Transformations

Computer Graphics
CMU 15-462/15-662
The Rasterization Pipeline

Rough sketch of rasterization pipeline:

1. Transform/position objects in the world
2. Project objects onto the screen
3. Sample triangle coverage
4. Sample texture maps / evaluate shaders
5. Interpolate triangle attributes at covered samples
6. Combine samples into final image (depth, alpha, …)
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
Transformations in Rigging
Transformations in Instancing
Basic idea: $f$ transforms $x$ to $f(x)$
What can we do with *linear* transformations?

- (What did *linear* mean?)

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

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

- Cheap to compute
- Composition of linear transformations is linear
  - Leads to uniform representation of transformations
  - E.g., in graphics card (GPU) or graphics APIs
Scale

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

Non-uniform scale??

Non-uniform scale??
Is scale a linear transform?

Yes!

\[
\begin{align*}
S_2(x) &= 2x \\
\alpha S_2(x) &= 2ax \\
S_2(ax) &= 2ax \\
S_2(ax) &= \alpha S_2(x) \\
S_2(x + y) &= 2(x + y) \\
S_2(x) + S_2(y) &= 2x + 2y \\
S_2(x + y) &= S_2(x) + S_2(y)
\end{align*}
\]
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: \[ |R_\theta(x)| = |x| \]
Is rotation linear?

Yes!
Translation

\[ T_b \quad \text{— “translate by } b \text{”} \]

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

No. Translation is affine.
Reflection

Reflecting a point $(a, b)$ across the y-axis:

$$Re_y(a, b) = (a, -b)$$

Reflecting a point $(a, b)$ across the x-axis:

$$Re_x(a, b) = (-a, b)$$
Shear (in $x$ direction)
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 4: 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 $\mathbf{e}_2$

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

$x = [2 \ 2]$

$x = [0.5 \ 1] \quad \text{in coordinate space defined by } \mathbf{e}_1 \text{ and } \mathbf{e}_2, \text{ with origin at } (1.5, 1)$

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

\[
\begin{bmatrix}
a & b \\
c & d
\end{bmatrix}
\begin{bmatrix}
x \\
y
\end{bmatrix} =
\]

\[
x \begin{bmatrix}
a \\
c
\end{bmatrix} + y \begin{bmatrix}
b \\
d
\end{bmatrix} =
\]

\[
\begin{bmatrix}
ax + by \\
cx + dy
\end{bmatrix}
\]

- Matrix multiplication is linear combination of columns
- Encodes a linear map!
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:
\[ s = \begin{bmatrix} 0.5 & 2 \end{bmatrix}^T \]

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

Arbitrary shear:
\[ H_{st} = \begin{bmatrix} 1 & s \\ t & 1 \end{bmatrix} \]

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 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

\[
\begin{align*}
  x_{\text{out}x} &= x_x + b_x \\
  x_{\text{out}y} &= x_y + b_y
\end{align*}
\]
2D homogeneous coordinates (2D-H)

Interesting 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 as 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 & Rotation in 2D-H Coords

- For transformations that are already linear, not much changes:

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

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

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

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

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

(Will get more interesting when we talk about perspective…)
Translation in 2D homogeneous coordinates

Translation expressed as 3x3 matrix multiplication:

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

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

(remember: 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
\( x \) 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

\[
T_{bx} = \begin{bmatrix}
1 & 0 & b_x \\
0 & 1 & b_y \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
w x_x \\
w x_y \\
w
\end{bmatrix}
= \begin{bmatrix}
w x_x + w b_x \\
w x_y + w b_y \\
w
\end{bmatrix}
\]
Homogeneous coordinates: points vs. vectors

2D-H points with \( \omega = 0 \) 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:

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

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:

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

Much more about rotations in next lecture!
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 change of coordinates: 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

Normalized coordinate space:

Screen (W x H output image) coordinate space:

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):

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

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

Input: point in 3D-H

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

After homogeneous divide:

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

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

\[ \begin{bmatrix} x_x / x_z \\ x_y / x_z \\ 1 \end{bmatrix}^T \]

(throw out third component)

Assumption:

Pinhole camera at (0,0) looking down z

Much more about perspective in later lecture!
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