Coordinate Spaces and Transformations

Interactive Computer Graphics
Stanford CS248, Winter 2019
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)
Consider drawing a cube person
Transformations in character rigging
Transformations in instancing
Basic idea: $f$ transforms $x$ to $f(x)$
What can we do with \textit{linear} transformations?

- What does \textit{linear} mean?

\[
\begin{align*}
  f(x + y) &= f(x) + f(y) \\
  f(ax) &= af(x)
\end{align*}
\]

- Cheap to compute
- Composition of linear transformations is linear
  - Leads to uniform representation of transformations
Linear transformation

\[ f(u + v) = f(u) + f(v) \]
\[ f(au) = af(u) \]

- In other words: if it doesn’t matter whether we add the vectors and then apply the map, or apply the map and then add the vectors (and likewise for scaling):
Linear transforms/maps—visualized

- Example:

Key idea: linear maps take lines to lines
Uniform scale:

\[ S_a(x) = a x \]
Is scale a linear transform?

Yes!

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

\[ S_2(x + y) = 2(x + y) \]
\[ S_2(x) + S_2(y) = 2x + 2y \]
\[ S_2(x + y) = S_2(x) + S_2(y) \]
Rotation

\[ R_\theta = \text{rotate counter-clockwise by } \theta \]
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: \[ |R_\theta(x)| = |x| \]
Yes!
**Translation**

\[ T_b \quad \text{"translate by } b \text{"} \]

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

No. Translation is affine.
Reflection

$Re_y = \text{reflection about } y$

$Re_x = \text{reflection about } x$

Stanford CS 248, Winter 2019
Shear (in $x$ direction)

Before shear:
- $x_0$, $x_1$, $x_2$, $x_3$

After shear:
- $H_x(x_0)$, $H_x(x_1)$, $H_x(x_2)$, $H_x(x_3)$
Compose basic transformations to construct more complicated ones

Note: order of composition matters

Top-right: scale, then translate
Bottom-right: translate, then scale
How would you perform these transformations?

Usually more than one way to do it!
Common task: rotate about a point x

Step 1: translate by -x

Step 2: rotate

Step 3: translate by x
Summary of basic transformations

Linear:
\[ f(x + y) = f(x) + f(y) \]
\[ f(ax) = af(x) \]

Scale
Rotation
Reflection
Shear

Affine:
Composition of linear transform + translation
(all examples on previous two slides)
\[ f(x) = g(x) + b \]

Not affine: perspective projection (will discuss later)

Euclidean: (Isometries)
Preserve distance between points (preserves length)
\[ |f(x) - f(y)| = |x - y| \]

Translation
Rotation
Reflection

“Rigid body” transformations are distance-preserving motions that also preserve orientation (i.e., does not include reflection)
Representing Transformations in Coordinates
Review: representing points in a coordinate space

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

\[ x = 2\mathbf{e}_1 + 2\mathbf{e}_2 \]
\[ x = [2 \quad 2] \]

\[ x = [0.5 \quad 1] \quad \text{in coordinate space defined by } \mathbf{e}_1 \text{ and } \mathbf{e}_2, \text{ with origin at (1.5, 1)} \]
\[ x = [\sqrt{8} \quad 0] \quad \text{in coordinate space defined by } \mathbf{e}_3 \text{ and } \mathbf{e}_4, \text{ with origin at (0, 0)} \]
Review: 2D matrix multiplication

\[
\begin{bmatrix}
a & b \\
c & d
\end{bmatrix}
\begin{bmatrix}
x \\
y
\end{bmatrix}
= \\
x
a
\begin{bmatrix}
1
0
\end{bmatrix}
+ y
b
\begin{bmatrix}
0
1
\end{bmatrix}
= \\
\begin{bmatrix}
ax + by \\
cx + dy
\end{bmatrix}
\]

- Matrix multiplication is linear combination of columns
- Encodes a linear map!
Linear maps via matrices

- Example: suppose I have a linear map
  \[ f(u) = u_1 a_1 + u_2 a_2 \]

- Encoding as a matrix: “a” vectors become matrix columns:
  \[
  A := \begin{bmatrix}
  a_{1,x} & a_{2,x} \\
  a_{1,y} & a_{2,y} \\
  a_{1,z} & a_{2,z}
  \end{bmatrix}
  \]

- Matrix-vector multiply computes same output as original map:
  \[
  \begin{bmatrix}
  a_{1,x} & a_{2,x} \\
  a_{1,y} & a_{2,y} \\
  a_{1,z} & a_{2,z}
  \end{bmatrix}
  \begin{bmatrix}
  u_1 \\
  u_2
  \end{bmatrix}
  = \begin{bmatrix}
  a_{1,x}u_1 + a_{2,x}u_2 \\
  a_{1,y}u_1 + a_{2,y}u_2 \\
  a_{1,z}u_1 + a_{2,z}u_2
  \end{bmatrix}
  = u_1 a_1 + u_2 a_2
  \]
Linear transformations 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:
\[
\begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} = \begin{bmatrix} 0.5 & 2 \\ 0 & 2 \end{bmatrix}
\]

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 θ?

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:**
\[ H_{ys} = \begin{bmatrix} 1 & 0 \\ s & 1 \end{bmatrix} \]

**Arbitrary shear:**
\[ H_{st} = \begin{bmatrix} 1 & s \\ t & 1 \end{bmatrix} \]
How do we compose linear transformations?

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

\[ f(x) = R_{\pi/4} S_{[1.5,1.5]} x \]

Enables simple, efficient implementation: reduce complex chain of transformations to a single matrix multiplication.
How do we deal with translation? (Not linear)

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

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)

Idea: represent 2D points with THREE values ("homogeneous coordinates")

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

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

Recover final 2D coordinates by dividing by "extra" (third) coordinate

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

(More on this later…)
Example: scale and rotation in 2D-H coords

For transformations that are already linear, not much changes:

\[
S_s = \begin{bmatrix}
x \cdot S_x & 0 & 0 \\
0 & y \cdot S_y & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

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

Notice that the last row/column doesn’t do anything interesting. E.g., for scaling:

\[
S_s \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x \cdot S_x \\ y \cdot S_y \\ 1 \end{bmatrix}
\]

Now we divide by the 3rd coordinate to get our final 2D coordinates (not too exciting!)

\[
\begin{bmatrix} x \cdot S_x \\ y \cdot S_y \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} x \cdot S_x / 1 \\ y \cdot S_y / 1 \\ 1 \end{bmatrix} = \begin{bmatrix} x \cdot S_x \\ y \cdot S_y \end{bmatrix}
\]

(Will get more interesting when we talk about perspective...)

Stanford CS 248, Winter 2019
Translation in 2D homogeneous coordinates

Translation expressed as 3x3 matrix multiplication:

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

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

(remember: just a linear combination of columns!)

Cool: homogeneous coordinates let us encode translations as linear transformations!
Homogeneous coordinates: some intuition

Many points in 2D-H correspond to same point in 2D
\( \mathbf{x} \) and \( w\mathbf{x} \) 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

\[
\mathbf{T}_b\mathbf{x} = \begin{bmatrix} 1 & 0 & b_x \\ 0 & 1 & b_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{wx} \\ \mathbf{wy} \\ w \end{bmatrix} = \begin{bmatrix} \mathbf{wx} + w\mathbf{b}_x \\ \mathbf{wy} + w\mathbf{b}_y \\ w \end{bmatrix}
\]
Translation = shear in homogeneous space

For simplicity, consider 1D-H:

Translate by $t=2$: \[ T = \begin{bmatrix} 1 & t \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} \]

Recall: this is a shear in homogeneous $x$.

1D translation is affine in 1D ($x + t$), but it is linear in 1D-H.
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 rotation $\leftrightarrow$ rotate around $w$

2D scale $\leftrightarrow$ scale $x$ and $y$; preserve $w$

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

2D translate $\leftrightarrow$ shear in 2D-H (LINEAR!)
Moving to 3D (and 3D-H)

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

Scale:

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

Shear (in x, based on y,z position):

\[
\begin{align*}
H_{x,d} & = \begin{bmatrix}
1 & d_y & d_z \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix} & H_{x,d} & = \begin{bmatrix}
1 & d_y & d_z & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\end{align*}
\]

Translate:

\[
\begin{align*}
T_b & = \begin{bmatrix}
1 & 0 & 0 & b_x \\
0 & 1 & 0 & b_y \\
0 & 0 & 1 & b_z \\
0 & 0 & 0 & 1
\end{bmatrix}
\end{align*}
\]
Commutativity of rotations—2D

- In 2D, order of rotations doesn’t matter:

Same result! (“2D rotations commute”)
Commutativity of rotations—3D

- What about in 3D?

- **IN-CLASS ACTIVITY:**
  - Rotate 90° around Y, then 90° around Z, then 90° around X
  - Rotate 90° around Z, then 90° around Y, then 90° around X
  - (Was there any difference?)

**CONCLUSION:** bad things can happen if we’re not careful about the order in which we apply rotations!
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:

$$R_{z, \theta} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

View looking down -x axis:

View looking down -y axis:

View looking down -z axis:
Representing rotations in 3D—euler angles

- How do we express rotations in 3D?
- One idea: we know how to do 2D rotations
- Why not simply apply rotations around the three axes? (X,Y,Z)
- Scheme is called *Euler angles*
- PROBLEM: “Gimbal Lock” [DEMO]
Alternative representations of 3D rotations

- Axis-angle rotations
- Quaternions (not today)
Another way to think about transformations: change of coordinates

Interpretation of transformations so far in this lecture: *points get moved*

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

---

Alternative interpretation:

Transformations induce a change of coordinate frame:
Representation of $x$ changes since point is now expressed in new coordinates
Screen transformation *

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 space
(-1,-1) in normalized space maps to (0,H) in screen space

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

* This slide adopts 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 positioned 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 new coordinate system where camera’s view direction is aligned with the -z axis *

What transform places in the object in a coordinate space where the camera is at the origin and the camera is looking directly down the -z 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 new 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!
Camera looking in a different direction

Consider camera at origin looking in direction \( \mathbf{w} \)

What transform places in the object in a coordinate space where the camera is at the origin and the camera is looking directly down the -z axis?

Form orthonormal basis around \( \mathbf{w} \): (see \( \mathbf{u} \) and \( \mathbf{v} \))

Consider rotation matrix: \( \mathbf{R} \)

\[
\mathbf{R} = \begin{bmatrix}
\mathbf{u}_x & \mathbf{v}_x & -\mathbf{w}_x \\
\mathbf{u}_y & \mathbf{v}_y & -\mathbf{w}_y \\
\mathbf{u}_z & \mathbf{v}_z & -\mathbf{w}_z
\end{bmatrix}
\]

\( \mathbf{R} \) maps x-axis to \( \mathbf{u} \), y-axis to \( \mathbf{v} \), z axis to -\( \mathbf{w} \)

How do we invert?

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

Why is that the inverse?

\[
\mathbf{R}^T \mathbf{u} = [\mathbf{u} \cdot \mathbf{u} \ \mathbf{v} \cdot \mathbf{u} \ -\mathbf{w} \cdot \mathbf{u}]^T = [1 \ 0 \ 0]^T
\]

\[
\mathbf{R}^T \mathbf{v} = [\mathbf{u} \cdot \mathbf{v} \ \mathbf{v} \cdot \mathbf{v} \ -\mathbf{w} \cdot \mathbf{v}]^T = [0 \ 1 \ 0]^T
\]

\[
\mathbf{R}^T \mathbf{w} = [\mathbf{u} \cdot \mathbf{w} \ \mathbf{v} \cdot \mathbf{w} \ -\mathbf{w} \cdot \mathbf{w}]^T = [0 \ 0 \ -1]^T
\]
Self-check exercise (for home)

- Given a camera position $P$
- And a camera orientation given by orthonormal basis $u,v,w$ (camera looking in $w$)
- What is a transformation matrix that places the scene in a coordinate space where...
  - The camera is at the origin
  - The camera is looking down -$z$. 
Let’s make a cube person
Skeleton - hierarchical representation

torso
  head
  right arm
    upper arm
    lower arm
    hand
  left arm
    upper arm
    lower arm
    hand
  right leg
    upper leg
    lower leg
    foot
  left leg
    upper leg
    lower leg
    foot
Hierarchical representation

- Grouped representation (tree)
  - Each group contains subgroups and/or shapes
  - Each group is associated with a transform relative to parent group
  - Transform on leaf-node shape is concatenation of all transforms on path from root node to leaf
  - Changing a group’s transform affects all parts
    - Allows high level editing by changing only one node
    - E.g. raising left arm requires changing only one transform for that group
Skeleton - hierarchical representation

```plaintext
translate(0, 10);
drawTorso();
pushmatrix(); // push a copy of transform onto stack
    translate(0, 5); // right-multiply onto current transform
    rotate(headRotation); // right-multiply onto current transform
drawHead();
popmatrix(); // pop current transform off stack
pushmatrix();
    translate(-2, 3);
    rotate(rightShoulderRotation);
drawUpperArm();
pushmatrix();
    translate(0, -3);
    rotate(elbowRotation);
drawLowerArm();
pushmatrix();
    translate(0, -3);
    rotate(wristRotation);
drawHand();
popmatrix();
popmatrix();
popmatrix();
....
```
translate(0, 10);

drawTorso();

pushmatrix();  // push a copy of transform onto stack
translate(0, 5);  // right-multiply onto current transform
rotate(headRotation);  // right-multiply onto current transform
drawHead();

popmatrix();  // pop current transform off stack

pushmatrix();                   
translate(-2, 3);
rotate(rightShoulderRotation);
drawUpperArm();

pushmatrix();                  
translate(0, -3);
rotate(elbowRotation);
drawLowerArm();

pushmatrix();                 
translate(0, -3);
rotate(wristRotation);
drawHand();

popmatrix();  

popmatrix();  

popmatrix();  

....
Basic perspective projection

Desired perspective projected result (2D point):
\[
p_{2D} = \begin{bmatrix} x_x / x_z & x_y / x_z \end{bmatrix}^T
\]

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

Input: point in 3D-H
\[
x = \begin{bmatrix} x_x & x_y & x_z & 1 \end{bmatrix}
\]

After applying \( P \): point in 3D-H
\[
P x = \begin{bmatrix} x_x & x_y & x_z & x_z \end{bmatrix}^T
\]

After homogeneous divide:
\[
\begin{bmatrix}
x_x / x_z & x_y / x_z & 1
\end{bmatrix}^T
\]

(throw out third component)

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

More about perspective next lecture!

Stanford CS 248, Winter 2019
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., translation, perspective projection) as matrix operations (linear transforms) in higher-dimensional space
  - Matrix representation affords simple implementation and efficient composition