Today

- Windowing and Viewing Transformations
  - Windows and viewports
  - Orthographic projection
  - Perspective projection
Screen Space

- Monitor has some number of pixels
  - e.g. 1024 x 768
- Some sub-region used for given program
  - You call it a window
  - Let’s call it a viewport instead

![Diagram: Screen Space](image)

Screen Space

- May not really be a “screen”
  - Image file
  - Printer
  - Other
  - Little pixel details
    - Sometimes odd
      - Upside down
      - Hexagonal

From Shirley textbook.
Screen Space

- Viewport is somewhere on screen
  - You probably don’t care where
  - Window System likely manages this detail
  - Sometimes you care exactly where
- Viewport has a size in pixels
  - Sometimes you care (images, text, etc.)
  - Sometimes you don’t (using high-level library)

Canonical View Space

- Canonical view region
  - 2D: [-1,-1] to [+1,+1]

From Shirley textbook.
Canonical View Space

- Canonical view region
  - 2D: [-1,-1] to [+1,+1]

\[
\begin{bmatrix}
    x' \\
y' \\
1
\end{bmatrix} = \begin{bmatrix}
    \frac{n_x}{2} & 0 & \frac{n_x-1}{2} \\
    0 & \frac{n_y}{2} & \frac{n_y-1}{2} \\
    0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
    x \\
y \\
1
\end{bmatrix}
\]

Canonical View Space

- Canonical view region
  - 2D: [-1,-1] to [+1,+1]

- Define arbitrary window and define objects
- Transform window to canonical region
- Do other things (we’ll see clipping latter)
- Transform canonical to screen space
- Draw it.

From Shirley textbook.
### Canonical View Space

- World Coordinates (Meters)
- Canonical
- Screen Space (Pixels)

> Note distortion issues...

### Projection

- Process of going from 3D to 2D
- Studies throughout history (e.g. painters)
- Different types of projection
  - Linear
    - Orthographic
    - Perspective
  - Nonlinear

> Many special cases in books just one of these two...

> Orthographic is special case of perspective...
Linear Projection

- Projection onto a planar surface
- Projection directions either
  - Converge to a point
  - Are parallel (converge at infinity)

Linear Projection

- A 2D view

Perspective
Orthographic
Linear Projection

Orthographic  Perspective

Linear Projection

Orthographic  Perspective
Linear Projection

- A 2D view

Note how different things can be seen

Parallel lines “meet” at infinity

Orthographic Projection

- No foreshortening
- Parallel lines stay parallel
- Poor depth cues
Canonical View Space

- Canonical view region
  - 3D: $[-1,-1,-1]$ to $[+1,+1,+1]$
- Assume looking down $-Z$ axis
  - Recall that “Z is in your face”

Orthographic Projection

- Convert arbitrary view volume to canonical
Orthographic Projection

Step 1: translate center to origin

*Assume up is perpendicular to view.
Orthographic Projection

- Step 1: translate center to origin
- Step 2: rotate view to $-Z$ and up to $+Y$
- Step 3: center view volume
Orthographic Projection

- Step 1: translate center to origin
- Step 2: rotate view to -$Z$ and up to $+Y$
- Step 3: center view volume
- Step 4: scale to canonical size

\[
\begin{align*}
M &= S \cdot T_2 \cdot R \cdot T_1 \\
M &= M_o \cdot M_v
\end{align*}
\]
Perspective Projection

- Foreshortening: further objects appear smaller
- Some parallel line stay parallel, most don’t
- Lines still look like lines
- $Z$ ordering preserved (where we care)

Pinhole a.k.a center of projection
Foreshortening: distant objects appear smaller

Vanishing points
- Depend on the scene
- Not intrinsic to camera

"One point perspective"
Perspective Projection

- Vanishing points
  - Depend on the scene
  - Not intrinsic to camera

“Two point perspective”

Perspective Projection

- Vanishing points
  - Depend on the scene
  - Not intrinsic to camera

“Three point perspective”
Perspective Projection

View Frustum

Distance to image plane $i$

Near $n$

Far $f$

Center

Top $t$

Bottom $b$

Up

View

Distance to image plane $i$
Perspective Projection

- Step 1: Translate center to origin
- Step 2: Rotate view to -Z, up to +Y
Perspective Projection

- Step 1: Translate center to origin
- Step 2: Rotate view to \(-Z, \text{ up to } +Y\)
- Step 3: Shear center-line to \(-Z\) axis

Perspective Projection

- Step 1: Translate center to origin
- Step 2: Rotate view to \(-Z, \text{ up to } +Y\)
- Step 3: Shear center-line to \(-Z\) axis
- Step 4: Perspective

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & \frac{i + f}{i} & f \\
0 & 0 & \frac{-1}{i} & 0
\end{bmatrix}
\]
Perspective Projection

- Step 4: Perspective
  - Points at $z=-i$ stay at $z=-i$
  - Points at $z=-f$ stay at $z=-f$
  - Points at $z=0$ goto $z=\pm \infty$
  - Points at $z=-\infty$ goto $z=-(i+f)$

- $x$ and $y$ values divided by $-\frac{z}{i}$

- Straight lines stay straight
- Depth ordering preserved in $[-i,-f]$  
- Movement along lines distorted

$$
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & \frac{i+f}{i} & f \\
0 & 0 & -\frac{1}{i} & 0 \\
\end{bmatrix}
$$

From Shirley textbook.

Perspective Projection
Perspective Projection

"Eye" plane

Top

Near | Far

Some horizontal lines

View vector

Visualizing division of $x$ and $y$ but not $z$
Perspective Projection

Motion in $x, y$

Note that points on near plane fixed
Perspective Projection

Recall that points on far plane will stay there...

When we also divide $z$ points must remain on straight lines
Perspective Projection

Lines extend outside view volume

Perspective Projection

Motion in \( \hat{z} \)
Perspective Projection

Motion in $z$
Perspective Projection

Step 1: Translate center to orange
Step 2: Rotate view to -Z, up to +Y
Step 3: Shear center-line to -Z axis
Step 4: Perspective
Step 5: Center view volume
Step 6: Scale to canonical size
Perspective Projection

- Step 1: Translate center to orange
- Step 2: Rotate view to -Z, up to +Y
- Step 3: Shear center-line to -Z axis
- Step 4: Perspective
- Step 5: center view volume
- Step 6: scale to canonical size

\[ M = M_o \cdot M_p \cdot M_v \]

There are other ways to set up the projection matrix
- View plane at z=0 zero
- Looking down another axis
- etc...
- Functionally equivalent
Vanishing Points

- Consider a ray:

\[ r(t) = p + t \mathbf{d} \]

- Ignore \( Z \) part of matrix 
- \( X \) and \( Y \) will give location in image plane  
- Assume image plane at \( z = -i \)

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
\text{whatever} & & & \\
0 & 0 & -1 & 0
\end{bmatrix}
\rightarrow
\begin{bmatrix}
I_x \\
I_y \\
I_w
\end{bmatrix} =
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & -1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix}
\]
Vanishing Points

\[
\begin{bmatrix}
I_x \\
I_y \\
I_w
\end{bmatrix} = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & -1
\end{bmatrix}\begin{bmatrix}
x \\
y \\
z
\end{bmatrix} = \begin{bmatrix}
x \\
y \\
-z
\end{bmatrix}
\]

\[
\begin{bmatrix}
I_x / I_w \\
I_y / I_w
\end{bmatrix} = \begin{bmatrix}
-x / z \\
-y / z
\end{bmatrix}
\]

Vanishing Points

- Assume \( dz = -1 \)

\[
\begin{bmatrix}
I_x / I_w \\
I_y / I_w
\end{bmatrix} = \begin{bmatrix}
-x / z \\
-y / z
\end{bmatrix} = \begin{bmatrix}
\frac{p_x + td_x}{-p_z + t} \\
\frac{p_y + td_y}{-p_z + t}
\end{bmatrix}
\]

\[
\text{Lim}_{t \to \pm \infty} = \begin{bmatrix}
dx \\
\ dy
\end{bmatrix}
\]
Vanishing Points

\[
\lim_{t \to \pm \infty} d = \begin{bmatrix} d_x \\ d_y \end{bmatrix}
\]

- All lines in direction \( d \) converge to same point in the image plane -- the vanishing point
- Every point in plane is a v.p. for some set of lines
- Lines parallel to image plane (\( d_z = 0 \)) vanish at infinity

What’s a horizon?

Ray Picking

- Pick object by picking point on screen
- Compute ray from pixel coordinates.

- Compute ray from pixel coordinates.
Ray Picking

- Transform from World to Screen is:

\[
\begin{bmatrix}
I_x \\
I_y \\
I_z \\
I_w
\end{bmatrix} = M \begin{bmatrix}
W_x \\
W_y \\
W_z \\
W_w
\end{bmatrix}
\]

- Inverse:

\[
\begin{bmatrix}
W_x \\
W_y \\
W_z \\
W_w
\end{bmatrix} = M^{-1} \begin{bmatrix}
I_x \\
I_y \\
I_z \\
I_w
\end{bmatrix}
\]

- What Z value?

Ray Picking

- Recall that:
  - Points at \( z=-i \) stay at \( z=-i \)
  - Points at \( z=-f \) stay at \( z=-f \)

\[r(t) = p + t \, d\]
\[r(t) = a_w + t(b_w - a_w)\]

\[
a_s = [s_x, s_y, -i]
\]
\[
b_s = [s_x, s_y, -f]
\]

Depends on screen details, YMMV

General idea should translate...
Suggested Reading

- Fundamentals of Computer Graphics by Pete Shirley
  - Chapter 6