Transformations I

Ronald A. Metoyer
Oregon State University
Transformation

Function mapping a set $X$ onto itself or another set

Important in Graphics for

- Modeling scenes
- Moving characters (and cameras)
- Changing coordinate systems/frames
Affine Transformations

*Preserves straight lines* and ratios of distances between points on a line

Characteristic of many physically important transformations
  - Rigid body transformations: rotation, translation
  - Scaling, shear

Importance in graphics is that we need only transform endpoints of line segments and let implementation draw line segment between the transformed endpoints

Parallel lines remain parallel and straight lines remain straight
Translation (2D)

\[ x' = x + dx \]
\[ y' = y + dy \]

\[ T = \begin{bmatrix} dx \\ dy \end{bmatrix} \quad P = \begin{bmatrix} x \\ y \end{bmatrix} \quad P' = \begin{bmatrix} x + dx \\ y + dy \end{bmatrix} \]

\[ P' = P + T \]
Rotation (of theta about origin)

\[ x' = r \cos \phi \]
\[ y' = r \sin \phi \]

\[ x' = r \cos(\theta + \phi) \]
\[ y' = r \sin(\theta + \phi) \]

\[ x' = r(\cos \theta \cos \phi - \sin \phi \sin \theta) \]
\[ y' = r(\cos \phi \sin \theta + \sin \phi \cos \theta) \]

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

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

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

\[ R = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \]

\[ P' = RP \]
Scale

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

\[ S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \quad P = \begin{bmatrix} x \\ y \end{bmatrix} \quad P' = \begin{bmatrix} s_x x \\ s_y y \end{bmatrix} \]

\[ P' = SP \]
X-Reflection

$(-1, 4) \rightarrow (1, 4)$
X-Reflection

Flip along the x axis

\[ x' = -x \]
\[ y' = y \]

\[ Rf_x = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \]

\[ P' = Rf_x P \]
Reflections
Efficiency Problem...

*Translation is performed with an addition, while scale and rotation are performed with multiplication.*

We’d like to treat them all the same to avoid extra bookkeeping!
- Optimize the hardware
- Also useful when we compose transformations

\[
P' = T + P \quad \text{Translation}
\]
\[
P' = SP \quad \text{Scale}
\]
\[
P' = RP \quad \text{Rotation}
\]
Homogenous Coordinates

Homogenous Point \( x,y \)

\[
\begin{bmatrix}
x \\
y \\
w
\end{bmatrix}
\]

Cartesian Point \( x,y \)

\[
\begin{bmatrix}
x / w \\
y / w \\
1
\end{bmatrix}
\]

When \( w=1 \) the homogenous point = cartesian point
Use \( w=0 \) to represent a homogenous direction vector (we don’t want direction vectors or offset vectors to move!)

*Treat translation as a multiplication*
Homogenous Coordinates - Translation

\[ x' = x + dx \]
\[ y' = y + dy \]

\[
T = \begin{bmatrix}
1 & 0 & dx \\
0 & 1 & dy \\
0 & 0 & 1
\end{bmatrix} \quad P = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad P' = \begin{bmatrix} x + dx \\ y + dy \\ 1 \end{bmatrix}
\]

\[ P' = TP \]

What if \( p \) is a vector? [\( x, y, 0 \)]
Homogenous Coordinates – T,R,S

\[
T = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} \quad R = \begin{bmatrix} c & -s & 0 \\ s & c & 0 \\ 0 & 0 & 1 \end{bmatrix}
\]

\[
S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}
\]
Composing Transformations

Series of transformations can be composited into a single transformation matrix

For example, how do we rotate around a point other than the origin?
Composition: rotation about center

Apply a series of transformations

- translate to the origin
- rotate
- translate back
Composition

\[ P' = T_2 R T_1 P \]

\[
T_1 = \begin{bmatrix}
1 & 0 & -a \\
0 & 1 & -b \\
0 & 0 & 1
\end{bmatrix} \\
R = \begin{bmatrix}
0 & -1 & 0 \\
1 & 0 & 0 \\
0 & 0 & 1
\end{bmatrix} \\
T_2 = \begin{bmatrix}
1 & 0 & a \\
0 & 1 & b \\
0 & 0 & 1
\end{bmatrix}
\]
Composition Example #2

translate by $-2$ in $x$
scale by 2 in $x,y$ (uniform scaling)

\[
T = \begin{bmatrix}
1 & 0 & -2 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

\[
S = \begin{bmatrix}
2 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

\[ST \neq TS\]
Composition Example

#1 Translate, Scale

(1,1) \rightarrow (1,1) \rightarrow (-1,1) \rightarrow (-2,2)
(3,1) \rightarrow (1,1) \rightarrow (2,2)
(2,3) \rightarrow (0,3) \rightarrow (0,6)

#2 Scale, Translate

(1,1) \rightarrow (2,2) \rightarrow (0,2)
(3,1) \rightarrow (6,2) \rightarrow (4,2)
(2,3) \rightarrow (4,6) \rightarrow (2,6)
Composition Example

\[ P' = STP \]

\[
\begin{bmatrix}
2 & 0 \\
0 & 2 \\
0 & 0
\end{bmatrix}
\begin{bmatrix}
1 & 0 & -2 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
= \begin{bmatrix}
2 & 0 & -4 \\
0 & 2 & 0 \\
0 & 0 & 1
\end{bmatrix} P
\]

\[ Scale(2.0,2.0); \]
\[ Translate(-2.0,0.0); \]
\[ drawTriangle(); \]

\[ P' = TSP \]

\[
\begin{bmatrix}
1 & 0 & -2 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
2 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 1
\end{bmatrix}
= \begin{bmatrix}
2 & 0 & -2 \\
0 & 2 & 0 \\
0 & 0 & 1
\end{bmatrix} P
\]

\[ Translate(-2.0, 0.0); \]
\[ Scale(2.0,2.0); \]
\[ drawTriangle(); \]

Matrix multiplication is not commutative!!! So the order of applying the transformations is important.
Thinking about Transformations

There are two ways to think about (or visualize) transformations

In a grand, fixed coordinate system
In a moving coordinate system

Depending on how you visualize your transformations, you must perform your multiplications in a specific order

right to left (bottom to top) w.r.t. grand fixed system
left to right (top to bottom) w.r.t. moving coordinate system
Order Example...

How do I achieve this new configuration?
Absolute: with respect to fixed reference

Absolute transformation. Execute the transforms Right to Left w.r.t. reference frame

\[ P' = TRP \]
Relative: wrt moving frame

Relative transformation. Left to right w.r.t. the new moving coordinate system

\[ P' = TRP \]
Absolute vs. Relative

Results are the same

Useful to think in terms of relative when building a hierarchical character

Think in terms of the system that makes the most sense
- apply transforms in the correct order
- also applies to graphics library routines
  - left to right means top to bottom code calls
Inverses

Although we could compute inverse matrices for transformations by general formulas, we can use simple geometric observations.

Translation: \( T^{-1}(d_x, d_y) = T(-d_x, -d_y) \)
Rotation: \( R^{-1}(q) = R(-q) \)

Since \( \cos(-q) = \cos(q) \) and \( \sin(-q) = -\sin(q) \)
\( R^{-1}(q) = R^{T}(q) \)

Scaling: \( S^{-1}(s_x, s_y) = S(1/s_x, 1/s_y) \)
What about 3D?
3D Homogenous Coordinates

Homogenous 3D point

\[
\begin{bmatrix}
  x \\
  y \\
  z \\
  w
\end{bmatrix}
\]

Cartesian 3D point = Homogenous 3D point when w=1

\[
\begin{bmatrix}
  x \\
  y \\
  z \\
  1
\end{bmatrix}
\]
Translation and Scale

Translation

\[
\begin{bmatrix}
1 & 0 & 0 & tx \\
0 & 1 & 0 & ty \\
0 & 0 & 1 & tz \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Scale

\[
\begin{bmatrix}
sx & 0 & 0 & 0 \\
0 & sy & 0 & 0 \\
0 & 0 & sz & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
3D Rotation

Now, have 3 axes we can rotate about

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

RotateX

Notice, X does not change
Looking down x negative rotating counter clockwise
3D Rotations

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

\text{RotateY}

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

\text{RotateZ}
Inverses – still hold!

Although we could compute inverse matrices for transformations by general formulas, we can use simple geometric observations:

Translation: \(T^{-1}(d_x, d_y, d_z) = T(-d_x, -d_y, -d_z)\)
Rotation: \(R^{-1}(q) = R(-q)\)

Since \(\cos(-q) = \cos(q)\) and \(\sin(-q) = -\sin(q)\)
\(R^{-1}(q) = R^T(q)\)

Scaling: \(S^{-1}(s_x, s_y, s_z) = S(1/s_x, 1/s_y, 1/s_z)\)