Announcements

- Office hours moved this week to Wed morning at 11am

Derivation of General Rotation Matrix

- Axis angle rotation

\[
\begin{align*}
\vec{r}_x &= (\vec{a} \times \vec{x}) \vec{a} \\
\vec{r}_z &= (\vec{r} - \vec{x}) = (\vec{r} - (\vec{a} \times \vec{x}) \vec{a}) \\
\vec{a} \times \vec{r}_z &= \vec{a} \times (\vec{r} - (\vec{a} \times \vec{x}) \vec{a}) = \vec{a} \times \vec{x}
\end{align*}
\]

Axis-angle ONB

\[
\text{Sym}(\vec{a}) = \begin{bmatrix}
  a_z & a_y & a_x \\
  a_y & a_z & -a_x \\
  a_x & a_y & a_z
\end{bmatrix}
\]

\[
\text{Skew}(\vec{a}) = \begin{bmatrix}
  0 & -a_z & a_y \\
  a_z & 0 & -a_x \\
  -a_y & a_x & 0
\end{bmatrix}
\]

\[
\text{Skew}(\vec{a}) \vec{x} = \hat{a} \times \vec{x}
\]

Axis-angle rotation

\[
\begin{align*}
x_{\text{rotated}} &= \alpha \hat{a} + \beta \hat{x} + \gamma \hat{a} \times \hat{x} \\
x_{\text{rotated}} &= \hat{x} + \cos \theta \hat{x} + \sin \theta \hat{a} \times \hat{x} \\
x_{\text{rotated}} &= (\hat{a} \times \hat{x}) (1 - \cos \theta) \hat{a} + \cos \theta \hat{x} + \sin \theta \hat{a} \times \hat{x}
\end{align*}
\]
**Viewing, backward and forward**

- So far have used the backward approach to viewing
  - start from pixel
  - ask what part of scene projects to pixel
  - explicitly construct the ray corresponding to the pixel
- Next will look at the forward approach
  - start from a point in 3D
  - compute its projection into the image
- Central tool is matrix transformations
  - combines seamlessly with coordinate transformations used to position camera and model
  - ultimate goal: single matrix operation to map any 3D point to its correct screen location.

**Forward viewing**

- Would like to just invert the ray generation process
- But ray generation produces rays, not points in scene
- Inverting the ray tracing process requires division for the perspective case

**Mathematics of projection**

- Always work in eye coords
  - assume eye point at 0 and plane perpendicular to z
- Orthographic case
  - a simple projection: just toss out z
- Perspective case: scale diminishes with z
  - increases with d

**Pipeline of transformations**

- Standard sequence of transforms

**Orthographic transformation chain**

- Start with coordinates in object’s local coordinates
- Transform into world coords (modeling transform, $M_m$)
- Transform into eye coords (camera xf, $M_{cam}$)
- Orthographic projection, $M_{orth}$
- Viewport transform, $M_{vp}$
  \[ p_\alpha = M_{vp} M_{orth} M_{cam} M_m p_o \]
Parallel projection: orthographic

![Parallel projection: orthographic diagram](image)

to implement orthographic, just toss out z:

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

Parallel projection: oblique

![Parallel projection: oblique diagram](image)

to implement oblique, shear then toss out z:

\[
\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} x + a z \\ y + b z \\ 1 \end{pmatrix} \begin{bmatrix} 1 & a & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{bmatrix}
\]

View volume: orthographic

![View volume: orthographic diagram](image)

Viewing a cube of size 2

- Start by looking at a restricted case: the canonical view volume
- It is the cube \([0,1]^3\), viewed from the z direction
- Matrix to project it into a square image in \([0,1]^2\) is trivial:

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

Revisiting ray tracing: Pixel-to-image mapping

- Pixel center was at \((0.5,0.5)\) offset from bottom left

\[
\begin{align*}
u &= l + (r - l)(i + 0.5)/n_x \\
v &= b + (t - b)(j + 0.5)/n_y
\end{align*}
\]

Revisiting ray tracing: Pixel-to-image mapping

- Instead make coordinates go through integers
Viewing a cube of size 2

- To draw in image, need coordinates in pixel units, though
- Exactly the opposite of mapping \((i, j)\) to \((u, v)\) in ray generation
- Pixel centers at integer values now

![Diagram showing pixel centers and mapping](image-url)