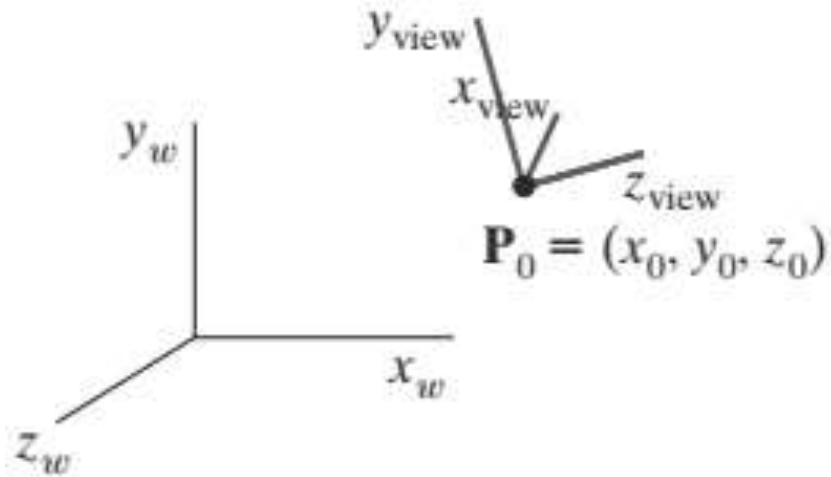


FIGURE 7
A right-handed viewing-coordinate system, with axes x_{view}, y_{view}, and z_{view}, relative to a right-handed world-coordinate frame.

A. 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 defined with a **view-plane normal vector N.**

- An additional scalar parameter is used to set the position of the view plane at some coordinate value *zvp* along the *z*view axis.

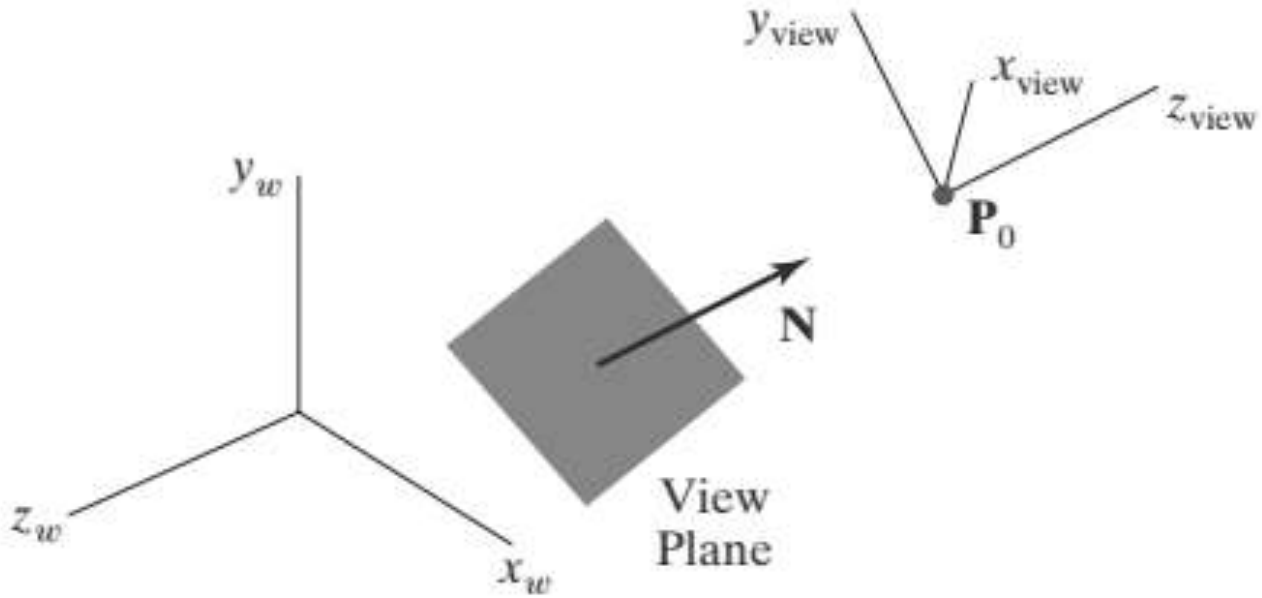


FIGURE 8
Orientation of the view plane and view-plane normal vector **N**.

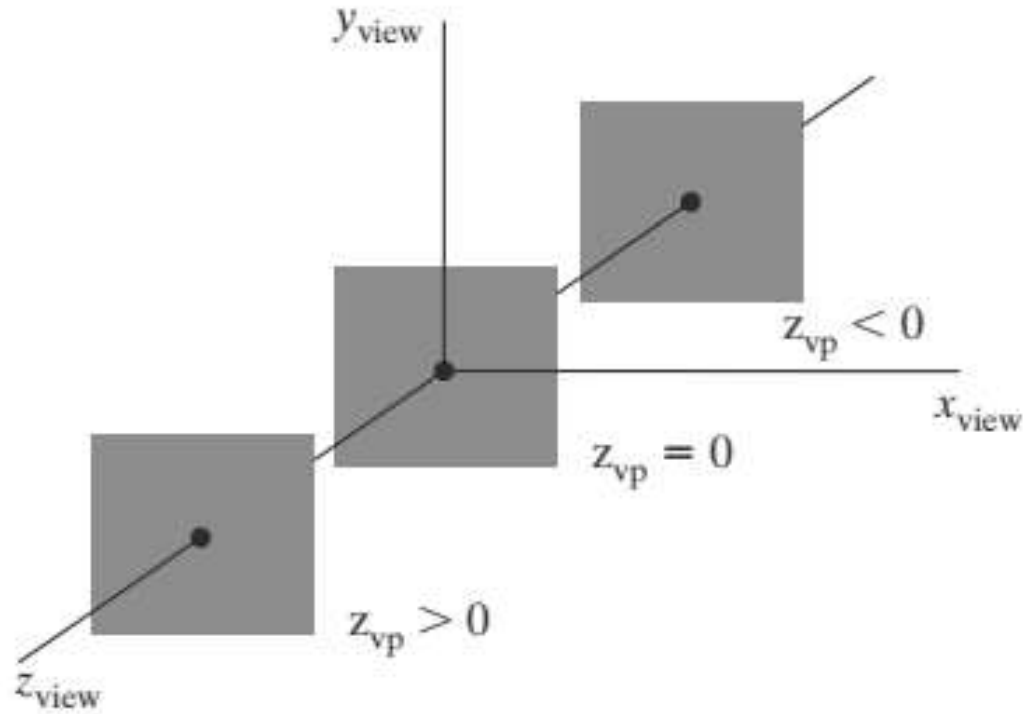


FIGURE 9
Three possible positions for the view plane along the z_{view} axis.

- This parameter value is usually specified as a distance from the viewing origin along the direction of viewing, which is often taken to be in the negative *z*view direction.
- Thus, the view plane is always parallel to the *x*view, *y*view plane, and the projection of objects to the view plane corresponds to the view of the scene that will be displayed on the output device.
- Vector **N** can be specified in various ways. In some graphics systems, the direction for **N** is defined to be along the line from the world-coordinate origin to a selected point position.
- Other systems take **N** to be in the direction from a reference point **P**ref to the viewing origin **P**0, as in Figure 10. In this case, the reference point is often referred to as a *look-at point* within the scene, with the viewing direction opposite to the direction of **N**.
- We could also define the view-plane normal vector, and other vector directions, using *direction angles*.
- These are the three angles,α,β, andγ, that a spatial line makes with the *x*, *y*, and *z* axes, respectively. But it is usually much easier to specify a vector direction with two point positions in a scene than with direction angles.

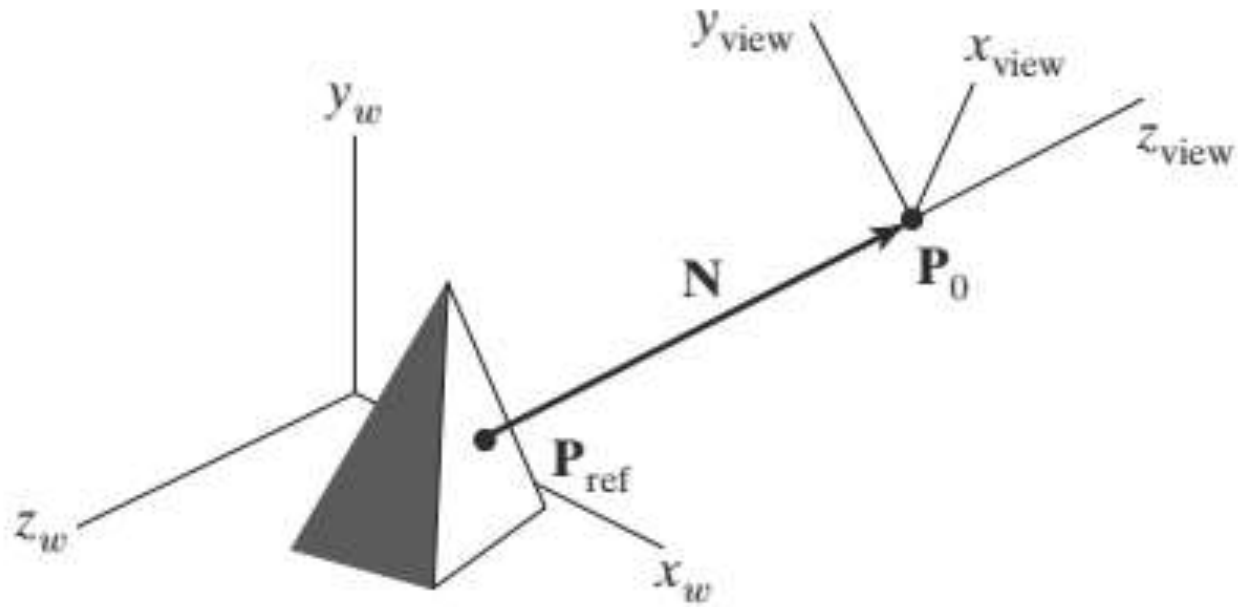


FIGURE 10
Specifying the view-plane normal vector **N** as the direction from a selected reference point $\mathbf{P}_{ref}$ to the viewing-coordinate origin $\mathbf{P}_0$.

B. The 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**.

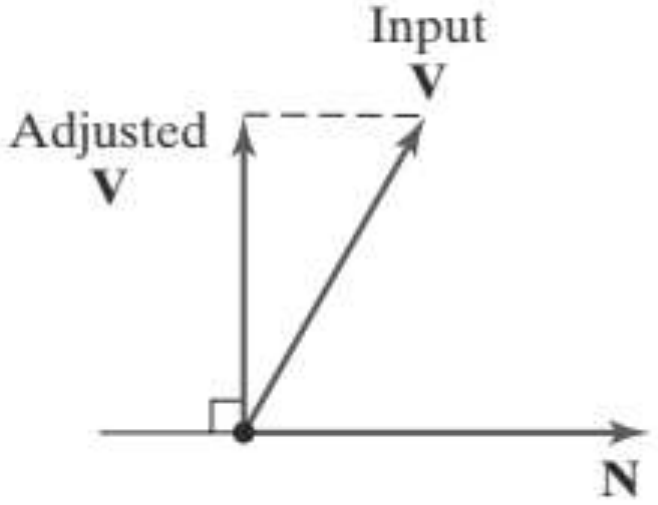


FIGURE 11
Adjusting the input direction of the view-up vector **V** to an orientation perpendicular to the view-plane normal vector **N**.

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

- Therefore, viewing routines typically adjust the user-defined orientation of vector **V**, as shown in Figure 11, so that **V** is projected onto a plane that is perpendicular to the view-plane normal vector.

- We can choose any direction for the view-up vector **V**, so long as it is not parallel to **N**.

- A convenient choice is often in a direction parallel to the world *yw* axis; that is, we could set **V** = *(*0, 1, 0*)*.

C. The uvn Viewing-Coordinate Reference Frame

- Left-handed viewing coordinates are sometimes used in graphics packages, with the viewing direction in the positive *z*view direction.
- With a left-handed system, increasing *z*view values are interpreted as being farther from the viewing position along the line of sight.
- But right-handed viewing systems are more common, because they have the same orientation as the world-reference frame.
- This allows a graphics package to deal with only one coordinate orientation for both world and viewing references.
- Because the view-plane normal **N** defines the direction for the *z*view axis and the view-up vector **V** is used to obtain the direction for the *y*view axis, we need only determine the direction for the *x*view axis.

- We determine the correct direction for **U** by taking the vector cross product of **V** and **N** so as to form a right-handed viewing frame.
- The vector cross product of **N** and **U** also produces the adjusted value for **V**, perpendicular to both **N** and **U**, along the positive *y*view axis. Following these procedures, we obtain the following set of unit axis vectors for a right-handed viewing coordinate system.

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

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

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

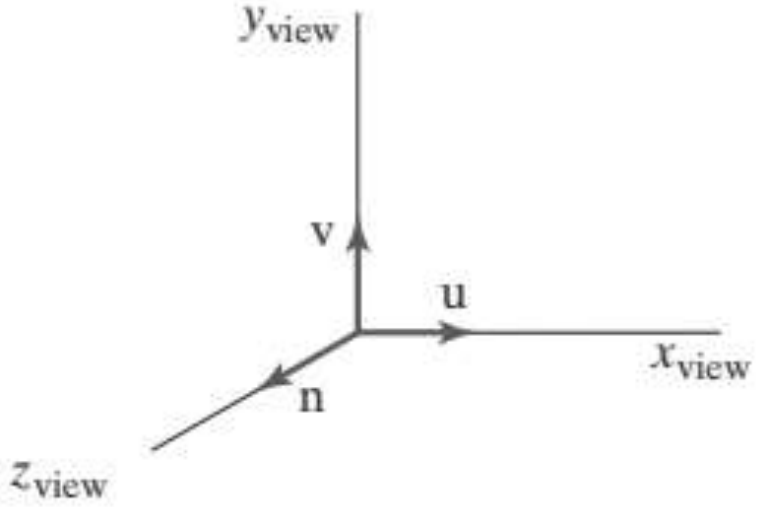


FIGURE 12
A right-handed viewing system defined with unit vectors **u**, **v**, and **n**.

D. Generating Three-dimensional Viewing Effects

- For instance, from a fixed viewing position, we could change the direction of **N** to display objects at positions around the viewing-coordinate origin.
- We could also vary **N** to create a composite display consisting of multiple views from a fixed camera position.
- We can simulate a wide viewing angle by producing seven views of the scene from the same viewing position, but with slight shifts in the viewing direction; the views are then combined to form a composite display.
- Similarly, we generate stereoscopic views by shifting the viewing direction as well as shifting the view point slightly to simulate the two eye positions.

- If we want to simulate an animation panning effect, as when a camera moves through a scene or follows an object that is moving through a scene, we can keep the direction for **N** fixed as we move the view point, as illustrated in Figure 13.

- And to display different views of an object, such as a side view and a front view, we could move the view point around the object, as in Figure 14.

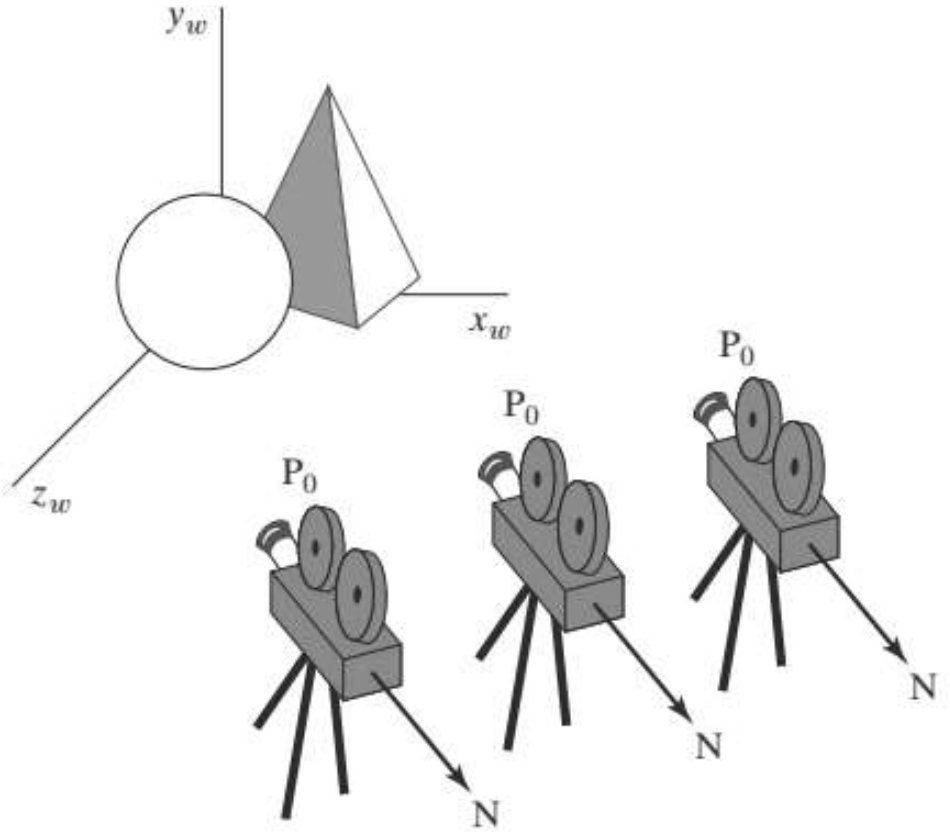


FIGURE 13
Panning across a scene by changing the viewing position, with a fixed direction for **N**.

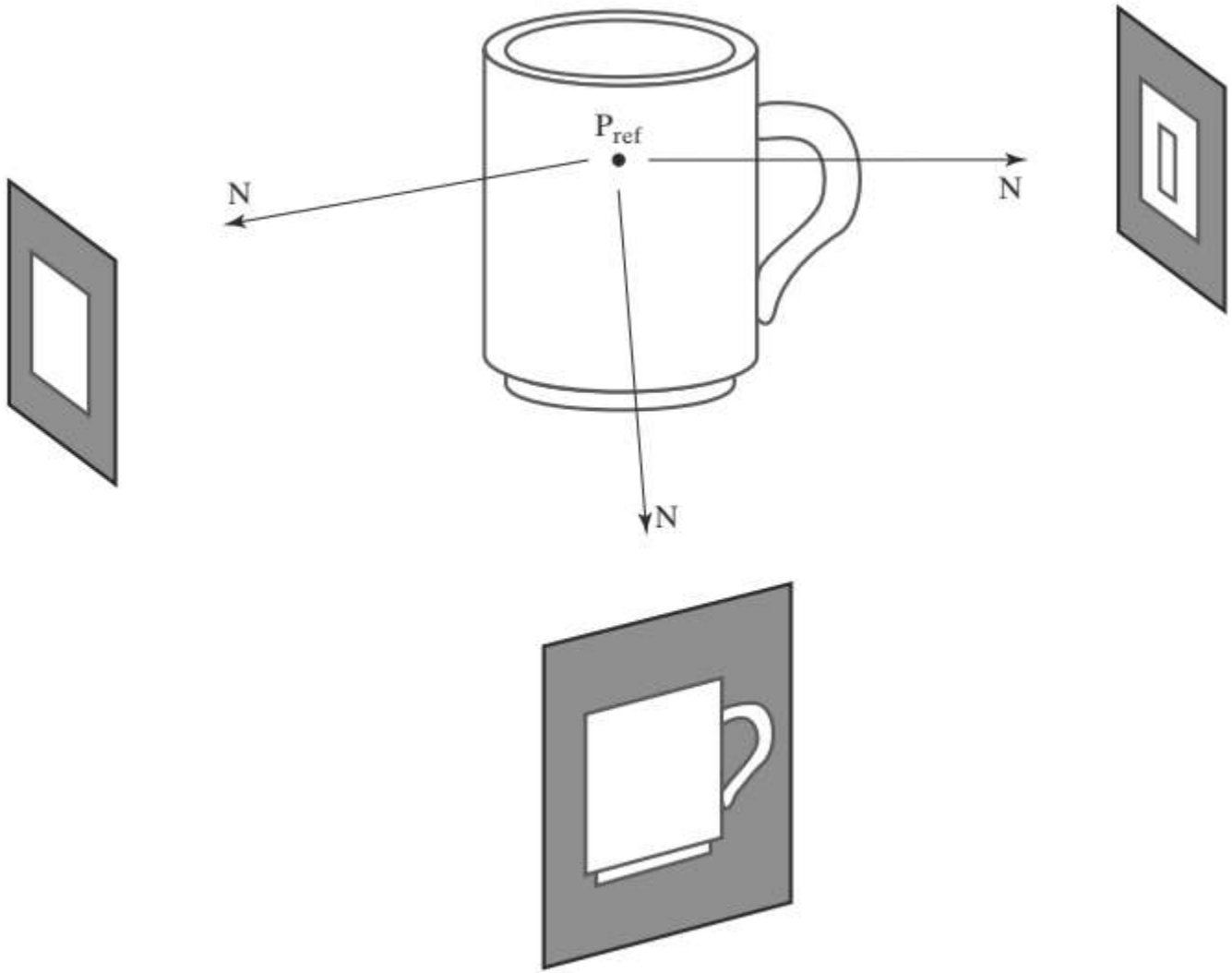


FIGURE 14
Viewing an object from different directions using a fixed reference point.

Transformation from World to Viewing Coordinates

- 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.
- We can accomplish this conversion using the methods for transforming between coordinate system :

  1. Translate the viewing-coordinate origin to the origin of the world-coordinate system.

  2. Apply rotations to align the *x*view, *y*view, and *z*view axes with the world *xw*, *yw*, and *zw* axes, respectively.

The viewing-coordinate origin is at world position $\mathbf{P_0} = (x_0, y_0, z_0)$. Therefore, the matrix for translating the viewing origin to the world origin is

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

For the rotation transformation, we can use the unit vectors **u**, **v**, and **n** to form the composite rotation matrix that superimposes the viewing axes onto the world frame. This transformation matrix is

$$\mathbf{R} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

where the elements of matrix **R** are the components of the **uvn** axis vectors.

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

$$\begin{aligned} \mathbf{M}_{WC,\,VC} &= \mathbf{R} \cdot \mathbf{T} \\ &= \begin{bmatrix} u_x & u_y & u_z & -\mathbf{u} \cdot \mathbf{P}_0 \\ v_x & v_y & v_z & -\mathbf{v} \cdot \mathbf{P}_0 \\ n_x & n_y & n_z & -\mathbf{n} \cdot \mathbf{P}_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{4}$$

Translation factors in this matrix are calculated as the vector dot product of each of the **u**, **v**, and **n** unit vectors with $\mathbf{P}_0$, which represents a vector from the world origin to the viewing origin. In other words, the translation factors are the negative projections of $\mathbf{P}_0$ on each of the viewing-coordinate axes (the negative components of $\mathbf{P}_0$ in viewing coordinates). These matrix elements are evaluated as

$$\begin{aligned} -\mathbf{u} \cdot \mathbf{P}_0 &= -x_0 u_x - y_0 u_y - z_0 u_z \\ -\mathbf{v} \cdot \mathbf{P}_0 &= -x_0 v_x - y_0 v_y - z_0 v_z \\ -\mathbf{n} \cdot \mathbf{P}_0 &= -x_0 n_x - y_0 n_y - z_0 n_z \end{aligned} \tag{5}$$

Matrix 4 transfers world-coordinate object descriptions to the viewing reference frame.

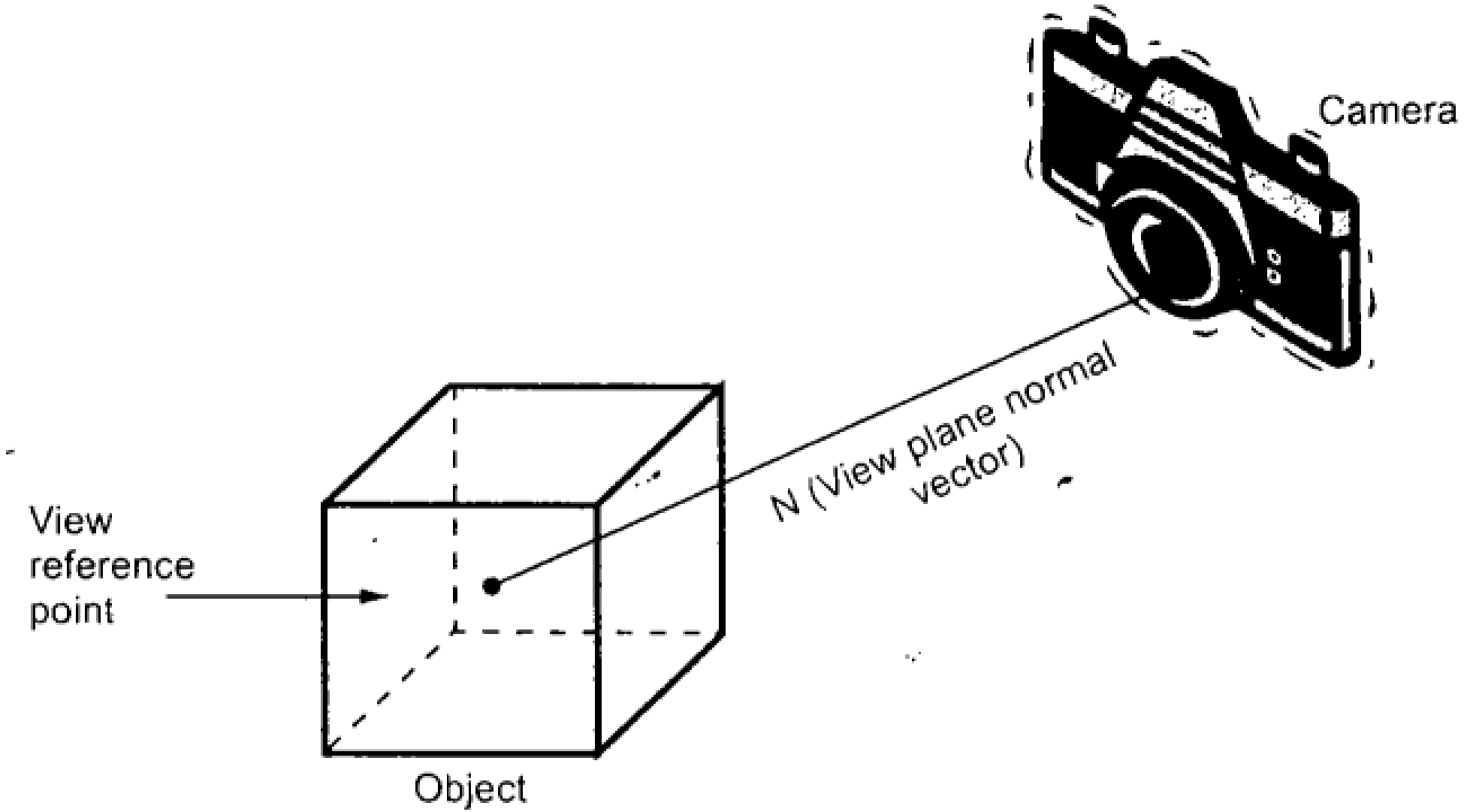
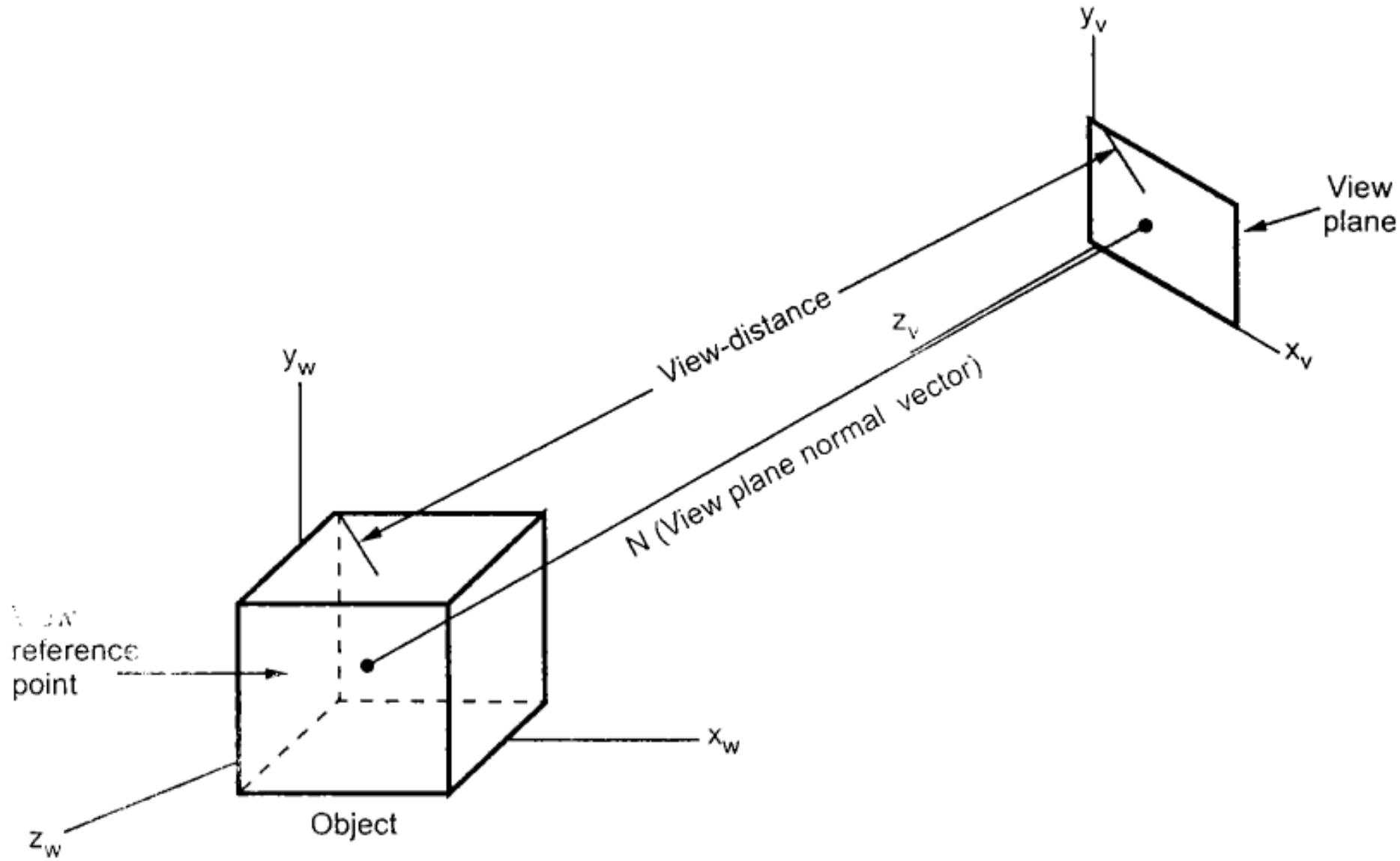


Fig. 7.3 View reference point and view plane normal vector


y_v
View plane
View-distance
z_v
x_v
y_w
N (View plane normal vector)
reference point
x_w
Object
z_w

Projection Transformations

- In the next phase of the three-dimensional viewing pipeline, after the transformation to viewing coordinates, object descriptions are projected to the view plane.
- Graphics packages generally support both parallel and perspective projections.
- In a **parallel projection,** coordinate positions are transferred 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.

- There are two general methods for obtaining a parallel-projection view of an object: We can project along lines that are perpendicular to the view plane, or we can project at an oblique angle to the view plane.

- For a **perspective projection,** object positions are transformed to projection coordinates along lines that converge to a point behind the view plane.

- Unlike a parallel projection, a perspective projection does not preserve relative proportions of objects.

- But perspective views of a scene are more realistic because distant objects in the projected display are reduced in size.

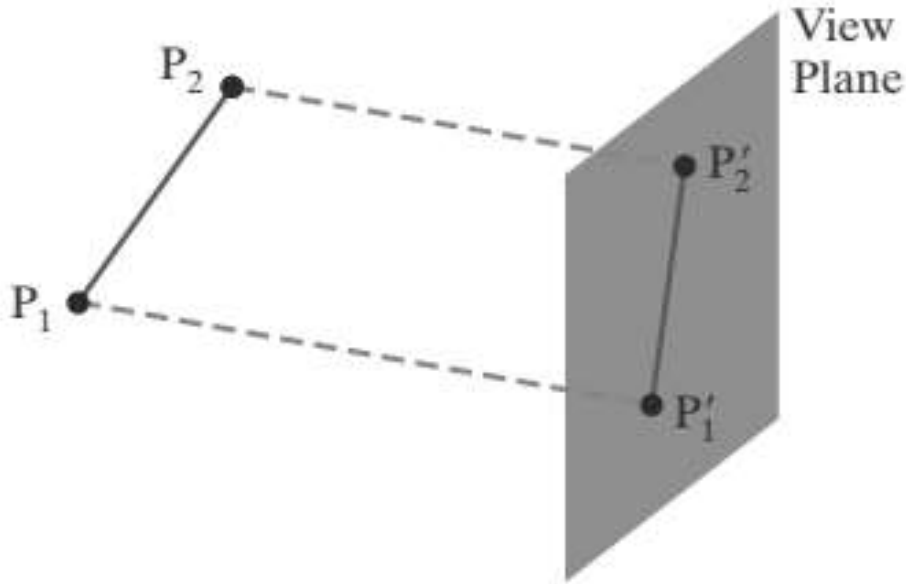


FIGURE 15
Parallel projection of a line segment onto a view plane.

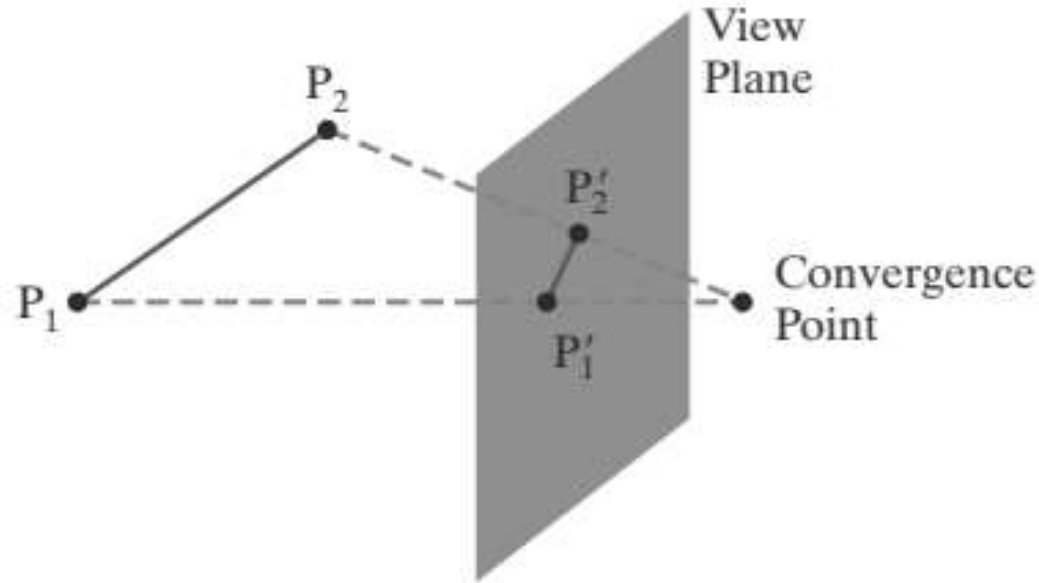
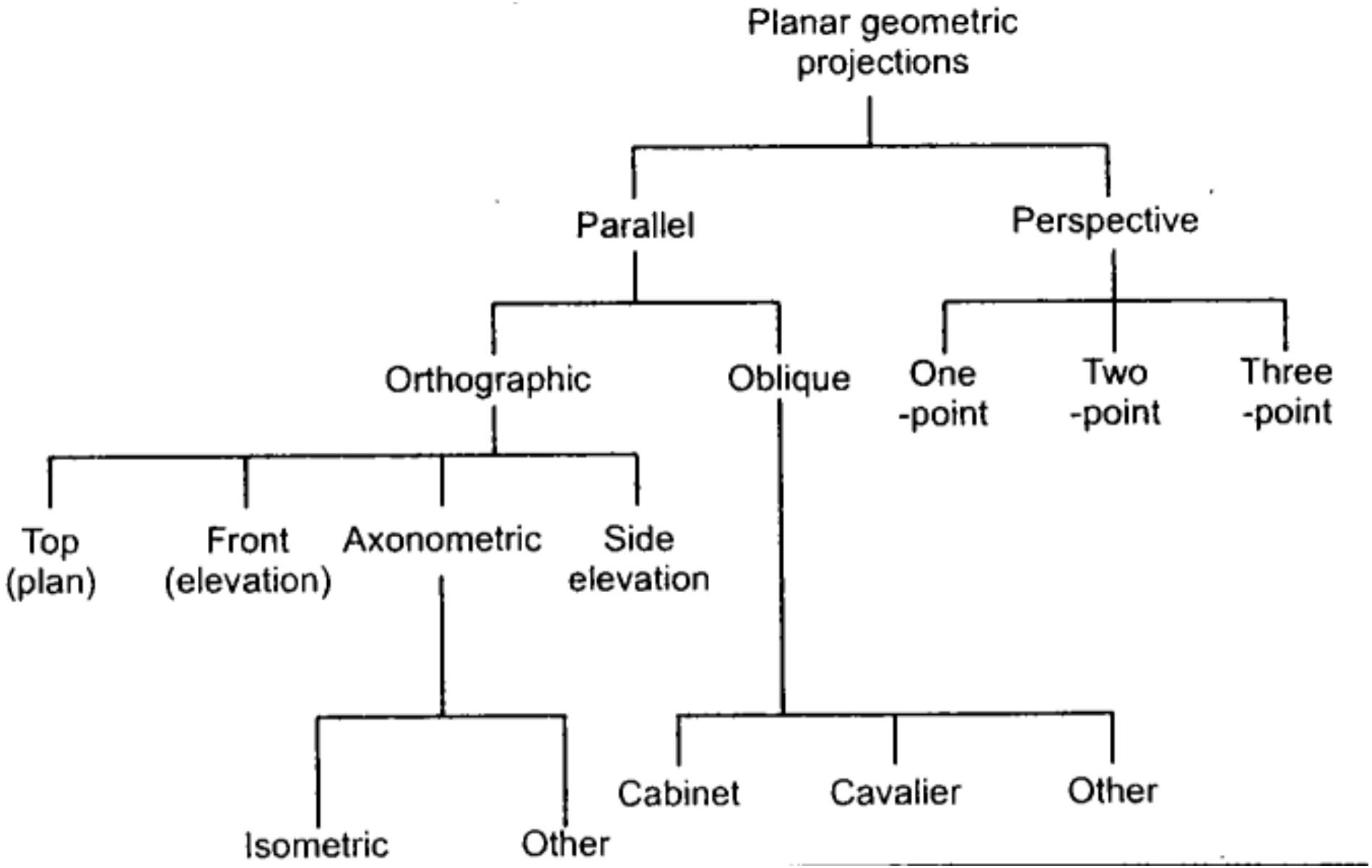
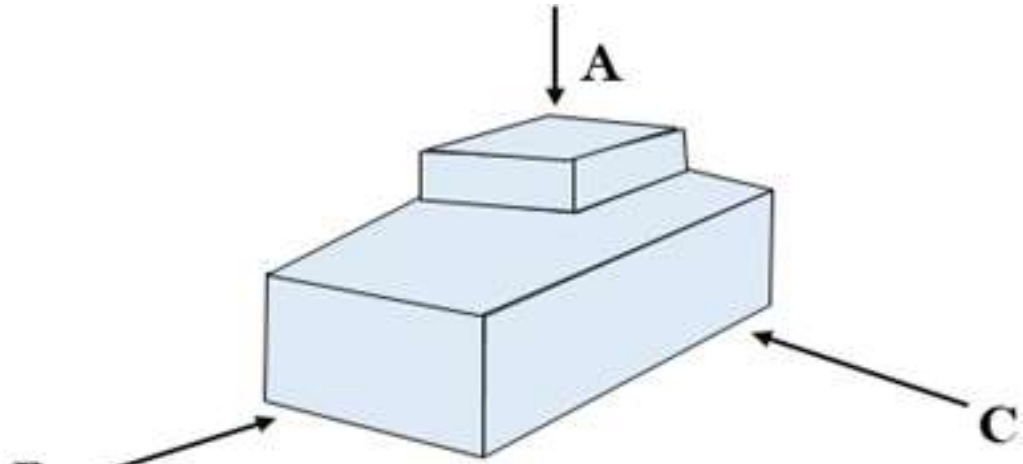


FIGURE 16
Perspective projection of a line segment onto a view plane.

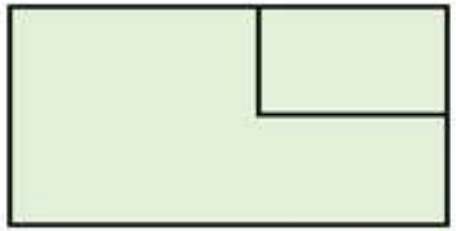

Planar geometric projections
Parallel
Perspective
Orthographic
Oblique
One -point
Two -point
Three -point
Top (plan)
Front (elevation)
Axonometric
Side elevation
Isometric
Other
Cabinet
Cavalier
Other

1. Parallel Projections

- Parallel Projection use to display picture in its true shape and size. When projectors are perpendicular to view plane then is called **orthographic projection**.
- The parallel projection is formed by extending parallel lines from each vertex on the object until they intersect the plane of the screen.
- The point of intersection is the projection of vertex.
- Parallel projections are used by architects and engineers for creating working drawing of the object, for complete representations require two or more views of an object using different planes.

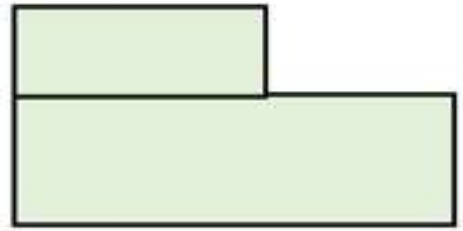


(a)



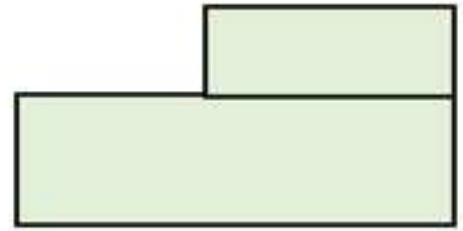
Parallel projection from top i.e. A direction

(b)



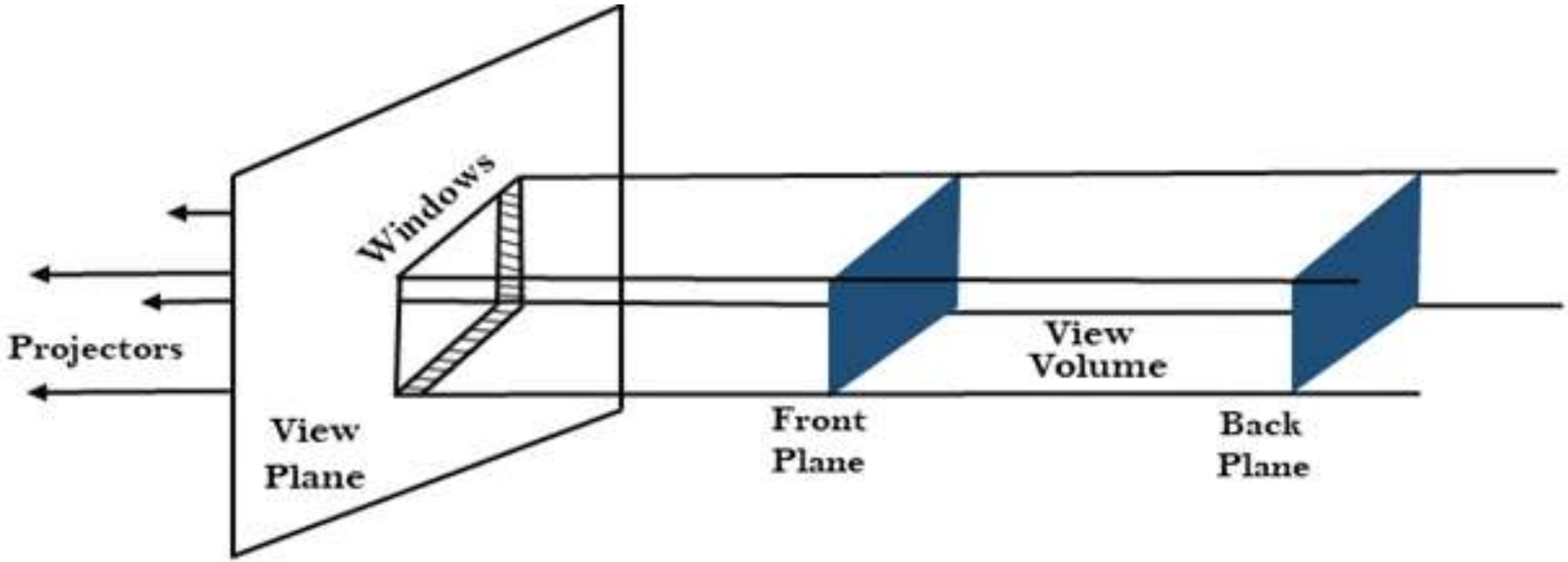
Parallel projection from top i.e. B direction

(c)



Parallel projection from top i.e. C direction

(d)



(a) Viewing Volume in orthographic projection

1.1. Orthogonal Projections

- 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** (or, equivalently, an **orthographic projection**).

- This produces a parallel-projection transformation in which the projection lines are perpendicular to the view plane.

- Orthogonal projections are most often used to produce the front, side, and top views of an object.

- **Elevations** and **plan view**: Front, side, and rear orthogonal projections of an object are called *elevations;* and a top orthogonal projection is called a *plan view*.

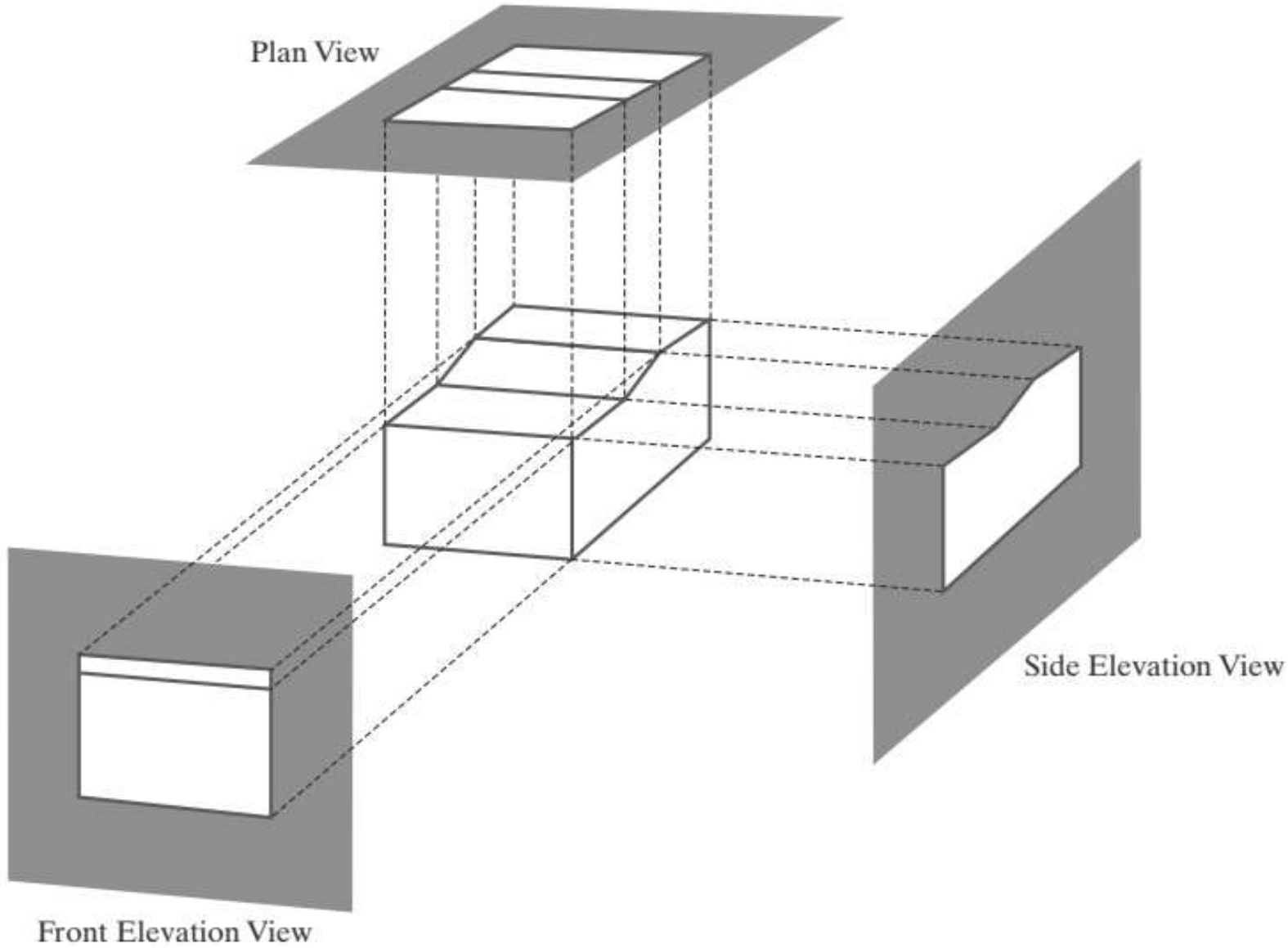


FIGURE 17
Orthogonal projections of an object, displaying plan and elevation views.

- Mostly used by drafters and engineers to create working drawings of an object which preserves its scale and shape.

- The distance between the COP and the projection plane is infinite i.e. The projectors are parallel to each other and have a fixed direction.

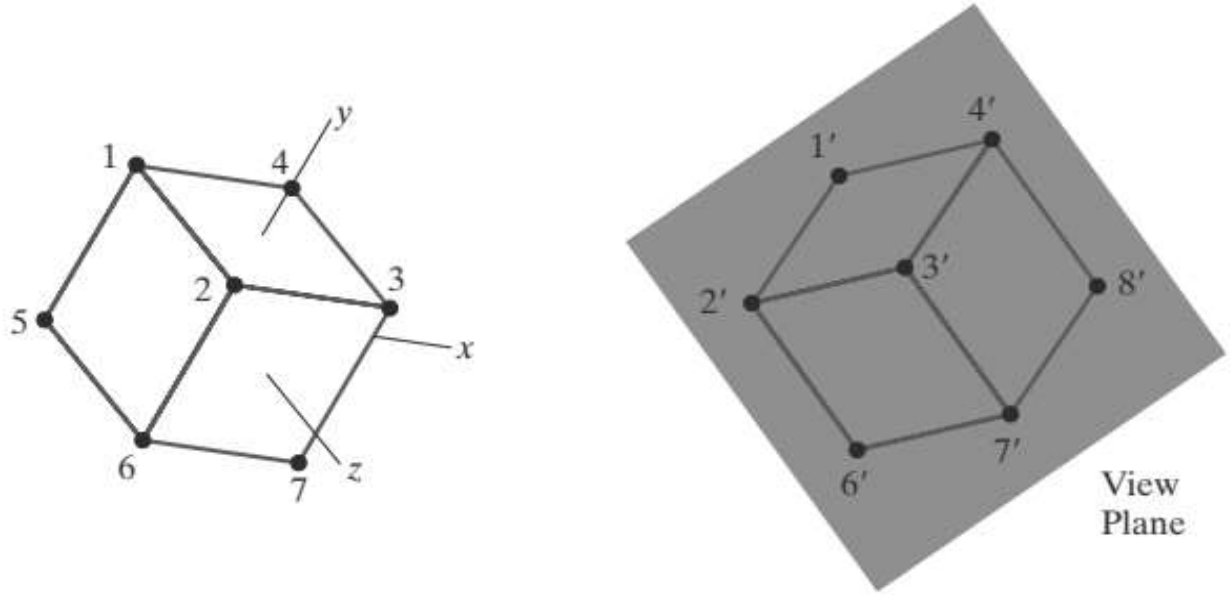


FIGURE 18
An isometric projection of a cube.

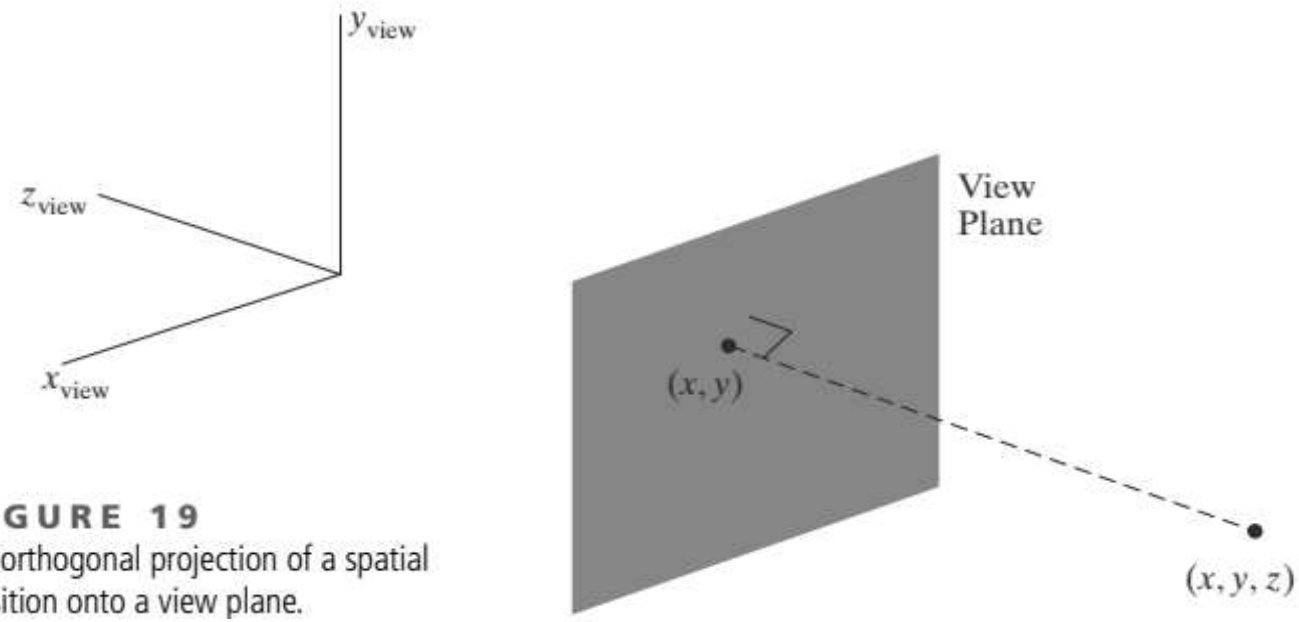


FIGURE 19
An orthogonal projection of a spatial position onto a view plane.

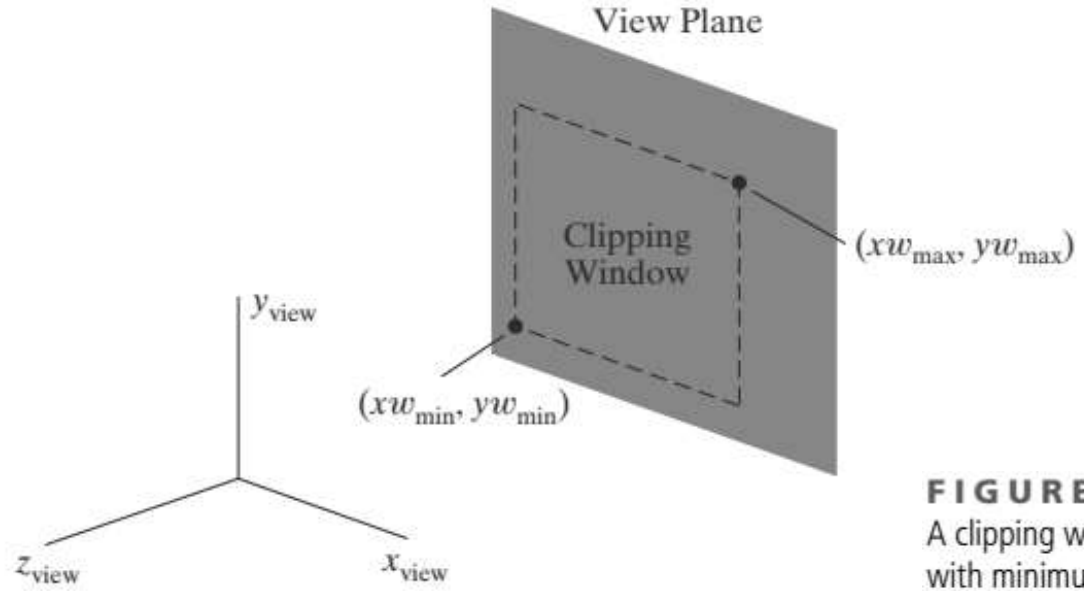


FIGURE 20
A clipping window on the view plane, with minimum and maximum coordinates given in the viewing reference system.

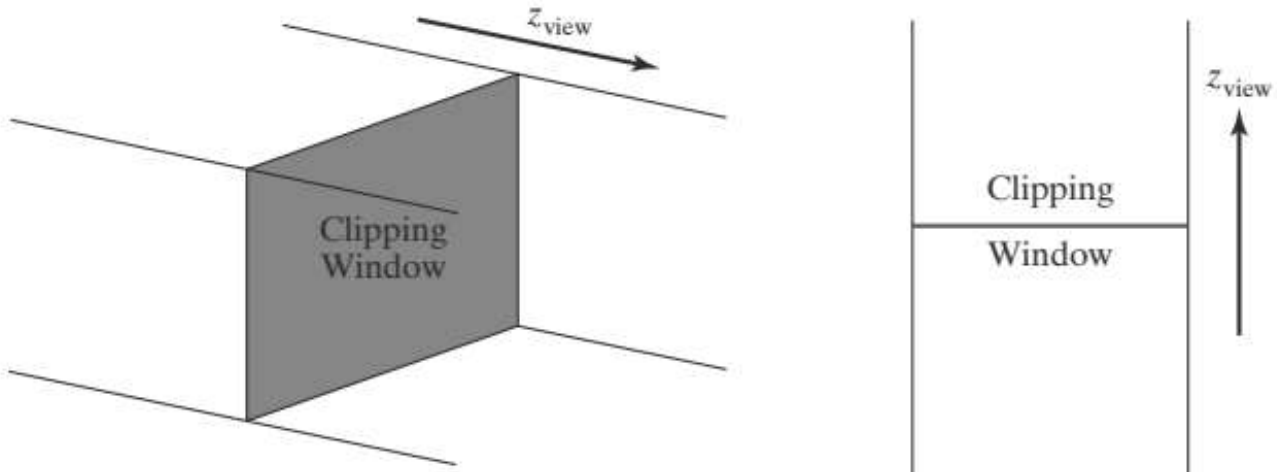


FIGURE 21
Infinite orthogonal-projection view volume.

Normalization Transformation for an Orthogonal Projection

- Using an orthogonal transfer of coordinate positions onto the view plane, we obtain the projected position of any spatial point *(x, y, z)* as simply *(x, y)*.
- Thus, 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.

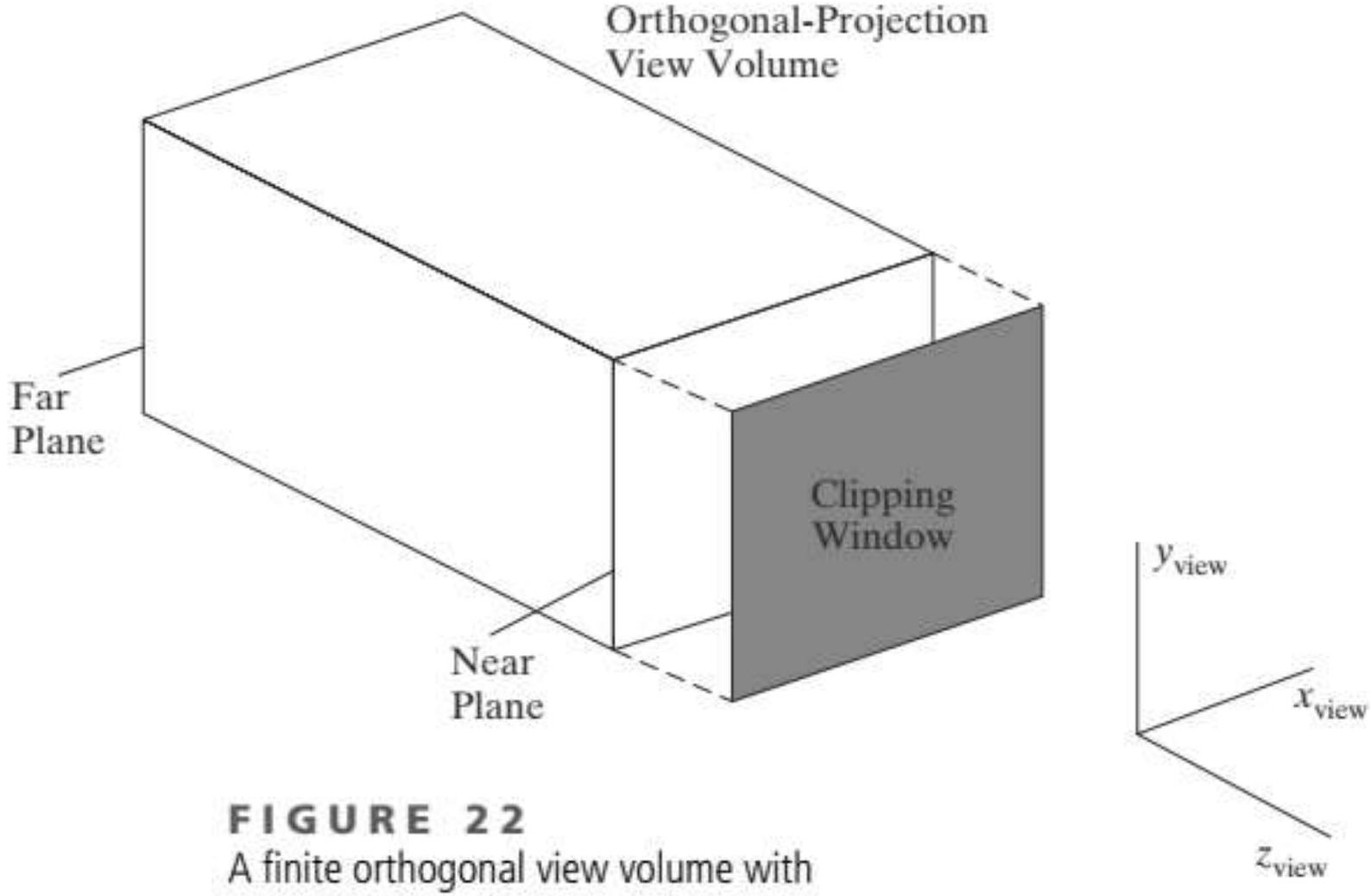


FIGURE 22
A finite orthogonal view volume with the view plane "in front" of the near plane.

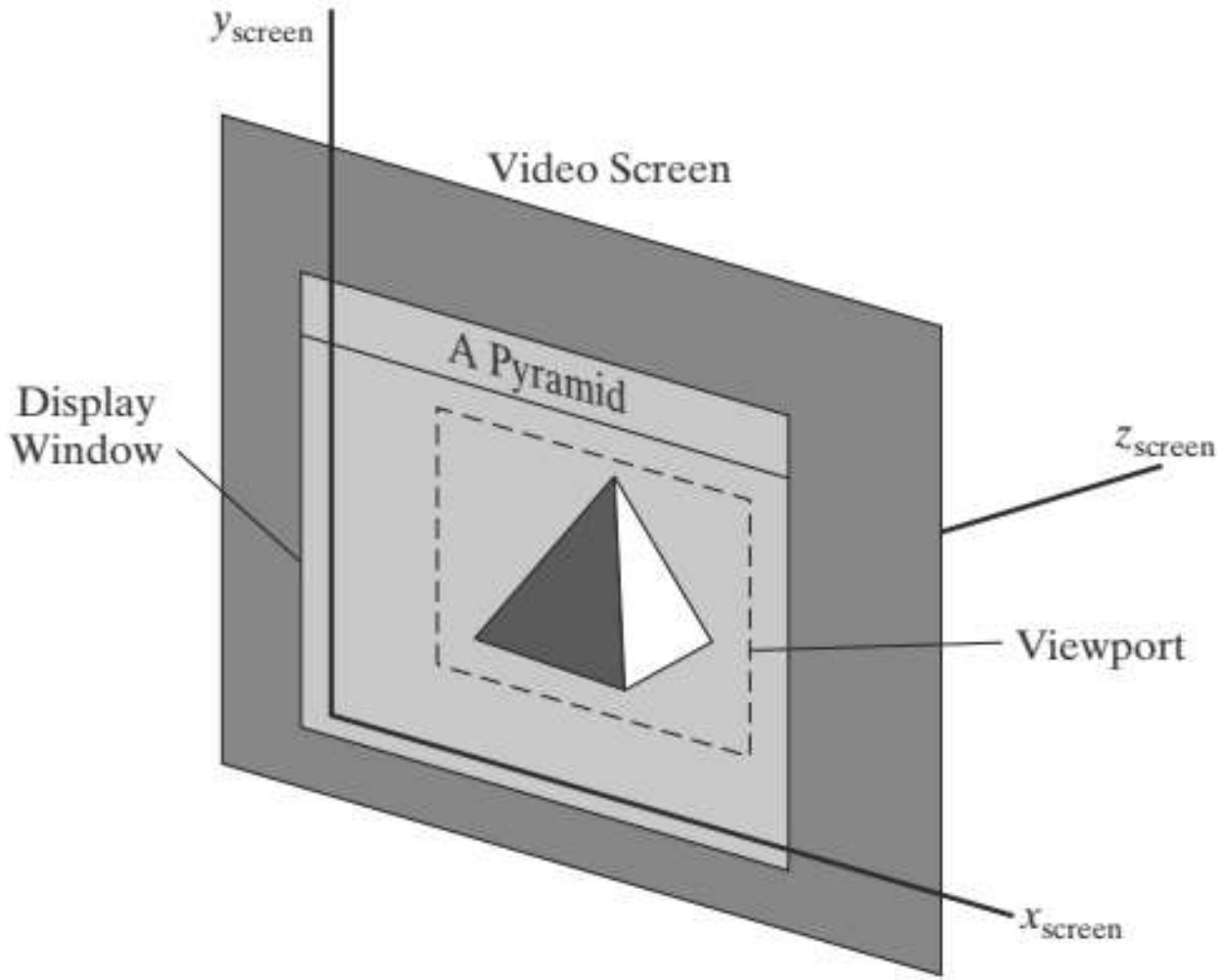


FIGURE 23
A left-handed screen-coordinate reference frame.

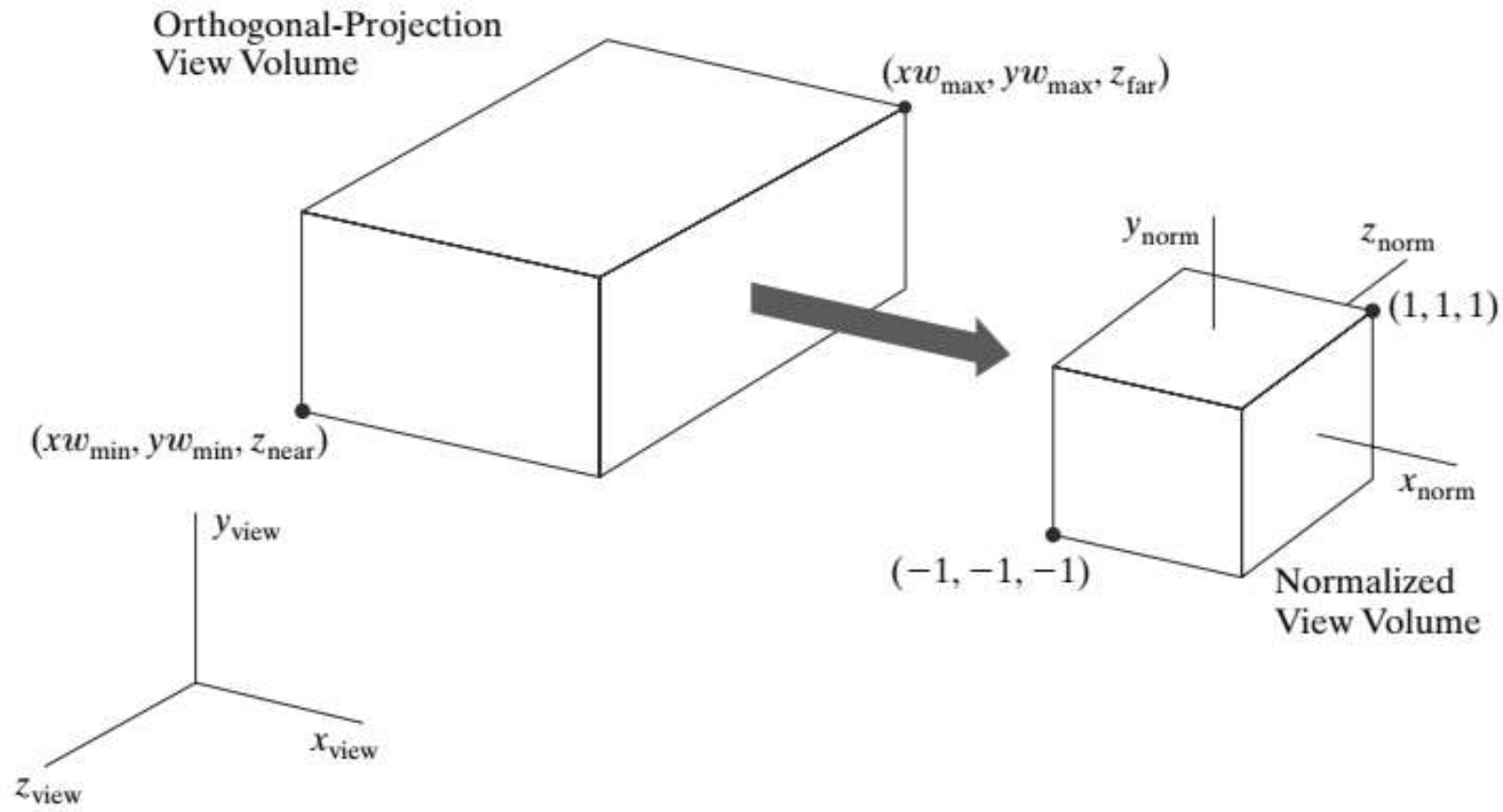


FIGURE 24
Normalization transformation from an orthogonal-projection view volume to the symmetric normalization cube within a left-handed reference frame.

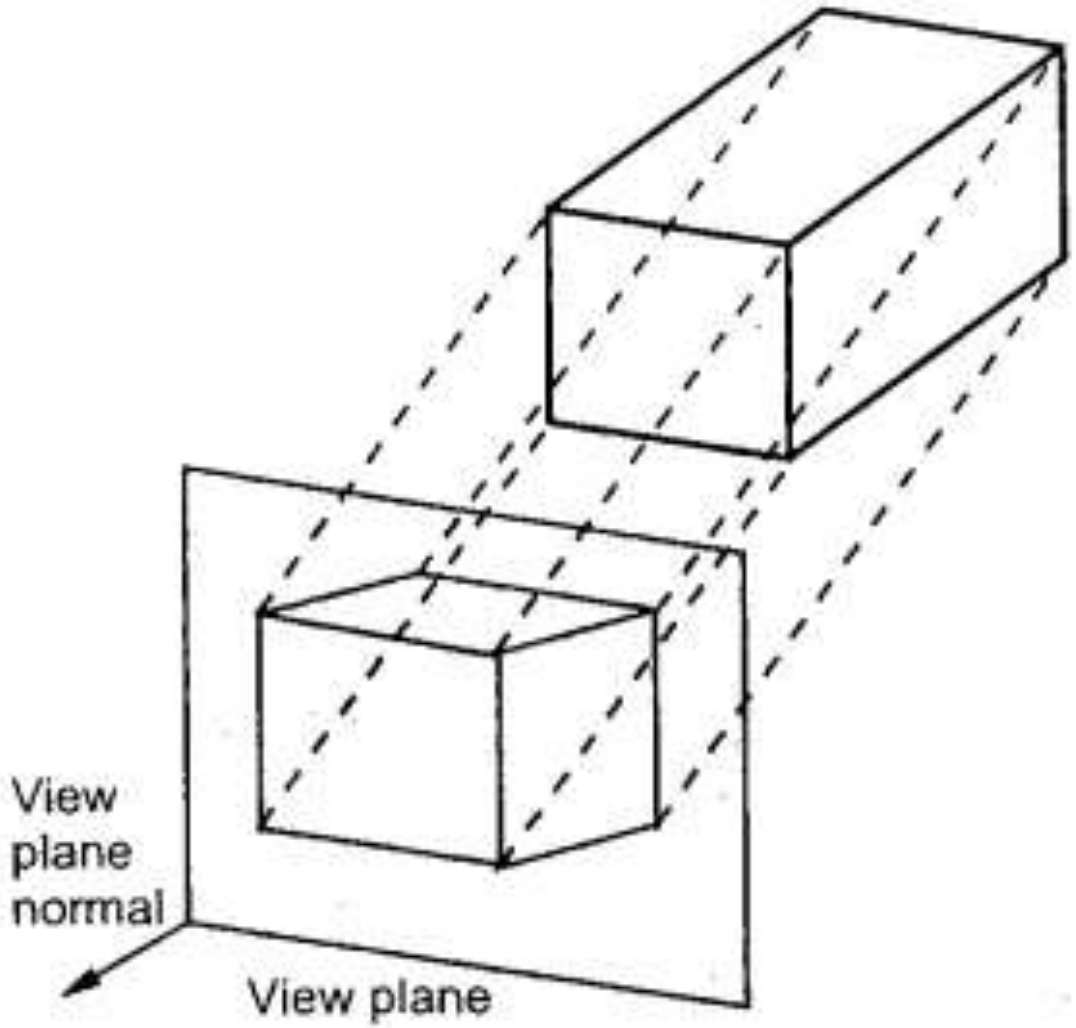
- To illustrate the normalization transformation, we assume that the orthogonal-projection view volume is to be mapped into the symmetric normalization cube within a left-handed reference frame.
- Also, *z*-coordinate positions for the near and far planes are denoted as *z*near and *z*far, respectively. Figure 24 illustrates this normalization transformation. Position *(x*min, *y*min, *z*near*)* is mapped to the normalized position *(*-1, -1, -1*)*, and position *(x*max, *y*max, *z*far*)* is mapped to *(*1, 1, 1*)*.
- The normalization transformation for the orthogonal view volume is:

Write on desk

$$\mathbf{M}_{\text{ortho,norm}} = \begin{bmatrix} \frac{2}{xw_{\max} - xw_{\min}} & 0 & 0 & -\frac{xw_{\max} + xw_{\min}}{xw_{\max} - xw_{\min}} \\ 0 & \frac{2}{yw_{\max} - yw_{\min}} & 0 & -\frac{yw_{\max} + yw_{\min}}{yw_{\max} - yw_{\min}} \\ 0 & 0 & \frac{-2}{z_{\text{near}} - z_{\text{far}}} & \frac{z_{\text{near}} + z_{\text{far}}}{z_{\text{near}} - z_{\text{far}}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

1.2. Oblique projection

- An oblique projection is obtained by projecting points along parallel lines that are not perpendicular to the projection plane.
- The view plane normal and the direction of projection are not the same.
- The oblique projections are further classified as the cavalier and cabinet projections.
- For the cavalier projections, the direction of projection makes a 45 degree angle with the view plane.
- The projection of a line perpendicular to the view plane has the same length as the line itself; that is, there is no foreshortening.



Write on Desk

$$\mathbf{M}_{\text{oblique}} = \begin{bmatrix} 1 & 0 & -\frac{V_{px}}{V_{pz}} & z_{vp}\frac{V_{px}}{V_{pz}} \\ 0 & 1 & -\frac{V_{py}}{V_{pz}} & z_{vp}\frac{V_{py}}{V_{pz}} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

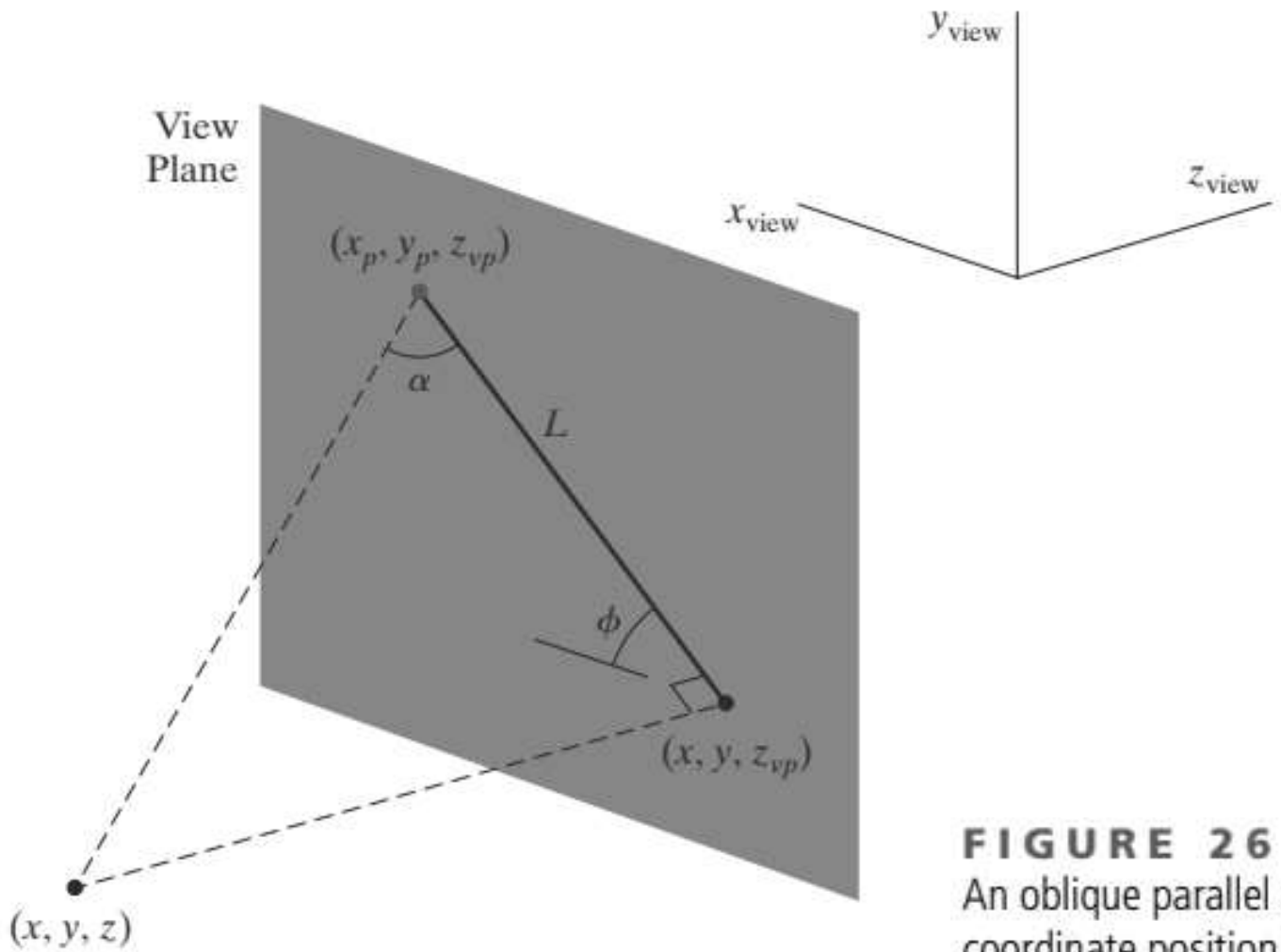


FIGURE 26
An oblique parallel projection of coordinate position (x, y, z) to position (x_p, y_p, z_{vp}) on a projection plane at position z_{vp} along the z_{view} axis.

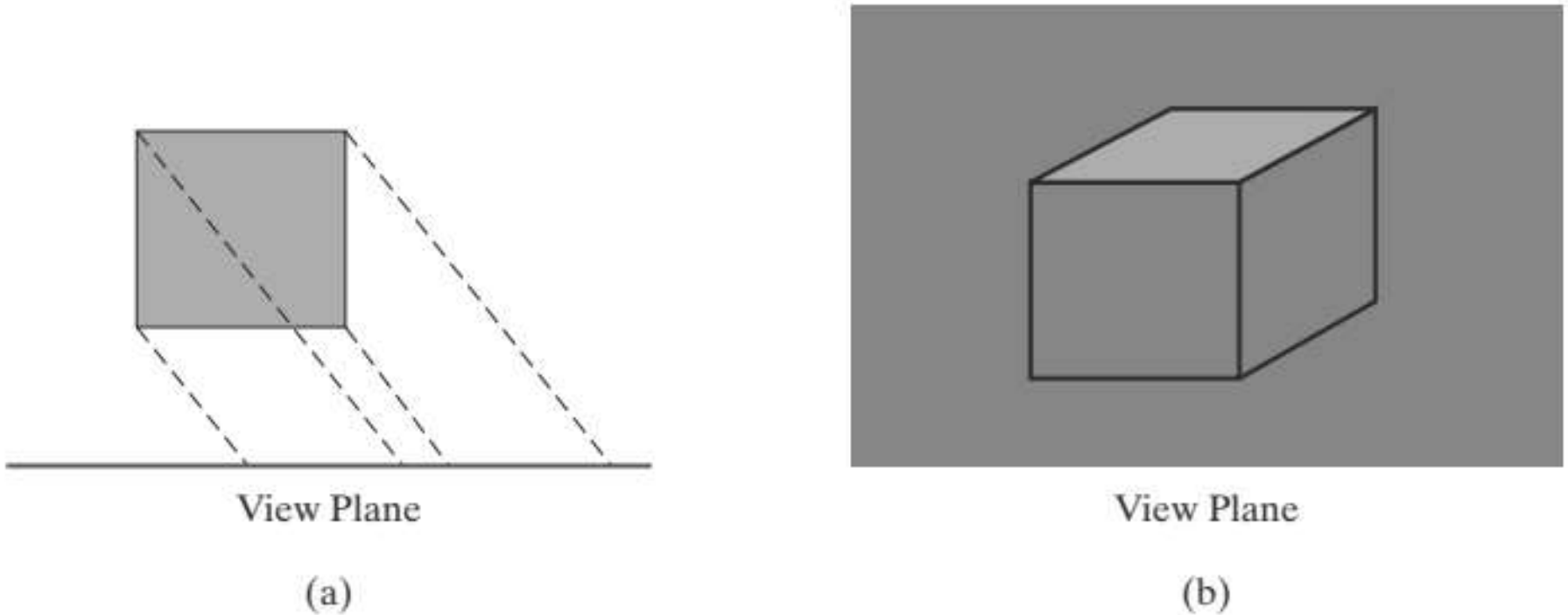


FIGURE 25
An oblique parallel projection of a cube, shown in a top view (a), produces a view (b) containing multiple surfaces of the cube.

$$x_p = x + L\cos\phi$$
$$y_p = y + L\sin\phi \tag{8}$$

Length L depends on the angle α and the perpendicular distance of the point (x, y, z) from the view plane:

$$\tan\alpha = \frac{z_{vp} - z}{L} \tag{9}$$

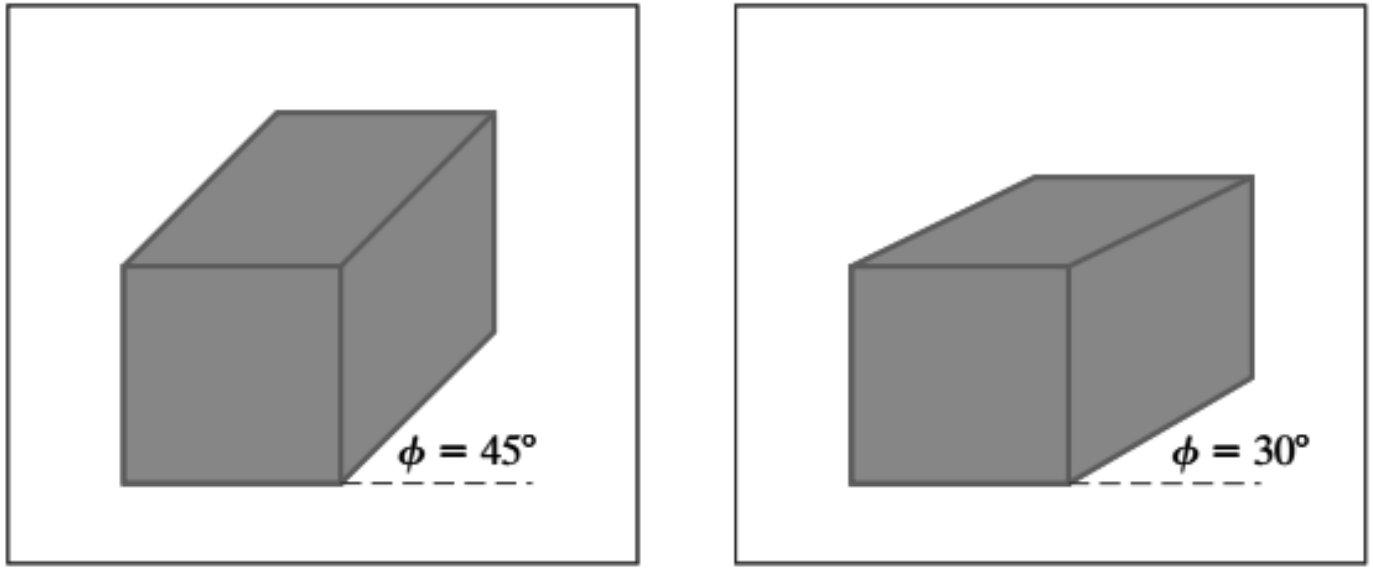
Thus

$$L = \frac{z_{vp} - z}{\tan\alpha} = L_1(z_{vp} - z) \tag{10}$$

where $L_1 = \cot\alpha$, which is also the value of L when $z_{vp} - z = 1$. We can then write the oblique parallel projection equations 8 as

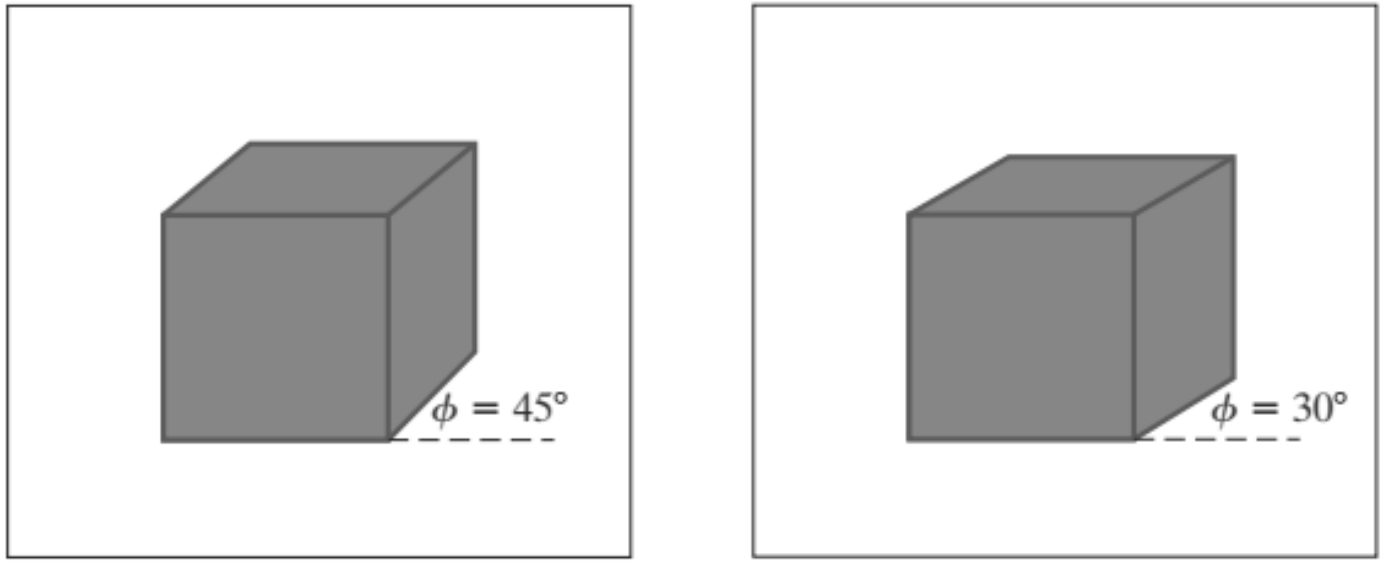
$$x_p = x + L_1(z_{vp} - z)\cos\phi$$
$$y_p = y + L_1(z_{vp} - z)\sin\phi \tag{11}$$

An orthogonal projection is obtained when $L_1 = 0$ (which occurs at the projection angle $\alpha = 90°$).



(a) (b)

FIGURE 28
Cavalier projections of a cube onto a view plane for two values of angle ϕ. The depth of the cube is projected with a length equal to that of the width and height.



(a) (b)

FIGURE 29
Cabinet projections of a cube onto a view plane for two values of angle ϕ. The depth is projected with a length that is one half that of the width and height of the cube.

Normalization Transformation for an Oblique Parallel Projection

- Because the oblique parallel-projection equations convert object descriptions to orthogonal-coordinate positions, we can apply the normalization procedures following this transformation.
- The oblique view volume has been converted to a rectangular parallelepiped.

$$\mathbf{M}_{\text{oblique,norm}} = \mathbf{M}_{\text{ortho,norm}} \cdot \mathbf{M}_{\text{oblique}}$$

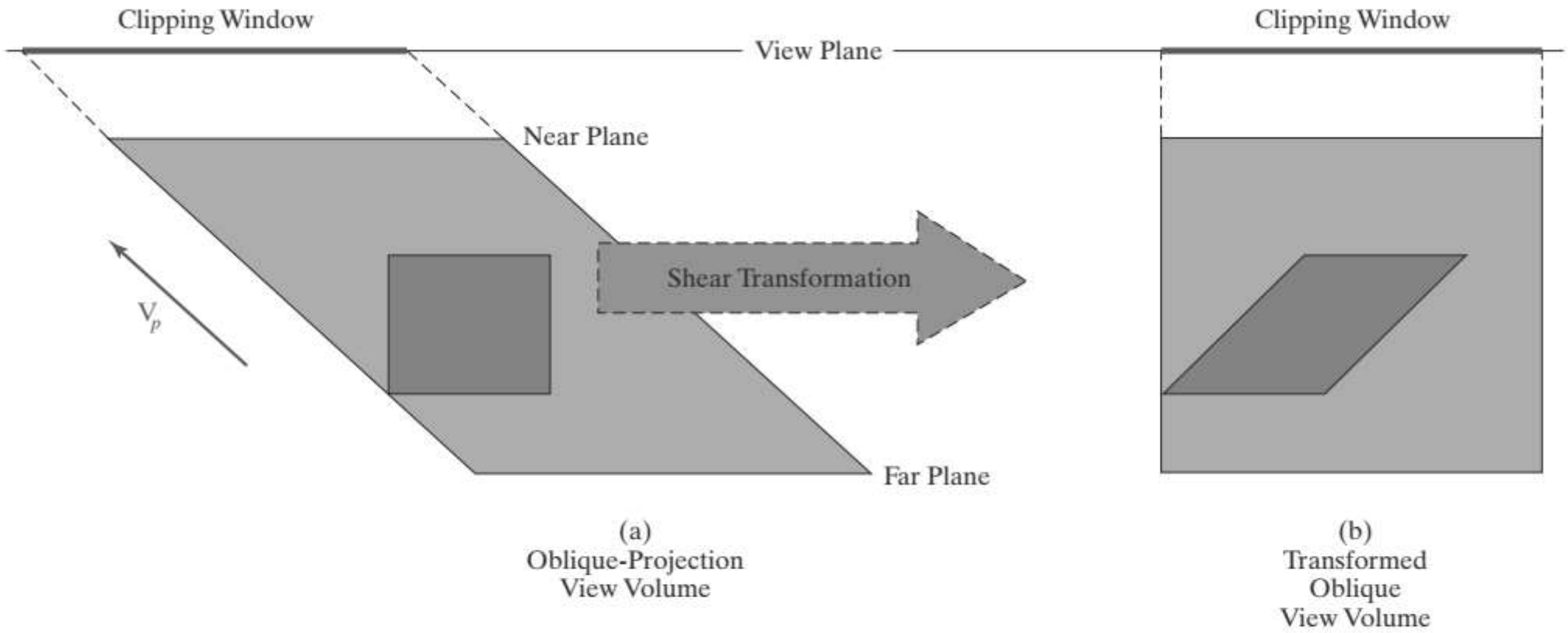
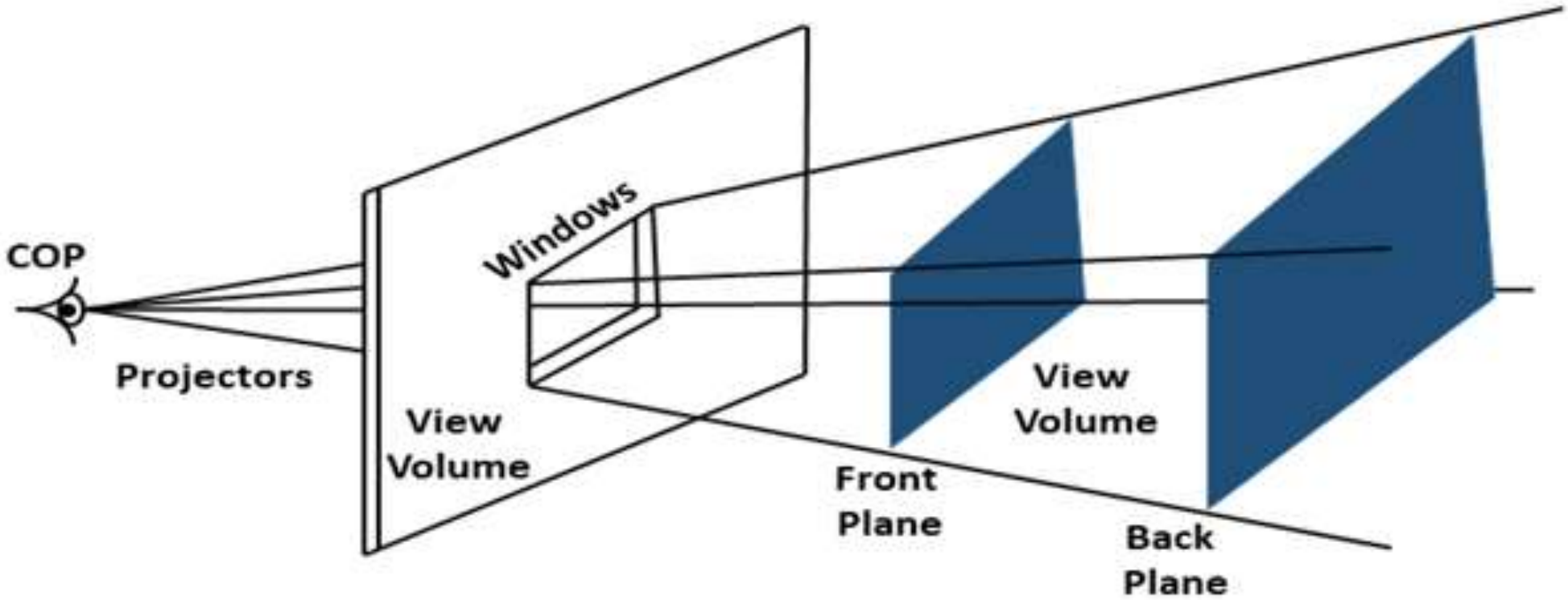


FIGURE 32
Top view of an oblique parallel-projection transformation. The oblique view volume is converted into a rectangular parallelepiped, and objects in the view volume, such as the green block, are mapped to orthogonal-projection coordinates.

2. Perspective Projections

- In perspective projection farther away object from the viewer, small it appears. This property of projection gives an idea about depth. The artist use perspective projection from drawing three-dimensional scenes.

- Two main characteristics of perspective are **vanishing points** and **perspective foreshortening**. Due to foreshortening object and lengths appear smaller from the center of projection. More we increase the distance from the center of projection, smaller will be the object appear.



(b) Viewing volume in perspective projection

Perspective-Projection Transformation Coordinates

- We can sometimes select the projection reference point as another viewing parameter in a graphics package, but some systems place this convergence point at a fixed position, such as at the view point.
- Figure 34 shows the projection path of a spatial position (*x*, *y*, *z*) to a general projection reference point at *(xpr p*, *ypr p*, *zpr p)*.
- The projection line intersects the view plane at the coordinate position *(xp*, *yp*, *zvp)*, where *zvp* is some selected position for the view plane on the zview axis.
- We can write equations describing coordinate positions along this perspective-projection line in parametric form as :

$$\begin{aligned} x' &= x - (x - x_{prp})u \\ y' &= y - (y - y_{prp})u \qquad 0 \le u \le 1 \\ z' &= z - (z - z_{prp})u \end{aligned} \tag{15}$$

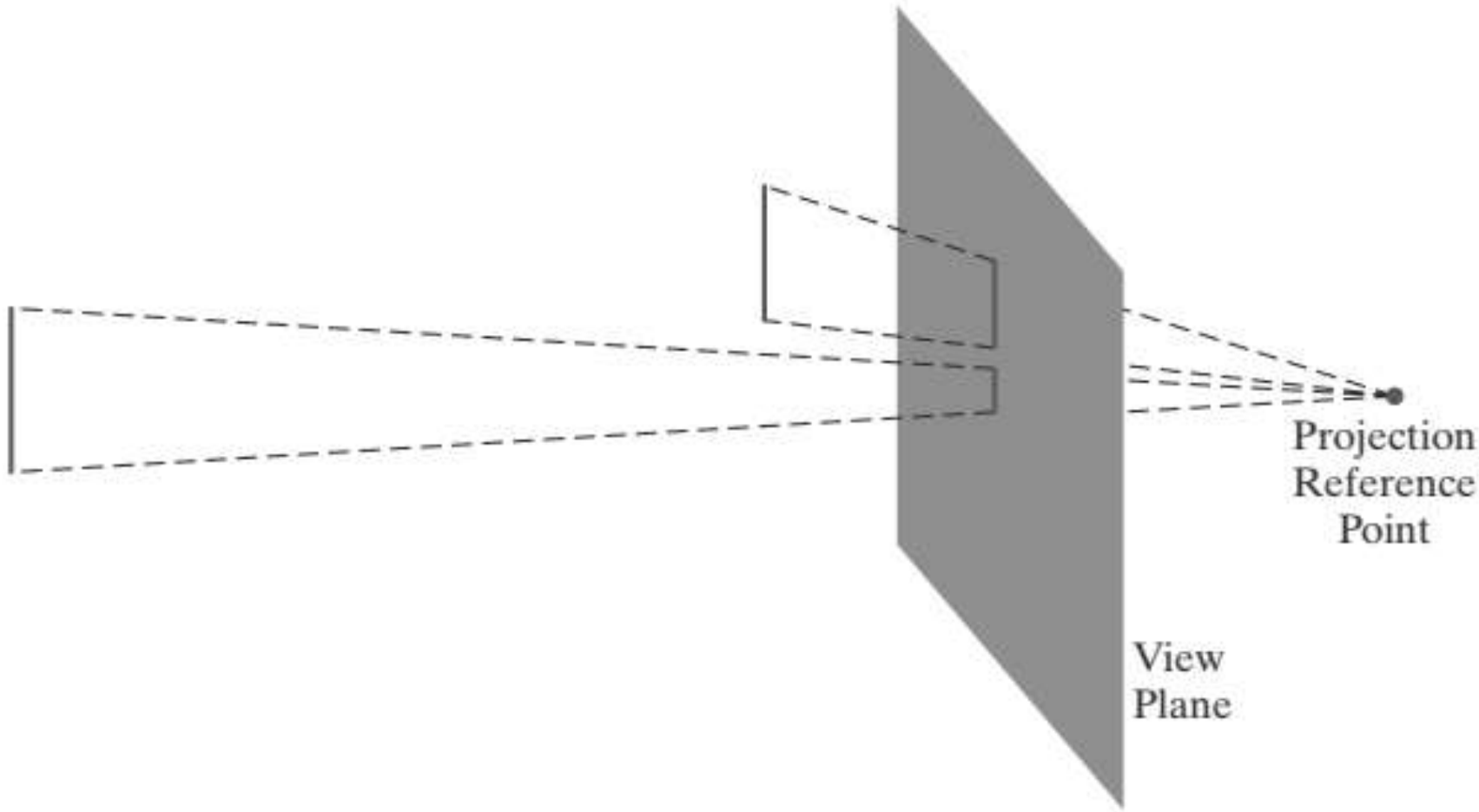


FIGURE 33
A perspective projection of two equal-length line segments at different distances from the view plane.

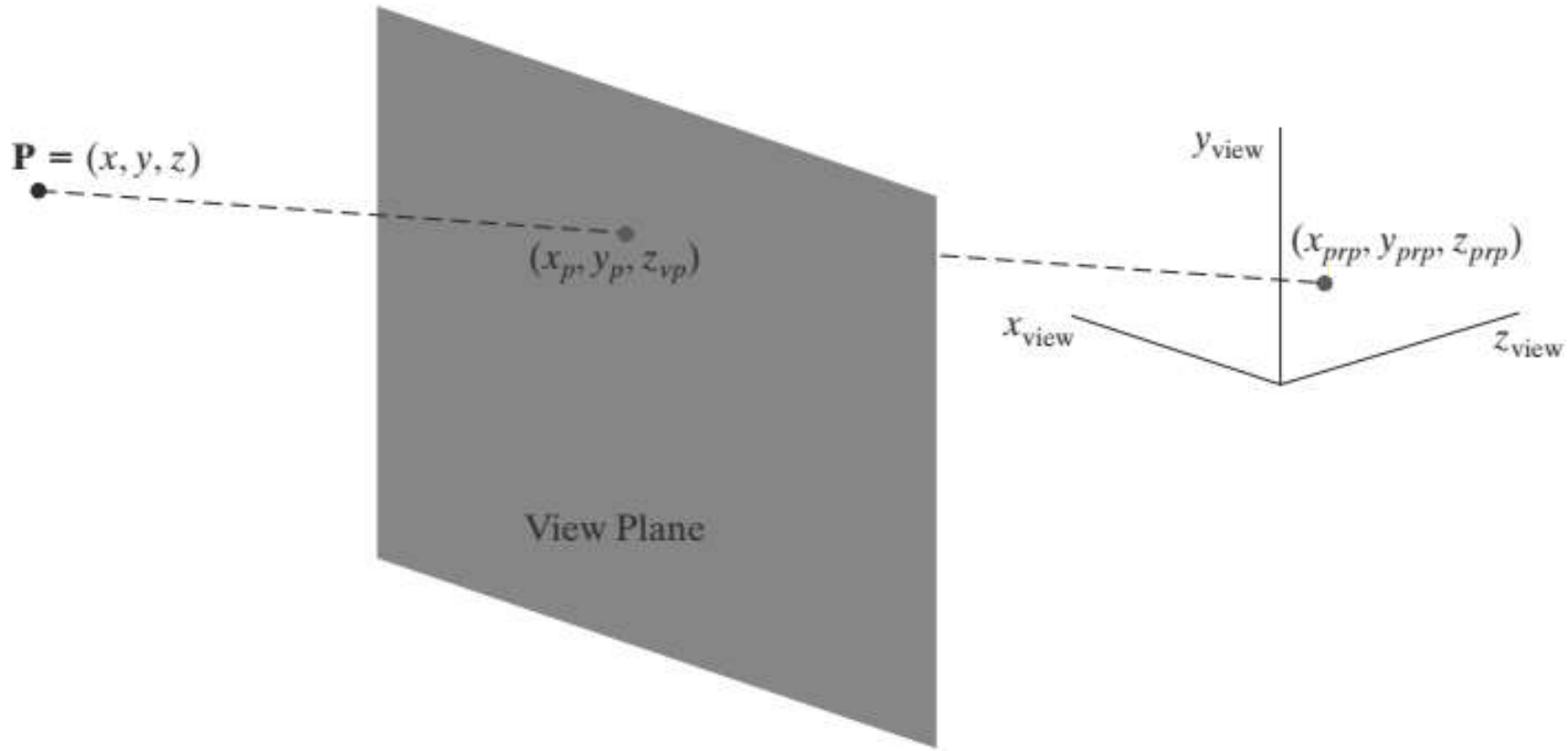


FIGURE 34
A perspective projection of a point **P** with coordinates (x, y, z) to a selected projection reference point. The intersection position on the view plane is (x_p, y_p, z_{vp}).

Coordinate position (x', y', z') represents any point along the projection line. When $u = 0$, we are at position $\mathbf{P} = (x, y, z)$. At the other end of the line, $u = 1$ and we have the projection reference-point coordinates $(x_{prp}, y_{prp}, z_{prp})$. On the view plane, $z' = z_{vp}$ and we can solve the z' equation for parameter u at this position along the projection line:

$$u = \frac{z_{vp} - z}{z_{prp} - z} \tag{16}$$

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

$$\begin{aligned} x_p &= x\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) + x_{prp}\left(\frac{z_{vp} - z}{z_{prp} - z}\right) \\ y_p &= y\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) + y_{prp}\left(\frac{z_{vp} - z}{z_{prp} - z}\right) \end{aligned} \tag{17}$$

Perspective-Projection Equations: Special Cases

Various restrictions are often placed on the parameters for a perspective projection. Depending on a particular graphics package, positioning for either the projection reference point or the view plane may not be completely optional.

To simplify the perspective calculations, the projection reference point could be limited to positions along the z_{view} axis, then

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

$$x_p = x\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right), \qquad y_p = y\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) \tag{18}$$

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

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

$$x_p = x\left(\frac{z_{vp}}{z}\right), \qquad y_p = y\left(\frac{z_{vp}}{z}\right) \tag{19}$$

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

3. $z_{vp} = 0$:

$$x_p = x\left(\frac{z_{prp}}{z_{prp} - z}\right) - x_{prp}\left(\frac{z}{z_{prp} - z}\right)$$
$$y_p = y\left(\frac{z_{prp}}{z_{prp} - z}\right) - y_{prp}\left(\frac{z}{z_{prp} - z}\right) \tag{20}$$

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

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

$$x_p = x\left(\frac{z_{prp}}{z_{prp} - z}\right), \qquad y_p = y\left(\frac{z_{prp}}{z_{prp} - z}\right) \tag{21}$$

Important terms in Perspective Projections

- **View plane:** It is an area of world coordinate system which is projected into viewing plane.

- **Center of Projection:** It is the location of the eye on which projected light rays converge.

- **Projectors:** It is also called a projection vector. These are rays start from the object scene and are used to create an image of the object on viewing or view plane.

It introduces several anomalies due to these object shape and appearance gets affected.

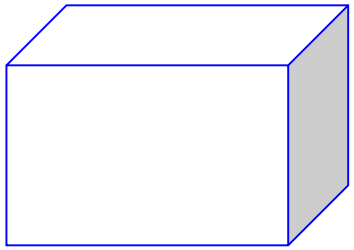
- **Perspective foreshortening:** The size of the object will be small of its distance from the center of projection increases.
- **Vanishing Point:** All lines appear to meet at some point in the view plane.
- **Distortion of Lines:** A range lies in front of the viewer to back of viewer is appearing to six rollers.

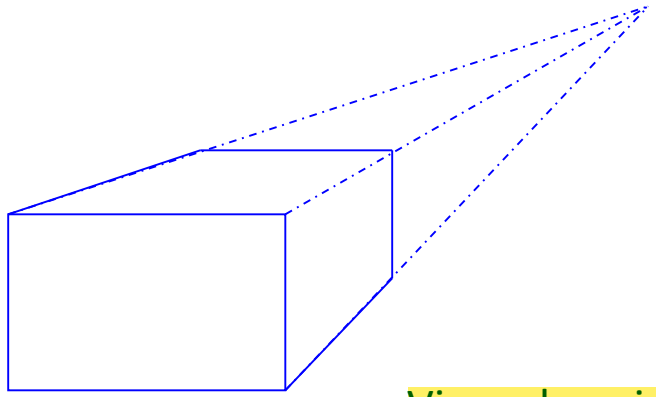
1. Perspective Projections: Vanishing points

- The point at which a set of projected parallel lines appears to converge is called a **vanishing point.**
- It is the point where all lines will appear to meet. There can be one point, two point, and three point perspectives.
- There is an illusion that certain sets of parallel lines (that are not parallel to the view plane) appear to meet at some point on the view plane.
- The vanishing point for any set of parallel lines that are parallel to one of principal axis is referred to as a principal vanishing point (PVP).
- The number of PVPs is determined by the number of principal axes intersected by the view plane.

One principal vanishing point projection

- occurs when the projection plane is perpendicular to one of the principal axes (x, y or z).



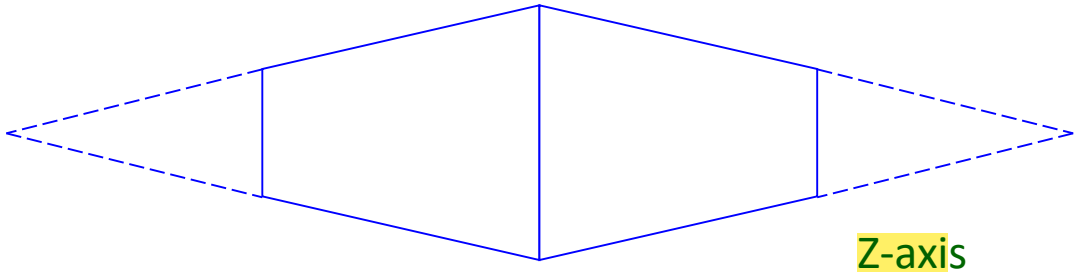


Vanishing point

View plane is parallel to XY-plane and intersects Z-axis only.

Two principal vanishing point projection

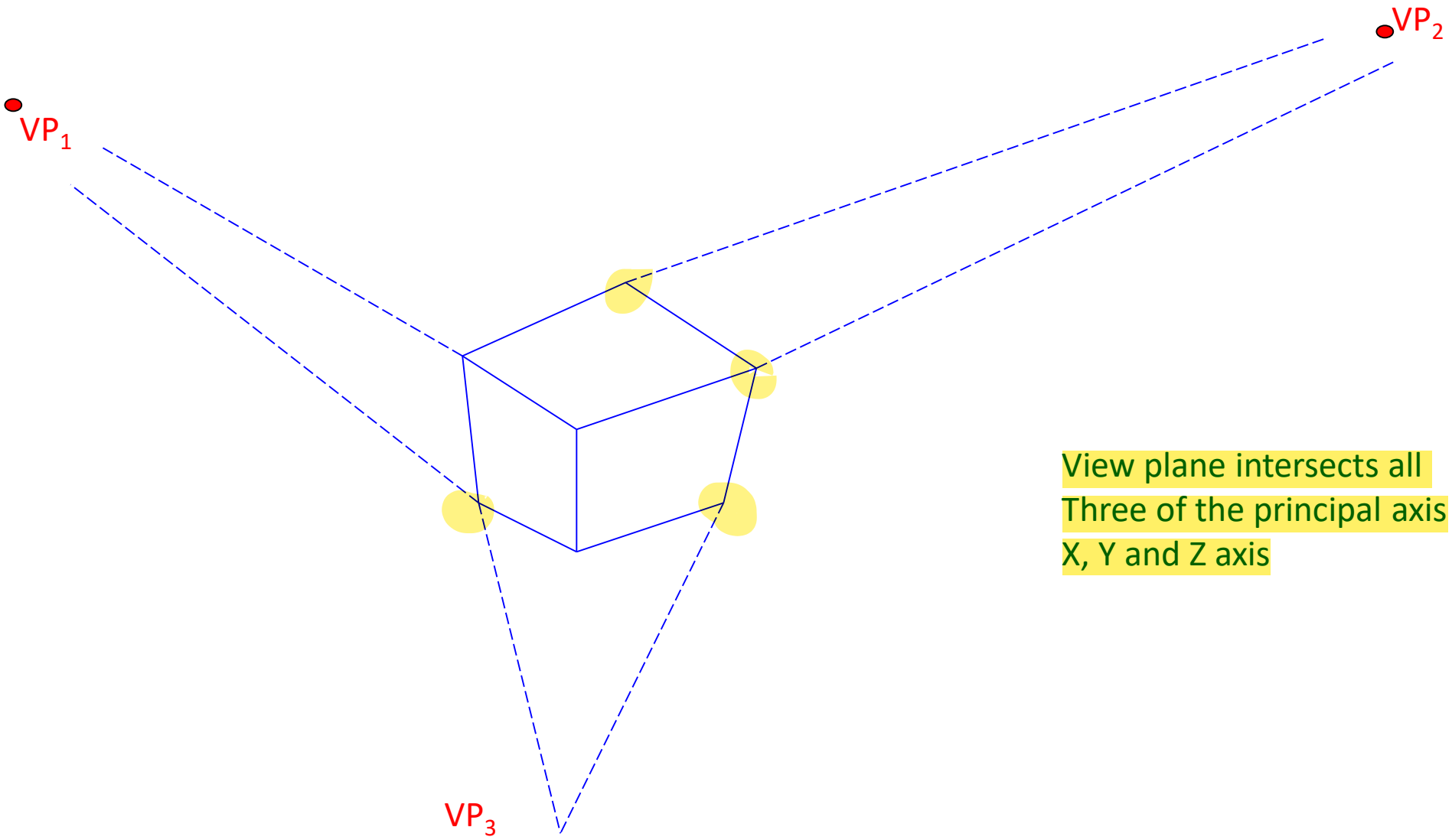
View plane intersects
Both X and Z axis but
not the Y axis



X-axis
Vanishing point

Z-axis
Vanishing point

- Three principal vanishing point intersection



View plane intersects all
Three of the principal axis
X, Y and Z axis

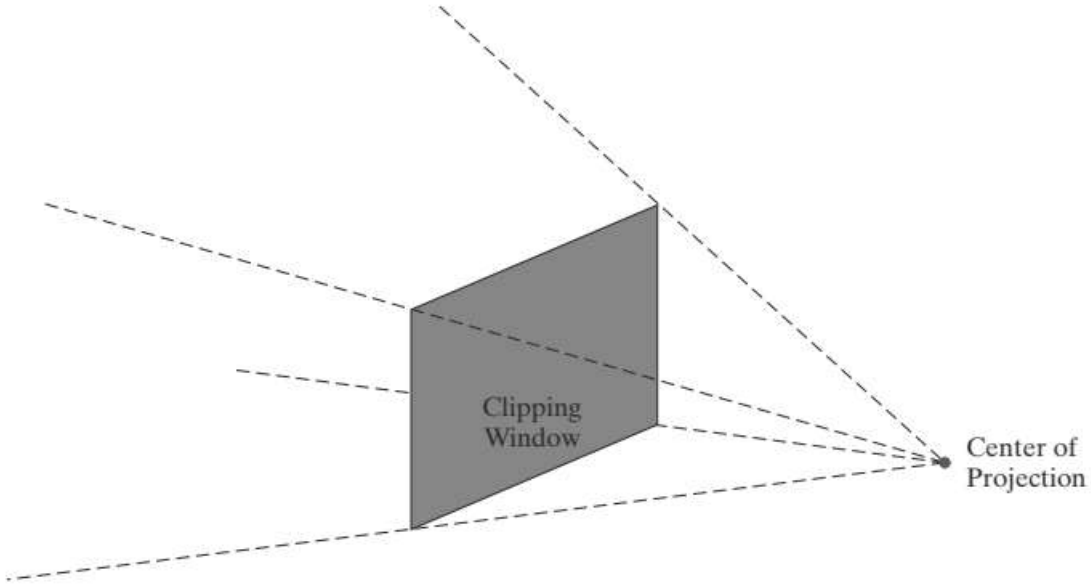


FIGURE 38
An infinite, pyramid view volume for a perspective projection.

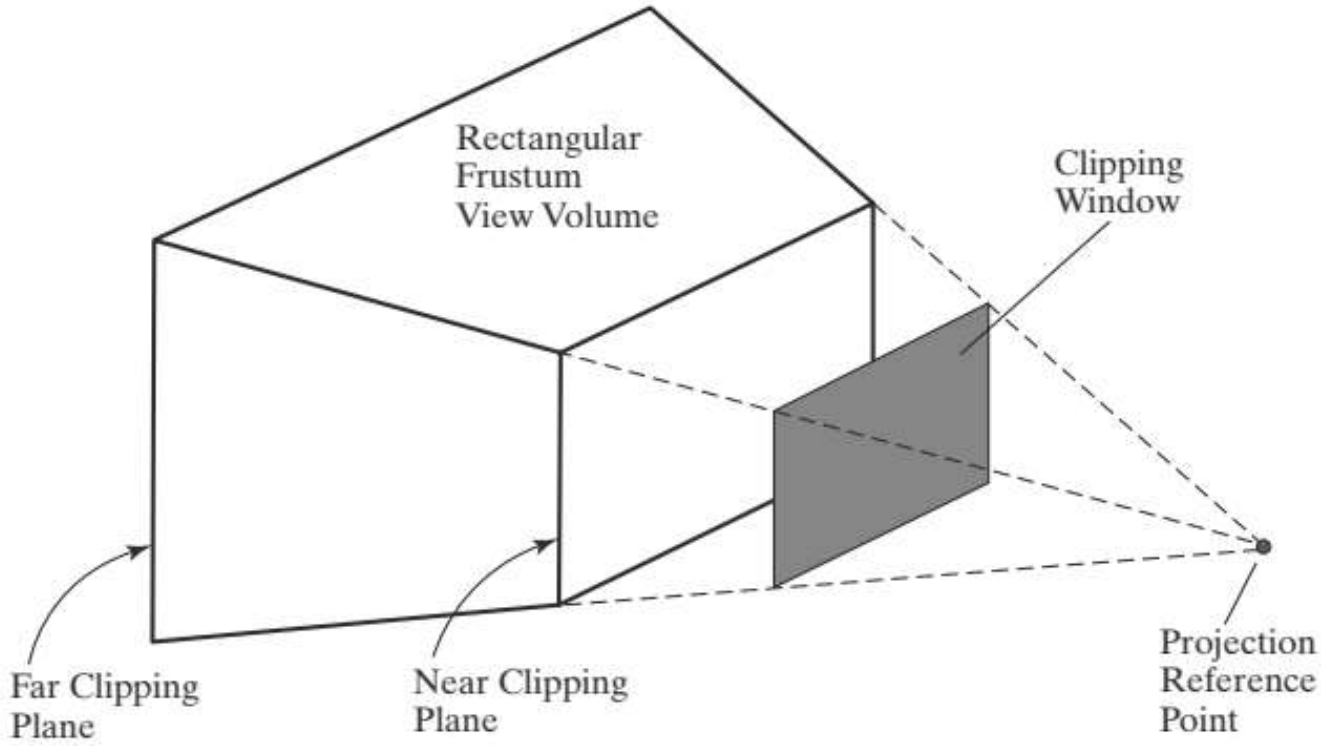


FIGURE 39
A perspective-projection frustum view volume with the view plane "in front" of the near clipping plane.

But we can use a three-dimensional, homogeneous-coordinate representation to express the perspective-projection equations in the form

$$x_p = \frac{x_h}{h}, \qquad y_p = \frac{y_h}{h} \tag{22}$$

where the homogeneous parameter has the value

$$h = z_{prp} - z \tag{23}$$

The numerators in 22 are the same as in equations 17:

$$\begin{aligned} x_h &= x(z_{prp} - z_{vp}) + x_{prp}(z_{vp} - z) \\ y_h &= y(z_{prp} - z_{vp}) + y_{prp}(z_{vp} - z) \end{aligned} \tag{24}$$

$$\mathbf{P}_h = \mathbf{M}_{\text{pers}} \cdot \mathbf{P} \tag{25}$$

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

Normalized Perspective-Projection Transformation Coordinates

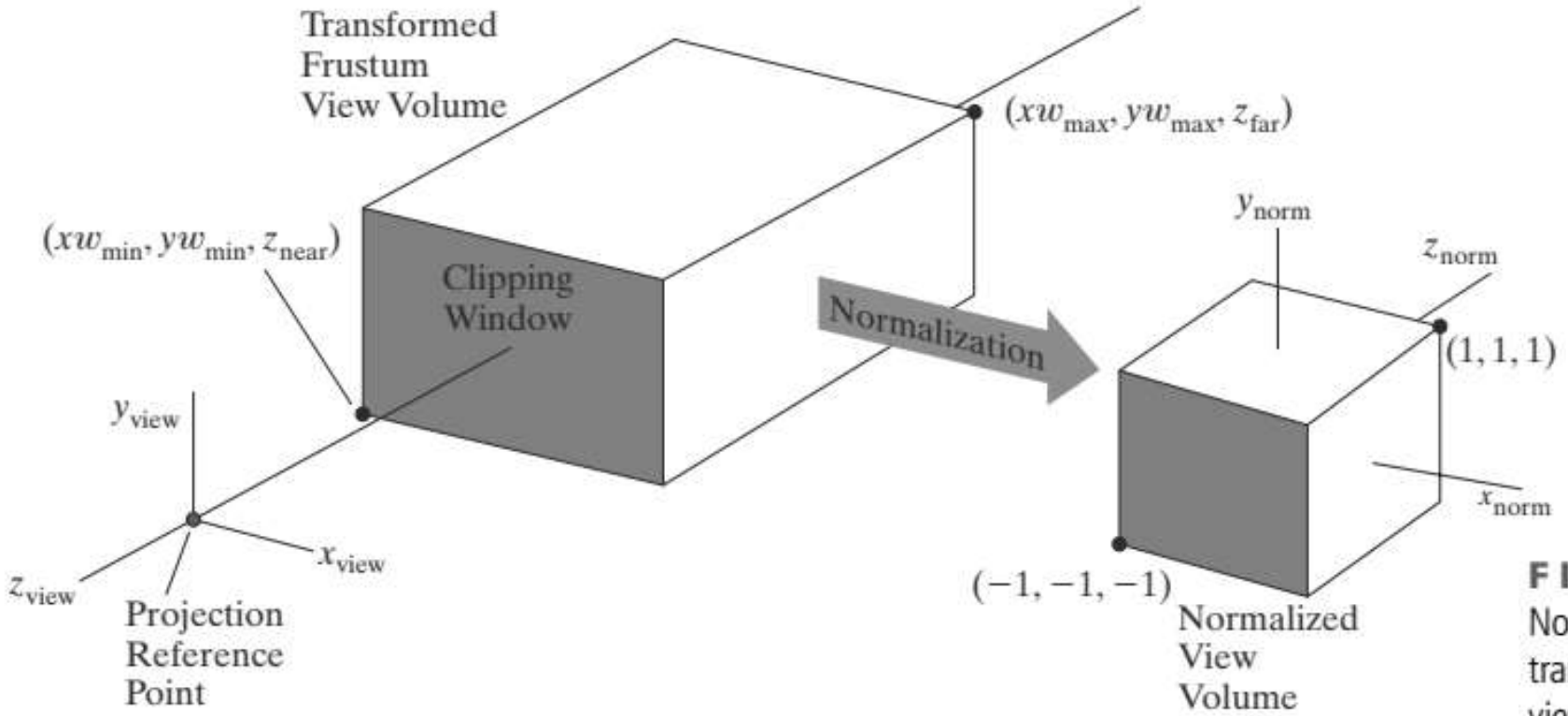


FIGURE 46
Normalization transformation from a transformed perspective-projection view volume (rectangular parallelepiped) to the symmetric normalization cube within a left-handed reference frame, with the near clipping plane as the view plane and the projection reference point at the viewing-coordinate origin.

$$\mathbf{M}_{xy\,\text{scale}} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{35}$$

Concatenating the xy-scaling matrix with matrix 34 produces the following normalization matrix for a perspective-projection transformation.

$$\begin{aligned} \mathbf{M}_{\text{normpers}} &= \mathbf{M}_{xy\,\text{scale}} \cdot \mathbf{M}_{\text{obliquepers}} \\ &= \begin{bmatrix} -z_{\text{near}} s_x & 0 & s_x \dfrac{xw_{\min} + xw_{\max}}{2} & 0 \\ 0 & -z_{\text{near}} s_y & s_y \dfrac{yw_{\min} + yw_{\max}}{2} & 0 \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & 0 \end{bmatrix} \end{aligned} \tag{36}$$

From this transformation, we obtain the homogeneous coordinates:

$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ h \end{bmatrix} = \mathbf{M}_{\text{normpers}} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{37}$$

And the projection coordinates are

$$
\begin{aligned}
x_p &= \frac{x_h}{h} = \frac{-z_{\text{near}} s_x x + s_x (x w_{\min} + x w_{\max})/2}{-z} \\
y_p &= \frac{y_h}{h} = \frac{-z_{\text{near}} s_y y + s_y (y w_{\min} + y w_{\max})/2}{-z} \\
z_p &= \frac{z_h}{h} = \frac{s_z z + t_z}{-z}
\end{aligned}
\tag{38}
$$

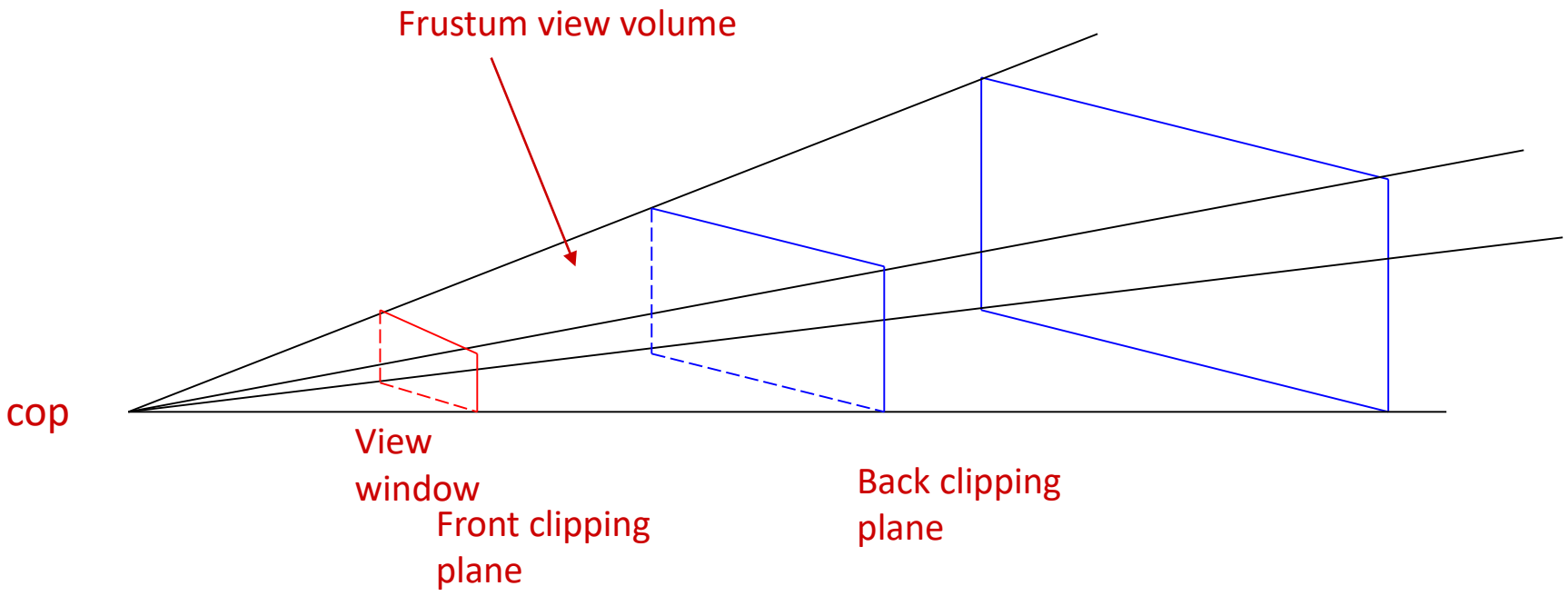
$$
\mathbf{M}_{\text{normpers}} = \begin{bmatrix}
\dfrac{-2z_{\text{near}}}{x w_{\max} - x w_{\min}} & 0 & \dfrac{x w_{\max} + x w_{\min}}{x w_{\max} - x w_{\min}} & 0 \\
0 & \dfrac{-2z_{\text{near}}}{y w_{\max} - y w_{\min}} & \dfrac{y w_{\max} + y w_{\min}}{y w_{\max} - y w_{\min}} & 0 \\
0 & 0 & \dfrac{z_{\text{near}} + z_{\text{far}}}{z_{\text{near}} - z_{\text{far}}} & -\dfrac{2z_{\text{near}} z_{\text{far}}}{z_{\text{near}} - z_{\text{far}}} \\
0 & 0 & -1 & 0
\end{bmatrix}
\tag{40}
$$

View Volume

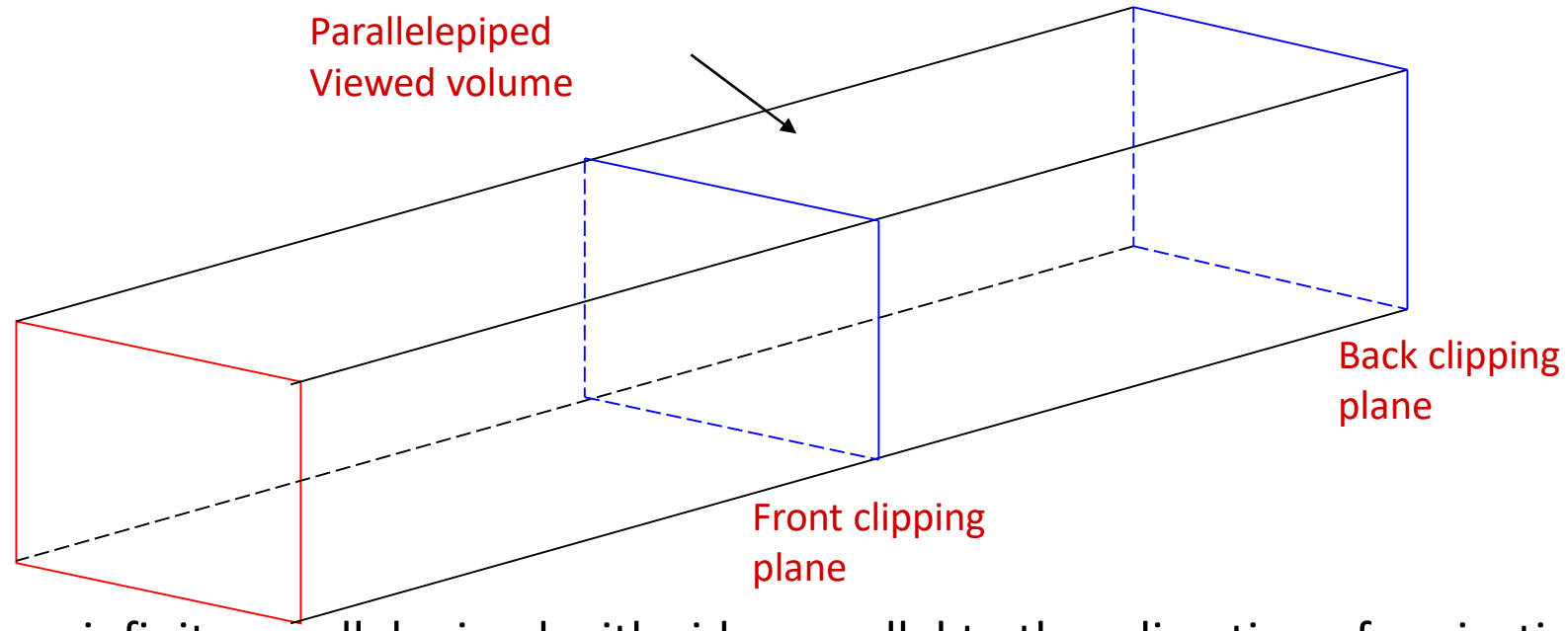
- The view volume bounds that portion of the 3D space that is to be clipped out and projected onto the view plane.

View Volume for Perspective Projection

- its shape is semi-infinite pyramid with apex at the view point and edge passing through the corners of the window.

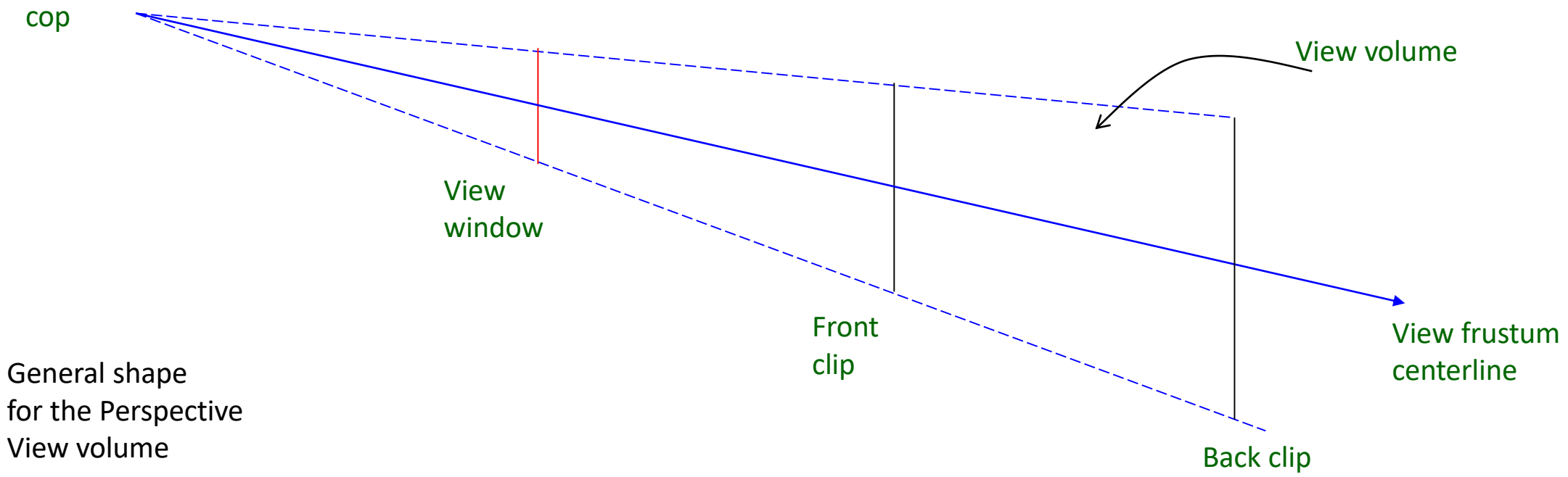


View Volume for Parallel Projection



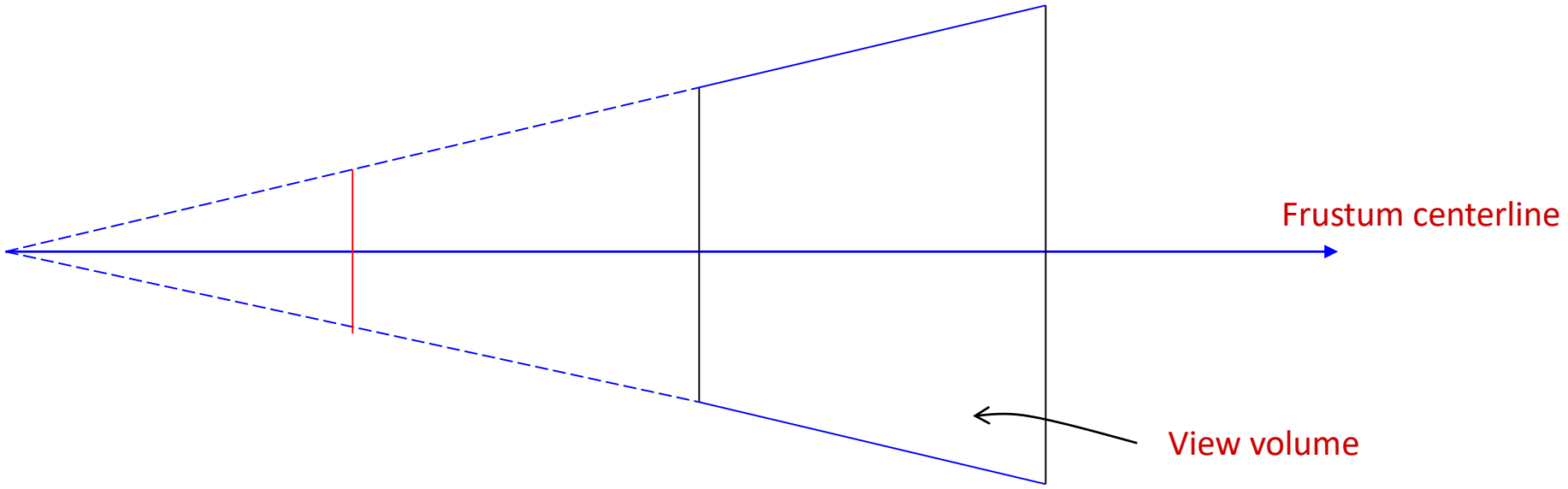
-It's shape is an infinite parallelepiped with sides parallel to the direction of projection.

Producing a Canonical view volume for a perspective projection

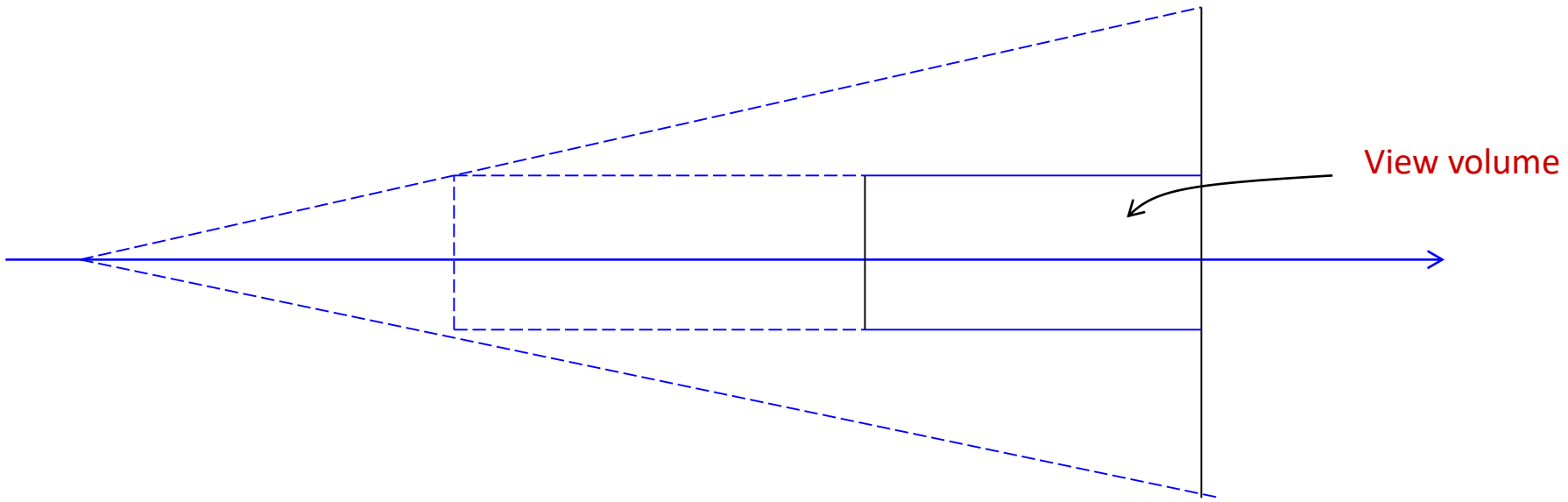


General shape
for the Perspective
View volume

Step 1: shear the view volume so that centerline of the frustum is perpendicular to the view plane and passes through the center of the view window.

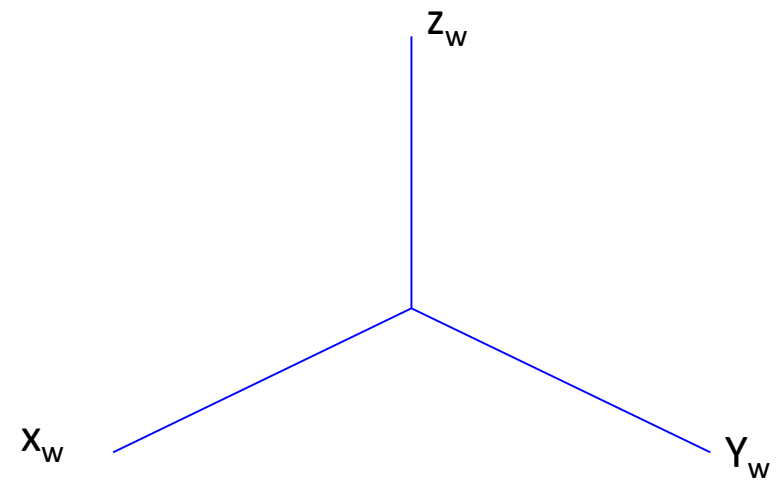


Step2: scale view volume inversely proportional to the distance from the view window, so that shape of view volume becomes rectangular parallelepiped.

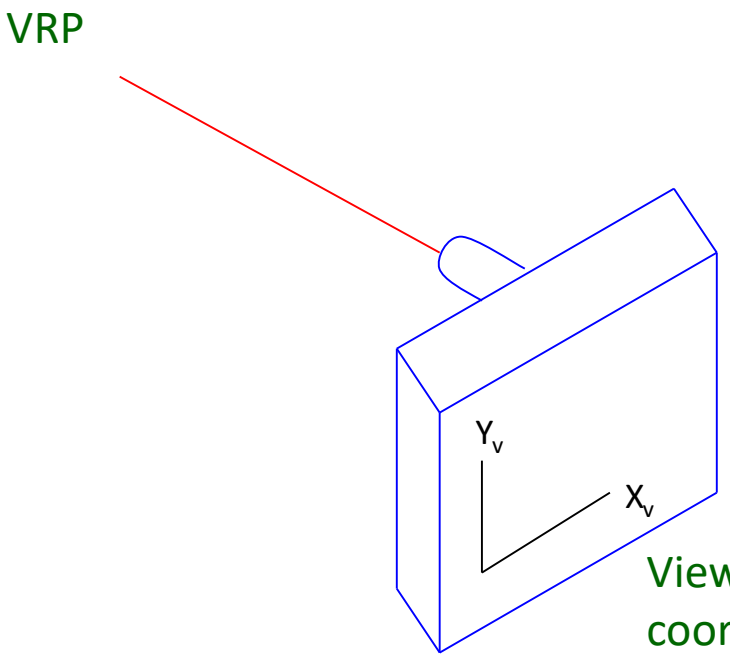


Converting object coordinates to view plane coordinates

- similar to the process of rotation about an arbitrary axis



World coordinate system



View plane (eye) coordinate system

Steps:

1. Translate origin to view reference point (VRP).

2. Translate along the view plane normal by view distance.

3. Align object coordinate's z-axis with view plane coordinates z-axis (the view plane normal).

a)- Rotate about x-axis to place the line (ie. Object coordinates z-axis) in the view plane coordinates xz-plane.

b)- Rotate about y-axis to move the z axis to its proper position.

c)- Rotate about the z-axis until x and y axis are in place in the view plane coordinates.

Three-Dimensional Clipping Algorithms

- All device-independent transformations (geometric and viewing) are concatenated and applied before executing the clipping routines.
- And each of the clipping boundaries for the normalized view volume is a plane that is parallel to one of the Cartesian planes, regardless of the projection type and original shape of the view volume.
- Depending on whether the view volume has been normalized to a unit cube or to a symmetric cube with edge length 2, the clipping planes have coordinate positions either at 0 and 1 or at -1 and 1.
- For the symmetric cube, the equations for the three-dimensional clipping planes are:

$$\begin{aligned} xw_{\min} &= -1, & xw_{\max} &= 1 \\ yw_{\min} &= -1, & yw_{\max} &= 1 \\ zw_{\min} &= -1, & zw_{\max} &= 1 \end{aligned} \qquad (43)$$

Clipping in 3D Homogeneous Coordinates

- Computer-graphics libraries process spatial positions as four-dimensional homogeneous coordinates so that all transformations can be represented as 4 by 4 matrices.
- As each coordinate position enters the viewing pipeline, it is converted to a four-dimensional representation:

$$(x, y, z) \rightarrow (x, y, z, 1)$$

After a position has passed through the geometric, viewing, and projection transformations, it is now in the homogeneous form

$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ h \end{bmatrix} = \mathbf{M} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad (44)$$

Three-dimensional Region Codes

FNTBRL

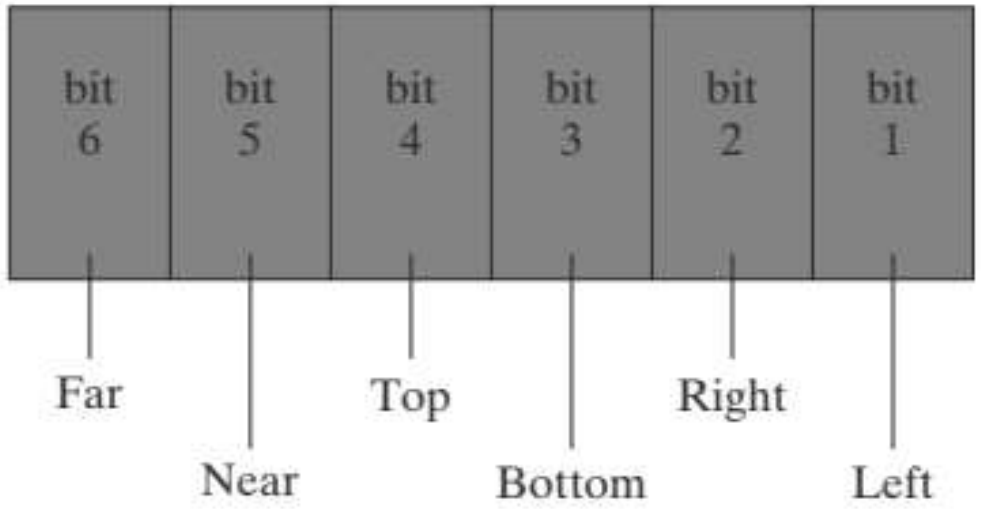
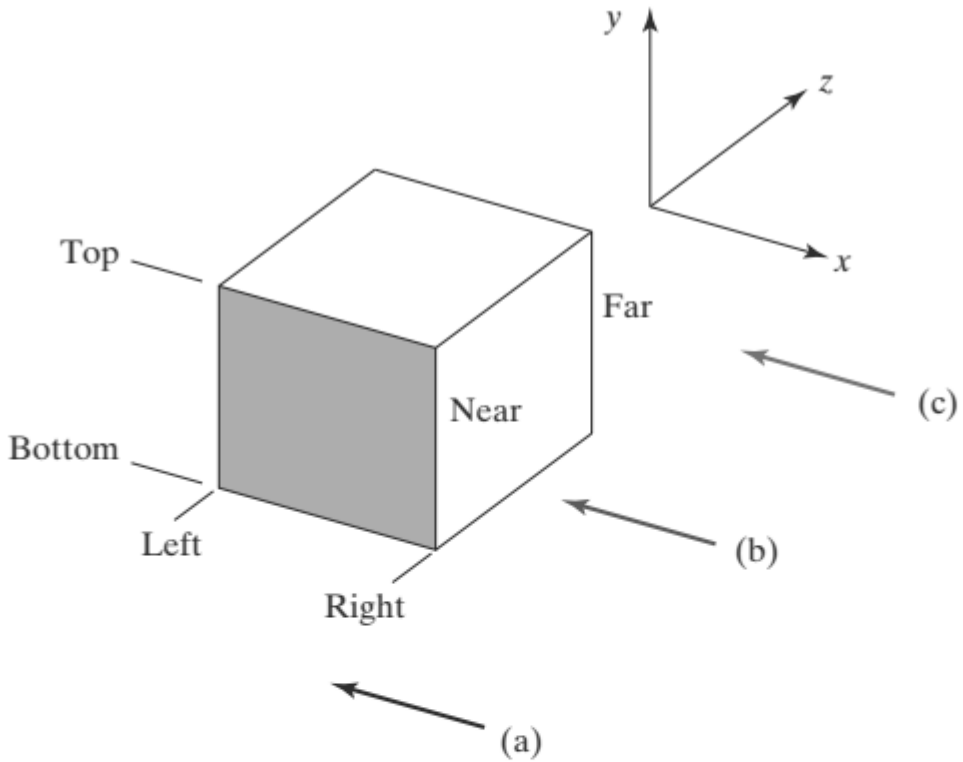


FIGURE 49
A possible ordering for the view-volume clipping boundaries corresponding to the region-code bit positions.

$$
\begin{aligned}
&\text{bit } 1 = 1 \quad \text{if } h + x_h < 0 \quad \text{(left)} \\
&\text{bit } 2 = 1 \quad \text{if } h - x_h < 0 \quad \text{(right)} \\
&\text{bit } 3 = 1 \quad \text{if } h + y_h < 0 \quad \text{(bottom)} \\
&\text{bit } 4 = 1 \quad \text{if } h - y_h < 0 \quad \text{(top)} \\
&\text{bit } 5 = 1 \quad \text{if } h + z_h < 0 \quad \text{(near)} \\
&\text{bit } 6 = 1 \quad \text{if } h - z_h < 0 \quad \text{(far)}
\end{aligned}
$$



011001	011000	011010
010001	010000	010010
010101	010100	010110

Region Codes
In Front of Near Plane
(a)

001001	001000	001010
000001	000000	000010
000101	000100	000110

Region Codes
Between Near and Far Planes
(b)

101001	101000	101010
100001	100000	100010
100101	100100	100110

Region Codes
Behind Far Plane
(c)

FIGURE 50
Values for the three-dimensional, six-bit region code that identifies spatial positions relative to the boundaries of a view volume.

3D Point and Line Clipping

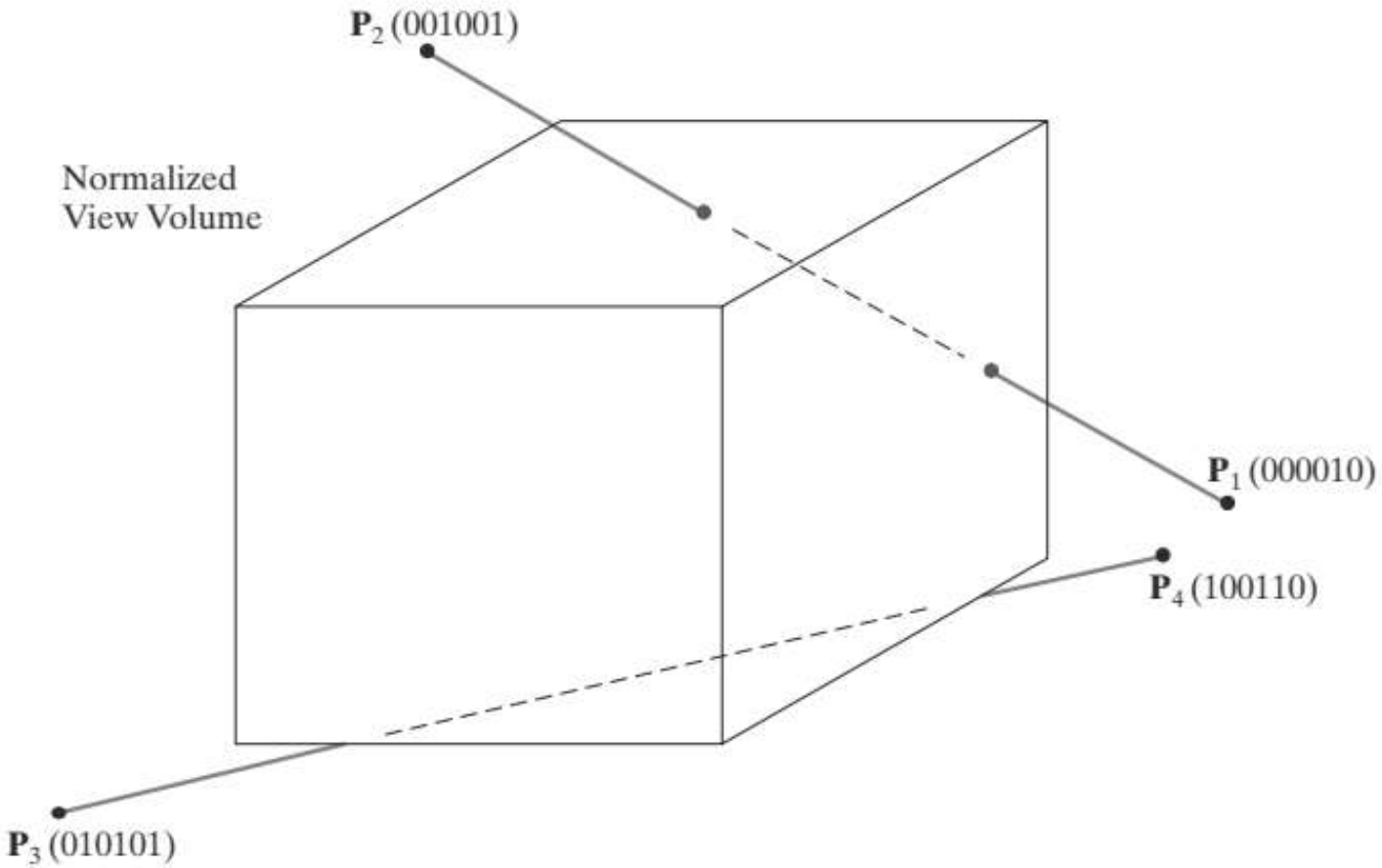


FIGURE 51
Three-dimensional region codes for two line segments. Line $\overline{P_1P_2}$ intersects the right and top clipping boundaries of the view volume, while line $\overline{P_3P_4}$ is completely below the bottom clipping plane.

Equations for three-dimensional line segments are conveniently expressed in parametric form, and the clipping methods of Cyrus-Beck or Liang-Barsky can be extended to three-dimensional scenes. For a line segment with endpoints $\mathbf{P}_1 = (x_{h1}, y_{h1}, z_{h1}, h_1)$ and $\mathbf{P}_2 = (x_{h2}, yh_2, zh_2, h_2)$, we can write the parametric equation describing any point position along the line as

$$\mathbf{P} = \mathbf{P}_1 + (\mathbf{P}_2 - \mathbf{P}_1)u \qquad 0 \le u \le 1 \tag{48}$$

When the line parameter has the value $u = 0$, we are at position $\mathbf{P}_1$. And $u = 1$ brings us to the other end of the line, $\mathbf{P}_2$. Writing the parametric line equation explicitly, in terms of the homogeneous coordinates, we have

$$\begin{aligned} x_h &= x_{h1} + (x_{h2} - x_{h1})u \\ y_h &= y_{h1} + (y_{h2} - y_{h1})u \\ z_h &= z_{h1} + (z_{h2} - z_{h1})u \\ h &= h_1 + (h_2 - h_1)u \end{aligned} \qquad 0 \le u \le 1 \tag{49}$$

3D Polygon Clipping

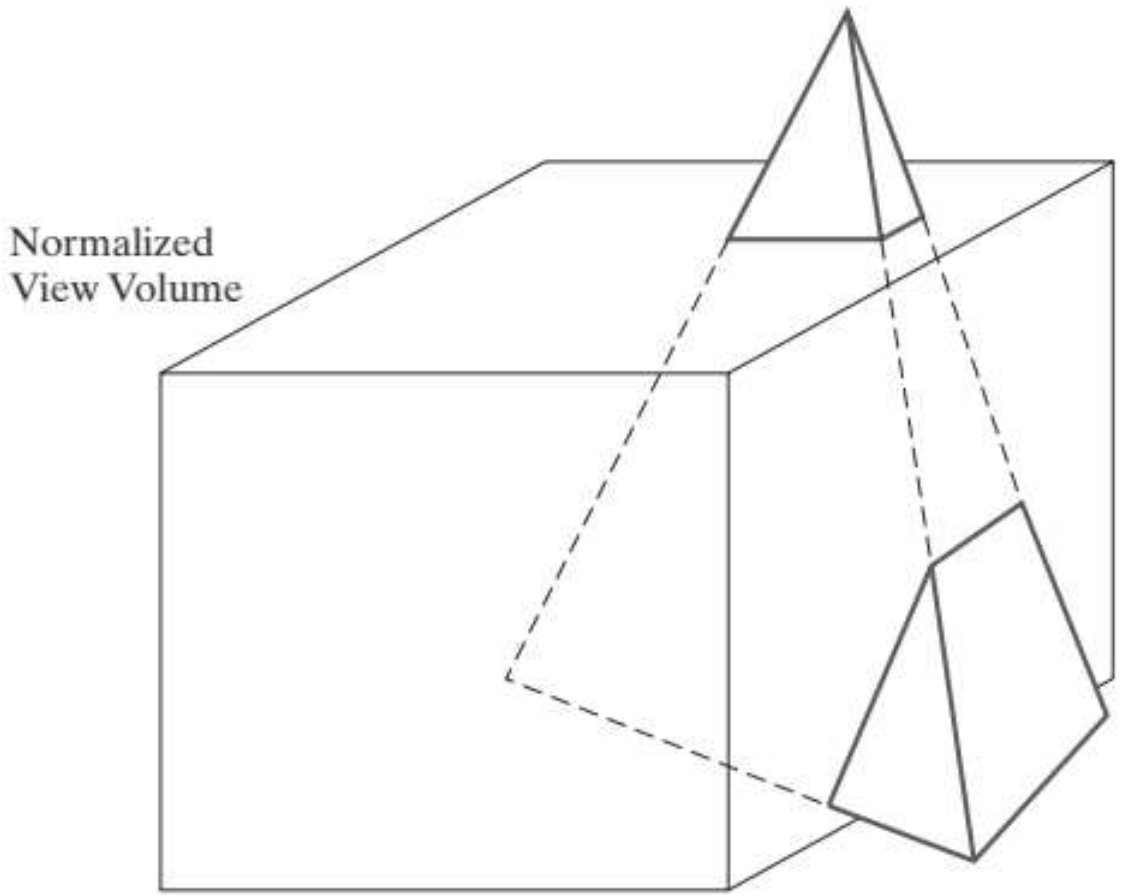


FIGURE 52
Three-dimensional object clipping. Surface sections that are outside the view-volume clipping planes are eliminated from the object description, and new surface facets may need to be constructed.

THREE-DIMENSIONAL (3D) OBJECT REPRESENTATION

General Modeling Techniques

- Polygon Mesh Models
- Curved Surfaces
- Quadric Surfaces
- Constructive Solid-Geometry
- BSP Trees
- Octrees
- Fractals

OTHER MODELING TECHNIQUES

- Particle Systems

  (for modeling objects that exhibit ‘fluid-like’ properties

  e.g. – smoke, fire, waterfalls etc)

- Volume Rendering

  (to show interior information of a data set e.g.- Seismic

  data or data set from a CT scanner)

- Physically-based Modeling

  (for modeling nonrigid objects and its behavior in terms

  of the interaction of external and internal forces e.g.- a

  rope, a piece of cloth or a soft rubber ball)

POLYGAN MESH MODELS

- Most commonly used method for polyhedron objects
- Object description is stored as sets of surface polygons
- Speeds up the surface rendering and display of objects
- Preview of scene in wireframe representation

Fig.- Hierarchical Structure for Polygon Mesh Models

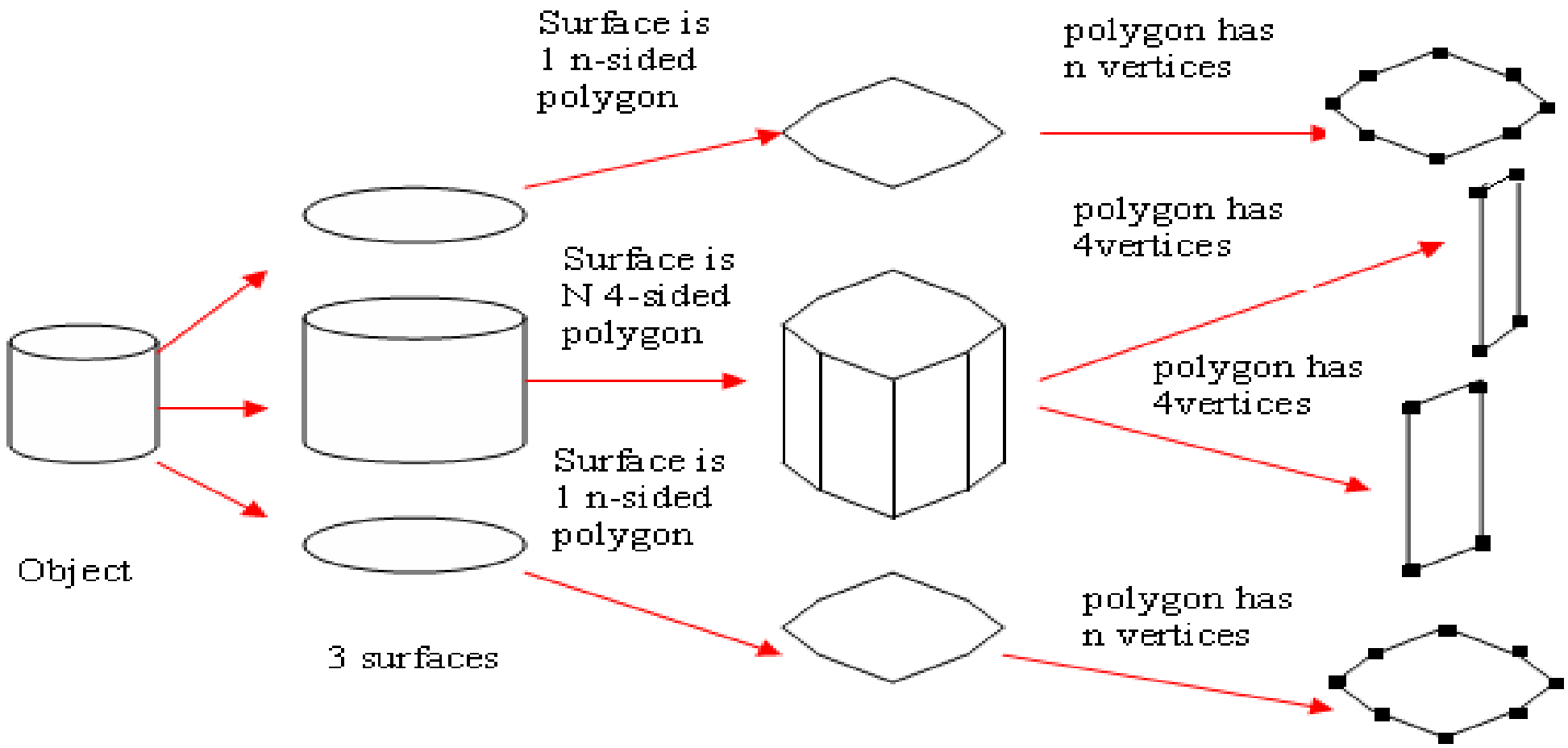
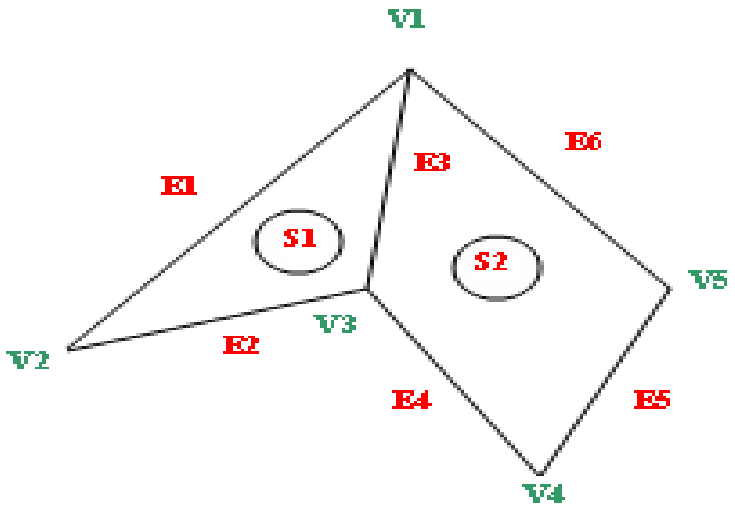


Fig. – Geometric Data Tables



Vertex table				
V1	:	x1	y1	z1
V2	:	x2	y2	z2
-			-	
-			-	
V5	:	x5	y5	z5

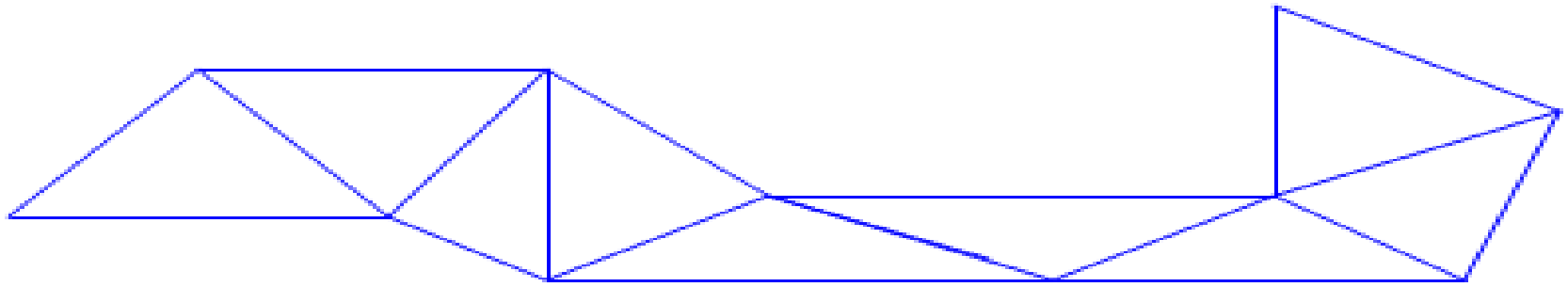
Edge Table		
E1	:	v1 ,v2
E2	:	v2 ,v3
E3	:	v3 ,v1
E4	:	v3 ,v4
E5	:	v4 ,v5
E6	:	v5 ,v1

Polygon surface Table		
S1	:	E1 ,E2 ,E3
S2	:	E3 , E4 ,E5 ,E6

POLYGON MESHES

- **USEFUL TO INCREASE THE SPEED OF POLYGON RENDERING**
- **ALSO HELPS TO DECREASE THE SIZE OF THE DATABASE**

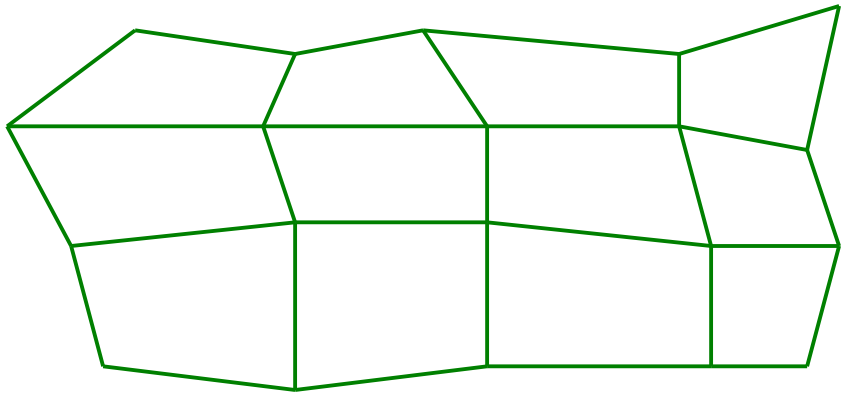
TRIANGULAR MESH



- For `n` vertices , there are `n-2`connected triangles.
- Major advantage is that they are always flat.

Quadrilateral Mesh

- For n by n array of vertices, a mesh of (n-1)by (m-1) quadrilaterals is generated.
- All 4 vertices of a quadrilateral may not lie in one plane due to numerical errors or error in selecting coordinate positions for the vertices.



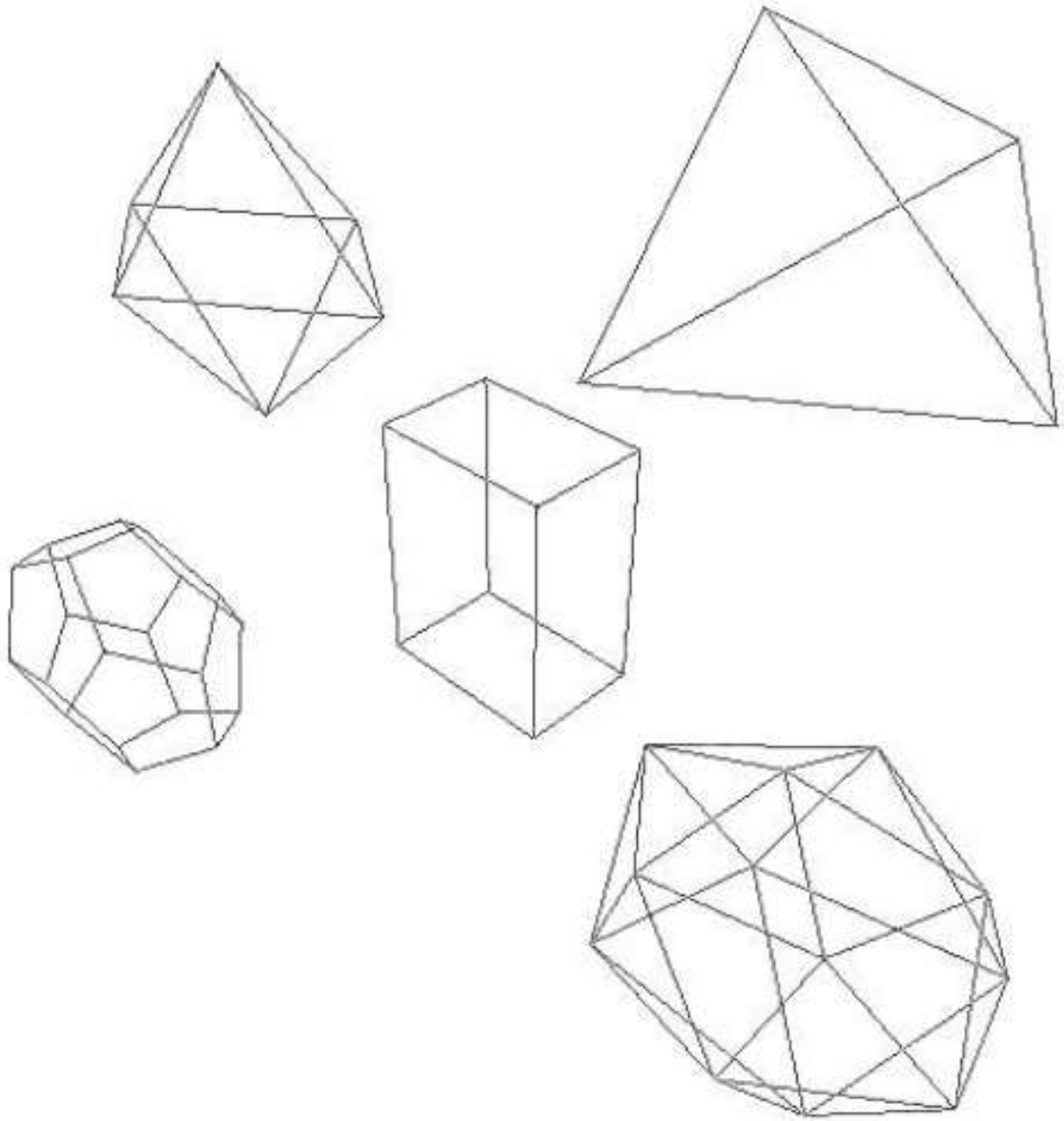


FIGURE 1
A perspective view of the five GLUT polyhedra, scaled and positioned within a display window by procedure `displayWirePolyhedra`.

Curved Surfaces

- Equations for objects with curved boundaries can be expressed in either a parametric or a nonparametric form.

- The various objects that are often useful in graphics applications include quadric surfaces, superquadrics, polynomial and exponential functions, and spline surfaces.

- These input object descriptions typically are tessellated to produce polygon-mesh approximations for the surfaces.

Quadric Surfaces

- A frequently used class of objects are the *quadric surfaces*, which are described with second-degree equations (quadratics).

- They include spheres, ellipsoids, tori, paraboloids, and hyperboloids. Quadric surfaces, particularly spheres and ellipsoids, are common elements of graphics scenes, and routines for generating these surfaces are often available in graphics packages.

- Also, quadric surfaces can be produced with rational spline representations.

Quadric Surfaces: Sphere

In Cartesian coordinates, a spherical surface with radius r centered on the coordinate origin is defined as the set of points (x, y, z) that satisfy the equation

$$x^2 + y^2 + z^2 = r^2 \tag{1}$$

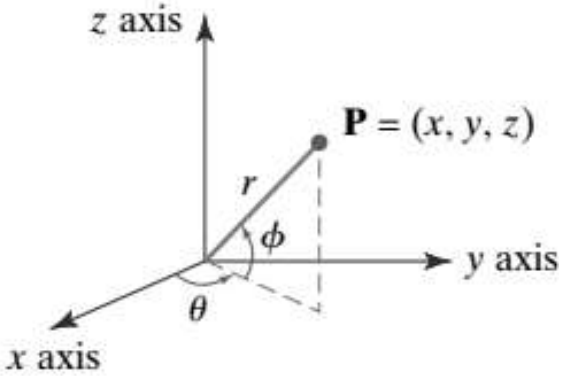


FIGURE 2
Parametric coordinate position (r, θ, ϕ) on the surface of a sphere with radius r.

We can also describe the spherical surface in parametric form, using latitude and longitude angles (Figure 2):

$$\begin{aligned} x &= r\cos\phi\cos\theta, \qquad -\pi/2 \le \phi \le \pi/2 \\ y &= r\cos\phi\sin\theta, \qquad -\pi \le \theta \le \pi \\ z &= r\sin\phi \end{aligned} \tag{2}$$

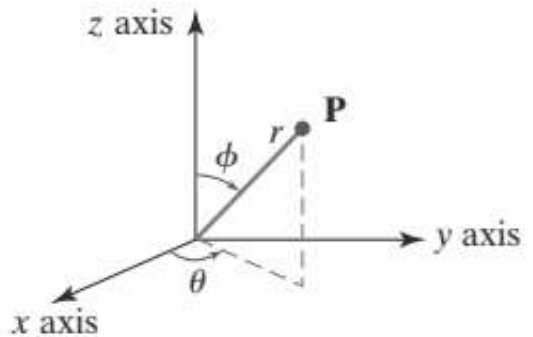


FIGURE 3
Spherical coordinate parameters (r, θ, ϕ), using colatitude for angle ϕ.

Quadric Surfaces: Ellipsoid

- An ellipsoidal surface can be described as an extension of a spherical surface where the radii in three mutually perpendicular directions can have different values (Figure 4). The Cartesian representation for points over the surface of an ellipsoid centered on the origin is:

$$\left(\frac{x}{r_x}\right)^2 + \left(\frac{y}{r_y}\right)^2 + \left(\frac{z}{r_z}\right)^2 = 1 \tag{3}$$

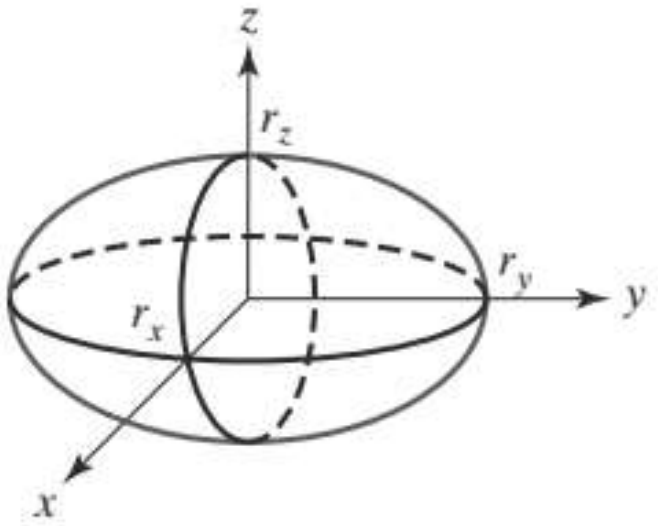


FIGURE 4
An ellipsoid with radii r_x, r_y, and r_z, centered on the coordinate origin.

And a parametric representation for the ellipsoid in terms of the latitude angle ϕ and the longitude angle θ in Figure 2 is

$$\begin{aligned} x &= r_x \cos\phi \cos\theta, & -\pi/2 \le \phi \le \pi/2 \\ y &= r_y \cos\phi \sin\theta, & -\pi \le \theta \le \pi \\ z &= r_z \sin\phi \end{aligned} \tag{4}$$

Quadric Surfaces: Torus

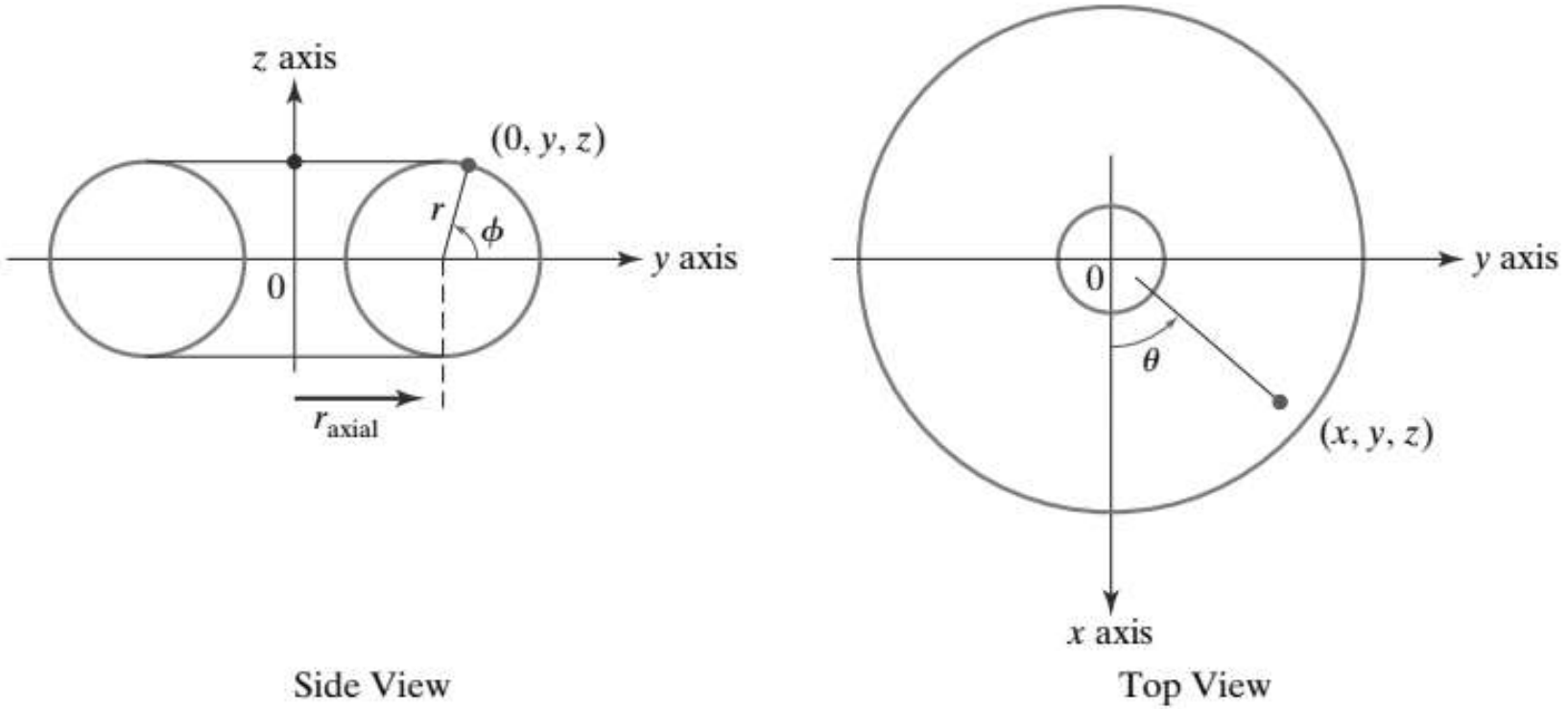


FIGURE 5
A torus, centered on the coordinate origin, with a circular cross-section and with the torus axis along the z axis.

The equation for the cross-sectional circle shown in the side view of Figure 5 is

$$(y - r_{\text{axial}})^2 + z^2 = r^2$$

Rotating this circle about the z axis produces the torus whose surface positions are described with the Cartesian equation

$$\left(\sqrt{x^2 + y^2} - r_{\text{axial}}\right)^2 + z^2 = r^2 \tag{5}$$

The corresponding parametric equations for the torus with a circular cross-section are

$$\begin{aligned} x &= (r_{\text{axial}} + r\cos\phi)\cos\theta, \qquad -\pi \le \phi \le \pi \\ y &= (r_{\text{axial}} + r\cos\phi)\sin\theta, \qquad -\pi \le \theta \le \pi \\ z &= r\sin\phi \end{aligned} \tag{6}$$

We could also generate a torus by rotating an ellipse, instead of a circle, about the z axis. For an ellipse in the yz plane with semimajor and semiminor axes denoted as r_y and r_z, we can write the ellipse equation as

$$\left(\frac{y - r_{\text{axial}}}{r_y}\right)^2 + \left(\frac{z}{r_z}\right)^2 = 1$$

where r_{axial} is the distance along the y axis from the rotation z axis to the ellipse center. This generates a torus that can be described with the Cartesian equation

$$\left(\frac{\sqrt{x^2 + y^2} - r_{\text{axial}}}{r_y}\right) + \left(\frac{z}{r_z}\right)^2 = 1 \tag{7}$$

The corresponding parametric representation for the torus with an elliptical cross-section is

$$\begin{aligned} x &= (r_{\text{axial}} + r_y \cos\phi)\cos\theta, \qquad -\pi \le \phi \le \pi \\ y &= (r_{\text{axial}} + r_y \cos\phi)\sin\theta, \qquad -\pi \le \theta \le \pi \\ z &= r_z \sin\phi \end{aligned} \tag{8}$$

Other variations on the preceding torus equations are possible. For example, we could generate a torus surface by rotating either a circle or an ellipse along an elliptical path around the rotation axis.

Superquadrics

- The class of objects called **Superquadrics** is a generalization of the quadric representations. Superquadrics are formed by incorporating additional parameters into the quadric equations to provide increased flexibility for adjusting object shapes.

- One additional parameter is added to curve equations, and two additional parameters are used in surface equations.

Superquadrics: Superellipse

One way to do this is to write the Cartesian superellipse equation in the form

$$\left(\frac{x}{r_x}\right)^{2/s} + \left(\frac{y}{r_y}\right)^{2/s} = 1 \quad \text{write on desk} \tag{9}$$

where parameter s can be assigned any real value. When $s = 1$, we have an ordinary ellipse.

Corresponding parametric equations for the superellipse of Equation 9 can be expressed as

$$\begin{aligned} x &= r_x \cos^s \theta, \qquad -\pi \le \theta \le \pi \\ y &= r_y \sin^s \theta \end{aligned} \tag{10}$$

Figure 6 illustrates superellipse shapes that can be generated using various values for parameter s.

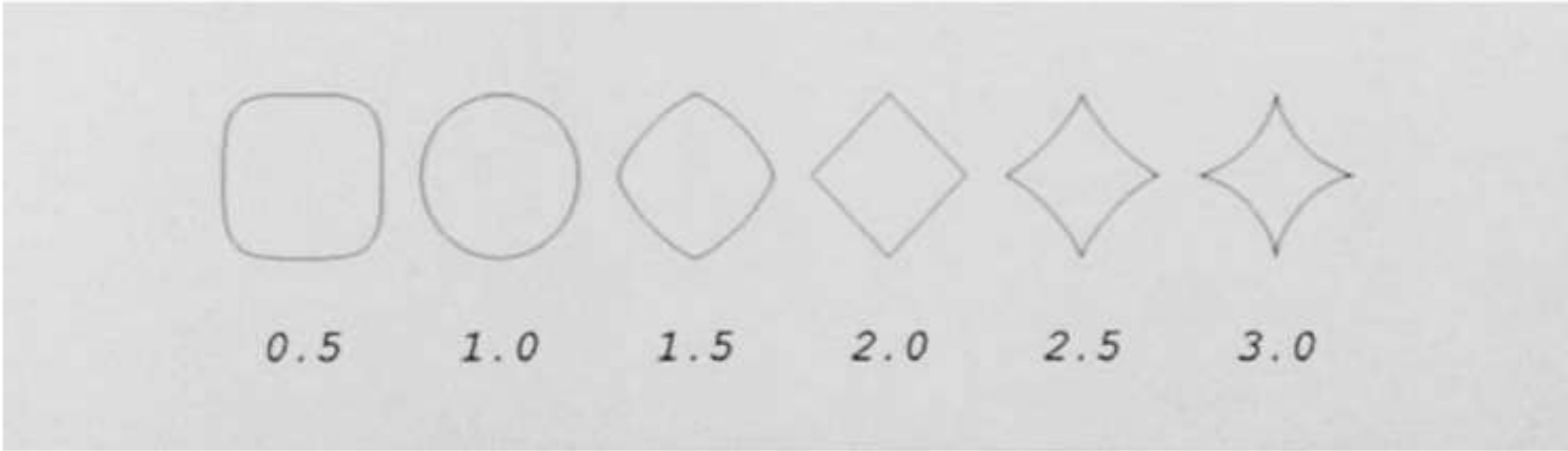


FIGURE 6
Superellipses plotted with values for parameter s ranging from 0.5 to 3.0 and with $r_x = r_y$.

Superquadrics: Superellipsoid

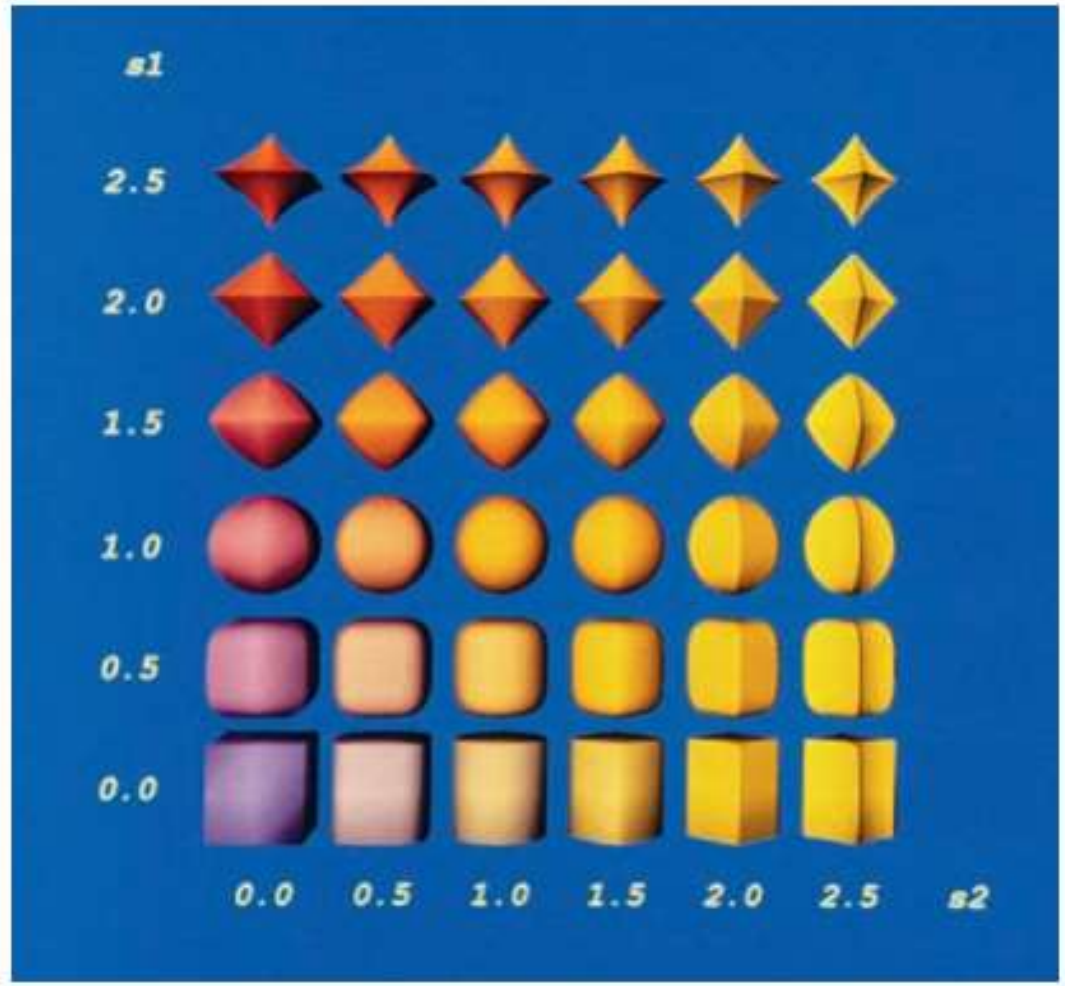
A Cartesian representation for a superellipsoid is obtained from the equation for an ellipsoid by incorporating two exponent parameters as follows:

$$\left[\left(\frac{x}{r_x}\right)^{2/s_2} + \left(\frac{y}{r_y}\right)^{2/s_2}\right]^{s_2/s_1} + \left(\frac{z}{r_z}\right)^{2/s_1} = 1 \tag{11}$$

For $s_1 = s_2 = 1$, we have an ordinary ellipsoid.

We can then write the corresponding parametric representation for the superellipsoid of Equation 11 as

$$\begin{aligned} x &= r_x \cos^{s_1}\phi \cos^{s_2}\theta, & -\pi/2 \le \phi \le \pi/2 \\ y &= r_y \cos^{s_1}\phi \sin^{s_2}\theta, & -\pi \le \theta \le \pi \\ z &= r_z \sin^{s_1}\phi \end{aligned} \tag{12}$$



Color Plate 10

Superellipsoids plotted with values for parameters s_1 and s_2 ranging from 0.0 to 2.5 and with $r_x = r_y = r_z$.

Curves and Surfaces

Bézier Curves & Surfaces

Representation of Curves

Brute Force Approach

- A curve can be approximated by a finite number of short straight line segments.
- To get a better approximation we can use more segments per unit length.
- This increases the amount of data required to store the curve and makes it difficult to manipulate.
- We clearly need a way of representing these curves in a more mathematical fashion.

Some Desirable Properties

- Reproducible - the representation should give the same curve every time;
- Computationally Quick;
- Easy to manipulate, especially important for design purposes;
- Flexible;
- Easy to combine with other segments of curve.

Categories of Curves

- **Interpolating Curves**

  - These curves will pass through the points used to describe it.

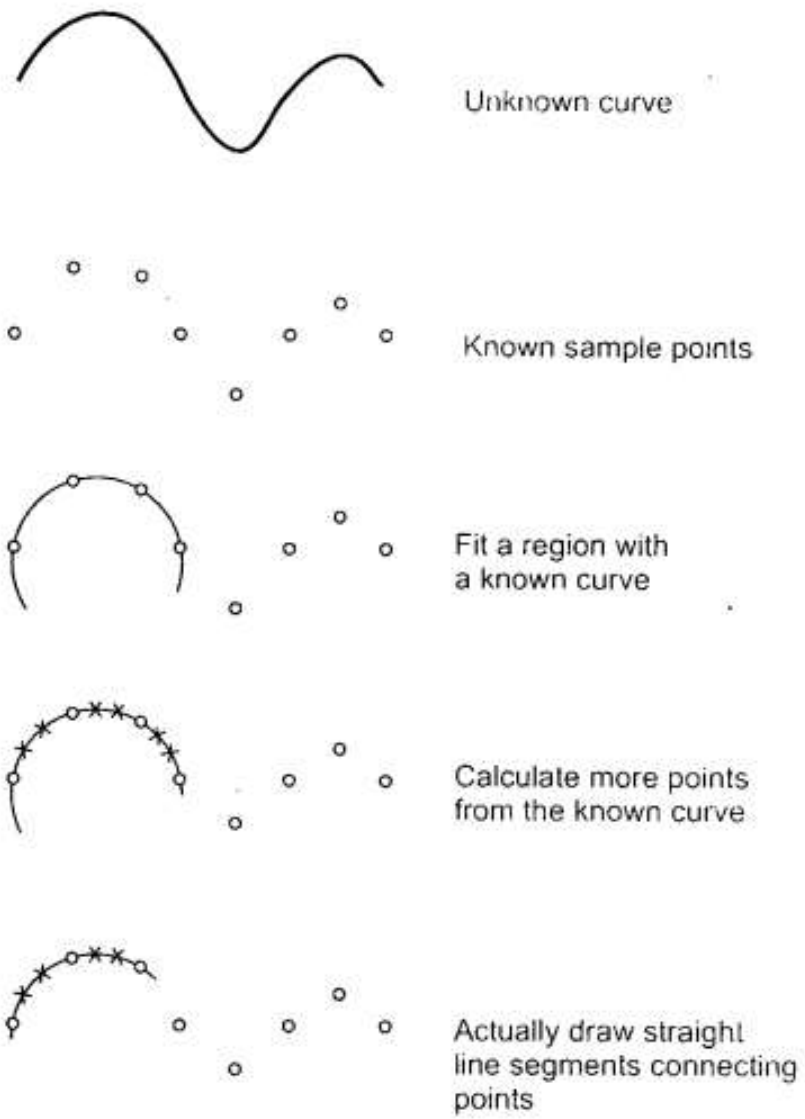
  - The points through which the curve passes are known as ***knots***.

- **Approximation Curves**

  - An approximating curve will get near to the points without necessarily passing through any of them.

Interpolation

FIGURE 1
A set of six control points interpolated with piecewise continuous polynomial sections.



Approximation

FIGURE 2
A set of six control points approximated with piecewise continuous polynomial sections.

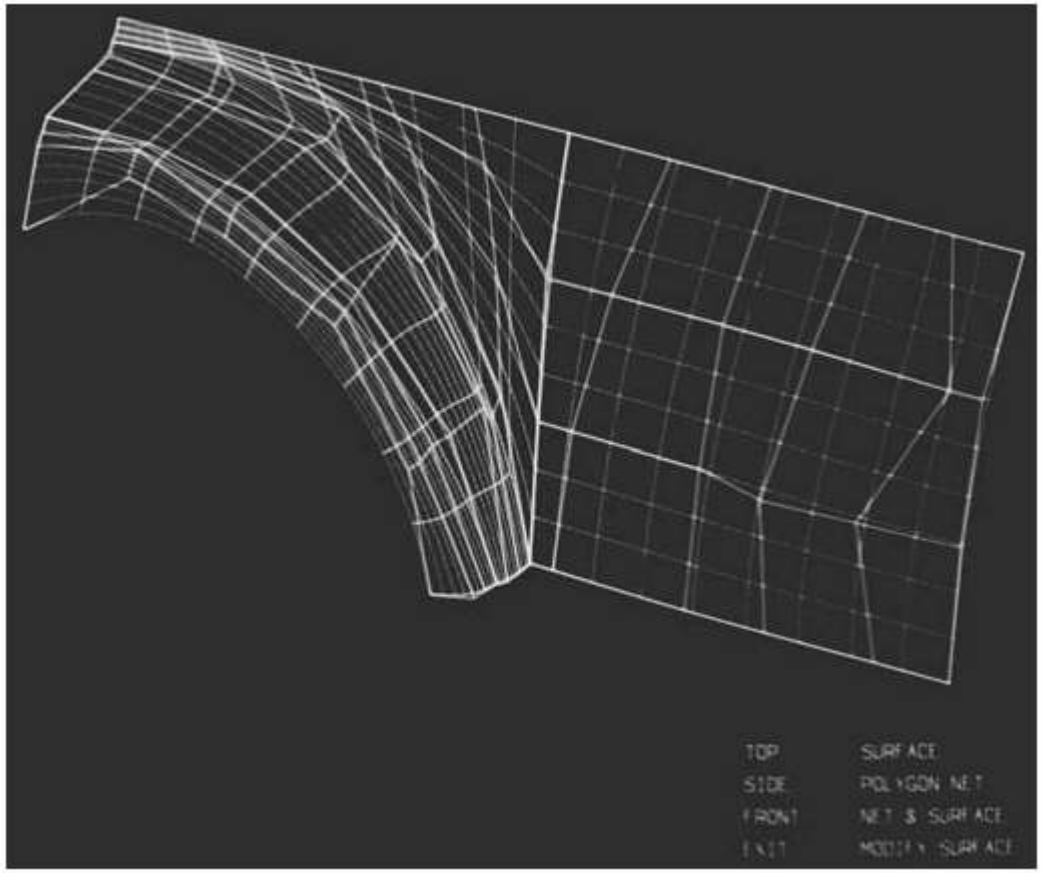


FIGURE 3
An approximation spline surface for a CAD application in automotive design. Surface contours are plotted with polynomial curve sections, and the surface control points are connected with straight-line segments. (*Courtesy of Evans & Sutherland.*)

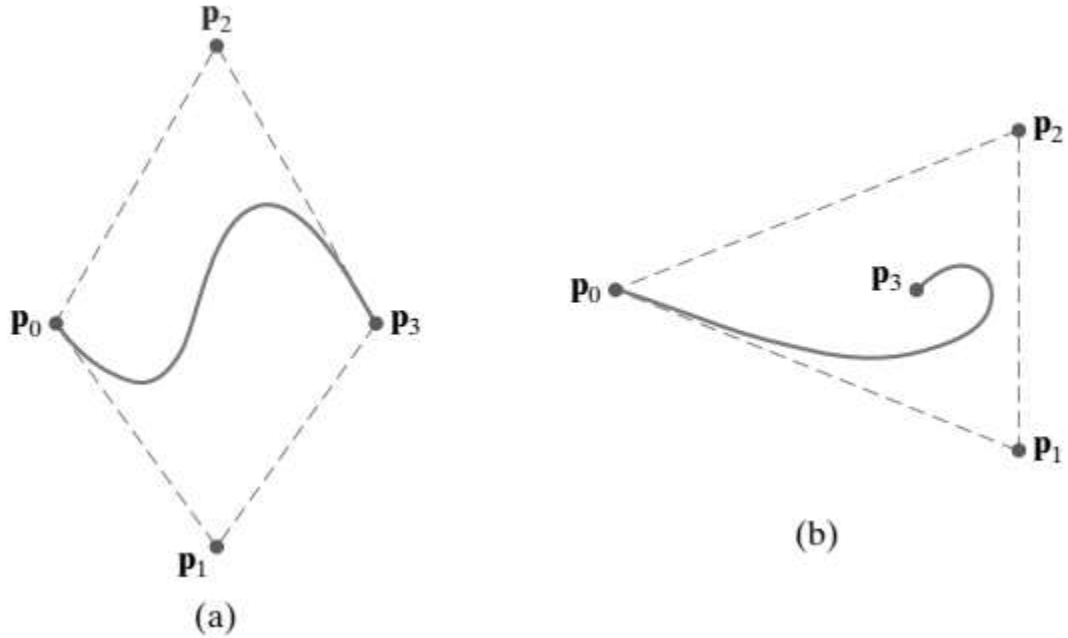


FIGURE 4
Convex-hull shapes (dashed lines) for two sets of control points in the *x y* plane.

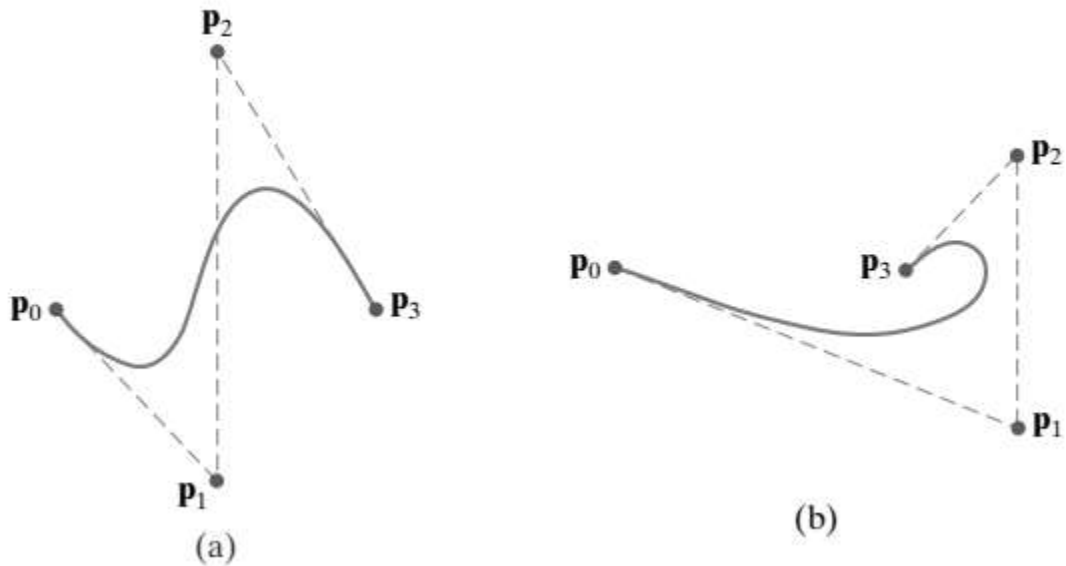


FIGURE 5
Control-graph shapes (dashed lines) for two sets of control points in the *x y* plane.

Curve Representation

Non-parametric Representation

- Implicit Representation of a circle

$$x^2 + y^2 - r^2 = 0$$

- Explicit Representation of a circle

$$y = + \sqrt{[(r^2 - x^2)]} \quad \textit{for the upper half}$$

$$y = - \sqrt{[(r^2 - x^2)]} \quad \textit{for the lower half}$$

Explicit vs. Implicit Rep. : Problems

- The Explicit form is satisfactory when the function is single-valued and the curve has no vertical tangents.
- The implicitly defined curves require the solution of a non-linear equation for each point and thus numerical procedures have to be employed.
- Both explicitly and implicitly represented curves are axis-dependent.

Parametric Representation of Curves

- The parametric form uses an auxiliary parameter to represent the position of a point.
- For example, a circle with center at origin may be represented by an angle parameter $u \in [0,2\pi]$:

$$x(u) = r.\cos(u), \quad y(u) = r.\sin(u)$$

Advantages of Parametric Representation

- Since a point on a parametric curve is specified by a single value of parameter, the use of parametric techniques free us from dependence on any particular system of coordinates.
- It avoids problems which can arise in representing closed or multiple-valued curves and curves with vertical tangents in a fixed coordinate system.
- The parametric methods lends itself to the piecewise description of curves and surfaces, which is the basic technique for the description of free-form shapes.

Bases and Control Points

- Most methods used in Computer Graphics depend upon polynomial representations. A polynomial of degree *k* in *t* can be written in terms of *k+1* coefficients as

$$C(t) = c_0 + c_1 t + c_2 t^2 \ldots c_k t^k \qquad \text{----- (1)}$$

- The representation above for a polynomial is not ideal. Its shape depends upon the values given to the coefficients c_i and it is not easy to predict the effect i.e. the effect of change in these coefficients on the shape of the curve.

Primitive Polynomial Basis

- In three-dimensional Euclidean space an arbitrary vector is described as a linear combination of three independent vectors.
- These vectors form a **basis** for the space and allow the definition of any vector in terms of the basis.
- The most obvious independent functions to use as a basis for a k-dimensional polynomial space are:

$$b_i(t) = t^i \qquad 0 \leq i \leq k$$

Representing Polynomials in terms of Basis Functions

- Using equation (1), a quadratic curve could be defined by:

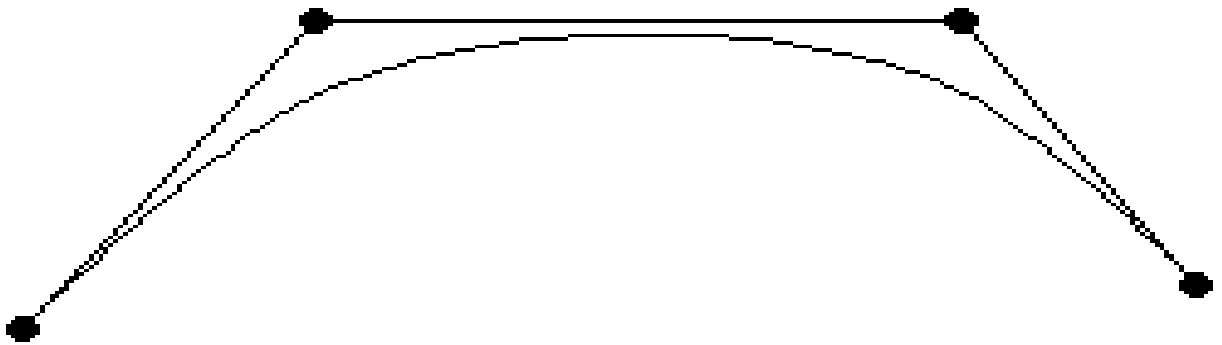
$$\begin{bmatrix} C_x(t) \\ C_y(t) \end{bmatrix} = \begin{bmatrix} 1.5 \\ 3.1 \end{bmatrix} + \begin{bmatrix} 4.2 \\ 9.5 \end{bmatrix} t + \begin{bmatrix} -1.5 \\ 6.1 \end{bmatrix} t^2$$

- By changing coefficients (points in 2D space), we can change the shape of the curve. Hence they are known as **control points $\mathbf{p_i}$**.

- Therefore the general equation of a curve in terms of basis functions $b_i(t)$ and control points p_i is as follows:

$$C(t) = \sum_{i=0}^{k} p_i b_i(t)$$

- when *k=3,* a cubic curve and its control points might appear as follows:



- Various Sets of Basis functions have been used in Computer Graphics for the specification of curves.
- Among these are the **Bézier Basis** and **B-Spline Basis**.

Advantages of Cubic Curves

1. A cubic curve is the lowest order curve which offers enough shape flexibility, i.e. it is the lowest order curve with a point of inflexion (a point on a curve at which that curvature changes from convex to concave or vice versa).
2. Higher order curves when fitted to data points can exhibit oscillations between the data points and take more time to evaluate.
3. Cubic segments can be fitted together with ***second degree continuity***, this allows smooth joins between segments.

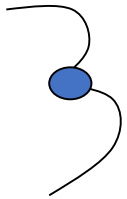
Parametric Continuity Conditions

- To ensure a smooth transition from one section of a *piecewise parametric curve* to the next, we can impose various continuity conditions at the connection points.
- Suppose each section of a connected curve is described with following parametric equations:

  $x=x(u), \quad y=y(u), \quad z=z(u), \quad u_1 \leq u \leq u_2$
- We set the parametric continuity by matching the parametric derivatives of adjoining curve sections at their common boundary.

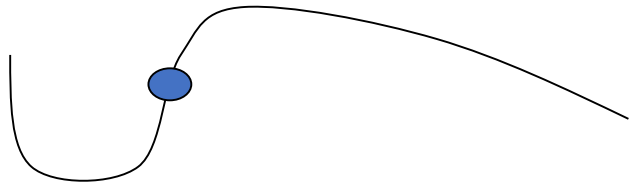
Zero Order C^0 Continuity

- ✓Last point of first section is same as first point of the second section.



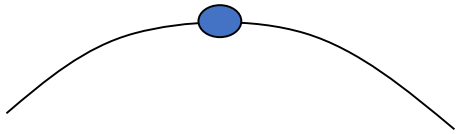
First Order C^1 Continuity

- ✓Tangents of the two sections are the same at their joining point.



Second Order C^2 Continuity

- ✓Both the first & second parametric derivatives of the two curve sections are the same at the join.



Geometric Continuity Conditions

- Another method for joining two successive curve sections is to specify conditions for **geometric continuity.**
- In this case, we require only that the parametric derivatives of the two sections are proportional to each other at their common boundary, instead of requiring equality.
- **Zero-order geometric continuity,** described as *G*0 continuity, is the same as zero-order parametric continuity. That is, two successive curve sections must have the same coordinate position at the boundary point.
- **First-order geometric continuity,** or *G*1 continuity, means that the parametric first derivatives are proportional at the intersection of two successive sections.
- **Second-order geometric continuity,** or *G*2 continuity, means that both the first and second parametric derivatives of the two curve sections are proportional at their boundary. Under *G*2 continuity, curvatures of two curve sections will match at the joining position.

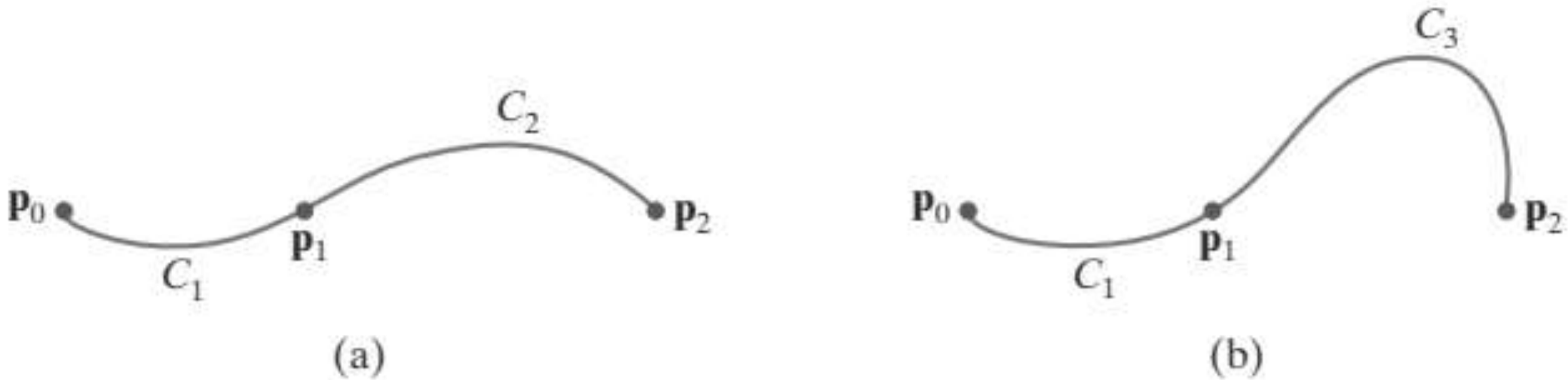
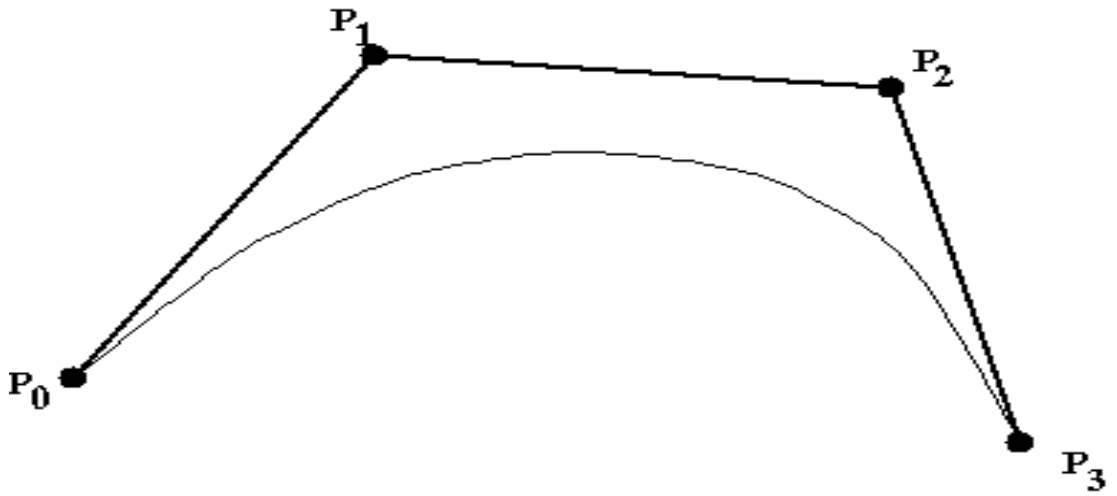


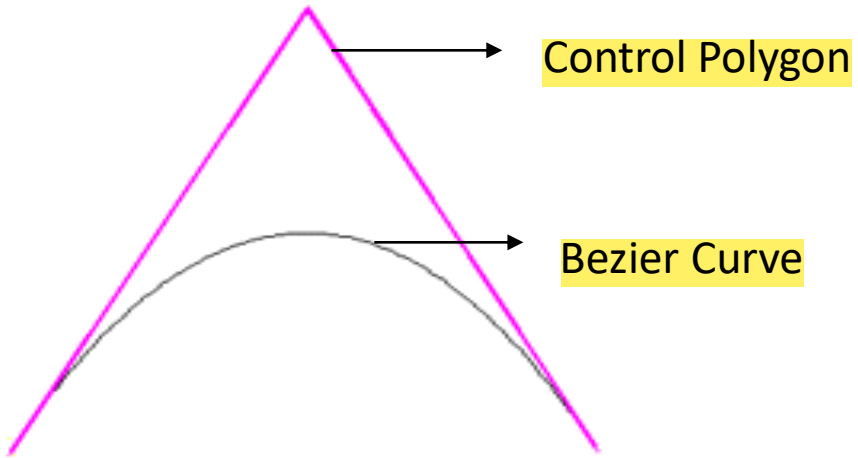
FIGURE 7
Three control points fitted with two curve sections joined with (a) parametric continuity and (b) geometric continuity, where the tangent vector of curve C_3 at point $\mathbf{P}_1$ has a greater magnitude than the tangent vector of curve C_1 at $\mathbf{P}_1$.

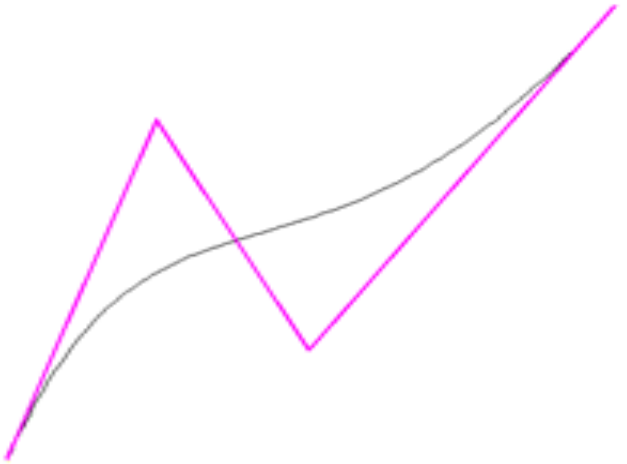
Bézier Curves

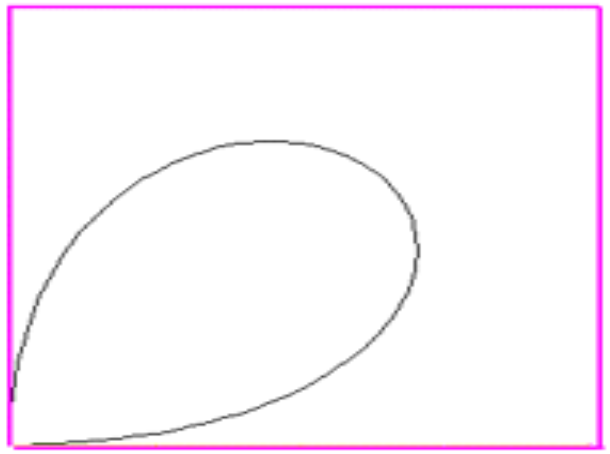
- Bézier functions are a set of polynomials which can be used instead of the primitive polynomial basis and have some useful properties for interactive curve design.
- The cubic Bézier Curve with the four control points P_0, P_1, P_2 and P_3 is illustrated below:

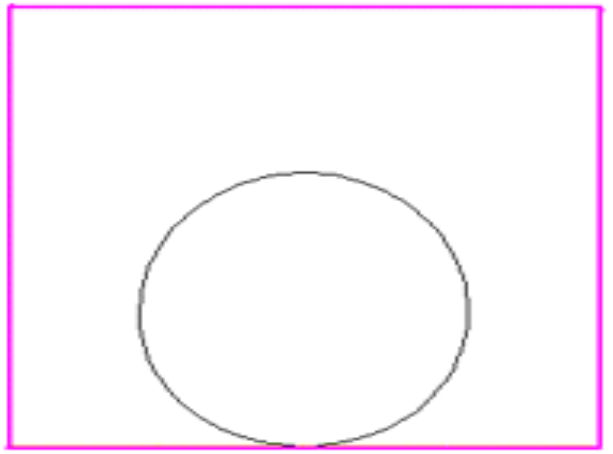


Examples: 2-D Bezier Curves









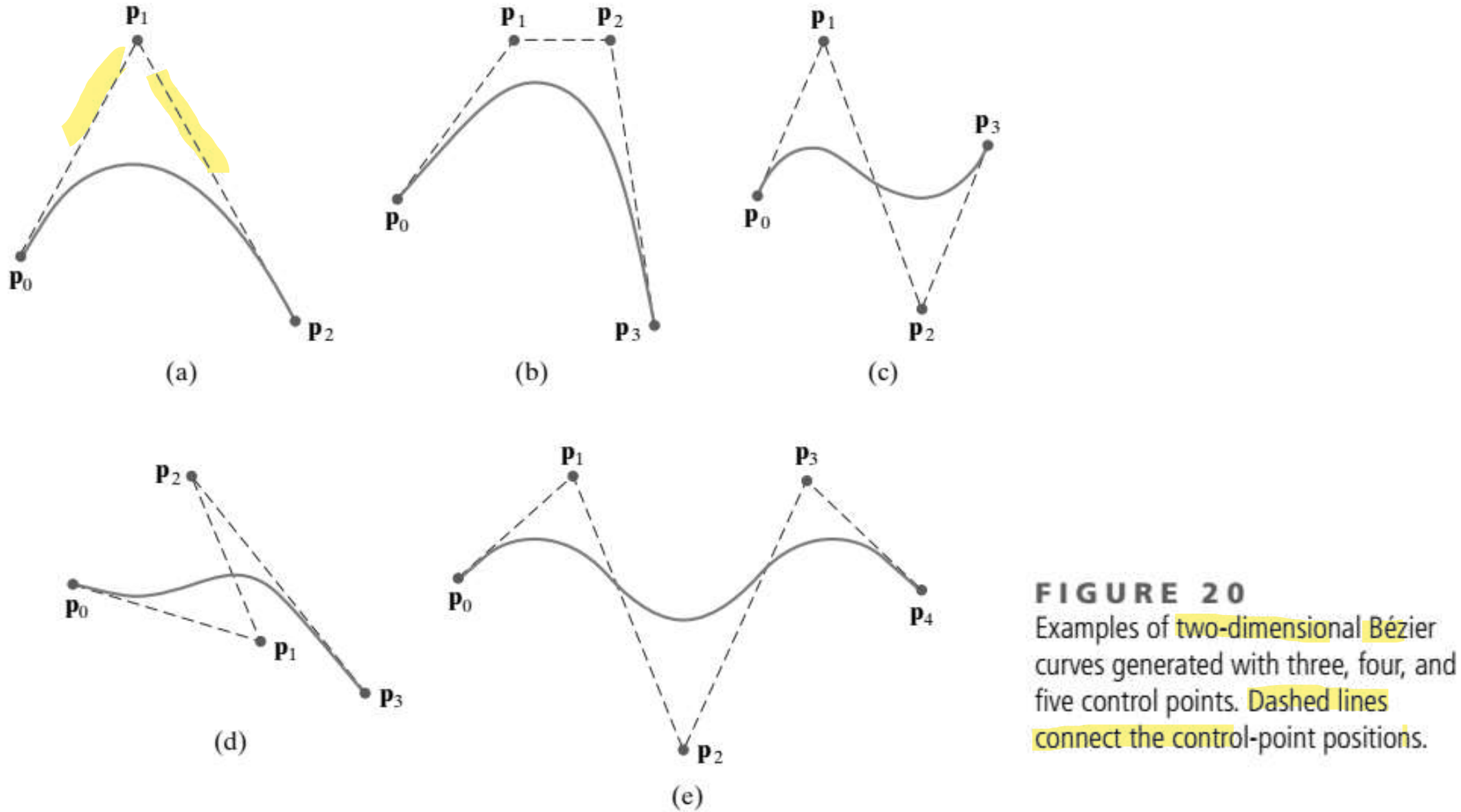


FIGURE 20
Examples of two-dimensional Bézier curves generated with three, four, and five control points. Dashed lines connect the control-point positions.

- The cubic Bézier curve is defined in the range *[0,1]* in *t* by:

Write on desk

$$B_3(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3$$

- The cubic Bézier (blending) functions are then:

Sare coefficient

$$(1-t)^3 \ , \ 3t(1-t)^2 \ , \ 3t^2(1-t) \ , \ t^3$$

- These are sometimes called the **Bernstein polynomials** of order three.

- Expanding **Bernstein polynomials** in terms of the primitive polynomial basis functions of degree three the curve definition can be written in the form:

$$C(t) = [t^3, t^2, t, 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

$$= [t^3, t^2, t, 1] M_b G_b$$

Write on desk

- where M_b and G_b are the Bézier **Geometric Basis Matrix** and **Geometric Vector** respectively.

- It is easy to generalize the above to curves of degree *n* with *n+1* control points. Thus:

$$C(t) = \sum_{i=0}^{n} b_{n,i}(t) P_i$$

write on desk

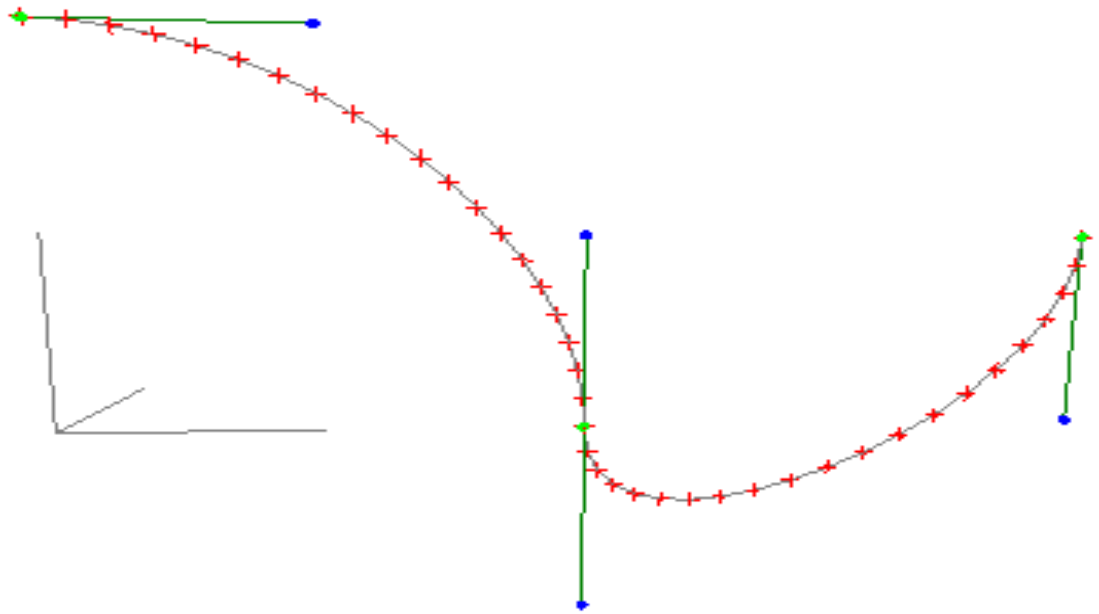
where the Bézier Blending Functions $b_{n,i}(t)$ are defined by:

$$b_{n,i}(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

- In practice it is better to use piecewise connected Bézier cubics rather than higher order Bézier curves.

Piecewise Cubic Bézier Curves

- Multiple curve pieces can be joined together to form longer continuous curves.
- The curve is made continuous by setting the tangents the same at the join (C^1 continuity).



0 0 1
0.5 0 1
1 0 0.5
1 0 0

1 0 0
1 0 -0.5
2 0 0
2 0 0.5

Construction of Bezier Curve

Recursive Subdivision approach (De Casteljau algorithm)

Recursive definition of any point on the surface as series of weighted combinations of control points

$$p_i^j(t) = (1-t)\, p_i^{j-1}(t) + t\, p_{i+1}^{j-1}(t)$$

for $i=0,1,\ldots, n$ and $j=0,1,2,\ldots, n-i$.

recurrence relation write on desk

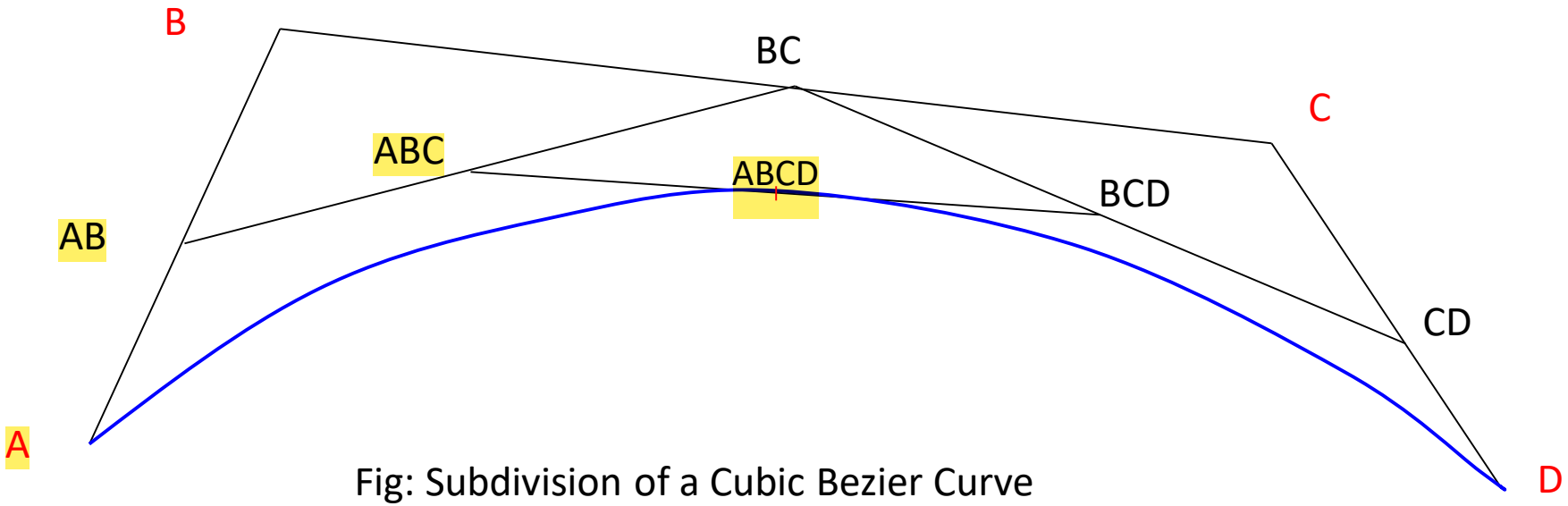


Fig: Subdivision of a Cubic Bezier Curve

- The point ABCD in on Bezier Curve.
- Control points for first section of Bezier curve

  A, AB, ABC and ABCD
- Control points for second section of Bezier curve

  ABCD, BCD, CD and D
- This splitting process continues until the sections are no bigger than individual pixels.

Properties of Bezier Curves

1. The curve passes through the start and finish points of the control polygon defining the curve.
2. The tangent to the curve at $t = 0$ lies in the direction of the line joining the first point to the second point. Also the tangent to the curve at $t=1$ is in the direction of the line joining the penultimate point to the last point.
3. Any point on the curve lies inside the **convex hull** of the control polygon.
4. All control points affect the entire curve.

5. The order of the curve is related to the number of control points. Hence using many control points to control the curve shape means evaluating high order polynomials.
6. The curve is transformed by applying any affine transformation to its control points and generating the transformed curve from the transformed control points.
7. No line can intersect the curve more than twice if the four control points form an open polygon. Thus there can be no loops in the curve and it must be smooth. This is called the **Variation Diminishing Property**.

Bezier Surfaces

- The Bezier curves can be extended to surfaces.
- We can construct a bicubic Bezier surface from a 4*4 grid of control points.
- Recursive subdivision can be used to draw such patches by repeatedly quartering them until they may be replaced by polygons or pixels.

SPLINE REPRESENTATION

Three representations/specifications of any spline are:

- The set of boundary conditions specifying the spline.
- The matrix that characterizes the spline.
- The set of blending functions that characterizes the spline.

A spline was originally a flexible strip of metal used by a draughtsman to draw curves.

CUBIC-SPLINES INTERPOLATION METHODS

Suppose we have n+1 control points specified with coordinates

$p_k = (x_k, y_k, z_k)$ $k=0,1,2,\ldots\ldots n$

point k

Equations describing cubic polynomials between each pair of control points:

$$x(u)= a_x u^3 + b_x u^2 + c_x u + d_x$$
$$y(u)= a_y u^3 + b_y u^2 + c_y u + d_y \quad (0 \le u \le 1)$$
$$z(u)= a_z u^3 + b_z u^2 + c_z u + d_z$$

Given a set of “n+1”control points, we want to define`n`different cubic splines that interpolate the control points. There are different method for setting the boundary conditions at the joints between curve sections so that we can obtain numerical values for all the coefficients.

NATURAL CUBIC SPLINES

- **The total curve has c^2 continuity.**

- **Given ‘n’ curve sections, we have ‘4n’coefficients to calculate.**

- **‘n-1’ interior points, each provide 4 constraints $\Rightarrow$**

  4n-4 equations

  - c^1 continuity for adjacent splines.

  - c^2 continuity for adjacent splines.

  - c^0 continuity : curve n ends at the point, and curve n+1 begins at the point.

- Exterior points provide 2 more constraints $\Rightarrow$ 4n -4 +2
  - the curve ends at each end-point.

- Need to specify two more constraints
  - -set the second derivatives at p_0 and p_n to 0, or
  - -specify two dummy points ,one at each end, so that there are ‘n+1’interior points.

Disadvantage- natural cubic splines are globally sensitive to changes at a single control points.

HERMITE SPLINE

Hermite spline allow local control of a spline. User specifies the tangent at each control point.

Hermite splines can be calculated for two control points p_k and p_{k+1} :

$p(0)=p_k$

$p(1)=p_{k+1}$

$p'(0)=D_{pk}$

$p'(1)=D_{pk+1}$

Matrix equivalent of Hermite-curve section

$p(u)=au^3+bu^2+cu+d, \qquad 0\leq u \leq 1$

is

$p(u)= [u^3 \quad u^2 \quad u \quad 1] \; [a \quad b \quad c \quad d]^T$

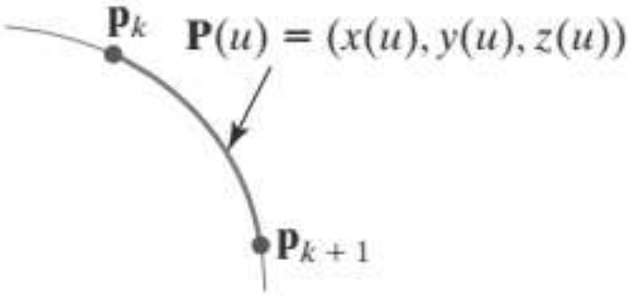


FIGURE 10
Parametric point function $\mathbf{P}(u)$ for a Hermite curve section between control points $\mathbf{p}_k$ and $\mathbf{p}_{k+1}$.

Contd…

Set up the derivate form

$$p'(u)=[3u^2 \;\; 2u \;\; 1 \;\; 0] \cdot \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

the complete matrix for the endpoint values **0** and **1**

$$\begin{bmatrix} Pk \\ Pk+1 \\ Dp\,k \\ Dpk+1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

Solving for polynomial coefficients ,we have

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} Pk \\ Pk+1 \\ Dp\,k \\ Dpk+1 \end{bmatrix} = M_h \begin{bmatrix} Pk \\ Pk+1 \\ Dp\,k \\ Dpk+1 \end{bmatrix}$$

Where M_h ,the Hermite matrix, is the inverse of the boundary constraint matrix.

Drawback –

It requires input values for the curve derivatives at the control points.

Note- Cardinal splines and Kochanek-Bartels splines (variant of Hermite splines) do not require above additional information.

APPROXIMATING SPLINES: B-SPLINE CURVES

B-splines have two advantages over Bézier splines:

(1) the degree of a B-spline polynomial can be set independently of the number of control points (with certain limitations), and

(2) B-splines allow local control over the shape of a spline.

The tradeoff is that B-splines are more complex than Bézier splines.

APPROXIMATING SPLINES: B-SPLINE CURVES...

There are two major disadvantages of Bezier curves, namely:

1. Their non-localness: while a control point mainly influence the shape of the curves close to it, it also affects the entire curve to some extent.

2. The fact that the degree of the curve is related to the number of control points. Thus either high order polynomials have to be used or multiple low-degree curve segments have to be used.

The **B-spline** curve gets around these difficulties.

APPROXIMATING SPLINES: B-SPLINE CURVES

Definition –

A B-spline is a set of piecewise (usually cubic) polynomial segments that pass close to a set of control points.

A B-spline curve P(u) of order k (or degree k-1) is defined as

$$P(u) = \sum_{i=0}^{n} PiNi,k(u)$$

Write on desk

Contd…

Where $\sum_{i=0}^{n} PiNi,k(u)$

- the $\{P_i : i=0,1,...,n\}$ are the control points.
- $N_{i,k}(u)$ are "normalized B-spline blending functions" of degree k-1.
- the parameter u defines a knot vector

  $u= \{u_0,u_1,.....u_m\}$

Here n (no. of control points), m (knot points) and k (order of curve) satisfy the following conditions

$m=n+k+1$

Blending functions $N_{i,k}$ for B-spline curve are defined by the Cox-deBoor recursion formula:

when k=1

$$N_{i,1}(u)= \begin{cases} 1, & u \in [u_i , u_{i+1}) \\ 0, & \text{otherwise} \end{cases}$$

write on desk

Contd…

And if k>1

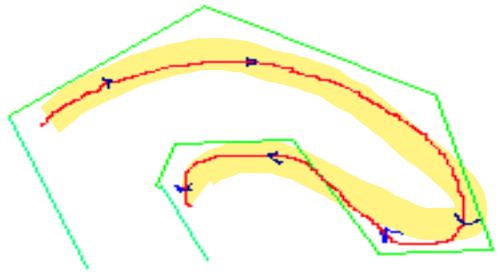
write on desk

$$N_{i,k}(u)= (u- u_i)/ (u_{i+k-1}-u_i) N_{i,k-1}(u) + (u_{i+ k} –u)/(u_{i+ k} –u_{i+1})N_{i+1,k-1} (u)$$

the order k is independent of the no. of control points (n+1).

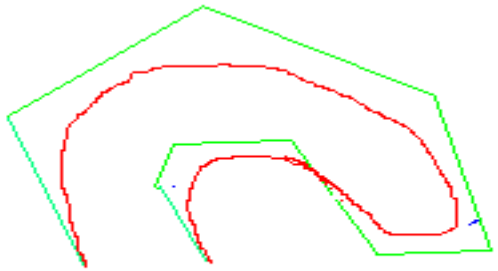
Open B-spline

If the knot vector does not have any particular structure, the generated curve will not touch the first and last lags of the control polygon.



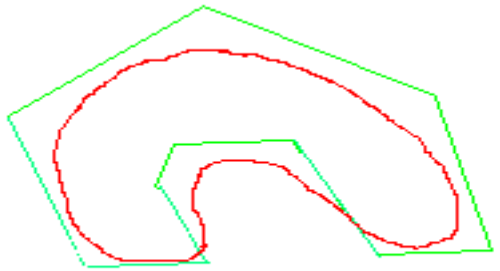
Clamped B-spline

We may want to clamp the curve so that it is tangent to first and last lags just like the Bezier curve does. To do so, the first knot and the last knot must be repeated (k+1) times. This type of B-spline curve is called clamped curve.



Closed B-spline

By repeating some knots and control points, the generated curve can be a closed one.



Repeated control points P_0, P_1, P_2 at the end of the sequence-

$P_0, P_1, \ldots\ldots P_m, P_0, P_1, P_2$

Matrix representation

The curve between P_i and P_{i+1} is defined by

write on desk

$$C_i(u) = 1/6 \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix}$$

Summary of properties of B-spline curve

Variation Diminishing and Convex Hull properties hold

- The polynomial curve has degree $d-1$ and C^{d-2} continuity over the range of u.
- For $n+1$ control points, the curve is described with $n+1$ blending functions.
- Each blending function $B_{k,d}$ is defined over d subintervals of the total range of u, starting at knot value u_k.
- The range of parameter u is divided into $n+d$ subintervals by the $n+d+1$ values specified in the knot vector.
- With knot values labeled as $\{u_0, u_1, \ldots, u_{n+d}\}$, the resulting B-spline curve is defined only in the interval from knot value u_{d-1} up to knot value u_{n+1}. (Some blending functions are undefined outside this interval.)
- Each section of the spline curve (between two successive knot values) is influenced by d control points.
- Any one control point can affect the shape of at most d curve sections.

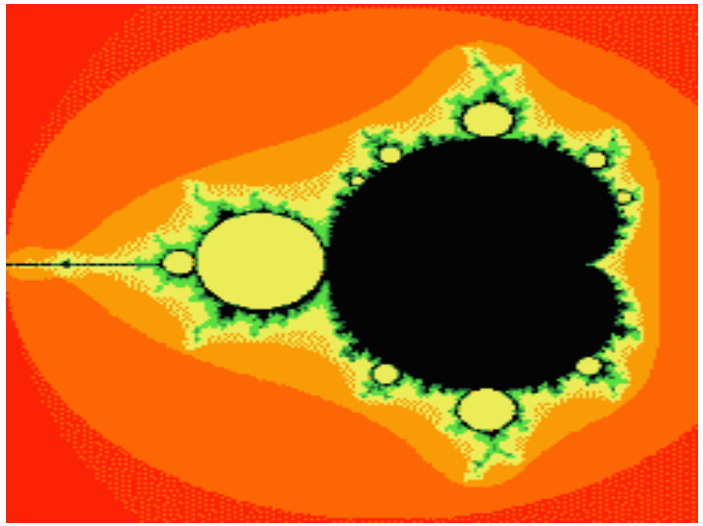
Contd…

5. Variation Diminishing and Convex Hull properties hold.

Limitation- B-spline curves (like Bezier curves) are polynomial curves and can not represent many useful curves such as circles and ellipses.

Thus a generalization of B-spline, called NURBS (Non- Uniform Rational B-Spline), is required.

Fractals



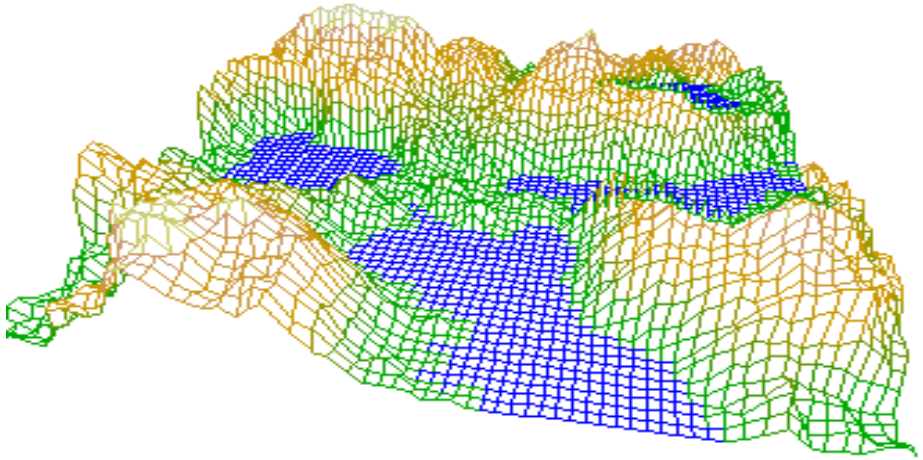
Modeling Complex Shapes

- Modeling of complex objects is a difficult process.
- In particular, geometric modeling of complex forms or mathematical modeling of complex systems having non-linear relationship have no unified methodology.
- Natural objects such as coastlines, mountains, cloud formations, trees etc. constitute an important class of complex objects.

Examples

Fractal fern

Fractal cloud



Fractal landscape

The Problem

- In order to represent these complex shapes, the traditional methods like the ***Euclidean geometry*** are unable to describe elegantly such complex objects.
- The Euclidean geometry deals with simple shapes like lines and planes, circles and spheres, triangle and cones etc.
- It is not able to describe complex shapes which may be irregular, fragmented, tangled, twisted or fractured.

The Solution

- The family of complex shapes may be described by what the French Mathematician Benoit Mandelbrot (year 1975) called ***fractals***.
- The word ‘fractal’ derives its origin from the Latin word *fractus* meaning ‘irregular and fragmented’.
- The fractal geometry has helped to reconnect pure mathematics with natural sciences and computing.

Euclidean and Fractal Geometry

Geometry is a mathematical language to describe, relate and manipulate shapes.

Euclidean

- Traditional (over 2000 years)
- Based on characteristic size & scale
- Suits for man-made objects
- Described by a non-recursive formula

Fractal

- Modern monsters (20 years)
- No specific size and scale
- Appropriate for natural shapes
- Described by algorithm or recursive formula

Fractal: Definition & Properties

- Fractals are infinitely magnifiable irregular objects with fractional dimension which can be produced by a small set of instructions and data.
- Important properties of fractals are:
  - Self-similarity
  - Fractional Dimensions
  - Ill defined characteristic scale of length
  - Formation by Iteration

1. Self-Similarity

- Geometric figures are ***similar*** if they have the same shape i.e. the corresponding sides are in proportion. For example, the following two squares are *similar*.

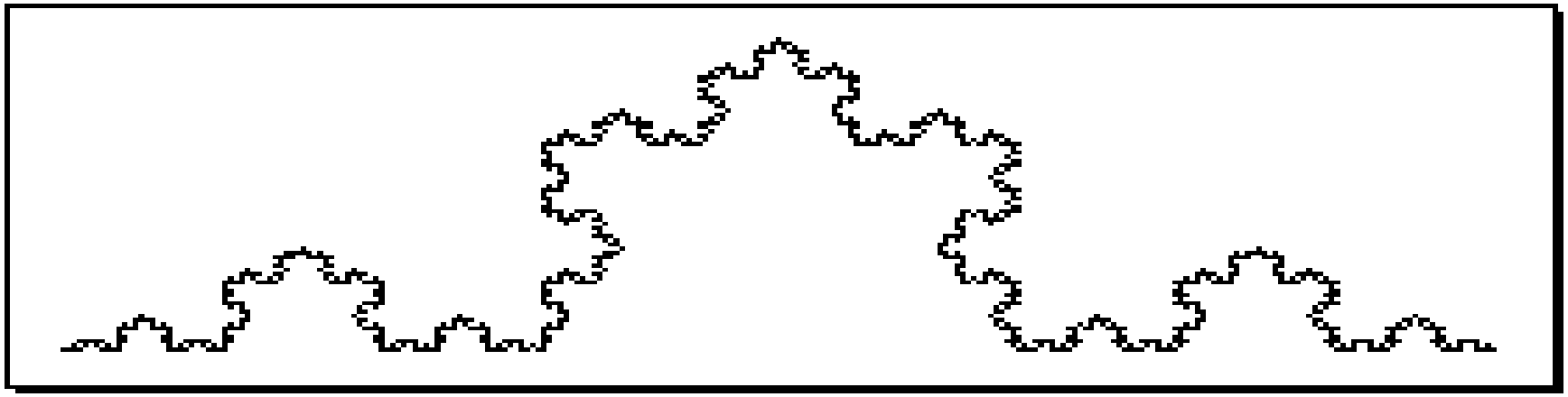
- The following two rectangles are *not similar*.

- But the two rectangles below are *similar*.

Self-Similarity of Fractals

- As fractal structure are examined at smaller and smaller scales i.e. as the view of them is magnified more and more, the smaller scale versions seem to resemble the large scale version.
- Self-similarity, in other words invariance against change in scale or size, is an attribute of many laws of nature.

Example: Self-similarity of fractals

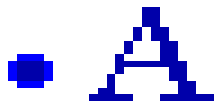


Koch Curve

2. Fractional Dimension

Dimension of Geometric Objects

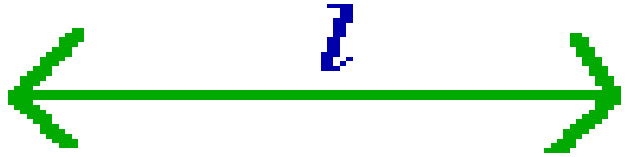
- A **point** has no dimensions - no length, no width, no height.
- That dot is obviously *way* too big to really represent a point. But we'll live with it, if we all just agree what a point really is.



Dimension of Geometric Objects

Contd…

- A **line** has one dimension - length. It has no width and no height, but infinite length.
- Again, this model of a line is really not very good, but until we learn how to draw a line with 0 width and infinite length, it'll have to do.



Dimension of Geometric Objects

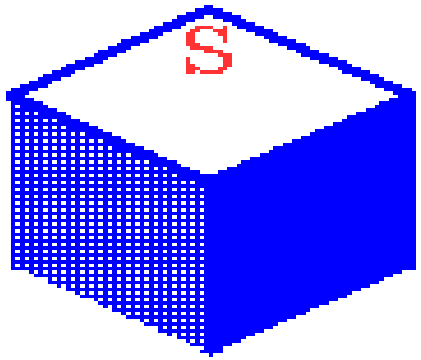
Contd…

- A **plane** has two dimensions - length and width, no depth.
- It's an absolutely flat tabletop extending out both ways to infinity.

Dimension of Geometric Objects

Contd…

- **Space,** a huge empty box, has three dimensions, length, width, and depth, extending to infinity in all three directions.
- Obviously following isn't a good representation of 3-D space. Besides its size, it's just a hexagon drawn to fool you into thinking it's a box.



Topological Dimension

- We observe that all the geometric objects residing in Euclidean space has **integer** dimension. Such a dimension is alternatively also known as *topological* dimension.
- In general, Euclidean space R^n has dimension n.
- Intuitively, the dimension of the space equals the number of real parameters necessary to describe different points in the space.

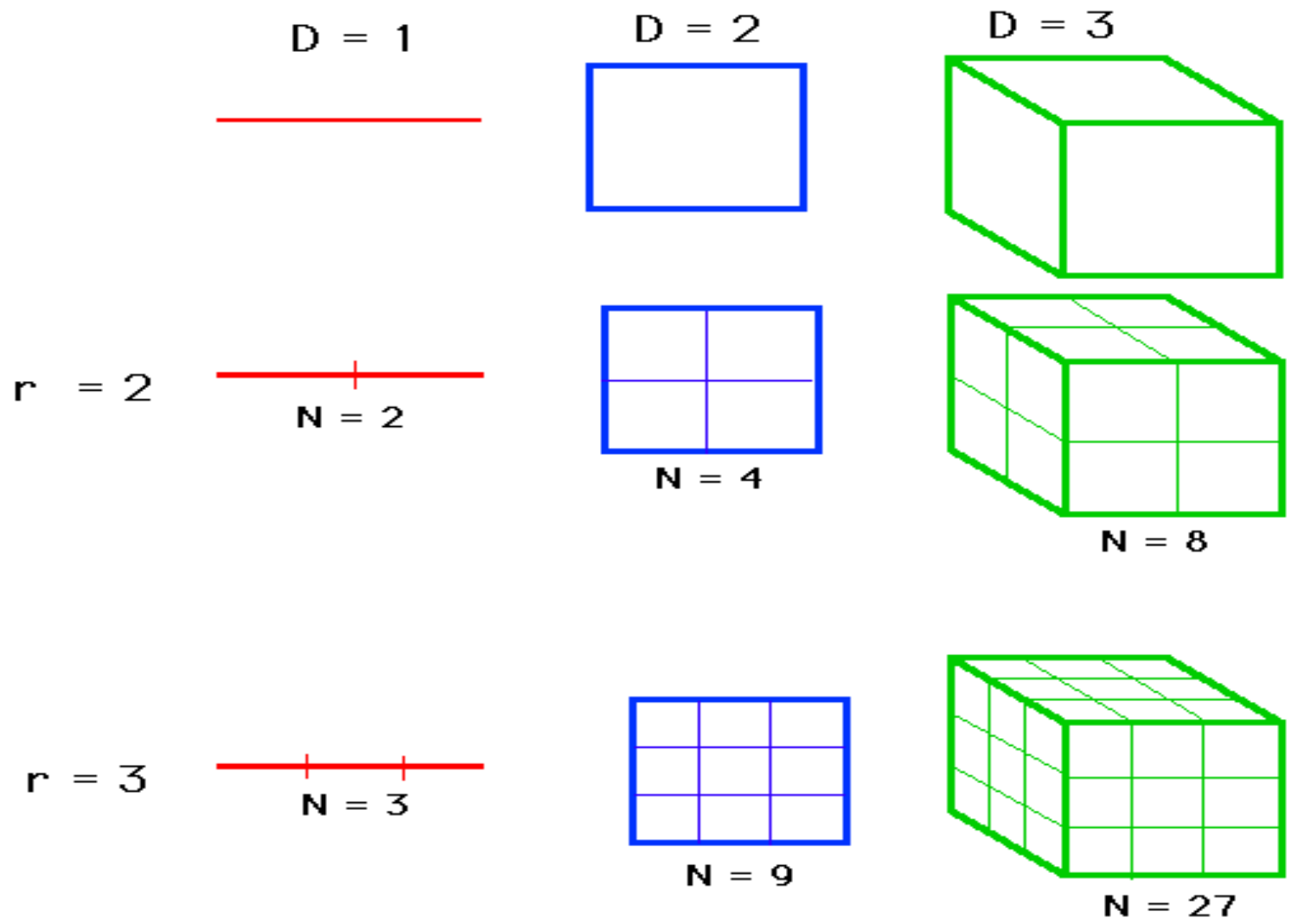
Fractional Dimension

- In fractal geometry, there is another concept of this dimension, that is, here dimension of an object is not an integer rather a fraction like 1.53, 2.71 etc.
- It is also known as **Hausdorff dimension or Fractal dimension**.
- Simply this implies that objects are possible where dimension is between 1 and 2.

Mathematical Interpretation

- If we take an object residing in Euclidean ***dimension*** **D** and reduce its linear size by 1/r in each spatial direction, its measure would increase to $N=r^D$ times the original (refer next figure).
- We consider $N=r^D$, take the log of both sides, and get log(N) = D log(r). If we solve for D, then **D = log(N)/log(r)**.

Figure: Understanding the concept of dimension



Fractional Dimension

Contd…

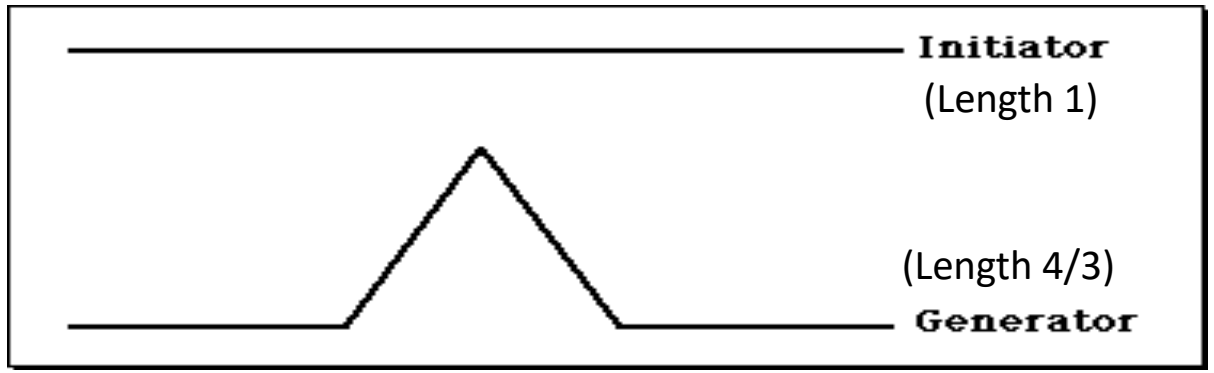
- In the equation D = log(N)/log(r), the dimension D need not be an integer, as it is in Euclidean geometry. In fractal geometry, D has fractional values.
- The fractal dimension is related to the degree of roughness or brokenness or irregularity in an object.
- It has proved useful for describing natural objects and for evaluating trajectories of dynamic systems.

Examples of Geometric Objects with Non-integer dimensions

Linear fractal- koch curve + sierpinski

Koch Curve

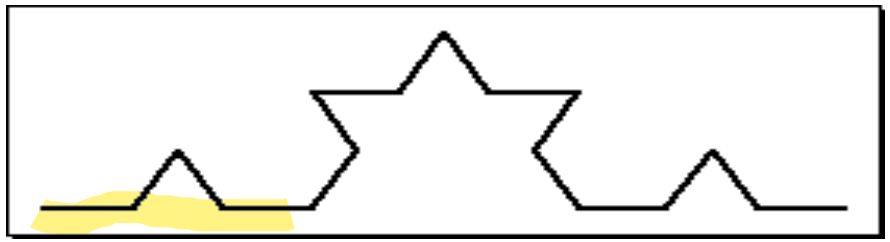
We begin with a straight line of length 1, called the **initiator**. We divide it in three equal parts and then replace the middle third one by the two adjacent sides (of length 1/3) of an equilateral triangle. This new form is called the **generator**, because it specifies a rule that is used to generate a new form.



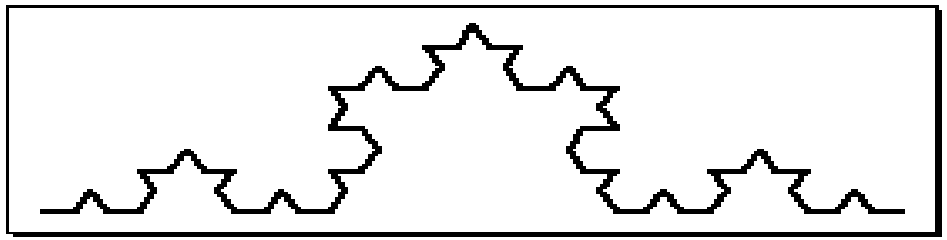
The Initiator and Generator for constructing the Koch Curve

Higher Levels of Koch Curve

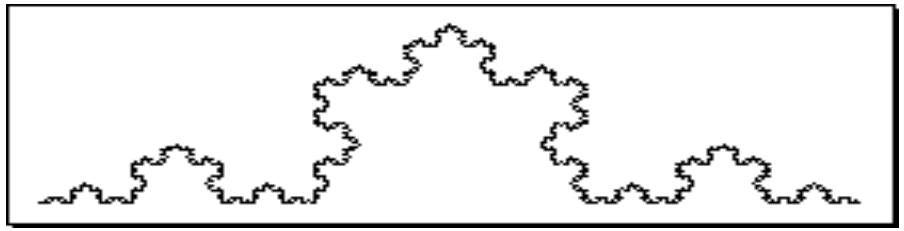
- The rule says to take each line and replace it with four lines, each one-third the length of the original.



Level 2 (Length 16/9)



Level 3 (Length 64/27)



Doing iteratively large number of times

Dimension of Koch Curve

- Here the line is reduced in scale by 3 i.e. r=3, and generates 4 equal pieces i.e. N = 4, then

$$4 = 3^D$$

N = r^D

Solving for D,

$$D = \log 4 / \log 3$$

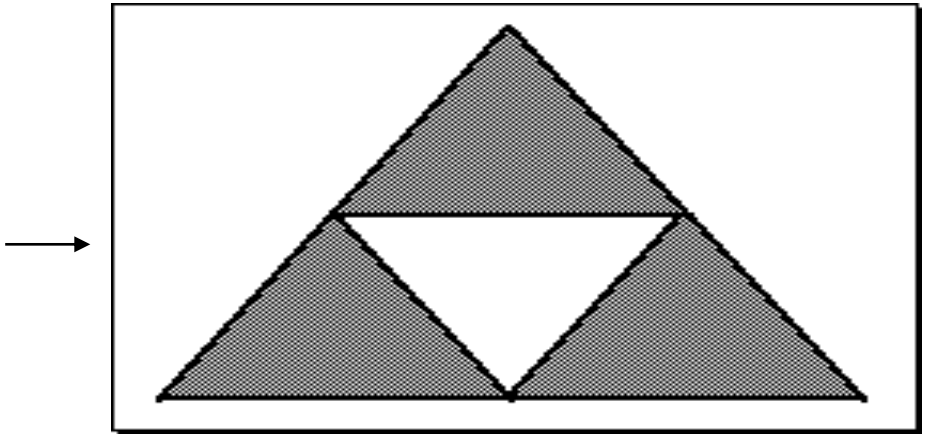
$$= 1.2618$$

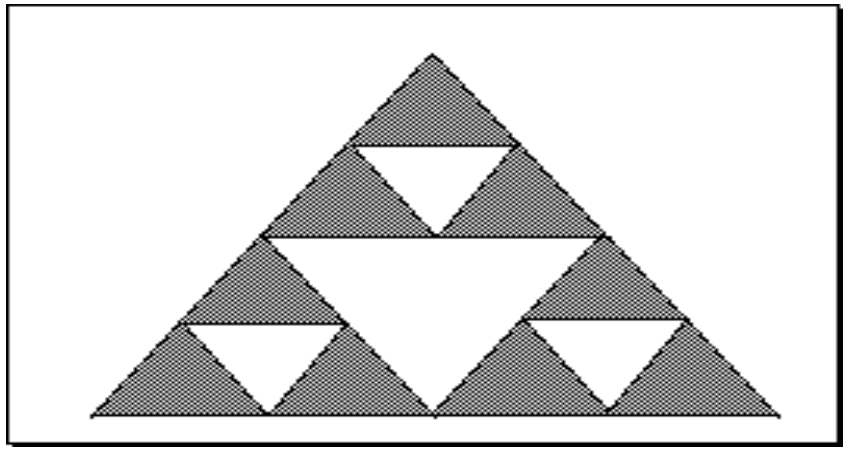
Interesting Features of Koch Curve

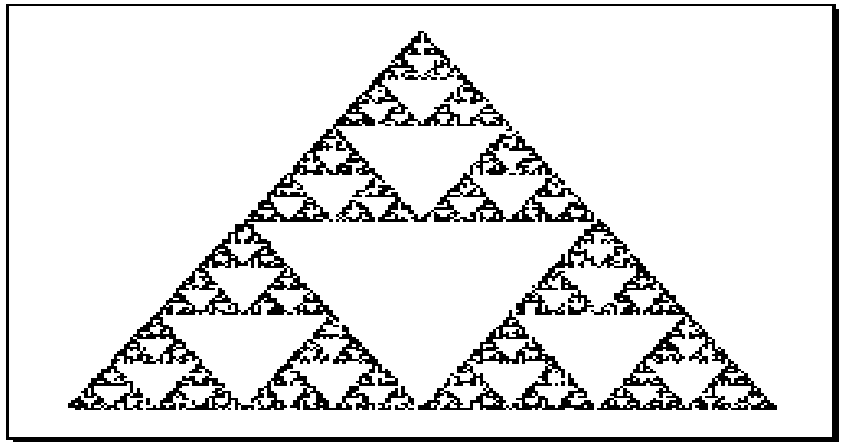
- It is a continuous loop that never intersects itself because the new triangles on each side are always small enough not to encroach upon each other.
- Other important feature of this curve is that its outline, which is of infinite length, encloses a ***finite area***.

Sierpinski Triangle

We start with an equilateral triangle, connect the mid-points of the three sides and remove the resulting inner triangle.







3. Ill defined Characteristic scale of length

- Unlike the Euclidean shapes which have characteristic sizes or scales (i.e. radius of sphere, the edge of a cube etc.), the length of fractal structures depends on the scale of measurement.

  (famous question: How long is the coast of Britain?)

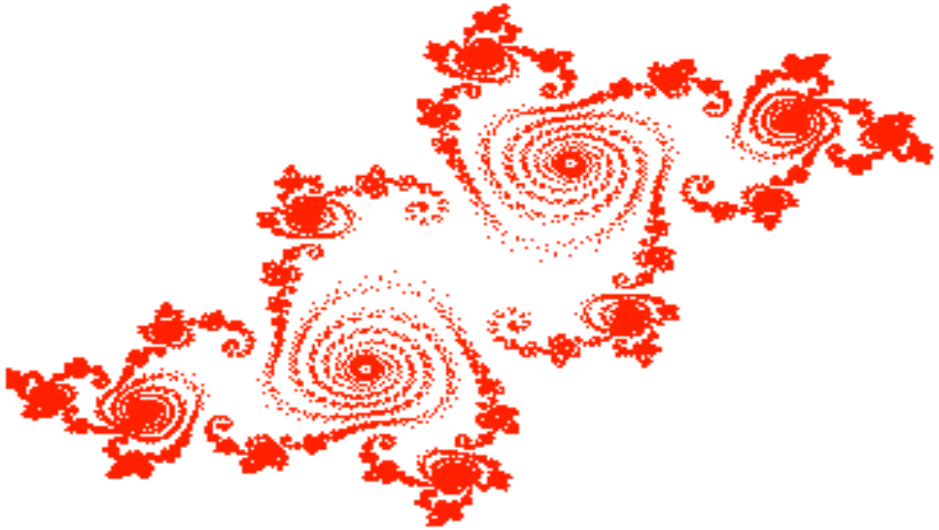
- The overall length of a fractal curve tends to infinity as we inspect closer and closer.

4. Formation by Iteration

- Fractals are often formed by what is called an *iterative* process.
- **To make a fractal:** Take a familiar geometric figure (a triangle or line segment, for example) and operate on it so that the new figure is more "complicated" in a special way.
- Then in the same way, operate on that resulting figure, and get an even more complicated figure.
- Do it again and again infinitely many times.

Non-linear Fractals

- The previous examples were from the examples of Linear fractal geometry. For the creation of such fractals, we use some rules and these rules are to be applied repeatedly a large number of times to get the final one.
- On the other hand, in case of Non-Linear fractal, a mathematical formula yields fractal.
- The examples are: Julia set and the Mandelbrot set.

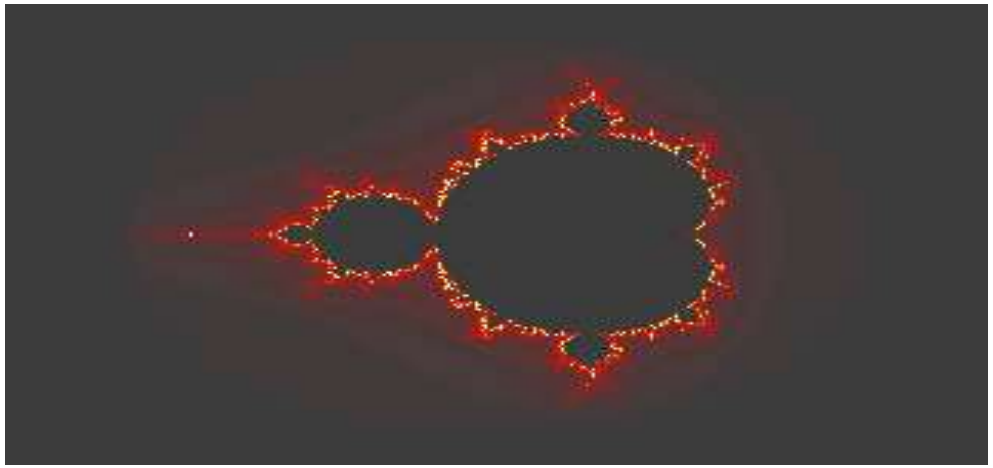


Julia Set

(Self-squaring fractal)

- Obtained by repeatedly applying the mapping $z \rightarrow z^2 + c$ to each complex number z for some non-zero value of c (*fixed c and varying starting z value*).
- In this process, some complex numbers will be attracted to infinity, some to finite numbers, and some will go toward neither. Drawing these boundary points, we get the Julia set. Thus the Julia set of f(z)=z*z is the unit circle.

Mandelbrot set



- Although the Mandelbrot set is self-similar at magnified scales, the small scale details are not *identical* to the whole.
- It's the the set of all complex numbers *z* for which sequence defined by the iteration (*varying c, z(0)=0*)

  $z(0) = z,\ z(n+1) = z(n)*z(n) + z,\ n=0,1,2,\ \ldots$ (1) remains bounded. This means that there is a number *B* such that the absolute value of all iterates *z(n)* never gets larger than *B*.

Applications of Fractals

- For describing dynamical systems and modeling chaos (e.g. modelling weather pattern).
- For image compression with very high compression ratio.
- For creating special effects in movies.
- In Astronomy for distribution of galaxies.
- For describing and predicting the location and timing of earthquakes.

Applications of Fractals

Contd…

Medical Applications: Human Anatomy

Anatomical Structure	Fractal Dimension
Bronchial Tubes	very close to 3
Arteries	2.7
Brain	2.73 – 2.79
Alveolar Membrane	2.17
Mitochondrial Membrane (outer)	2.09
Mitochondrial Membrane (inner)	2.53
Endoplasmic Reticulum	1.72