Topic 6:

3D Transformations

- Homogeneous 3D transformations
- Scene Hierarchies
- Change of basis and rotations in 3D
Showtime:
Logistics

• Assignment 1 Due Tomorrow
• Assignment 2 available today/tomorrow
• For assignment questions use the bulletin board or email:
  • csc418tas@cs.toronto.edu
• “When will your slides be online ?”
  • Today 😊
Representing 2D transforms as a 3x3 matrix

Translate a point \([x \ y]^T\) by \([t_x \ t_y]^T\):

\[
\begin{pmatrix}
  x' \\
  y' \\
  1
\end{pmatrix} =
\begin{pmatrix}
  1 & 0 & t_x \\
  0 & 1 & t_y \\
  0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
  x \\
  y \\
  1
\end{pmatrix}
\]

Rotate a point \([x \ y]^T\) by an angle \(t\):

\[
\begin{pmatrix}
  x' \\
  y' \\
  1
\end{pmatrix} =
\begin{pmatrix}
  \cos t & -\sin t & 0 \\
  \sin t & \cos t & 0 \\
  0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
  x \\
  y \\
  1
\end{pmatrix}
\]

Scale a point \([x \ y]^T\) by a factor \([s_x \ s_y]^T\)

\[
\begin{pmatrix}
  x' \\
  y' \\
  1
\end{pmatrix} =
\begin{pmatrix}
  s_x & 0 & 0 \\
  0 & s_y & 0 \\
  0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
  x \\
  y \\
  1
\end{pmatrix}
\]
Representing 3D transforms as a 4x4 matrix

Translate a point \([x \ y \ z]^T\) by \([t_x \ t_y \ t_z]^T\):

\[
\begin{pmatrix}
    x' \\
y' \\
z' \\
1
\end{pmatrix} =
\begin{pmatrix}
    1 & 0 & 0 & t_x \\
    0 & 1 & 0 & t_y \\
    0 & 0 & 1 & t_z \\
    0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
    x \\
y \\
z \\
1
\end{pmatrix}
\]

Rotate a point \([x \ y \ z]^T\) by an angle \(t\) around z axis:

\[
\begin{pmatrix}
    x' \\
y' \\
z' \\
1
\end{pmatrix} =
\begin{pmatrix}
    \cos t & -\sin t & 0 & 0 \\
    \sin t & \cos t & 0 & 0 \\
    0 & 0 & 1 & 0 \\
    0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
    x \\
y \\
z \\
1
\end{pmatrix}
\]

Scale a point \([x \ y \ z]^T\) by a factor \([s_x \ s_y \ s_z]^T\)

\[
\begin{pmatrix}
    x' \\
y' \\
z' \\
1
\end{pmatrix} =
\begin{pmatrix}
    s_x & 0 & 0 & 0 \\
    0 & s_y & 0 & 0 \\
    0 & 0 & s_z & 0 \\
    0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
    x \\
y \\
z \\
1
\end{pmatrix}
\]
Elementary Rotations in 3D

Rotation by $\theta$ about $x$ axis:

$$H_x^\theta = \begin{bmatrix} A_x \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & \sin \theta \\ 0 & -\sin \theta & \cos \theta \end{bmatrix}$$

Rotation by $\theta$ about $y$ axis:

$$H_y^\theta = \begin{bmatrix} A_y \end{bmatrix} = \begin{bmatrix} \cos \theta & 0 & -\sin \theta \\ 0 & 1 & 0 \\ \sin \theta & 0 & \cos \theta \end{bmatrix}$$

Rotation by $\theta$ about $z$ axis:

$$H_z^\theta = \begin{bmatrix} A_z \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
Rotation About Arbitrary Vector?

Question: How do we define $A$ when it is a rotation about an arbitrary vector $\vec{v}$?
Rotation About Arbitrary Vector?

Question: How do we define \( A \) when it is a rotation about an arbitrary vector \( \mathbf{v} \)?

Answer: Express it as a composition of the three elementary matrices \( A_x, A_y, A_z \).

Rotation by \( \theta \) about \( x \) axis:
\[
H_x^\theta = \begin{bmatrix} A_x & 0 \\ 0 & 1 \end{bmatrix} A_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{bmatrix}
\]

Rotation by \( \theta \) about \( y \) axis:
\[
H_y^\theta = \begin{bmatrix} A_y & 0 \\ 0 & 1 \end{bmatrix} A_y = \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix}
\]

Rotation by \( \theta \) about \( z \) axis:
\[
H_z^\theta = \begin{bmatrix} A_z & 0 \\ 0 & 1 \end{bmatrix} A_z = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}
\]
Rotation About Arbitrary Vector: Construction

Question: How do we define $A$ when it is a rotation of $\Phi$ about an arbitrary vector $\vec{v}$?

Answer: Express it as a composition of the three elementary matrices $A_x, A_y, A_z$

Basic Idea: Since we know how to do rotations about $\hat{z}$, we will do the following:

* Align $\vec{v}$ with the $\hat{z}$ axis "temporarily" (using axis-aligned rotations)
* Rotate about $\vec{v}$ using $A_z$
* Undo the temporary alignment
Scene Hierarchies
Change of reference frame/basis matrix

\[ \mathbf{p} = a \mathbf{p}_x' + b \mathbf{p}_y' + c \mathbf{p}_z' + \mathbf{o} \]

\[ \mathbf{p} = \begin{bmatrix} a & b & c & o \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{p}' \]

\[ \mathbf{p}' = \begin{bmatrix} a & b & c & o \end{bmatrix}^{-1} \begin{bmatrix} a & b & c & o \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{p} \]
After translation to align origins
\[ \vec{p} = (x, y, z) \]

A) In Euclidean Basis
\[ \vec{p} = x \left[ \begin{array}{c} 1 \\ 0 \\ 0 \end{array} \right] + y \left[ \begin{array}{c} 0 \\ 1 \\ 0 \end{array} \right] + z \left[ \begin{array}{c} 0 \\ 0 \\ 1 \end{array} \right] \]

But in \( \hat{\omega}\hat{u}\hat{v} \) space
\[ \vec{p} = a \hat{\omega} + b \hat{u} + c \hat{v} \]

This defines a system of equations

B) \( a \hat{\omega} + b \hat{u} + c \hat{v} = x \hat{e}_1 + y \hat{e}_2 + z \hat{e}_3 \)

Exploit orthogonality:
\[ a = x \hat{\omega}^T \hat{e}_1 + y \hat{\omega}^T \hat{e}_2 + z \hat{\omega}^T \hat{e}_3 \]
\[ b = \ldots \]
\[ c = \ldots \]

\[ \Rightarrow \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} \hat{\omega}^T \hat{e}_1 \\ \hat{\omega}^T \hat{e}_2 \\ \hat{\omega}^T \hat{e}_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \]

\[ \cos(\theta_{\hat{e}_3}) \]

Transformation between frames
Viewing Pipeline

object → modeling transform → world → viewing transform → camera → projection transform → screen → viewport transform → cannonical 2D → cartesianize

- perspective divide
- cannonical view vol. 4D
Topic 7:

3D Viewing

- Camera Model
- Orthographic projection
- The world-to-camera transformation
- Perspective projection
- The transformation chain for 3D viewing
"Camera obscura"
(principle known since Aristotle ~ 350BC,
first built ~ 1000 AD
by Ibn al-Haitham)

projection of world point

pinhole

world point

www.lensculture.com/morell.html
Tim’s Vermeer
A Penn & Teller Film
The Pinhole Camera

- viewing plane
- plane center (origin)
- projection of world point
- "pinhole" aperture
- optical axis
- world point
- world point
Camera model

Ideal pinhole camera

Real pinhole camera
Camera model

Real pinhole camera

Camera with a lens
Camera model

Camera with a lens

aperture

object

Depth of Field
We will only consider the idealized pinhole model here (a.k.a. perspective projection).

Simplification #1: Take plane-to-pinhole distance = f
Simplification #2: "Undo" image reversal by placing viewing plane in front of pinhole.
Camera model
Viewing Pipeline

- Object
  - Modeling transform
  - World
  - Viewing transform
  - Camera
  - Projection transform
  - Canonical view vol. 4D
  - Screen
  - Viewport transform
  - Canonical 2D
  - Cartesianize
  - Perspective divide
Viewing Transform
Camera model
Viewing Transform

\[ w = \frac{P_{\text{eye}} - P_{\text{ref}}}{||P_{\text{eye}} - P_{\text{ref}}||} \]

\[ V_{\text{up}} \]

\[ P_{\text{eye}} \]

\[ P_{\text{ref}} \]
Camera model
Viewing Transform

\[ u = \frac{(V_{\text{up}} \times w)}{|| V_{\text{up}} \times w ||} \]
Viewing Transform
After translation to align origins

\( \vec{P} = (x, y, z) \)

A) In Euclidean Basis

\[
\vec{P} = x \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + y \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + z \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}
\]

But in \( \omega \vec{w} \) space

\[\vec{\tilde{P}} = a \vec{\omega} + b \vec{w} + c \vec{u}\]

This defines a system of equations

B) \( a \vec{\omega} + b \vec{w} + c \vec{u} = x \vec{e}_1 + y \vec{e}_2 + z \vec{e}_3 \)

Exploit orthogonality:

\[
\begin{align*}
a &= x \omega^T \vec{e}_1 + y \omega^T \vec{e}_2 + z \omega^T \vec{e}_3 \\
b &= \vdots \\
c &= \vdots \\
\end{align*}
\]

\[\Rightarrow \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} \omega^T \vec{e}_1 & \omega^T \vec{e}_2 & \omega^T \vec{e}_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \]

\[\text{cos} (\vec{\omega} \vec{e}_3)\]

Transformation Between Frames
Change-of-basis Matrix

\[ M_{camera} = \begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \]
Change-of-basis Matrix

\[ M_{\text{camera}} = \begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -\text{eye}_{x} \\ 0 & 1 & 0 & -\text{eye}_{y} \\ 0 & 0 & 1 & -\text{eye}_{z} \\ 0 & 0 & 0 & 1 \end{pmatrix} \]
Viewing Pipeline

- **Modeling transform**
- **Viewing transform**
- **Projection transform**
- **Viewport transform**
- **Cartesianize**

- **$M_{camera}$**
- **Canonical view vol. 4D**
- **Perspective divide**

- **Object** → **World** → **Camera**
Camera model

What is the difference between these images?
What is the difference between these images?
object-to-camera distance affects an object's projection. In real photos, parallel lines do converge in the distance!
Parallel vs Perspective projection

- Parallel: usage in mechanical and architectural drawings
- Perspective projection: more natural and realistic

- How to get 3D objects perspectively correct on 2D screen?
- Note: usually your API takes care of most of this, but it’s good to know what’s going on behind those function calls (esp. when debugging your code)
Orthographic projection

\[ p' = [x \ y \ 1]^T \]

\[ p = [x \ y \ z \ 1]^T \]

\[ P_{ortho} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \]
Orthographic projection

Is $|p-q| = |p'-q'|$ ?

If $m = (p+q)/2$,
Is $m' = (p'+q')/2$?

$p = [x\ y\ z\ 1]^T$

$p' = [x\ y\ 1]^T$

$q'$

$m'$

$m$
Parallel vs Perspective projection

- Parallel: usage in mechanical and architectural drawings
- Perspective projection: more natural and realistic

How to get 3D objects perspectively correct on 2D screen?
Note: usually your API takes care of most of this, but it’s good to know what’s going on behind those function calls (esp. when debugging your code)
Camera model

Perspective Projection
Perspective projection
The view frustum (aka view volume) specifies everything that the camera can see. It’s defined by:

- the **left plane** $l$
- the **right plane** $r$
- the **top plane** $t$
- the **bottom plane** $b$
- the **near plane** $n$
- the **far plane** $f$

For now, we assume *wireframe models* that are completely within the view frustum.
Simple Perspective

\[
P(P(x,y,z)) = P'(x',y',z') \text{ at } (0,0,d)
\]
Simple Perspective

$$P_2 = \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix}$$

$$P_1 = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}$$
\[ P_{persp} = \begin{pmatrix} \frac{x}{w} \\ \frac{y}{w} \\ z \\ w \\ 1 \end{pmatrix} \]

\[ w = \frac{z}{d} \]
Using Homogenous Coordinates

With homogeneous coordinates, the vector

\[(x, y, z, 1)\] represents the point \((x, y, z)\).

Now we extend this in a way that the homogeneous vector

\[(x, y, z, w)\] represents the point \((x/w, y/w, z/w)\).

And matrix transformation becomes:

\[
\begin{pmatrix}
\tilde{x} \\
\tilde{y} \\
\tilde{z} \\
\tilde{w}
\end{pmatrix} = \begin{pmatrix}
a_1 & b_1 & c_1 & d_1 \\
a_2 & b_2 & c_2 & d_2 \\
a_3 & b_3 & c_3 & d_3 \\
e & f & g & h
\end{pmatrix} \begin{pmatrix}
x \\
y \\
z \\
1
\end{pmatrix}
\]
\[ P_{persp} = \begin{pmatrix} \frac{x}{w} \\ \frac{y}{w} \\ \frac{z}{w} \\ 1 \end{pmatrix} \]

\[ w = \frac{z}{d} \]
\[ P_{\text{persp}} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{d} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \]
Homogenous Coordinate after Orthographic Projection

\[ M_{ortho} \]
\[
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1
\end{pmatrix}
\]

Homogenous Coordinate after Perspective Projection

\[ M_{persp} \]
\[
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & \frac{1}{d} & 0 & 0 \\
0 & 0 & 0 & 0 & 1
\end{pmatrix}
\]
Viewing volumes

Projected image
We have to transform the view frustum into the orthographic view volume. The transformation needs to:

- Map lines through the origin to lines parallel to the $z$ axis.
- Map points on the viewing plane to themselves.
- Map points on the far plane to (other) points on the far plane.
- Preserve the near-to-far order of points on a line.
Cannonical view volume

Map 3D to a cube centered at the origin of side length 2!
The orthographic view volume

... how do we get the data from the axis-aligned box \([l, r] \times [b, t] \times [n, f]\) to a \(2 \times 2 \times 2\) box around the origin?
First we need to move the center to the origin:

$$\begin{pmatrix}
1 & 0 & 0 & \frac{l+r}{2} \\
0 & 1 & 0 & \frac{b+t}{2} \\
0 & 0 & 1 & \frac{n+f}{2} \\
0 & 0 & 0 & 1
\end{pmatrix}$$
The orthographic view volume

Then we have to scale everything to $[-1, 1]$:

$$
\begin{pmatrix}
\frac{2}{r-l} & 0 & 0 & 0 & 0 \\
0 & \frac{2}{t-b} & 0 & 0 & 0 \\
0 & 0 & \frac{2}{n-f} & 0 & 0 \\
0 & 0 & 0 & 0 & 1 \\
\end{pmatrix}
$$

Note: we divide by the length, e.g. $\frac{1}{r-l}$ for the $x$-coordinate value
The orthographic view volume

Since these are just matrix multiplications (associative!), we can combine them into one matrix:

\[
M_{orth} = \begin{pmatrix}
\frac{2}{r-l} & 0 & 0 & 0 & -\frac{l+r}{2} \\
0 & \frac{2}{t-b} & 0 & 0 & -\frac{b+t}{2} \\
0 & 0 & \frac{2}{n-f} & 0 & -\frac{n+f}{2} \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1
\end{pmatrix}
\]

\[
= \begin{pmatrix}
\frac{2}{r-l} & 0 & 0 & 0 & -\frac{r+l}{2} \\
0 & \frac{2}{t-b} & 0 & 0 & -\frac{t+b}{2} \\
0 & 0 & \frac{2}{n-f} & 0 & -\frac{n+f}{n-f} \\
0 & 0 & 0 & 0 & 1
\end{pmatrix}
\]
Homogeneous Coords and Perspective

Camera Space

Warped to look like Orthographic Projection
Perspective Matrix

The following matrix will do the trick:

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & \frac{n+f}{n} & -f \\
0 & 0 & \frac{1}{n} & 0
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
z \\
1
\end{pmatrix}
\]

Remember that
- we are looking in negative Z-direction
- \(n, f\) denote the near and far plane of the view frustum
- \(n\) serves as projection plane

Let’s verify that …
Perspective Matrix

\[ M_{persp} = \begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & \frac{1}{n} & -f \\
0 & 0 & \frac{1}{n} & 0
\end{pmatrix} \begin{pmatrix}
x \\
y \\
z \\
1
\end{pmatrix} = \begin{pmatrix}
x \\
y \\
z \frac{n+f}{n} - f \\
z
\end{pmatrix} \text{ homogenize } \begin{pmatrix}
\frac{nx}{n} \\
\frac{ny}{n} \\
\frac{nz}{n} - f \\
1
\end{pmatrix}

Indeed, that gives the correct values for \( x_s \) and \( y_s \).

But what about \( z \)? Remember our requirements for \( z \):

- stays the same for all points on the near and far planes
- does not change the order along the \( Z \)-axis for all other points
Verify

We have \( z_s = n + f - \frac{fn}{z} \) and need to prove that ...

- points on the near plane are mapped to themselves, i.e. if \( z = n \), then \( z_s = n \):
  \[
  z_s = n + f - \frac{fn}{n} = n + f - f = n
  \]
  and obviously \( x_s = \frac{nx}{n} = x \) and \( y_s = \frac{ny}{n} = y \).

- points on the far plane stay on the far plane, i.e. if \( z = f \), then \( z_s = f \):
  \[
  z_s = n + f - \frac{fn}{f} = n + f - n = f
  \]
  and ...
We have $z_s = n + f - \frac{fn}{z}$ and need to prove that ...

- $z$-values for points within the view frustum stay within the view frustum,
i.e. if $z > n$ then $z_s > n$:

$$z_s = n + f - \frac{fn}{z} > n + f - \frac{fn}{n} = n$$

and if $z < f$ then $z_s < f$:

$$z_s = n + f - \frac{fn}{z} < n + f - \frac{fn}{f} = f$$

and ...
We have \( z_s = n + f - \frac{f_n}{z} \) and need to prove that . . .

- the order along the \( Z \)-axis is preserved, i.e. if \( 0 > n \geq z_1 > z_2 \geq f \) then \( z_{1s} > z_{2s} \):

With \( z_{1s} = n + f - \frac{f_n}{z_1} \) and \( z_{2s} = n + f - \frac{f_n}{z_2} \) we get:

\[
z_{1s} - z_{2s} = \frac{f_n}{z_2} - \frac{f_n}{z_1} = \frac{(z_1 - z_2)f_n}{z_1 z_2}.
\]

Because of \( f, z_1, z_2, n < 0 \) we have \( \frac{f_n}{z_1 z_2} > 0 \), and because of \( z_1 > z_2 \), we have \( z_1 - z_2 > 0 \), so

\[
z_{1s} - z_{2s} > 0 \text{ or } z_{1s} > z_{2s}
\]
Viewing Pipeline

\[ M_{camera} \]

1. Modeling transform
2. Viewing transform
3. Projection transform
4. Cartesianize

- Object
- World
- Camera
- Screen
- Canonical view vol. 4D
- Canonical 2D
- Perspective divide

\[
\begin{pmatrix}
\frac{nx}{z} \\
\frac{ny}{z} \\
\frac{z}{n + f - fn/z} \\
1
\end{pmatrix}
\]
Now all that’s left is a parallel projection along the $Z$-axis (every easy) and ...
Viewport Transform

Now all that’s left is a parallel projection along the Z-axis (every easy) and ...

\[
M_{2D} = \begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 \\
\end{pmatrix}
\]
Viewing Pipeline

Object → modeling transform → world → viewing transform → camera → projection transform → screen

\( M_{\text{camera}} \)

\( M_{\text{orth}} \cdot M_{\text{persp}} \)

cartesianize perspective divide

\( M_{2D} \)

\[
\begin{pmatrix}
\frac{nx}{z} \\
\frac{ny}{z} \\
n + f - \frac{fn}{z} \\
1
\end{pmatrix}
\]
...a windowing transformation in order to display the square $[-1, 1]^2$ onto an $n_x \times n_y$ image.

Again, these are just some (simple) matrix multiplications