Transformations
Translation, Rotation, Scale
Composite transformations

Homogeneous Coordinates

• Homogeneous coordinates are key to all computer graphics systems

• Hardware pipeline all work with 4 dimensional representations

• All standard transformations (rotation, translation, scaling) can be implemented by matrix multiplications with 4 x 4 matrices
A Single Representation

With these rules, we can keep track of the difference:

\[ \mathbf{v} = \alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \alpha_3 \mathbf{v}_3 = [\alpha_1 \, \alpha_2 \, \alpha_3 \, 0] \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \\ \mathbf{P}_0 \end{bmatrix}^T \]

\[ \mathbf{P} = \mathbf{P}_0 + \beta_1 \mathbf{v}_1 + \beta_2 \mathbf{v}_2 + \beta_3 \mathbf{v}_3 = [\beta_1 \, \beta_2 \, \beta_3 \, 1] \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \\ \mathbf{P}_0 \end{bmatrix}^T \]

Thus we obtain a four-dimensional representation for both:

\[ \mathbf{v} = [\alpha_1 \, \alpha_2 \, \alpha_3 \, 0]^T \]

\[ \mathbf{p} = [\beta_1 \, \beta_2 \, \beta_3 \, 1]^T \]

Affine Transformations

- Line preserving
- Characteristic of many physically important transformations
  - Rigid body transformations: translation, rotation
  - Non-rigid: Scaling, shear
- Importance in graphics is that we need only transform vertices (points) of line segments and polygons, then system draws between the transformed points
Translation

• Move (translate, displace) a point to a new location

• Displacement determined by a vector \( d \)
  - Three degrees of freedom
  - \( P' = P + d \)

Moving objects

When we move a point on an object to a new location, to preserve the object, we must move all other points on the object in the same way.

Object translation: every point displaced by the same vector, \( d \)
Translation Using Representations

Using the homogeneous coordinate representation in some frame
\[ p = [x \ y \ z \ 1]^T \]
\[ p' = [x' \ y' \ z' \ 1]^T \]
\[ d = [dx \ dy \ dz \ 0]^T \]
Hence \( p' = p + d \) or
\[ x' = x + dx \]
\[ y' = y + dy \]
\[ z' = z + dz \]
note that this expression is in four dimensions and expresses that point = vector + point

Translation Matrix

We can also express translation using a 4 x 4 matrix \( T \) in homogeneous coordinates
\[ p' =Tp \] where
\[ T = T(d_x, \ d_y, \ d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \]
This form is better for implementation because all affine transformations can be expressed this way and multiple transformations can be concatenated together
Rotation (2D)

• Consider rotation about the origin by \( \theta \) degrees
  - radius stays the same, angle increases by \( \theta \)

What is this rotation about the z axis?

\[
x' = x \cos \theta - y \sin \theta \\
y' = x \sin \theta + y \cos \theta
\]
Rotation about the z axis

- Rotation about z axis in three dimensions leaves all points with the same z
  - Equivalent to rotation in two dimensions in planes of constant z
    \[ x' = x \cos \theta - y \sin \theta \]
    \[ y' = x \sin \theta + y \cos \theta \]
    \[ z' = z \]
  - or in matrix notation (with p as a column)
    \[ p' = R_z(\theta)p \]

Rotation Matrix

Homogeneous Coordinates:

\[ R = R_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]
Rotation about x and y axes

- Same argument as for rotation about z axis
  - For rotation about x axis, x is unchanged
  - For rotation about y axis, y is unchanged

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

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

Scaling

Expand or contract along each axis (fixed point of origin)

\[
x' = s_x x \\
y' = s_y y \\
z' = s_z z
\]

\[
p' = Sp
\]

\[
S = S(s_x, s_y, s_z) = \begin{bmatrix}
s_x & 0 & 0 & 0 \\
0 & s_y & 0 & 0 \\
0 & 0 & s_z & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
Reflection

corresponds to negative scale factors

\[ s_x = -1 \quad s_y = 1 \]

\[ s_x = -1 \quad s_y = -1 \]

\[ s_x = 1 \quad s_y = -1 \]

Basic transforms in OpenGL

\[
\begin{align*}
glTranslatef(a,b,c) &; \\
glTranslated(a,b,c) &; \\
glScalef(a,b,c) &; \\
glScaled(a,b,c) &; \\
glRotatef(angle,x,y,z) &; \\
glRotated(angle,x,y,z) &;
\end{align*}
\]
Inverses

- Although we could compute inverse matrices by general formulas, we can also use simple geometric observations, for example:
  - Translation: $T^{-1}(d_x, d_y, d_z) = T(-d_x, -d_y, -d_z)$
  - Rotation: $R^{-1}(\theta) = R(-\theta)$
    * Holds for any rotation matrix
    * Note that since $\cos(-\theta) = \cos(\theta)$ and $\sin(-\theta) = -\sin(\theta)$
      $R^{-1}(\theta) = R^T(\theta)$
  - Scaling: $S^{-1}(s_x, s_y, s_z) = S(1/s_x, 1/s_y, 1/s_z)$

Concatenation

- We can form arbitrary affine transformation matrices by multiplying together rotation, translation, and scaling matrices
- Because the same transformation is applied to many vertices, the cost of forming a matrix $M = ABCD$ is not significant compared to the cost of computing $Mp$ for many vertices $p$
- The combination of transformations must be managed with care, b/c order matters
Order of Transformations

• Note that matrix on the right is the first applied
• Mathematically, the following are equivalent
  \[ p' = ABCp = A(B(Cp)) \]
  (but does not \( = \) \( CBA \ p) \)
• Some references use row matrices to present points. In terms of rows, we get
  \[ p^T' = p^T C^T B^T A^T \]

Order of Transformations

• In modeling, we often start with a simple object centered at the origin, oriented with the axis, and at a standard size
• We apply an composite transformation to its vertices to
  • Scale
  • Orient
  • Locate
Composite Transformations

• Scaling about a fixed point
  - Applying the scale transformation also moves the object being scaled.

![Diagram showing scaling about a fixed point](image)

Composite Transformations

• Exception: Scaling about origin -> no movement
• Origin is a fixed point for the scale transformation
• We use composite transformations to create scale transformations with different fixed points

![Diagram showing scaling about the origin](image)
Composite Transformations

- **Fixed point scale transformation**
  - Move fixed point \((px, py, pz)\) to origin
  - Scale by desired amount
  - Move fixed point back to original position

\[
M = T(px, py, pz) S(s_x, s_y, s_z) T(-px, -py, -pz)
\]

Composite Transformations

- Rotating about a fixed point
  - **basic** rotation alone will rotate about origin but we want:
Composite Transformations

- Rotating about a fixed point
- Move fixed point \((px, py, pz)\) to origin
- Rotate by desired amount
- Move fixed point back to original position

\[
M = T(px, py, pz) R_x(\theta) T(-px, -py, -pz)
\]
Rotation about an arbitrary axis

Rotating about an axis by theta degrees

- Rotate about x to bring axis to xz plane
- Rotate about y to align axis with z-axis
- Rotate theta degrees about z
- Unrotate about y, unrotate about x

\[ M = R_x^{-1} R_y^{-1} R_z(\theta) R_y R_x \]

- Can you determine the values of \( R_x \) and \( R_y \)?

---

Composite transformations

A series of transformations on an object can be applied as a series of matrix multiplications

\[ p = T(x_0, y_0, z_0) R(\phi_3) R(\sigma_0) T(0, h_0, 0) R(\phi_1) R(\sigma_1) T(0, h_1, 0) R(\phi_2) T(0, h_2, 0) R(\phi_0) R(\sigma_0) x \]

- \( p \): position in the global coordinate
- \( x \): position in the local coordinate

\( (h_3, 0, 0) \)