Transformations

Geometry vs. Topology

Where things are (e.g., coordinates)

Geometry:
How things are connected

Topology:
How things are connected

Geometry = changed
Topology = same (1-2-3-4-1)

Geometry = same
Topology = changed (1-2-4-3-1)

Original Object

Transformations

Suppose you have a point P and you want to move it over by 2 units in X.

How would you change P's coordinates?

This is known as a coordinate transformation

Translation Equations

General Form of 3D Linear Transformations

It's called a "Linear Transformation" because all of the coordinates are raised to the 1st power, that is, there are no x^2, x^3, etc. terms.

Transform the geometry - leave the topology as is
Scaling About the Origin

\[ x' = x \cdot S_x \]
\[ y' = y \cdot S_y \]
\[ z' = z \cdot S_z \]

2D Rotation About the Origin

\[ x' = x \cos \theta - y \sin \theta \]
\[ y' = x \sin \theta + y \cos \theta \]

Linear Equations in Matrix Form

\[ x' = Ax + By + Cz + D \]
\[ y' = Ex + Fy + Gz + H \]
\[ z' = Ix + Jy + Kz + L \]

Identity Matrix (\([ I ]\))

\[ [I] \] signifies that "Nothing has changed"

Matrix Inverse

\([M] \cdot [M]^{-1} = [I]\]

"Whatever \([M]\) does, \([M]^{-1}\) undoes"
Quick! What is the inverse of this matrix?

3D Rotation Matrix About Z

Right-handed coordinates

Right-handed positive rotation rule

+90° rotation gives: $y' = x$

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

3D Rotation Matrix About Y

+90° rotation gives: $x' = z$

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

3D Rotation Matrix About X

+90° rotation gives: $x = y$

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

How it Really Works :-)

Compound Transformations

Q: Our rotation matrices only work around the origin. What if we want to rotate about an arbitrary point (A, B)?

A: Use more than one matrix.

Write it

Say it
**Matrix Multiplication is not Commutative**

\[
\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} T_{x_{-a,b}} & \cdot & \cdot & \cdot \\ \cdot & T_{z_{-a,b}} & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}
\]

Rotate, then translate

Translate, then rotate

**Matrix Multiplication is Associative**

\[
\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \left( \left[ \begin{array}{ccc} T_{x_{-a,b}} & \cdot & \cdot \\ \cdot & T_{z_{-a,b}} & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{array} \right] \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \right)
\]

One matrix – the Current Transformation Matrix, or CTM

**OpenGL Will Do the Transformation Compounding for You!**

```cpp
for( ; ; ) {
    << Turn mouse position into Xrot and Yrot >>
    glLoadIdentity( );
    glTranslatef( A, B, C );
    glRotatef(  (GLfloat)Yrot, 0., 1., 0. );
    glRotatef(  (GLfloat)Xrot, 1., 0., 0. );
    glScalef(  (GLfloat)Scale, (GLfloat)Scale, (GLfloat)Scale  );
    glCallList( BoxList );
}
```

**OpenGL Will Do the Transformation Compounding for You!**

Typically objects are modeled around their own local origin, so the `glTranslate(-A, -B, -C)` step is unnecessary.

```cpp
for( ; ; ) {
    if( <mouse position> )
        << Turn mouse position into Xrot and Yrot >>
    glLoadIdentity( );
    glTranslatef( A, B, C );
    glRotatef(  (GLfloat)Yrot, 0., 1., 0. );
    glRotatef(  (GLfloat)Xrot, 1., 0., 0. );
    glScalef(  (GLfloat)Scale, (GLfloat)Scale, (GLfloat)Scale  );
    glCallList( BoxList );
}
```

**The Funky Rotation Matrix for an Arbitrary Axis and Angle**

\[
Q = \hat{A} \times P_1 = \hat{A} \times (P_1 + P) = \hat{A} \times P_1 + \hat{A} \times P = \hat{A} \times (P_1 + P)
\]
The Funky Rotation Matrix for an Arbitrary Axis and Angle

\[ P' = P_1 \parallel P_1' \parallel + P_1' \perp \]

\[ P' = \hat{A}(\hat{A} \cdot P) + \cos \theta [P - \hat{A}(\hat{A} \cdot P)] + \sin \theta [\hat{A} \times P] \]

For this to work, \( A \) must be a unit vector.