3D Transformations

Foundations of Computer Graphics
Torsten Möller
Schedule

• Geometry basics
• Affine transformations
• Use of homogeneous coordinates
• Concatenation of transformations
• 3D transformations
• Transformation of coordinate systems
• Transform the transforms
• Transformations in OpenGL
Transformations in 3D

• Add a z-axis to (x, y) plane
  – right-handed system:
    • positive z pointing towards us
    • positive rotation counter-clockwise
    • Standard math convention (used in our presentation and WebGL)
  – left-handed system:
    • positive z pointing away from us
    • positive rotation clockwise
    • Used in some graphics systems (z-axis as depth), e.g., POV-Ray, and Renderman
Translation in 3D

• Again, we use homogeneous coordinates

\[ T(t_x, t_y, t_z) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \]
Scaling in 3D

\[ S(s_x, s_y, s_z) = \begin{bmatrix}
  s_x & 0 & 0 & 0 \\
  0 & s_y & 0 & 0 \\
  0 & 0 & s_z & 0 \\
  0 & 0 & 0 & 1
\end{bmatrix} \]
Rotation in 3D

• Around z-axis

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

• Around x-axis

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

• Around y-axis

\[ R_y(\alpha) = \begin{bmatrix}
\cos \alpha & 0 & \sin \alpha & 0 \\
0 & 1 & 0 & 0 \\
-\sin \alpha & 0 & \cos \alpha & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \]
Properties of rotation matrix

- **Property 1:** columns and rows are mutually orthogonal unit vectors, i.e., orthonormal
- **Property 2:**
  
  determinant of $M = 1$

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

- product of any pair of orthonormal matrices is also orthonormal
- orthonormality: inverse = transpose ($P^T = P^{-1}$)
Another nice property

- row vectors: unit vectors which rotate into principal axes, i.e., 
  \[ [1 \ 0 \ 0]^T, [0 \ 1 \ 0]^T, \text{and} \ [0 \ 0 \ 1]^T \]
- column vectors: unit vectors into which principle axes rotate (obviously)

\[
\begin{bmatrix}
1 \\
0 \\
0
\end{bmatrix}
= 
\begin{bmatrix}
r_{11} & r_{12} & r_{13} \\
r_{21} & r_{22} & r_{23} \\
r_{31} & r_{32} & r_{33}
\end{bmatrix}
\begin{bmatrix}
r_{11} \\
r_{12} \\
r_{13}
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 \\
1 \\
0
\end{bmatrix}
= 
\begin{bmatrix}
r_{11} & r_{12} & r_{13} \\
r_{21} & r_{22} & r_{23} \\
r_{31} & r_{32} & r_{33}
\end{bmatrix}
\begin{bmatrix}
r_{21} \\
r_{22} \\
r_{23}
\end{bmatrix}
\begin{bmatrix}
0 \\
0 \\
1
\end{bmatrix}
= 
\begin{bmatrix}
r_{11} & r_{12} & r_{13} \\
r_{21} & r_{22} & r_{23} \\
r_{31} & r_{32} & r_{33}
\end{bmatrix}
\begin{bmatrix}
r_{31} \\
r_{32} \\
r_{33}
\end{bmatrix}
\]
Shearing in 3D

• In (y, z) w.r.t. x value  
  \[ SH_{yz} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ sh_y & 1 & 0 & 0 \\ sh_z & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

• In (z, x) w.r.t. y value  
  \[ SH_{xz} = \begin{bmatrix} 1 & sh_x & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & sh_z & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

• In (x, y) w.r.t. z value  
  \[ SH_{xy} = \begin{bmatrix} 1 & 0 & sh_x & 0 \\ 0 & 1 & sh_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]
Inverse Transforms

- Translation: negate $t_x$, $t_y$, $t_z$
- Scaling: change $s_x$ to $1/s_x$, etc.
- Rotation: negate the angle
- Shearing: negate $sh_y$, $sh_z$, etc.
General 3D transformations

- Any arbitrary sequence of rotation, translation scaling, and shear can be represented as:

\[ M = \begin{bmatrix}
  r_{11} & r_{12} & r_{13} & t_x \\
  r_{21} & r_{22} & r_{23} & t_y \\
  r_{31} & r_{32} & r_{33} & t_z \\
  0 & 0 & 0 & 1
\end{bmatrix} \]

- where upper left $3 \times 3$ is the combined scaling, rotation, and shearing; $[t_x \ t_y \ t_z]^T$ for translation
Compound transforms

• Just like in 2D, however …
• Rotation is about more than one axis

P_3 on (y, z) plane

• How should we do this?
Compute compound transform

• Use of right-hand CS

• Translation by $P_1$ so that $P_1$ is at origin

• Rotation about $y$ by $(\theta - 90^\circ)$ to get $P_1P_2$ onto the $(y, z)$ plane
Compute compound transform

- Rotation about x by $\phi$ to get $P_1P_2$ to align with the positive z-axis
- Rotation about z by $\alpha$ to get $P_1P_3$ onto the (y, z) plane
Compute compound transform

- Combined transformation:

\[ R_z(\alpha) \times R_x(\phi) \times R_y(\theta - 90^\circ) \times T(-P_1) \]
Alternative composition

• Recall the “nice” properties of the rotation matrices:

\[
R = \begin{bmatrix}
    r_{1x} & r_{2x} & r_{3x} \\
    r_{1y} & r_{2y} & r_{3y} \\
    r_{1z} & r_{2z} & r_{3z}
\end{bmatrix}
= \begin{bmatrix} R_x \\ R_y \\ R_z \end{bmatrix}
\]

• \( R_i \)'s are the unit row vectors which rotate into principal coordinate axes, e.g., \( RR_x^T = [1 \ 0 \ 0]^T \)

• Let us try to construct these directly, assuming the translation \( T(-P_1) \) is already done.
Alternative composition

- the unit vector to move to lie on the positive $z$ axis is:

$$R_z = \frac{\overrightarrow{P_1P_2}}{|\overrightarrow{P_1P_2}|}$$

- the unit vector that rotates into $x$ is normal to the plane $P_1P_2P_3$. 

$$R_x = \frac{\overrightarrow{P_1P_3} \times \overrightarrow{P_1P_2}}{|\overrightarrow{P_1P_3} \times \overrightarrow{P_1P_2}|}$$
Alternative composition

• By definition, $R_z \times R_x$ must rotate into the remaining y-axis and:

$$R_y = R_z \times R_x$$

• We are done:

$$M = \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix} \times T(-P_1), \text{ where } R = \begin{bmatrix} R_x \\ R_y \\ R_z \end{bmatrix}$$
Exercise

• How to get the jet into the desired direction of flight (DOF)?
Special transformations

• Points: we have been doing this so far
• Lines: just transform the endpoint of a line
• Planes: trickier
  – if defined by 3 points, can transform points,
  – but ... often defined by a plane equation

\[ Ax + By + Cz + D = 0 \]
Plane transform

• By homogeneous coordinates we can write:

\[ N = \begin{bmatrix} A & B & C & D \end{bmatrix}^T \]

• with \( P = [x \, y \, z \, 1]^T \):

\[ N^T P = 0 \]

• Now, suppose we want to transform our space by matrix \( M \)

• To maintain \( N^T P = 0 \), we must also transform \( N \). Let this transform be \( Q \).
Plane transform: derivation

• After the transform we have:

\[ N_n = QN \quad P_n = MP \]

• and we would like to have:

\[ N^T_n P_n = 0 \]

• now some algebra:

\[ N^T_n P_n = (QN)^T (MP) \]

\[ = N^T (Q^T M) P \]

\[ = 0 \]
Plane transform: result

• This will hold when:

\[ Q^T M = kI \]

• hence:

\[ Q = M^{-T} \]
Schedule

- Geometry basics
- Affine transformations
- Use of homogeneous coordinates
- Concatenation of transformations
- 3D transformations
- Transformation of coordinate systems
- Transform the transforms
- Transformations in OpenGL
Transformation of CS

• So far: transform points on one object with respect to the same coordinate system (CS)
• Sometimes need to change CS
• e.g. we may have many objects, each in its own CS, and we want to express all of them in some GLOBAL CS
Transformation of CS

- $P^{(i)} = \text{point in coordinate system } i$
- $M_{2\leftarrow1}$ converts representation of point in $CS_1$ to representation of point in $CS_2$
- Alternate interpretation: $M_{2\leftarrow1}$ transforms axes of $CS_2$ into axes of $CS_1$

<table>
<thead>
<tr>
<th>CS1: P=P(6,5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS2: P=P(2,3)</td>
</tr>
</tbody>
</table>

© Machiraju/Zhang/Möller
Derivation

• By definition: \( P^{(2)} = M_{2 \leftarrow 1} P^{(1)} \)

• Hence: \( CS_2 P^{(2)} = CS_1 P^{(1)} \)

• Therefore: \( CS_2 M_{2 \leftarrow 1} = CS_1 \)

• with other words, \( M_{2 \leftarrow 1} \) transforms \( CS_2 \) into \( CS_1 \)
Transform of CS: example

• Example: $M_{2\leftarrow 1} = T(-4, -2)$, this is seen by inspection
• $(2,3)^T = T(-4, -2)(6,5)^T$
• $CS_1 = CS_2 \cdot T(-4, -2)$
Transform of CS: transitivity

- Observe transitivity of this operator:
  - Given \( P(j) = M_{j \leftarrow i} P(i) \)
  - then \( P(k) = M_{k \leftarrow j} P(j) \)
  - so that \( M_{k \leftarrow i} = M_{k \leftarrow j} M_{j \leftarrow i} \)

- this is/was our basic concatenation of transformations
Transformation of CS

• Example: what is $M_{2\leftarrow 3}$?
  - $M_{2\leftarrow 3} = M_{2\leftarrow 3'} \ M_{3'\leftarrow 3}$ with CS$_{3'}$ aligned with CS$_3$ but having the same scale as CS$_2$
  - $M_{3'\leftarrow 3}$ transforms CS$_{3'}$ to CS$_3$: $S(0.5, 0.5)$
  - $M_{2\leftarrow 3'}$ transforms CS$_2$ to CS$_{3'}$: $T(2, 3)$
  - $M_{2\leftarrow 3} = T(2, 3) S(0.5, 0.5)$
  - Verify:
  - Alternative: $M_{2\leftarrow 3} = S(0.5, 0.5) \ T(4, 6)$

\[
\begin{bmatrix}
6 \\
6
\end{bmatrix} = \begin{bmatrix}
1/2 & 0 \\
0 & 1/2
\end{bmatrix} \begin{bmatrix}
8 \\
6
\end{bmatrix} + \begin{bmatrix}
2 \\
3
\end{bmatrix}
\]
Transform of CS: example

• What is $M_3 \leftarrow 4$?
  - $M_3 \leftarrow 4 = M_3 \leftarrow 4'$ $M_4' \leftarrow 4$
    with CS$_4'$ as shown
  - $M_4' \leftarrow 4$ transforms CS$_4'$
    to CS$_4$: $R(+45)$
  - $M_3 \leftarrow 4'$ transforms CS$_3$
    to CS$_4'$: $T(6.7, 1.8)$
  - $M_3 \leftarrow 4 = T(6.7, 1.8)R(+45)$
  - Verify:

\[
\begin{bmatrix}
8 \\
6
\end{bmatrix} = \begin{bmatrix}
\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\
\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2}
\end{bmatrix} \begin{bmatrix} 4 \\ 2 \end{bmatrix} + \begin{bmatrix} 6.7 \\ 1.8 \end{bmatrix}
\]

$P^{(1)} = (10,8) \quad P^{(2)} = (6,6)$

$P^{(3)} = (8,6) \quad P^{(4)} = (4,2)$
Schedule

• Geometry basics
• Affine transformations
• Use of homogeneous coordinates
• Concatenation of transformations
• 3D transformations
• Transformation of coordinate systems
• Transform the transforms
• Transformations in OpenGL
Transforming the transforms

• Suppose $Q_j$ is a transformation in $CS_j$
• Need $Q_i$ that acts on points, with respect to $CS_i$, just like $Q_j$ would on the same points
• Assume that we know $M_{i \leftarrow j}$

\[
P_i = M_{i \leftarrow j} P_j \quad P'_i = M_{i \leftarrow j} P'_j
\]

\[
P'_i = M_{i \leftarrow j} P'_j = M_{i \leftarrow j} Q_j P_j
\]

\[
P'_j = Q_j P_j = [M_{i \leftarrow j} Q_j M_{i \leftarrow j}^{-1}] P_i
\]

\[
Q_i = M_{i \leftarrow j} Q_j M_{i \leftarrow j}^{-1}
\]
Transforming the transforms

Example: How does a point P on the front tricycle wheel move in the world CS (wo) when the wheel rotates forward by an angle of $\alpha$ about its own $z_{wh}$?
Transforming the transforms

\[ P^{'}(w_o) = M_{w_o \leftarrow wh} P^{'}(wh) \]

\[ = M_{w_o \leftarrow wh} M_{wh \leftarrow wh^{'}} P^{'}(wh^{'}) \]

\[ = M_{w_o \leftarrow wh} T(\alpha r, 0, 0) P^{'}(wh^{'}) \]

\[ = M_{w_o \leftarrow wh} T(\alpha r, 0, 0) R_z(-\alpha) P^{(wh)} \]

P: original point

P': transformed point

\(P^{'}(w_o)\) is the transformed point.
Schedule

- Geometry basics
- Affine transformations
- Use of homogeneous coordinates
- Concatenation of transformations
- 3D transformations
- Transformation of coordinate systems
- Transform the transforms
- Transformations in OpenGL
Coordinate Systems

• The units in **points** are determined by the application and are called
  – *object* (or *model*) coordinates
  – *world* coordinates

• Viewing specifications usually are also in object coordinates

• transformed through
  – *eye* (or *camera*) coordinates
  – *clip* coordinates
  – normalized *device* coordinates
  – *window* (or *screen*) coordinates
CTM in OpenGL

- OpenGL had a model-view and a projection matrix in the pipeline which were concatenated together to form the CTM
- Angel emulates this process
Rotation, Translation, Scaling

• Create an identity matrix:

\[
\text{mat4 } m = \text{Identity}();
\]

• Multiply on right by rotation matrix of \text{theta} in degrees where (\text{vx}, \text{vy}, \text{vz}) define axis of rotation

\[
\text{mat4 } r = \text{Rotate}(\text{theta}, \text{vx}, \text{vy}, \text{vz}) \quad m = m*r;
\]

• Do same with translation and scaling:

\[
\text{mat4 } s = \text{Scale}( \text{sx}, \text{sy}, \text{sz}) \quad \text{mat4 } t = \text{Translate}(\text{dx}, \text{dy}, \text{dz}); \quad m = m*s*t;
\]
Example

- Rotation about z axis by 30 degrees with a fixed point of (1.0, 2.0, 3.0)

```
mat4 m = Identity();
    m = Translate(1.0, 2.0, 3.0) * 
        Rotate(30.0, 0.0, 0.0, 1.0) * 
        Translate(-1.0, -2.0, -3.0);
```

- Remember that last matrix specified in the program is the first applied
Arbitrary Matrices

- Can load and multiply by matrices defined in the application program
- Matrices are stored as one dimensional array of 16 elements which are the components of the desired 4 x 4 matrix stored by columns
- WebGL functions that have matrices as parameters allow the application to send the matrix or its transpose