Geometrical Transformations

2D Linear transformations:

\[
\begin{bmatrix}
X' \\
Y'
\end{bmatrix} =
\begin{bmatrix}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{bmatrix}
\begin{bmatrix}
X \\
Y
\end{bmatrix} \quad X' = a_{11}X + a_{12}Y
\quad Y' = a_{21}X + a_{22}Y
\]

Premultiplication form: transformation matrix is written before the position column vector.

Linear because it satisfies the following two conditions necessary for any linear function \( L(x) \):
1) \( L(x+y) = L(x) + L(y) \) \( \Leftarrow \) superposition
2) \( L(cx) = cL(x) \) for any scalar \( c \) and position vectors \( x \) and \( y \)

Scaling: \[
\begin{bmatrix}
S_x & 0 \\
0 & S_y
\end{bmatrix}
\]

Rotation: \[
\begin{bmatrix}
cos\theta & -sin\theta \\
sin\theta & cos\theta
\end{bmatrix}
\]

Horizontal shear: \[
\begin{bmatrix}
1 & S \\
0 & 1
\end{bmatrix}
\]

Vertical shear: \[
\begin{bmatrix}
1 & 0 \\
S & 1
\end{bmatrix}
\]

42
Note that linear transformations are a sum of scaled input coords; they do not account for simple translation.

We want
\[ x' = a_{11}x + a_{12}y + a_{13} \]
\[ y' = a_{21}x + a_{22}y + a_{23} \]

\[
\begin{bmatrix}
  x' \\
  y' \\
  w'
\end{bmatrix}
= 
\begin{bmatrix}
  a_{11} & a_{12} & a_{13} \\
  a_{21} & a_{22} & a_{23} \\
  a_{31} & a_{32} & a_{33}
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix}
\]

Points are expressed in homogeneous coords.

* All transformations can now be treated as matrix multiplications.

In homogeneous coords, we add a third coord to a pt \( \Rightarrow (x, y) \) is represented by \((x, y, w)\).

\( w \) refers to the plane upon which the transformation operates.
The representation of a point is no longer unique:
\[ [8, 16, 2] = [4, 8, 1] = [16, 32, 4] \]
To recover any 2-D position vector \( p = \begin{bmatrix} x \\ y \end{bmatrix} \) from \( p_h = \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} \), divide by the homogeneous coordinate \( w' \).
\[
\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ w' \\ (w/w') \end{bmatrix} \Rightarrow \begin{align*}
x &= \frac{x'}{w'} \\
y &= \frac{y'}{w'}
\end{align*}
\]
The points with \( w = 0 \) are called points at infinity.

If we take all triples representing the same point \((tx, ty, tw)\) with \( t \neq 0\), then we get a line in 3-space.

For convenience, all input pts are made to lie on \( w = 1 \) plane to trivially facilitate translation.
**Affine Transformations**

\[
\begin{bmatrix}
X' \\
Y'
\end{bmatrix} = \begin{bmatrix}
a_{11} & a_{12} & a_{13} \\
a_{21} & a_{22} & a_{23} \\
0 & 0 & 1
\end{bmatrix}\begin{bmatrix}
X \\
Y \\
1
\end{bmatrix} \quad \leftarrow \text{General representation}
\]

Note: matrix multiplication is not commutative

\[AB \neq BA\]

However,

\[(AB)^T = B^T A^T\]

Therefore, in postmultiplication form

\[
\begin{bmatrix}
X' \\
Y' \\
1
\end{bmatrix} = \begin{bmatrix}
X \\
Y \\
1
\end{bmatrix} \begin{bmatrix}
a_{11} & a_{12} & 0 \\
a_{21} & a_{22} & 0 \\
a_{13} & a_{23} & 1
\end{bmatrix} \quad \leftarrow \text{used in Hill book}
\]

(In these notes we shall use the premult form though)

Division by \(w'\) is avoided by selecting \(w = w' = 1\).

All affine mappings have \([0 \ 0 \ 1]\) as the last row. Handles translation, rotation, scale, and shearing (skewing).
Translation: \[
\begin{bmatrix}
X' \\
Y'
\end{bmatrix} = \begin{bmatrix}
1 & 0 & T_x \\
0 & 1 & T_y
\end{bmatrix}
\begin{bmatrix}
X \\
Y
\end{bmatrix} \rightarrow \square \rightarrow \square
\]

Rotation: \[
\begin{bmatrix}
X' \\
Y'
\end{bmatrix} = \begin{bmatrix}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0
\end{bmatrix}
\begin{bmatrix}
X \\
Y
\end{bmatrix} \quad \text{CCW rotation}
\square \rightarrow \square
\]

Scale: \[
\begin{bmatrix}
X' \\
Y'
\end{bmatrix} = \begin{bmatrix}
S_x & 0 & 0 \\
0 & S_y & 0
\end{bmatrix}
\begin{bmatrix}
X \\
Y
\end{bmatrix} \quad \square \rightarrow \square
\]

Horizontal shear: \[
\begin{bmatrix}
X' \\
Y'
\end{bmatrix} = \begin{bmatrix}
1 & \lambda & 0 \\
0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
X \\
Y
\end{bmatrix} \quad x \text{ is linearly dep. on } y \text{ as well as } x
\square \rightarrow \square
\]

Vertical shear: \[
\begin{bmatrix}
X' \\
Y'
\end{bmatrix} = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & \lambda
\end{bmatrix}
\begin{bmatrix}
X \\
Y
\end{bmatrix} \quad \square \rightarrow \diamond
\]

Properties: affine transformations preserve parallel lines, but not lengths or angles.

Ex: \[\begin{array}{c}
\text{A} \\
\text{B} \\
\text{C}
\end{array}\quad \rightarrow \begin{array}{c}
\text{A} \\
\text{B} \\
\text{C}
\end{array}\quad \rightarrow \begin{array}{c}
\text{A} \\
\text{B} \\
\text{C}
\end{array}\quad \text{A} \parallel \text{C} \quad \text{B} \parallel \text{D} \quad \text{but } 45^\circ \text{ and lengths changed}
\]

Note: We transform lines by transforming endpts + redrawing line through transformed endpts.
Composition of 2D Transformations

To rotate some object about an arbitrary point $P_i$: translate such that $P_i \to \text{origin}$, rotate, translate such that pt at origin $\to P_i$ (pre-multiplication shown here)

$$
\begin{bmatrix}
X' \\
Y' \\
1
\end{bmatrix} =
\begin{bmatrix}
1 & 0 & X_i \\
0 & 1 & Y_i \\
0 & 0 & 1
\end{bmatrix} \times
\begin{bmatrix}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{bmatrix} \times
\begin{bmatrix}
1 & 0 & -X_i \\
0 & 1 & -Y_i \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
X \\
Y \\
1
\end{bmatrix}
$$

translate origin to $P_i$, rotate about origin, translate $P_i \to$ origin

$$
\begin{bmatrix}
\cos \theta & -\sin \theta & X_i(1-\cos \theta) + Y_i \sin \theta \\
\sin \theta & \cos \theta & Y_i(1-\cos \theta) - X_i \sin \theta \\
0 & 0 & 1
\end{bmatrix}
$$

Note: $(A \cdot (B \cdot C)) = ((A \cdot B) \cdot C)$ associative
$A \cdot B \neq B \cdot A$ not always commutative
**Window-to-Viewport Transformation**

![Diagram showing window and viewport transformation](image)

(uses unit meaningful to appln. program)

"World" refers to models being created or displayed to user.

**Note:** what we actually refer to as a window on the screen is known as a viewport in the computer graphics literature. CG preceded modern window systems.

Transformation matrix that maps world to viewport coordinates

\[
MWV = T(U_{min}, V_{min}) \cdot S \left( \frac{U_{max} - U_{min}}{X_{max} - X_{min}}, \frac{V_{max} - V_{min}}{Y_{max} - Y_{min}} \right) \cdot T(-X_{min}, -Y_{min})
\]

\[
MWV = \begin{bmatrix}
1 & 0 & U_{min} \\
0 & 1 & V_{min} \\
0 & 0 & 1
\end{bmatrix} \cdot \begin{bmatrix}
\frac{U_{max} - U_{min}}{X_{max} - X_{min}} & 0 & 0 \\
0 & \frac{V_{max} - V_{min}}{Y_{max} - Y_{min}} & 0 \\
0 & 0 & 1
\end{bmatrix} \cdot \begin{bmatrix}
1 & 0 & -X_{min} \\
0 & 1 & -Y_{min} \\
0 & 0 & 1
\end{bmatrix}
\]

\[
MWV = \begin{bmatrix}
\frac{U_{max} - U_{min}}{X_{max} - X_{min}} & 0 & -X_{min} \cdot \frac{U_{max} - U_{min}}{X_{max} - X_{min}} + U_{min} \\
0 & \frac{V_{max} - V_{min}}{Y_{max} - Y_{min}} & -Y_{min} \cdot \frac{V_{max} - V_{min}}{Y_{max} - Y_{min}} + V_{min} \\
0 & 0 & 1
\end{bmatrix}
\]

\[p' = MWVP\]
3D Transformations

\[
\begin{bmatrix}
X \\
Y \\
Z \\
1
\end{bmatrix}
\]
is a 3D point in homogeneous coordinates.

Homogenizing \( \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \) gives us \( \begin{bmatrix} X/w \\ Y/w \\ Z/w \\ 1 \end{bmatrix} \)

\[ \text{homogeneous coord.} \quad \text{3D coord.} \]

We use right-handed (RH) coordinate system.

Cross product

Translation: \( \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \)

Rotation:

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

R_x = \[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & cos\theta & -sin\theta & 0 \\
0 & sin\theta & cos\theta & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

R_y = \[
\begin{bmatrix}
cos\theta & 0 & sin\theta & 0 \\
0 & 1 & 0 & 0 \\
-sin\theta & 0 & cos\theta & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Scaling:

\[
\begin{bmatrix}
x' \\
y' \\
z' \\
1
\end{bmatrix} = \begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix} \begin{bmatrix}
S_x & 0 & 0 & 0 \\
0 & S_y & 0 & 0 \\
0 & 0 & S_z & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

49
All these matrices have inverses:

\[ I = T^{-1} \cdot T \]

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{bmatrix} =
\begin{bmatrix}
1 & 0 & -T_x \\
0 & 1 & -T_y \\
0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
1 & 0 & T_x \\
0 & 1 & T_y \\
0 & 0 & 1 \\
\end{bmatrix} \leftarrow 2D case (same applies to 3D)
\]

\[ I = S^{-1} \cdot S \]

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{bmatrix} =
\begin{bmatrix}
\frac{1}{S_x} & 0 & 0 \\
0 & \frac{1}{S_y} & 0 \\
0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
S_x & 0 & 0 \\
0 & S_y & 0 \\
0 & 0 & 1 \\
\end{bmatrix}
\]

\[ I = R^{-1} \cdot R \]

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{bmatrix} =
\begin{bmatrix}
\cos(-\theta) & -\sin(-\theta) & 0 \\
\sin(-\theta) & \cos(-\theta) & 0 \\
0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
\cos\theta & \sin\theta & 0 \\
-\sin\theta & \cos\theta & 0 \\
0 & 0 & 1 \\
\end{bmatrix}
\]

Note: \[ \cos(-\theta) = \cos\theta \]
\[ \sin(-\theta) = -\sin\theta \]
Composition of 3D Transformations

Initial

Final

1) Translate \( P_1 \) to origin
2) Rotate about \( y \)-axis such that \( P_1P_2 \) lies in \( yz \)-plane
3) Rotate about \( x \)-axis such that \( P_1P_2 \) lies on \( z \)-axis
4) Rotate about \( z \)-axis such that \( P_1P_3 \) lies on \( yz \)-plane

After step (1):

After step (2):

After step (3):

After step (4):

Composite matrix:

\[ R_z(\alpha) \cdot R_x(\beta) \cdot R_y(\theta-90) \cdot T(-x, y, z) \]
Rotation About An Arbitrary Axis

So far, we only know \( R_x, R_y, R_z \) but not \( R_{\text{arbitrary axis}} \)

**Strategy:**

1. Translate \( P_0 \) to origin
2. Perform up to 2 rotations to make \( P_1 \) lie on \( z \)-axis (make rot. axis coincident with \( z \)-axis)
3. Rotate about \( z \)-axis by angle \( \delta \)
4. Perform inverse of (3)
5. Perform inverse of (1)

**Step (2):**

To bring \( P_1(x, y, z) \) onto \( z \)-axis, we must have \( x, y \), become 0.

Rotate about \( x \): \( P_1 \rightarrow P'_1 \) or \( xy \) plane

\[
R_x = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \cos \alpha & -\sin \alpha & 0 \\
0 & \sin \alpha & \cos \alpha & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\( \rightarrow \) \( x \) remains the same

What is \( x \)?

\[
d = \sqrt{c_y^2 + c_z^2} \quad \text{(length of projection onto } yz \text{ plane)}
\]

\[
\cos \alpha = \frac{c_z}{d} \quad \sin \alpha = \frac{c_y}{d}
\]
After rotation about x-axis, x-axis lies on x-axis, rotate by desired 

\[ \mathbf{R}_y = \begin{bmatrix} \cos(-\theta) & 0 & \sin(-\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(-\theta) & 0 & \cos(-\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

use -\theta for clockwise rotation (not ccw)

Now P(x, y, z) lies on z-axis; rotate by desired \[ \mathbf{R}_z = \begin{bmatrix} \cos \phi & -\sin \phi & 0 & 0 \\ \sin \phi & \cos \phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

\[ \mathbf{M} = \mathbf{T}^{-1} \cdot \mathbf{R}_x^{-1} \cdot \mathbf{R}_y^{-1} \cdot \mathbf{R}_z \cdot \mathbf{R}_y \cdot \mathbf{R}_x \cdot \mathbf{T} \]

composite matrix \( (x, y, z) \) - \( x, y, z \)

\[ \begin{bmatrix} C_x \\ C_y \\ C_z \end{bmatrix} = \begin{bmatrix} X_i - X_0 \\ Y_i - Y_0 \\ Z_i - Z_0 \end{bmatrix} \]

\[ \sqrt{(x_i - x_0)^2 + (y_i - y_0)^2 + (z_i - z_0)^2} \]
1) Compute $C_x, C_y, C_z:\ \begin{bmatrix} C_x \\ C_y \\ C_z \end{bmatrix} = \begin{bmatrix} 3-2 \\ 2-1 \\ 2-1 \end{bmatrix} \cdot \frac{1}{\sqrt{(3-2)^2 + (2-1)^2 + (2-1)^2}} = \begin{bmatrix} 1/\sqrt{3} \\ 1/\sqrt{3} \\ 1/\sqrt{3} \end{bmatrix}$

2) Translate $F$ to origin: \( T = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \)

3) Rotate about $X$:
\[ d = \sqrt{C_y^2 + C_z^2} = \sqrt{\frac{1}{3} + \frac{1}{3}} = \frac{\sqrt{2}}{3} \]
\[ \cos\alpha = \frac{C_z}{d} = \frac{1}{\sqrt{2}} \quad \sin\alpha = \frac{C_y}{d} = \frac{1}{\sqrt{2}} \]
\[ R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ 0 & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

4) Rotate about $Y$:
\[ \cos\beta = d \quad \sin\beta = C_x \]
\[ \text{We must use } -\beta \Rightarrow \cos(-\beta) = d \quad \sin(-\beta) = -C_x \]
\[ R_y = \begin{bmatrix} d & 0 & -C_x & 0 \\ 0 & 1 & 0 & 0 \\ C_x & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]
\[ \begin{bmatrix} 1/\sqrt{3} & 0 & -1/\sqrt{3} & 0 \\ 0 & 1 & 0 & 0 \\ 1/\sqrt{3} & 0 & \sqrt{2/3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

51.3
5) Rotate about $z$:

$$R_z = \begin{bmatrix}
\cos \delta & -\sin \delta & 0 & 0 \\
\sin \delta & \cos \delta & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}$$

$$M = T^{-1} \cdot R_x^{-1} \cdot R_y^{-1} \cdot R_z \cdot R_y \cdot R_x \cdot T$$

Transform all object vertices:

$$[X'] = [M][X]$$
Coordinate System Transformation

Thus far, we have transformed object points in the same coordinate system.

$\begin{align*}
\mathbf{P}_1 & \quad \mathbf{P}_2' \\
\mathbf{P}_1 & \quad \mathbf{P}_2
\end{align*}$

$\mathbf{P}_1\mathbf{P}_2$ rotates in the same coord system

$\mathbf{P}_1\mathbf{P}_2$ remains fixed in the different coord system

Change of coordinate system is useful when multiple objects, each defined in its local coordinate system, are combined and we wish to express these objects' coordinates in a single global coordinate system (known as the world coordinate system).
Def: $M_{i \leftarrow j}$ is a transformation matrix that converts the representation of a point in coordinate system $j$ into its representation in coordinate system $i$.

$p(i)$ is the representation of a point in coordinate system $i$,

$p(j) = M_{i \leftarrow j} p(i)$

$p(j) = M_{j \leftarrow k} p(k)$

$p(i) = M_{i \leftarrow j} \cdot M_{j \leftarrow k} \cdot p(k) = M_{i \leftarrow k} \cdot p(k)$

$\therefore M_{i \leftarrow k} = M_{i \leftarrow j} \cdot M_{j \leftarrow k}$

Ex:

$M_{i \leftarrow 2} = T(4,2)$

$(0,0)^{(2)} \rightarrow (4,2)^{(1)}$

$M_{2 \leftarrow 3} = T(2,3) \cdot S((5,5))$

Note: first scale by $(5,5)$ and then translate by $(2,3)$. Scaling about origin leaves origin alone so that it can later be translated $(2,3)^{(2)}$

$M_{i \leftarrow 3} = M_{i \leftarrow 2} \cdot M_{2 \leftarrow 3} = T(4,2) \cdot T(2,3) \cdot S((5,5))$

53
Projectors (3D to 2D)

2D points

clip rect

2D

window

clipped points

viewport
transform into
viewport device
coordinates

3D

3D points

clip against
view volume

3D

clipped 3D
points

projection
plane
(window)

viewport
2D device
coods

Projectors transform points in a coord system of dimension n into pts. in a coord system of dimension < n (eg 3D to 2D)

We will deal with planar geometric projections (PnP) onto a plane uses straight projectors

PnP can be divided into two basic classes: perspective and parallel.
**Perspective projection**

*Distance between COP and projection plane is finite*

**Parallel projection**

*Distance between COP and projection plane is infinite*

---

When defining a perspective projection, we must specify its center of proj.

When defining a parallel projection, we must specify its direction of proj.

Center of projection (COP): \( (x, y, z, 1) \) \( \leftarrow \) 3D pt rep. with homogeneous coords

Direction of projection is a vector (difference between pts):

\[
d = (x, y, z, 1) - (x', y', z', 1) = (a, b, c, 0)
\]

\( \rightarrow \) pt at \( \infty \) in direction \((a, b, c)\)

\( \circ \) direction and pts at \( \infty \) correspond naturally

A perspective projection whose COP is a pt at \( \infty \) \( \Rightarrow \) parallel projection
**Hierarchy of projections**

- **Parallel**
  - Orthographic
  - Axonometric
  - Oblique
    - Trimetric
    - Cavalier
    - Cabinet
  - Dimetric
  - Isometric
- **Perspective**
  - Single pt.
  - 2-pt
  - 3-pt

**P&G:** intersection of lines (projectors) with proj plane
Projectors are lines from an arbitrary pt. (cep) through each point in an object.

**Perspective:** cep is located at a finite pt. in 3-space
**Parallel:** cep is located at infinity \( \Rightarrow \) all projectors are parallel
**Orthographic:** projection onto one of the 3 coordinate planes \((x=0, y=0, \text{ or } z=0)\)

\[
\begin{align*}
\hat{p}_z &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\
\hat{p}_x &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\
\hat{p}_y &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}
\end{align*}
\]

- Projection onto \( z=0 \) plane \( \hat{z} \) drops out when \( z=0 \)
Multiple orthographic projections are needed to reconstruct shape of object:

Orthographic projections show "true" size and shape of a single object face if it is parallel to the coordinate plane (proj. plane). Parallel lines and angles are preserved.

Axonometric: rotate/translate object before ortho. proj. such that at least 3 adjacent faces are shown. This is equivalent to using a projection plane that is not normal to a principal axis. Again, true shape of face is not given unless it is parallel to the projection plane. However, the relative lengths of parallel lines remain constant (parallel lines are equally foreshortened). Angles are not preserved.
Trimetric: an axonometric projection whereby the foreshortening ratios for each projected principal axis \((x, y, z)\) are all different.

Dimetric: a trimetric projection with 2 of the 3 foreshortening factors equal.

Isometric: a dimetric projection with all 3 principal axes equally foreshortened.

The principal axes project so as to make equal angles with one another.

8 possible directions satisfy this condition:

\[ \pm 45^\circ, \pm 35.26^\circ \]

Oblique: parallel projectors are not perpendicular to the plane of projection.

Only faces parallel to projection plane are shown at their true \(\pm 90^\circ\) size and shape (angles + lengths are preserved for these faces only, as in ortho. proj.)

Otherwise, size and shape are distorted.

Cavalier: angle between oblique projectors and proj. plane is \(45^\circ\)

The foreshortening factors for all 3 dirs are equal.

The resulting figure appears too thick.

Cabinet: foreshortening factor for edges \(\perp\) to plane of proj. = \(\frac{1}{2}\)

Used to correct deficiency of cavalier proj.
**Perspective:**

A perspective projection is obtained by concatenating an orthographic proj. w/ persp. transformation:

\[
\begin{bmatrix} X' \\ Y' \\ Z' \\ W' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix}
\]

\[
X' = \frac{X}{r_z W}, \quad Y' = \frac{Y}{r_z W}, \quad Z' = \frac{Z}{r_z W}
\]

**Perspective foreshortening:**

The size of the projection varies inversely with the distance of the object from COP.

Parallel lines converge.

**Similar triangles:**

\[
\frac{X^*}{Z_c} = \frac{X}{Z_c - Z} \Rightarrow X^* = \frac{X}{1 - \frac{Z}{Z_c}}
\]

\[
\frac{Y^*}{\sqrt{X^*^2 + Z_c^2}} = \frac{Y}{\sqrt{X^2 + (Z_c - Z)^2}} \Rightarrow Y^* = \frac{Y}{1 - \frac{Z}{Z_c}}
\]

Let \( r = \frac{-1}{Z_c} \) in matrix above

\[
X' = \frac{1 - \frac{Z}{Z_c}}{W'} \quad Y' = \frac{Y}{1 - \frac{Z}{Z_c}}
\]

Interpret COP to be \(-\frac{1}{r} \)

Notice, as \( r \to 0 \), COP \to \infty, parallel projection results.

Also, pts on plane of proj (\( Z = 0 \)) are not distorted (r.t.s)
Observations
1) $A'B'$ intersects $z=0$ plane at same pt as $AB$
2) $A'B'$ intersects $z$-axis at $z = \frac{1}{r}$
3) Perspective transformation has transformed the intersection pt. at $\infty$ of $AB$ and $z$-axis to finite pt. $z = \frac{1}{r}$  
This pt. is called the vanishing pt. (parallel lines) 
4) Vanishing pt. lies equal distance on the opposite side of proj. plane from COP

Confirmation:
\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
0 \\
0 \\
1 \\
0 \\
\end{bmatrix} =
\begin{bmatrix}
0 \\
0 \\
0 \\
1 \\
\end{bmatrix} \Rightarrow
\begin{align*}
x' &= 0 \\
y' &= 0 \\
z' &= \frac{1}{r} \\
\end{align*}
\]

1 v.p. ← 1-pt perspective: last row has only 2 0s
2 v.p. ← 2-pt perspective: last row has only 1 0
3 v.p. ← 3-pt perspective: last row has no zeros
Ex:

\[ \begin{bmatrix} 1 & .6 \\ .667 & .4 \\ 1.333 & 1.6 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 3 \\ 2 & 2 \\ 4 & 8 \\ 3 & 5 \end{bmatrix} = \begin{bmatrix} 1000 \\ 0100 \\ 0010 \\ 00.51 \end{bmatrix} \]

\[ r = -\frac{1}{z_c} = .5 \]

3D pts after perp. transformation

\[ 3D \ pts \ before \ perp. \ transf. \]

Parametric equation of \( A'B' \):

\[ P(t) = A' + (B' - A')t \quad 0 \leq t \leq 1 \]

\[ = \begin{bmatrix} .667 \\ 1.333 \\ 1 \end{bmatrix} + \begin{bmatrix} -.4 \\ .267 \\ 0 \end{bmatrix} t \]

Intersection of \( A'B' \) with the \( x=0, y=0, \) and \( z=0 \) planes yields:

\[ x(t) = 0 = 1 - .4t \Rightarrow t = 2.5 \]

\[ y(t) = 0 = .667 - .267t \Rightarrow t = 2.5 \]

\[ z(t) = 0 = 1.333 + .267t \Rightarrow t = -5 \]

Vanishing pt: \( z(2.5) = 1.333 + (.267)(2.5) = 2.0 \Rightarrow \) intersection of \( A'B' \) with \( z-axis \) at \( z = \pm f \) (opposite COP)

Intersection of \( A'B' \) with \( z=0 \) plane

\[ x(-5) = 1 - (.4)(-5) = 3 \]

\[ y(-5) = .667 - (.267)(5) = 2 \Rightarrow \) Same as intersection of \( AB \) with \( z=0 \) plane

Proj. of \( A'B' \) onto \( z=0 \) plane:

\[ \begin{bmatrix} 1 & .6 \\ .667 & .4 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1000 \\ 0100 \\ 00.01 \end{bmatrix} \begin{bmatrix} A' \\ B' \end{bmatrix} \]

\[ \begin{bmatrix} 1 & 1 \\ 1.667 & .4 \\ 1.333 & 1.6 \end{bmatrix} \]

61
1-pt perspective

lines parallel to x and y axes do not converge; only lines parallel to z-axis do so.

2-pt perspective

lines parallel to x- and y-axes converge

3-pt perspective is used less since it adds little realism beyond 2-pt perspective.

Properties of perspective projections:

angles are preserved only on those faces of the object that are parallel to the projection plane. Parallel lines do not in general project to parallel lines.

smaller but angles preserved

projection plane

face parallel to proj plane

more foreshortening

(unequal because some object parts are closer to proj plane than others)

Angles are not preserved
Specifying an Arbitrary 3D View

Viewing coord sys (LH) COP = projection reference pt
(COP)

Window in viewplane

Synthetic camera: viewplane in which window is defined, viewing coord system (LH sys is more natural), and eye

Depth of pt from eye in positive N direction

Pin-hole (real) camera:

Note: Pinhole is COP and it lies between object and projection plane (film).

Image is upside down

Synthetic camera:

Notes: projection plane lies between object and COP.

Image is upright
Mathematical description of synthetic camera:
Specify viewing reference coord (VRC) system, position + orientation of viewplane, window borders, and position of eye (PRP).

1) View plane (= projection plane)

   a) Define view reference point (VRP) to be origin of coord sys. It is at (F_x, F_y, F_z) in WC (world coords).
   b) Define viewplane normal (VPN); given as a unit vector \( \mathbf{n} \) with WC components \( (n_x, n_y, n_z) \). VPN establishes \( n \)-axis

2) Viewing coordinate system

   a) Define VUP vector to designate view "up" direction. This points in the general direction but is not guaranteed to be \( \perp \) to VPN (N)
   b) Project VUP onto VP along N. This continues to point up but also \( \perp \) to \( \mathbf{n} \). Call this projected line (from VRP to VUP) the \( v \)-axis
      \[ V_U P' = V_U P - \left( V_U P \cdot n \right) n \]
      \[ v = \frac{V_U P'}{\|V_U P'\|} \]
      Fails if \( V_U P' \parallel N \) (up cannot be along \( N \))
   c) Make \( u \)-axis \( \perp \) \( \mathbf{n} \) and \( v \); \( u = n \times v \) \( \perp \) forms LH sys.
3) Window borders: \((U_{\text{min}}, V_{\text{min}})\) to \((U_{\text{max}}, V_{\text{max}})\)
in VRC system. VRP is not in general, the
center of the window. The window borders
specify what part of the view plane is to
be displayed.

4) Eye position \((e_u, e_v, e_n)\) is given in VRC sys.
   It can be anywhere but usually on
   \(-n\)-axis. Otherwise, the view would be oblique

**Flexibility of Camera Model**

**Fly by:** move VRP (position of viewer). Akin to moving
head without changing direction of looking

**Look around:** move VPN. Akin to swiveling/pivoting head

**Head tilting:** change VUP. Akin to keeping eye fixed
on object while tilting head. Appears rotated.

**Hints:** VRP is usually near center of object to be
viewed. Normalize VPN: \[ n = \frac{\text{norm}}{\|\text{norm}\|}\]
Have VPN point to WC origin or point of interest.
Transform Object Points into VRC

Represent all points in VRC for consistency with eye position and window boundaries already given in VRC.

Find: viewing coords \((a, b, c)\) of WC pt. \(p(x, y, z)\).
\[
\begin{bmatrix}
x' \\
y' \\
z'
\end{bmatrix} = \begin{bmatrix} a \\ b \\ c
\end{bmatrix} + \begin{bmatrix} r_x \\ r_y \\ r_z
\end{bmatrix}
\]
where \(M_{vw} = \begin{pmatrix} u_x & v_x & n_x & 0 \\ u_y & v_y & n_y & 0 \\ u_z & v_z & n_z & 0 \\ 0 & 0 & 0 & 1
\end{pmatrix}\)
\[
\begin{bmatrix} a \\ b \\ c
\end{bmatrix} = M_{vw}^{-1}(p - r) = M_{vw}^T p - M_{vw}^T r
\]
\(\text{translation } r'\)
\(r' = (-r.u, -r.v, -r.n)\)
\(M_{wv} = \begin{pmatrix} u_x & u_y & u_z & r_x' \\ v_x & v_y & v_z & r_y' \\ n_x & n_y & n_z & r_z' \\ 0 & 0 & 0 & 1
\end{pmatrix}\)

WC to VRC

Ex: \(u = (-1, 0, 0)\) \(v = (0, \frac{4}{5}, \frac{3}{5})\) \(n = (0, \frac{-3}{5}, \frac{4}{5})\)
\(-r.u = 2\) \(-r.v = \frac{-9}{5}\) \(-r.n = \frac{-13}{5}\)
\(r = (2, 3, -1)\)
\(M_{wv} = \begin{pmatrix} -1 & 0 & 0 & \frac{2}{5} \\ 0 & \frac{4}{5} & \frac{3}{5} & -\frac{9}{5} \\ 0 & -\frac{3}{5} & \frac{4}{5} & \frac{13}{5} \\ 0 & 0 & 0 & 1
\end{pmatrix}\)

\[
\begin{bmatrix} 4 \\ 7 \\ 2 \\ 1
\end{bmatrix} \rightarrow \begin{bmatrix} -2 \\ 5 \\ 0 \\ 1
\end{bmatrix} \rightarrow \text{on viewplane}
\]

66
**Purpose:** Convert from WC to VRC to simplify remaining operations of clipping, projection, and rendering pictures of a scene.

For all vertices \( \{ \}

\[
UVN_{\text{vert} [i]} = \text{worldToView}(\text{vert} [i], u, v, n, r);
\]

\( \} \)

\[
\begin{array}{c}
\text{VRC sys.} \\
\text{WC sys.}
\end{array}
\]

\[
\begin{array}{c}
\text{COP} \\
\text{PRP)}
\end{array}
\]

\( \text{projection reference pt.} \)

\[
\begin{array}{c}
\text{LH sys.} \\
\text{Window in VP}
\end{array}
\]

\[
\begin{array}{c}
\text{RH system}
\end{array}
\]

\[
\begin{array}{c}
\text{convert everything to VRC}
\end{array}
\]

\[
\begin{array}{c}
\text{NDC viewport}
\end{array}
\]

\[
\begin{array}{c}
\text{Device Coords}
\end{array}
\]

\[
\begin{array}{c}
\text{map clipped edge to viewport in NDC}
\end{array}
\]

\[
\begin{array}{c}
\text{Window in VP}
\end{array}
\]

\[
\begin{array}{c}
\text{map clipped edge to viewport in NDC}
\end{array}
\]

\[
\begin{array}{c}
\text{Window in VP}
\end{array}
\]
Ray from eye at \( e = (e_u, e_v, e_n) \) to \( p \) is:
\[
\vec{r}(t) = \vec{e} (1-t) + \vec{p} t
\]

It pierces VP at \( t' \) when \( n \)-component = 0:
\[
e_n(1-t') + p_n t' = 0 \Rightarrow t' = \frac{e_n}{e_n - p_n}
\]

Note: this fails if \( e_n = p_n \) (if \( \vec{p} \perp \vec{N} \)) only if we look parallel to VP

Usual case: \( e_n < 0, \ p_n > e_n \Rightarrow 0 < t' < 1 \)

Plugging \( t' \) into \( \vec{r}(t) \) yields the \( u^*, v^* \) components:

\[
\begin{aligned}
\text{General} \quad u^* &= e_u(1-t') + p_u t' = e_u\left(\frac{-p_n}{e_n - p_n}\right) + p_u\left(\frac{e_n}{e_n - p_n}\right) \\
&= \frac{e_n p_u - e_u p_n}{e_n - p_n} \\

\text{Soln.} \quad \begin{cases} \\
(\text{CP may be anywhere}) \\
\end{cases}
\end{aligned}
\]

\[
\begin{aligned}
\text{General} \quad v^* &= e_v(1-t') + p_v t' = e_v\left(\frac{-p_n}{e_n - p_n}\right) + p_v\left(\frac{e_n}{e_n - p_n}\right) \\
&= \frac{e_n p_v - e_v p_n}{e_n - p_n} \\
\end{aligned}
\]
Special case: eye on N-axis (usual case)

\[ e_u = e_v = 0 \]

\[ U^* = \frac{e_u p_u - e_v p_n}{e_n - p_n} = \frac{e_u p_u}{e_n - p_n} = \frac{p_u}{1 - \frac{p_n}{e_n}} \]

\[ V^* = \frac{e_u p_v - e_v p_n}{e_n - p_n} = \frac{e_u p_v}{e_n - p_n} = \frac{p_v}{1 - \frac{p_n}{e_n}} \]

Fails if \( e_n = p_n \) (\( r \perp N \)) or \( e_n = 0 \) (eye on VP)

\[ e_n < 0 \]

SAME RESULT \( z_c < 0 \)

Let \( r = \frac{1}{2e} \)

\[
\begin{bmatrix}
X' \\
Y' \\
Z'
\end{bmatrix} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}\begin{bmatrix}
X \\
Y \\
Z \\
1
\end{bmatrix}
\]

\[ X' = \frac{X}{1+rz_c} = \frac{X}{1-\frac{e_n}{z_c}} \]

\[ Y' = \frac{Y}{1+rz_c} = \frac{Y}{1-\frac{e_n}{z_c}} \]

Foreshortening factor is \( \frac{1}{1 - \frac{p_n}{e_n}} \) \( \Rightarrow \) No foreshortening if \( p_n = 0 \) (on VP)

Since \( e_n < 0 \), \( p_n > 0 \), \( (-p_n) > 0 \)

As \( p_n \uparrow \) foresh. factor \( \downarrow \)

inverse relationship between distance + size

We can rewrite above matrix as:

\[
M_p = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & \frac{1}{e_n} & 1
\end{bmatrix}
\]
Back to general case: eye located off N-axis

Numerators of \( u^* \), \( v^* \) are linear combinations of \( p \):

\[
p^* = M_p M_s \begin{bmatrix} p_u \\ p_v \\ p_n \end{bmatrix}
\]

where \( M_s = \begin{bmatrix} 1 & 0 & -e_u/e_n \\ 0 & 1 & -e_v/e_n \\ 0 & 0 & 1 \end{bmatrix} \)

shear transf.

Verify:

\[
\begin{bmatrix} e_n u - e_u n \\ e_n v - e_v n \\ e_n p_n \\ e_n \end{bmatrix} = \begin{bmatrix} p_u - e_u n \\ p_v - e_v n \\ p_n \end{bmatrix} = \begin{bmatrix} 1 & 0 & -e_u/e_n \\ 0 & 1 & -e_v/e_n \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_u \\ p_v \\ p_n \end{bmatrix}
\]

\[
U^* = \frac{e_n u - e_u n}{e_n - p_n}
\]

\[
V^* = \frac{e_n v - e_v n}{e_n - p_n}
\]

\[
\begin{bmatrix} e_n u - e_u n \\ e_n v - e_v n \\ e_n p_n \\ e_n \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & e_n & 1 \end{bmatrix} \begin{bmatrix} e_n u \\ e_n v \\ e_n p_n \\ e_n \end{bmatrix}
\]

\[
\therefore \text{effect of moving eye off N-axis is to shear the image. } +ve \text{ shifts projected pt upward}
\]
View Volume is region in space that is to be projected and drawn. It is part of the synthetic camera \( \Rightarrow \) defined in VRC. F and B planes are parallel to view plane. They chop pyramid into frustum. Lines are clipped to view volume, \( e_n < F_n < B_n \). For parallel projection, view volume becomes parallelepiped:

\[
\text{COP at } \infty
\]
Summary Thus Far:

\[ WC \rightarrow VC \rightarrow \text{shear} \rightarrow \text{perspective} \]

\[
\begin{bmatrix}
P_u \\
P_v \\
P_n \\
P_w \\
\end{bmatrix}
= M_p \cdot M_s \cdot M_{wv}
\begin{bmatrix}
P_x \\
P_y \\
P_z \\
1 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
U^* \\
V^* \\
N^* \\
\end{bmatrix}
= \begin{bmatrix}
P_u/P_w \\
P_v/P_w \\
P_n/P_w \\
\end{bmatrix}
\text{ and } P_w = 1 - \frac{P_n}{E_n} = \frac{E_n - P_n}{E_n}
\]

The effect of \( M_p M_s \) is called prewarping.

This moves each vertex of the object in 3 dimensions such that its \((u, v)\) coordinates are in their final position.

Before prewarping

After prewarping
The front of the block is on the viewplane so its projection remains the same (no distortion).

The rear appears smaller.

Once objects have been prewarped, we perform orthographic projection. This sets the n-component to 0.

Perspective projection consists of:
1) perspective transformation
2) recover Cartesian coordinates (divide by W)
3) orthographic projection

Notice that prewarping transforms view volume from frustum to parallelepiped, a much simpler shape!

Note that:
1) prewarping preserves planes \((F, VP, B)\)
2) planes parallel to \(VP\) are just shifted to \(\frac{F}{1 - \frac{F}{F_n}}\) and \(\frac{B}{1 - \frac{B}{B_n}}\)
3) the 4 side walls of the view volume are warped into 4 parallel planes:
   \(U = U_{\text{min}}, U = U_{\text{max}}, V = V_{\text{min}}, V = V_{\text{max}}\)
Normalized (Canonical) View Volume.

A view volume is used to map coordinates to viewport, given in NDC. Normalizing it helps simplify clipping.

\[
M_N = \begin{pmatrix}
S_u & 0 & 0 & R_u \\
0 & S_v & 0 & R_v \\
0 & 0 & S_h & R_h \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

where

\[
S_u = \frac{V_r - V_l}{U_{\text{max}} - U_{\text{min}}} \\
S_v = \frac{V_t - V_b}{V_{\text{max}} - V_{\text{min}}} \\
S_h = \frac{(e_n - B)(e_n - F)}{e_n^2(B - F)} \\
R_u = \frac{V_l U_{\text{max}} - V_r U_{\text{min}}}{U_{\text{max}} - U_{\text{min}}} \\
R_v = \frac{V_b V_{\text{max}} - V_t V_{\text{min}}}{V_{\text{max}} - V_{\text{min}}} \\
R_h = \frac{F(e_n - B)}{e_n(F - B)}
\]

Recall:

\[
\begin{aligned}
&V_t \quad (u_t, v_t) \\
&V_l \quad (u_l, v_l) \\
&V_r \quad (u_r, v_r) \\
&V_b \quad (u_b, v_b)
\end{aligned}
\]

\[
\begin{aligned}
&x = S_u u + t_u \\
y = S_v v + t_v
\end{aligned}
\]

where

\[
S_u = \frac{x_u - x_l}{u_r - u_l} \\
S_v = \frac{y_r - y_l}{v_r - v_l}
\]

\[
\begin{bmatrix}
x_1 \\
x_2
\end{bmatrix} = \begin{bmatrix}
u_1 & 1 \\
u_2 & 1
\end{bmatrix} \begin{bmatrix}
S_u \\
t_u
\end{bmatrix} \quad \Rightarrow \quad
\begin{aligned}
t_x &= \frac{x_1 u_2 - x_2 u_1}{u_2 - u_1} \\
t_y &= \frac{y_1 v_2 - y_2 v_1}{v_2 - v_1}
\end{aligned}
\]

Note: If \( M = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \) then \( M^{-1} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} \)
Composite matrix $M_c$

$$M_c = M_N M_P M_s M_w v$$

$$P(u,v) = M_c P(x,y,z)$$

transforms pts from WC to NDC

Clipping

Clipping to a normalized view volume is now easy.
Do clipping in homogeneous coords to avoid division on clipped points.
Use 3D Liang-Barsky algorithm.