Lecture 3:
Transforms

Computer Graphics
CMU 15-462/15-662, Fall 2015
Cube

(-1, -1, 1)  (1, -1, 1)

(-1, -1, -1) (1, -1, -1)

(1, 1, -1)   (1, 1, 1)

(-1, 1, -1)  (1, 1, -1)

(-1, 1, 1)   (1, 1, 1)
Cube man
$f$ transforms $x$ to $f(x)$
Linear transforms

\[ f(x + y) = f(x) + f(y) \]

\[ f(ax) = af(x) \]
Scale

Uniform scale:
\[ S_a(x) = ax \]

Non-uniform scale??

CMU 15-462/662, Fall 2015
Is scale a linear transform?

Yes!

\[
S_2(x) = 2x \\
aS_2(x) = 2ax \\
S_2(ax) = 2ax \\
S_2(ax) = aS_2(x)
\]

\[
S_2(x + y) = 2(x + y) \\
S_2(x) + S_2(y) = 2x + 2y \\
S_2(x + y) = S_2(x) + S_2(y)
\]
$R_\theta = \text{rotate counter-clockwise by } \theta$
Rotation as Circular Motion

\[ R_\theta = \text{rotate counter-clockwise by } \theta \]

As angle changes, points move along \textit{circular} trajectories.

Hence, rotations preserve length of vectors: 
\[ |R_\theta(x)| = |x| \]
Is rotation linear?

Yes!
Translation

\[ T_b(x) = \text{translate by } b \]

\[ T_b(x) = x + b \]
Is translation linear?

No. Translation is affine.
Reflection

\[ \text{Re}_y = \text{reflection about } y \]

\[ \text{Re}_x = \text{reflection about } x \]
Shear (in $x$ direction)
Compose basic transforms to construct more complex transforms

Note: order of composition matters
Top-right: scale, then translate
Bottom-right: translate, then scale
How would you perform these transformations?
Common pattern: rotation about point \(x\)

1. Step 1: translate by \(-x\)
2. Step 2: rotate
3. Step 4: translate by \(x\)
Summary of basic transforms

Linear:

- Composition of linear transform + translation
- (all examples on previous two slides)

Not linear:

- Translation

Affine:

- Composition of linear transform + translation
- (all examples on previous two slides)

Not affine: perspective projection (will discuss later)

Euclidean: (Isometries)

- Preserve distance between points (preserves length)

- "Rigid body" transforms are Euclidean transforms that also preserve "winding" (does not include reflection)
Representing Transforms
Review: representing points in a coordinate space

Consider coordinate space defined by orthogonal vectors $\mathbf{e}_1$ and $\mathbf{e}_2$

$x = 2\mathbf{e}_1 + 2\mathbf{e}_2$

$x = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$

$x = \begin{bmatrix} 0.5 \\ 1 \end{bmatrix}$ in coordinate space defined by $\mathbf{e}_1$ and $\mathbf{e}_2$, with origin at (1.5, 1)

$x = \begin{bmatrix} \sqrt{8} \\ 0 \end{bmatrix}$ in coordinate space defined by $\mathbf{e}_3$ and $\mathbf{e}_4$, with origin at (0, 0)
Review: 2D matrix multiplication

\[
\begin{bmatrix}
ax + by \\
cx + dy
\end{bmatrix}
= \begin{bmatrix} a & b \\ c & d \end{bmatrix}
\begin{bmatrix} x \\ y \end{bmatrix}
\]

\[
x \begin{bmatrix} a \\ c \end{bmatrix} + y \begin{bmatrix} b \\ d \end{bmatrix} =
\]
Linear transforms in 2D can be represented as 2x2 matrices

Consider non-uniform scale: \( S_s = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \)

Scaling amounts in each direction:
\( s = \begin{bmatrix} 0.5 & 2 \end{bmatrix}^T \)

Matrix representing scale transform:
\( S_s = \begin{bmatrix} 0.5 & 0 \\ 0 & 2 \end{bmatrix} \)
Rotation matrix (2D)

Question: what happens to (1, 0) and (0,1) after rotation by $\theta$?

Reminder: rotation moves points along circular trajectories.

(Recall that $\cos \theta$ and $\sin \theta$ are the coordinates of a point on the unit circle.)

Answer:

\[
R_\theta(1, 0) = (\cos(\theta), \sin(\theta))
\]
\[
R_\theta(0, 1) = (\cos(\theta + \pi/2), \sin(\theta + \pi/2))
\]

Which means the matrix must look like:

\[
R_\theta = \begin{bmatrix}
\cos(\theta) & \cos(\theta + \pi/2) \\
\sin(\theta) & \sin(\theta + \pi/2)
\end{bmatrix}
\]
\[
= \begin{bmatrix}
\cos(\theta) & -\sin(\theta) \\
\sin(\theta) & \cos(\theta)
\end{bmatrix}
\]
Rotation matrix (2D): another way...

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

Shear in x:
\[ H_{xs} = \begin{bmatrix} 1 & s \\ 0 & 1 \end{bmatrix} \]

Shear in y:
\[ H_{ys} = \begin{bmatrix} 1 & 0 \\ s & 1 \end{bmatrix} \]
How do we compose linear transforms?

Compose linear transforms via matrix multiplication. This example: uniform scale, followed by rotation

$$f(x) = \frac{\pi}{4} S_{[1.5,1.5]} x$$

Enables simple, efficient implementation: reduce complex chain of transforms to a single matrix multiplication.
Translation?

\[ T_b(x) = x + b \]

Recall: translation is not a linear transform

→ Output coefficients are not a linear combination of input coefficients
→ Translation operation cannot be represented by a 2x2 matrix

\[
\begin{align*}
x_{\text{out}x} &= x_x + b_x \\
x_{\text{out}y} &= x_y + b_y
\end{align*}
\]

Translation math
2D homogeneous coordinates (2D-H)

Key idea: represent 2D points in 3D coordinate space

So the point \((x, y)\) is represented as the 3-vector: 
\[
\begin{bmatrix}
x \\
y \\
1
\end{bmatrix}^T
\]

And transforms are represented a 3x3 matrices that transform these vectors.

For example: here are 2D scale and rotation transforms written in 2D homogeneous form:

\[
S_s = \begin{bmatrix}
S_x & 0 & 0 \\
0 & S_y & 0 \\
0 & 0 & 1
\end{bmatrix} \quad R_\theta = \begin{bmatrix}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

Observe:

In these examples, the last row just propagates third coordinate of input to output.
Expressing transformations in 2D-H coords

Translation expressed as 3x3 matrix multiplication:

\[
T_b = \begin{bmatrix}
1 & 0 & b_x \\
0 & 1 & b_y \\
0 & 0 & 1
\end{bmatrix}
\]

\[
T_b x = \begin{bmatrix}
1 & 0 & b_x \\
0 & 1 & b_y \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x_x \\
x_y \\
1
\end{bmatrix} = \begin{bmatrix}
x_x + b_x \\
x_y + b_y \\
1
\end{bmatrix}
\]

Homogeneous representation enables composition of affine transforms!

Example: rotation about point \( b \)

\[
T_b R_\theta T_{-b}
\]
Homogeneous coordinates: some intuition

Many points in 2D-H correspond to same point in 2D

\( x \) and \( wx \) correspond to the same 2D point
(divide by \( w \) to convert 2D-H back to 2D)

Translation is a shear in \( x \) and \( y \) in 2D-H space

\[
T_{b}x = \begin{bmatrix} 1 & 0 & b_x \\ 0 & 1 & b_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} wx_x \\ wx_y \\ w \end{bmatrix} = \begin{bmatrix} wx_x + wb_x \\ wx_y + wb_y \\ w \end{bmatrix}
\]
Homogeneous coordinates: points vs. vectors

2D-H points with $w=0$ represent 2D vectors (think: directions are points at infinity)

Unlike 2D, points and directions are distinguishable by their representation in 2D-H

Note: translation does not modify directions:

$$T_b v = \begin{bmatrix} 1 & 0 & b_x \\ 0 & 1 & b_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ 0 \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ 0 \end{bmatrix}$$
Visualizing 2D transformations in 2D-H

Original shape in 2D can be viewed as many copies, uniformly scaled by \( w \).

2D scale \( \leftrightarrow \) scale \( x \) and \( y \); preserve \( w \) (Question: what happens to 2D shape if you scale \( x, y, \) and \( w \) uniformly?)

2D rotation \( \leftrightarrow \) rotate around \( w \)

2D translate \( \leftrightarrow \) shear in \( xy \)
Moving to 3D (and 3D-H)

Represent 3D transforms as 3x3 matrices and 3D-H transforms as 4x4 matrices

**Scale:**  
\[
S_s = \begin{bmatrix}
S_x & 0 & 0 \\
0 & S_y & 0 \\
0 & 0 & S_z
\end{bmatrix} \quad \begin{bmatrix}
S_x & 0 & 0 & 0 \\
0 & S_y & 0 & 0 \\
0 & 0 & S_z & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

**Shear (in x, based on y,z position):**  
\[
H_{x,d} = \begin{bmatrix}
1 & d_y & d_z \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix} \quad H_{x,d} = \begin{bmatrix}
1 & d_y & d_z & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

**Translate:**  
\[
T_b = \begin{bmatrix}
1 & 0 & 0 & b_x \\
0 & 1 & 0 & b_y \\
0 & 0 & 1 & b_z \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
Rotations in 3D

Rotation about x axis:

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

Rotation about y axis:

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

Rotation about z axis:

\[ R_{z,\theta} = \begin{bmatrix}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{bmatrix} \]
Rotation about an arbitrary axis

To rotate by $\theta$ about $\mathbf{w}$:

1. Form orthonormal basis around $\mathbf{w}$ (see $\mathbf{u}$ and $\mathbf{v}$ in figure)

2. Rotate to map $\mathbf{w}$ to $[0 \ 0 \ 1]$ (change in coordinate space)

$$
\mathbf{R}_{uvw} = \begin{bmatrix}
\mathbf{u}_x & \mathbf{u}_y & \mathbf{u}_z \\
\mathbf{v}_x & \mathbf{v}_y & \mathbf{v}_z \\
\mathbf{w}_x & \mathbf{w}_y & \mathbf{w}_z
\end{bmatrix}
$$

$$
\mathbf{R}_{uvw} \mathbf{u} = [1 \ 0 \ 0]
$$

$$
\mathbf{R}_{uvw} \mathbf{v} = [0 \ 1 \ 0]
$$

$$
\mathbf{R}_{uvw} \mathbf{w} = [0 \ 0 \ 1]
$$

3. Perform rotation about $z$: $\mathbf{R}_{z,\theta}$

4. Rotate back to original coordinate space: $\mathbf{R}_{uvw}^T$

$$
\mathbf{R}_{uvw}^{-1} = \mathbf{R}_{uvw}^T = \begin{bmatrix}
\mathbf{u}_x & \mathbf{v}_x & \mathbf{w}_x \\
\mathbf{u}_y & \mathbf{v}_y & \mathbf{w}_y \\
\mathbf{u}_z & \mathbf{v}_z & \mathbf{w}_z
\end{bmatrix}
$$

$$
\mathbf{R}_{w,\theta} = \mathbf{R}_{uvw}^T \mathbf{R}_{z,\theta} \mathbf{R}_{uvw}
$$
Alternative representation for rotations: complex numbers

\[ z = a + bi \]

\[ i^2 = -1 \]
\[ (a + bi)(c + di) = (ac - bd) + (bc + ad)i \]

\[ iz = i(a + bi) = -b + ai \]  
(multiplication by \( i \) → rotation by \( \pi/2 \))

\[ i(iz) = -a - bi = -z \]  
(multiplication by \( i^2 \) → rotation by \( \pi \))

\[ \mathbf{R}_\theta = e^{i\theta} = \cos \theta + i \sin \theta \]
Alternative representation for rotations: complex numbers

Quaternions are a representation of 3D rotations based on complex numbers [see further reading on web site]

\[ Q = (q_v, q_w) = i q_x + j q_y + k q_z + q_w \]
Another way to think about transformations: change in coordinate space

Interpretation of transforms so far in this lecture: transforms move points

Point $x$ moved to new position $f(x)$

Alternative interpretation:

Transformations induce of change of coordinate space: Representation of $x$ changes since point is now described in a new coordinate space.
Review from last time: screen transform *

Convert points in normalized coordinate space to screen pixel coordinates

Example:
All points within (-1,1) to (1,1) region are on screen
(1,1) in normalized space maps to (W,0) in screen

Step 1: reflect about \( x \)
Step 2: translate by \( (1,1) \)
Step 3: scale by \( (W/2,H/2) \)

* Adopting convention that top-left of screen is \( (0,0) \) to match SVG convention in Assignment 1.
Many 3D graphics systems like OpenGL place \( (0,0) \) in bottom-left. In this case what would the transform be?
Example: simple camera transform

- Consider object in world at (10, 2, 0)
- Consider camera at (4, 2, 0), looking down x axis

- Translating object vertex positions by (-4, -2, 0) yields position relative to camera.
- Rotation about $y$ by $-\pi/2$ gives position of object in coordinate system where camera’s view direction is aligned with the $z$ axis *

* The convenience of such a coordinate system will become clear on the next slide!
Basic perspective projection

Desired perspective projected result (2D point):

\[ p_{2D} = \begin{bmatrix} x_x / x_z & x_y / x_z \end{bmatrix}^T \]

Input: point in 3D-H

\[ x = \begin{bmatrix} x_x & x_y & x_z & 1 \end{bmatrix} \]

After applying \( P \): point in 3D-H

\[ Px = \begin{bmatrix} x_x & x_y & x_z & x_z \end{bmatrix}^T \]

Point in 2D-H (drop z coord)

\[ P_{2D-H} = \begin{bmatrix} x_x & x_y & x_z \end{bmatrix}^T \]

Point in 2D (homogeneous divide)

\[ P_{2D} = \begin{bmatrix} x_x / x_z & x_y / x_z \end{bmatrix}^T \]

Assumption:
Pinhole camera at (0,0) looking down z
Transformations summary

- Transformations can be interpreted as operations that move points in space
  - e.g., for modeling, animation

- Or as a change of coordinate system
  - e.g., screen and view transforms

- Construct complex transformations as compositions of basic transforms

- Homogeneous coordinate representation allows for expression of non-linear transforms (e.g., affine, perspective projection) as matrix operations (linear transforms) in higher-dimensional space
  - Matrix representation affords simple implementation and efficient composition