Today

- 2D Transformations
  - “Primitive” Operations
    - Scale, Rotate, Shear, Flip, Translate
  - Homogenous Coordinates
  - SVD
  - Start thinking about rotations...
Introduction

- **Transformation:**
  An operation that changes one configuration into another

- **For images, shapes, etc.**
  A geometric transformation maps positions that define the object to other positions

  Linear transformation means the transformation is defined by a linear function... which is what matrices are good for.
Some Examples

Images from *Conan The Destroyer*, 1984
Mapping Function

\[ f(x) = x \text{ in old image} \]

\[ c(x) = [195, 120, 58] \]

\[ c' x = c(f(x)) \]
Linear -vs- Nonlinear

Linear (shear)

Nonlinear (swirl)
Geometric -vs- Color Space

Color Space Transform
(edge finding)

Linear Geometric
(flip)
Instancing

M.C. Escher, from Ghostscript 8.0 Distribution
Instancing

- Reuse geometric descriptions
- Saves memory
Linear is Linear

- Polygons defined by points
- Edges defined by interpolation between two points
- Interior defined by interpolation between all points
- *Linear* interpolation
Linear is Linear

- Composing two linear function is still linear
- Transform polygon by transforming vertices
Linear is Linear

- Composing two linear functions is still linear
- Transform polygon by transforming vertices

\[ f(x) = a + bx \quad g(f) = c + df \]

\[ g(x) = c + df \quad f(x) = c + ad + bdx \]

\[ g(x) = a' + b'x \]
Points in Space

- Represent point in space by vector in $\mathbb{R}^n$
  - Relative to some origin!
  - Relative to some coordinate axes!
- Later we’ll add something extra...

$\mathbf{p} = [4, 2]^T$
Basic Transformations

- Basic transforms are: rotate, scale, and translate
- Shear is a composite transformation!
Linear Functions in 2D

\[ x' = f(x, y) = c_1 + c_2x + c_3y \]
\[ y' = f(x, y) = d_1 + d_2x + d_3y \]

\[
\begin{bmatrix}
x'
y'
\end{bmatrix} = \begin{bmatrix} t_x \\ t_y \end{bmatrix} + \begin{bmatrix} M_{xx} & M_{xy} \\ M_{yx} & M_{yy} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}
\]

\[ x' = t + M \cdot x \]
Rotations

\[ p' = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} p \]

Rotate

45 degree rotation

\[
\begin{bmatrix}
\begin{array}{c}
.707 \\
-.707
\end{array}
\end{bmatrix}
\begin{bmatrix}
\begin{array}{c}
.707 \\
.707
\end{array}
\end{bmatrix}
\]
Rotations

- Rotations are positive counter-clockwise
- Consistent w/ right-hand rule
- Don’t be different...

Note:
- rotate by zero degrees give identity
- rotations are modulo 360 (or $2\pi$)
Rotations

- Preserve lengths and distance to origin
- Rotation matrices are orthonormal
- $\det(R) = 1 \neq -1$
- In 2D rotations commute...
  - But in 3D they won’t!
Scales

Uniform/isotropic

Non-uniform/anisotropic

\[
p' = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} p
\]
Scales

- Diagonal matrices
  - Diagonal parts are scale in X and scale in Y directions
  - Negative values flip
  - Two negatives make a positive (180 deg. rotation)
  - Really, axis-aligned scales
Shears

\[ p' = \begin{bmatrix} 1 & H_{yx} \\ H_{xy} & 1 \end{bmatrix} p \]
Shears

- Shears are not really primitive transforms
- Related to non-axis-aligned scales
- More shortly.....
Translation

- This is the not-so-useful way:

\[ p' = p + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \]

Translate

Note that its not like the others.
Arbitrary Matrices

- For everything but translations we have:
  \[ x' = A \cdot x \]

- Soon, translations will be assimilated as well

- What does an arbitrary matrix mean?
Singular Value Decomposition

For any matrix, $A$, we can write SVD:

$$A = QR^T$$

where $Q$ and $R$ are orthonormal and $S$ is diagonal

Can also write Polar Decomposition

$$A = QR SR^T$$

where $Q$ is still orthonormal

not the same $Q$
Decomposing Matrices

- We can force $Q$ and $R$ to have $\text{Det}=1$ so they are rotations.
- Any matrix is now:
  - Rotation:Rotation:Scale:Rotation
  - See, shear is just a mix of rotations and scales.
Composition

- Matrix multiplication composites matrices

\[ p' = BA p \]

“Apply A to p and then apply B to that result.”

\[ p' = B(Ap) = (BA)p = C_p \]

- Several translations composted to one

- Translations still left out...

\[ p' = B(Ap + t) = \Box A_p + Bt = C_p + u \]
Composition

Transformations built up from others

SVD builds from scale and rotations

Also build other ways

i.e. 45 deg rotation built from shears
Homogeneous Coordinates

- Move to one higher dimensional space
  - Append a 1 at the end of the vectors

\[
\mathbf{p} = \begin{bmatrix} p_x \\ p_y \end{bmatrix} \quad \tilde{\mathbf{p}} = \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix}
\]

- For directions the extra coordinate is a zero
Homogeneous Translation

\[
\tilde{p}' = \begin{bmatrix}
1 & 0 & t_x \\
0 & 1 & t_y \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
p_x \\
p_y \\
1
\end{bmatrix}
\]

\[\tilde{p}' = \tilde{A}\tilde{p}\]

The tildes are for clarity to distinguish homogenized from non-homogenized vectors.
Homogeneous Others

Now everything looks the same...
Hence the term “homogenized!”
Compositing Matrices

- Rotations and scales always about the origin
- How to rotate/scale about another point?
Rotate About Arb. Point

- Step 1: Translate point to origin

Translate \((-C)\)
Rotate About Arb. Point

- Step 1: Translate point to origin
- Step 2: Rotate as desired

Translate (-C)
Rotate (θ)
Rotate About Arb. Point

- Step 1: Translate point to origin
- Step 2: Rotate as desired
- Step 3: Put back where it was

\[ \tilde{p}' = (-T)RT\tilde{p} = A\tilde{p} \]

Don’t negate the 1...
Scale About Arb. Axis

- Diagonal matrices scale about coordinate axes only:

Not axis-aligned
Scale About Arb. Axis

- Step 1: Translate axis to origin
Scale About Arb. Axis

- Step 1: Translate axis to origin
- Step 2: Rotate axis to align with one of the coordinate axes
Scale About Arb. Axis

- Step 1: Translate axis to origin
- Step 2: Rotate axis to align with one of the coordinate axes
- Step 3: Scale as desired
Scale About Arb. Axis

- Step 1: Translate axis to origin
- Step 2: Rotate axis to align with one of the coordinate axes
- Step 3: Scale as desired
- Steps 4&5: Undo 2 and 1 (reverse order)
Order Matters!

- The order that matrices appear in matters
  \[ A \cdot B \neq BA \]
- Some special cases work, but they are special
- But matrices are associative
  \[ (A \cdot B) \cdot C = A \cdot (B \cdot C) \]
- Think about efficiency when you have many points to transform...
Matrix Inverses

- In general: $A^{-1}$ undoes effect of $A$

- Special cases:
  - Translation: negate $t_x$ and $t_y$
  - Rotation: transpose
  - Scale: invert diagonal (axis-aligned scales)

- Others:
  - Invert matrix
  - Invert SVD matrices
Point Vectors / Direction Vectors

- Points in space have a 1 for the “w” coordinate

- What should we have for $\mathbf{a} - \mathbf{b}$?
  - $w = 0$

- Directions not the same as positions
- Difference of positions is a direction
- Position + direction is a position
- Direction + direction is a direction
- Position + position is nonsense
For example normals do not transform normally

$$M(a \times b) \neq (Ma) \times (Mb)$$

Use inverse transpose of the matrix for normals. See text book.
Suggested Reading

- Fundamentals of Computer Graphics by Pete Shirley
  - Chapter 5
  - And re-read chapter 4 if your linear algebra is rusty!
CS-184: Computer Graphics

Lecture #5: 3D Transformations and Rotations

Prof. James O’Brien
University of California, Berkeley
Today

- Transformations in 3D
- Rotations
  - Matrices
  - Euler angles
  - Exponential maps
  - Quaternions
3D Transformations

- Generally, the extension from 2D to 3D is straightforward
  - Vectors get longer by one
  - Matrices get extra column and row
  - SVD still works the same way
  - Scale, Translation, and Shear all basically the same
- Rotations get interesting
Translations

\[ \tilde{A} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \quad \text{For 2D} \]

\[ \tilde{A} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{For 3D} \]
\[ \tilde{A} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \]  

For 2D

\[ \tilde{A} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]  

For 3D

(Axis-aligned!)
Shears

\[ \tilde{\mathbf{A}} = \begin{bmatrix} 1 & h_{xy} & 0 \\ h_{yx} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \]

For 2D

\[ \mathbf{A} = \begin{bmatrix} 1 & h_{xy} & h_{xz} & 0 \\ h_{yx} & 1 & h_{yz} & 0 \\ h_{zx} & h_{zy} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

For 3D

(Axis-aligned!)
Shears

\[ \tilde{A} = \begin{bmatrix}
1 & h_{xy} & h_{xz} & 0 \\
 h_{yx} & 1 & h_{yz} & 0 \\
 h_{zx} & h_{zy} & 1 & 0 \\
 0 & 0 & 0 & 1 \\
\end{bmatrix} \]

Shears \( y \) into \( x \)
Rotations

- 3D Rotations fundamentally more complex than in 2D
  - 2D: amount of rotation
  - 3D: amount and axis of rotation
Rotations

- Rotations still orthonormal
- $\text{Det}(R) = 1 \neq -1$
- Preserve lengths and distance to origin
- 3D rotations **DO NOT COMMUTE!**
- Right-hand rule
- Unique matrices
Axis-aligned 3D Rotations

- 2D rotations implicitly rotate about a third out of plane axis
Axis-aligned 3D Rotations

- 2D rotations implicitly rotate about a third out of plane axis

\[
R = \begin{bmatrix}
\cos(\theta) & -\sin(\theta) \\
\sin(\theta) & \cos(\theta)
\end{bmatrix}
\]

\[
\hat{R} = \begin{bmatrix}
\cos(\theta) & -\sin(\theta) & 0 \\
\sin(\theta) & \cos(\theta) & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

Note: looks same as \( \hat{R} \)
Axis-aligned 3D Rotations

\[ \mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} \]

\[ \mathbf{R}_y = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \]

\[ \mathbf{R}_z = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \]
Axis-aligned 3D Rotations

\[
\mathbf{R}_x = \begin{bmatrix}
1 & 0 & 0 \\
0 & \cos(\theta) & -\sin(\theta) \\
0 & \sin(\theta) & \cos(\theta)
\end{bmatrix}
\]

\[
\mathbf{R}_y = \begin{bmatrix}
\cos(\theta) & 0 & \sin(\theta) \\
0 & 1 & 0 \\
-\sin(\theta) & 0 & \cos(\theta)
\end{bmatrix}
\]

\[
\mathbf{R}_z = \begin{bmatrix}
\cos(\theta) & -\sin(\theta) & 0 \\
\sin(\theta) & \cos(\theta) & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

Also right handed “Zup”
Axis-aligned 3D Rotations

Also known as “direction-cosine” matrices

\[
R_x = \begin{bmatrix}
1 & 0 & 0 \\
0 & \cos(\theta) & -\sin(\theta) \\
0 & \sin(\theta) & \cos(\theta)
\end{bmatrix} \quad R_y = \begin{bmatrix}
\cos(\theta) & 0 & \sin(\theta) \\
0 & 1 & 0 \\
-\sin(\theta) & 0 & \cos(\theta)
\end{bmatrix} \quad R_z = \begin{bmatrix}
\cos(\theta) & -\sin(\theta) & 0 \\
\sin(\theta) & \cos(\theta) & 0 \\
0 & 0 & 1
\end{bmatrix}
\]
Arbitrary Rotations

- Can be built from axis-aligned matrices:
  \[ \mathbf{R} = \mathbf{R}_z \cdot \mathbf{R}_y \cdot \mathbf{R}_x \]

- Result due to Euler... hence called Euler Angles

- Easy to store in vector
  \[ \mathbf{R} = \text{rot}(x, y, z) \]

- But NOT a vector.
Arbitrary Rotations

\[ \mathbf{R} = \mathbf{R}_{\hat{z}} \cdot \mathbf{R}_{\hat{y}} \cdot \mathbf{R}_{\hat{x}} \]
Arbitrary Rotations

- Allows tumbling
- Euler angles are non-unique
- Gimbal-lock
- Moving -vs- fixed axes
  - Reverse of each other
Exponential Maps

- Direct representation of arbitrary rotation
- AKA: axis-angle, angular displacement vector
- Rotate $\theta$ degrees about some axis
- Encode $\theta$ by length of vector

$$\theta = |\mathbf{r}|$$
Exponential Maps

- Given vector $\mathbf{r}$, how to get matrix $\mathbf{R}$
- Method from text:
  1. rotate about $x$ axis to put $\mathbf{r}$ into the $x$-$y$ plane
  2. rotate about $z$ axis align $\mathbf{r}$ with the $x$ axis
  3. rotate $\theta$ degrees about $x$ axis
  4. undo #2 and then #1
  5. composite together
Vector expressing a point has two parts

- $\mathbf{X}_\parallel$ does not change
- $\mathbf{X}_\perp$ rotates like a 2D point
Exponential Maps

\[ x' = x_{\parallel} + x_{\perp} \sin(\theta) + x_{\perp} \cos(\theta) \]
Exponential Maps

- Rodriguez Formula

\[ x' = \hat{r}(\hat{r} \cdot x) + \sin(\theta)(\hat{r} \times x) - \cos(\theta)(\hat{r} \times (\hat{r} \times x)) \]

Linear in \( x \)

Actually a minor variation ...
Exponential Maps

- Building the matrix

\[ x' = ((\hat{r}\hat{r}^t) + \sin(\theta)(\hat{r} \times) - \cos(\theta)(\hat{r} \times)(\hat{r} \times))x \]

\[ (\hat{r} \times) = \begin{bmatrix} 0 & -\hat{r}_z & \hat{r}_y \\ \hat{r}_z & 0 & -\hat{r}_x \\ -\hat{r}_y & \hat{r}_x & 0 \end{bmatrix} \]

Antisymmetric matrix

\((a \times)b = a \times b\)

Easy to verify by expansion
Exponential Maps

- Allows tumbling
- No gimbal-lock!
- Orientations are space within $\pi$-radius ball
- Nearly unique representation
- Singularities on shells at $2\pi$
- Nice for interpolation
Exponential Maps

- Why exponential?
- Recall series expansion of $e^x$

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$
Exponential Maps

- Why exponential?
- Recall series expansion of $e^x$
- Euler: what happens if you put in $i\theta$ for $x$

$$e^{i\theta} = 1 + \frac{i\theta}{1!} + \frac{-\theta^2}{2!} + \frac{-i\theta^3}{3!} + \frac{\theta^4}{4!} + \cdots$$

$$= \left( 1 + \frac{-\theta^2}{2!} + \frac{\theta^4}{4!} + \cdots \right) + i \left( \frac{\theta}{1!} + \frac{-\theta^3}{3!} + \cdots \right)$$

$$= \cos(\theta) + i\sin(\theta)$$
Exponential Maps

- Why exponential?

\[
e^{(\hat{r} \times) \theta} = \mathbf{I} + \frac{(\hat{r} \times) \theta}{1!} + \frac{(\hat{r} \times)^2 \theta^2}{2!} + \frac{(\hat{r} \times)^3 \theta^3}{3!} + \frac{(\hat{r} \times)^4 \theta^4}{4!} + \ldots
\]

But notice that: \((\hat{r} \times)^3 = -(\hat{r} \times)\)

\[
e^{(\hat{r} \times) \theta} = \mathbf{I} + \frac{(\hat{r} \times) \theta}{1!} + \frac{(\hat{r} \times)^2 \theta^2}{2!} + \frac{-(\hat{r} \times) \theta^3}{3!} + \frac{-(\hat{r} \times)^2 \theta^4}{4!} + \ldots
\]
Exponential Maps

\[ e^{(\hat{r} \times) \theta} = \mathbf{I} + \frac{(\hat{r} \times) \theta}{1!} + \frac{(\hat{r} \times)^2 \theta^2}{2!} + \frac{-(\hat{r} \times) \theta^3}{3!} + \frac{-(\hat{r} \times)^2 \theta^4}{4!} + \cdots \]

\[ e^{(\hat{r} \times) \theta} = (\hat{r} \times) \left( \frac{\theta}{1!} - \frac{\theta^3}{3!} + \cdots \right) + \mathbf{I} + (\hat{r} \times)^2 \left( \frac{\theta^2}{2!} - \frac{\theta^4}{4!} + \cdots \right) \]

\[ e^{(\hat{r} \times) \theta} = (\hat{r} \times) \sin(\theta) + \mathbf{I} + (\hat{r} \times)^2 (1 - \cos(\theta)) \]
Quaternions

- More popular than exponential maps
- Natural extension of $e^{i\theta} = \cos(\theta) + i\sin(\theta)$
- Due to Hamilton (1843)
  - Interesting history
  - Involves “hermaphroditic monsters”
Quaternions

- Uber-Complex Numbers

\[ q = (z_1, z_2, z_3, s) = (z, s) \]

\[ q = i z_1 + j z_2 + k z_3 + s \]

\[ i^2 = j^2 = k^2 = -1 \]

- \( i j = k \) \quad \( j i = -k \)
- \( j k = i \) \quad \( k j = -i \)
- \( k i = j \) \quad \( i k = -j \)
Quaternions

- **Multiplication** natural consequence of defn.
  \[ q \cdot p = (z_q s_p + z_p s_q + z_p \times z_q, \ s_p s_q - z_p \cdot z_q) \]

- **Conjugate**
  \[ q^{*} = (-z, s) \]

- **Magnitude**
  \[ ||q||^2 = z \cdot z + s^2 = q \cdot q^{*} \]
Quaternions

- Vectors as quaternions
  \[ v = (v, 0) \]

- Rotations as quaternions
  \[ r = (\hat{r}\sin\frac{\theta}{2}, \cos\frac{\theta}{2}) \]

- Rotating a vector
  \[ x' = r \cdot x \cdot r^* \]

- Composing rotations
  \[ r = r_1 \cdot r_2 \]

Compare to Exp. Map
Quaternions

- No tumbling
- No gimbal-lock
- Orientations are “double unique”
- Surface of a 3-sphere in 4D $||r|| = 1$
- Nice for interpolation
Interpolation
Rotation Matrices

- Eigen system
  - One real eigenvalue
  - Real axis is axis of rotation
  - Imaginary values are 2D rotation as complex number

- Logarithmic formula

\[
(\hat{\mathbf{r}} \times) = \ln(\mathbf{R}) = \frac{\theta}{2 \sin \theta} (\mathbf{R} - \mathbf{R}^T)
\]

\[
\theta = \cos^{-1}\left(\frac{\text{Tr}(\mathbf{R}) - 1}{2}\right)
\]

Similar formulae as for exponential...
Rotation Matrices

- Consider:

\[
RI = \begin{bmatrix}
  r_{xx} & r_{xy} & r_{xz} \\
  r_{yx} & r_{yy} & r_{yz} \\
  r_{zx} & r_{zy} & r_{zz}
\end{bmatrix} \begin{bmatrix}
  1 & 0 & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 1
\end{bmatrix}
\]

- Columns are coordinate axes after transformation (true for general matrices)

- Rows are original axes in original system (not true for general matrices)
Note:

- Rotation stuff in the book is a bit weak... luckily you have these nice slides!
CS-184: Computer Graphics

Lecture #8: Projection

Prof. James O’Brien
University of California, Berkeley
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
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)
Screen Space

\[ nx-0.5, ny-0.5 \]

Integer Pixel Addresses

\[ i=3 \]

\[ j=5 \]

-0.5,-0.5

10 × 10 Image Resolution
Screen Space

Float Pixel Coordinates

\[ u = 0.35 = \frac{(i + 0.5)}{nx} \]

\[ v = 0.55 = \frac{(j + 0.5)}{ny} \]
Canonical View Space

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

$\begin{align*}
\text{x} &= 0.0, \quad \text{y} = 0.0
\end{align*}$
Canonical View Space

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

From Shirley textbook. (Image coordinates are up-side-down.)

Remove minus for right-side-up
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...
Perspective Projections
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

Perspective

Orthographic
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

- **View vector**
- **Up vector**
- **Right** = view $\times$ up
- **Center**
- **Origin**

*Assume up is perpendicular to view.*
Orthographic Projection

- Step 1: translate center to origin
Orthographic Projection

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

\[ M = S \cdot T_2 \cdot R \cdot T_1 \]
\[ M = M_o \cdot M_v \]
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)
Perspective Projection

Pinhole a.k.a center of projection

Image from D. Forsyth
Perspective Projection

Foreshortening: distant objects appear smaller
Perspective Projection

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

“One point perspective”
Perspective Projection

- Vanishing points
  - Depend on the scene
  - Nor intrinsic to camera

“Two point perspective”
Perspective Projection

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

“Three point perspective”
Perspective Projection
Perspective Projection

- Near $n$
- Far $f$
- Up
- Center
- View
- Distance to image plane $i$
- Top $t$
- Bottom $b$
Perspective Projection

- Step 1: Translate center to origin
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 \(-Z\), *up* to \(+Y\)
- Step 3: Shear center-line to \(-Z\) axis
Perspective Projection

- Step 1: Translate center to origin
- Step 2: Rotate view to \(-Z\), up to \(+Y\)
- Step 3: Shear center-line to \(-Z\) axis
- Step 4: Perspective
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 $-z/i$
- Straight lines stay straight
- Depth ordering preserved in $[-i,-f]$
- Movement along lines distorted

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

**WRONG!**

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

“Eye” plane

Top

Near

Far

Some horizontal lines

View vector

\[ \hat{z} \]
Perspective Projection

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

Motion in $x,y$
Perspective Projection

Note that points on near plane fixed
Perspective Projection

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

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

Lines extend outside view volume
Perspective Projection

Motion in $z$
Perspective Projection

Motion in $z$
Perspective Projection

Motion in $z$
Perspective Projection

Total motion
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

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

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

- Consider a ray:

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

- Ignore \( Z \) part of matrix
- \( \bf{X} \) and \( \bf{Y} \) will give location in image plane
- Assume image plane at \( z = -1 \)

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

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

\[
\text{Lim } t \to \pm \infty = \begin{bmatrix}
d_x \\
d_y
\end{bmatrix}
\]
Vanishing Points

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

- All lines in direction \( \mathbf{d} \) 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?
Perspective Tricks
Right Looks Wrong (Sometimes)

From Correction of Geometric Perceptual Distortions in Pictures, Zorin and Barr SIGGRAPH 1995
Right Looks Wrong (Sometimes)

That iPhone Marketshare Chart: WTF?

- U.S. Smartphone Marketshare
- How the pie chart is distorted
- Let's correct the photo perspective
- Chart perspective fixed
- Here it is overlaid with a pie chart using the same data. There's no funny business here — except for the perspective trick!
Strangeness

The Ambassadors by Hans Holbein the Younger
Ray Picking

- Pick object by picking point on screen

- Compute ray from pixel coordinates.
Ray Picking

- Transform from World to Screen is:

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

- Inverse:

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

- What Z value?
Ray Picking

- **Recall that:**
  - Points at $z=-i$ stay at $z=-i$
  - Points at $z=-f$ stay at $z=-f$

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

\[
\mathbf{r}(t) = \mathbf{a}_w + t(\mathbf{b}_w - \mathbf{a}_w)
\]

\[
\mathbf{a}_s = \begin{bmatrix} s_x, s_y, -i \end{bmatrix}
\]

\[
\mathbf{b}_s = \begin{bmatrix} s_x, s_y, -f \end{bmatrix}
\]

Depends on screen details, YMMV

General idea should translate...
Depth Distortion

- Recall depth distortion from perspective
  - Interpolating in screen space different than in world
  - Ok, for shading (mostly)
  - Bad for texture
Depth Distortion

\[ S_1 = \frac{P_1}{h_1} \]
\[ S_2 = \frac{P_2}{h_2} \]
\[ S_3 = \frac{P_3}{h_3} \]
\[ S_4 = \frac{P_4}{h_4} \]
Depth Distortion

We know the $S_i$, $P_i$, and $b_i$, but not the $a_i$. 

\[ S_1 = \frac{P_1}{h_1} \]
\[ S_2 = \frac{P_2}{h_2} \]
\[ S_3 = \frac{P_3}{h_3} \]
\[ S_4 = \frac{P_4}{h_4} \]
\[ X = \sum_i S_i b_i \]
\[ Q = \sum_i P_i a_i \]
Depth Distortion

\[ S_1 = \frac{P_1}{h_1} \]

\[ S_2 = \frac{P_2}{h_2} \]

\[ S_3 = \frac{P_3}{h_3} \]

\[ S_4 = \frac{P_4}{h_4} \]

\[ X = \sum_i S_i b_i \]

\[ X = \frac{Q}{h} = \frac{\left( \sum_i P_i a_i \right)}{\left( \sum_j h_j a_j \right)} \]
Depth Distortion

\[ S_1 = P_1/h_1 \]

\[ S_2 = P_2/h_2 \]

\[ S_3 = P_3/h_3 \]

\[ S_4 = P_4/h_4 \]

\[ X = \sum_i S_i b_i \]

\[ Q = \sum_i P_i a_i \]

\[ \sum_i S_i b_i = \left( \sum_i P_i a_i \right) / \left( \sum_j h_j a_j \right) \]
Depth Distortion

\[ S_1 = P_1 / h_1 \]

\[ S_2 = P_2 / h_2 \]

\[ S_3 = P_3 / h_3 \]

\[ S_4 = P_4 / h_4 \]

\[ X = \sum_i S_i b_i \]

\[ \sum_i P_i b_i / h_i = \left( \sum_i P_i a_i \right) / \left( \sum_j h_j a_j \right) \]
Depth Distortion

\[ S_1 = P_1/h_1 \]
\[ S_4 = P_4/h_4 \]
\[ S_3 = P_3/h_3 \]
\[ X = \sum_i S_ib_i \]
\[ Q = \sum_i P_ia_i \]

Independent of given vertex locations.

\[ \sum_i P_i b_i/h_i = \left( \sum_i P_ia_i \right) / \left( \sum_j h_ja_j \right) \]

\[ b_i/h_i = a_i/ \left( \sum_j h_ja_j \right) \quad \forall i \]
**Depth Distortion**

\[ S_1 = P_1 / h_1 \]

\[ S_2 = P_2 / h_2 \]

\[ S_3 = P_3 / h_3 \]

\[ S_4 = P_4 / h_4 \]

\[ X = \sum_i S_i b_i \]

\[ Q = \sum_i P_i a_i \]

\[ b_i / h_i = a_i / \left( \sum_j h_j a_j \right) \quad \forall i \]

**Linear equations in the** \( a_i \).

\[ \left( \sum_j h_j a_j \right) b_i / h_i - a_i = 0 \quad \forall i \]
Depth Distortion

Linear equations in the $a_i$.

$$\left( \sum_j h_j a_j \right) \frac{b_i}{h_i} - a_i = 0 \quad \forall i$$

Not invertible so add some extra constraints.

$$\sum_i a_i = \sum_i b_i = 1$$
Depth Distortion

For a line:
\[ a_1 = \frac{h_2 b_i}{(b_1 h_2 + h_1 b_2)} \]

For a triangle:
\[ a_1 = \frac{h_2 h_3 b_1}{(h_2 h_3 b_1 + h_1 h_3 b_2 + h_1 h_2 b_3)} \]

Obvious Permutations for other coefficients.