CS4620/5620: Lecture 11

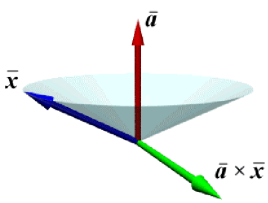
Viewing

Announcements

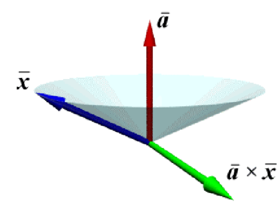
- Office hours moved this week to Wed morning at 11am

Derivation of General Rotation Matrix

- Axis angle rotation



Axis-angle ONB



$$\vec{x}_{\parallel} = (\vec{a}.\vec{x})\vec{a}$$
$$\vec{x}_{\perp} = (\vec{x} - \vec{x}_{\parallel}) = (\vec{x} - (\vec{a}.\vec{x})\vec{a})$$
$$\vec{a} \times \vec{x}_{\perp} = \vec{a} \times (\vec{x} - \vec{x}_{\parallel}) = \vec{a} \times (\vec{x} - (\vec{a}.\vec{x})\vec{a}) = \vec{a} \times \vec{x}$$

Axis-angle rotation

$$x_{rotated} = \alpha\ \vec{a} + \beta\ \vec{x}_{\perp} + \gamma\ \vec{a} \times \vec{x}$$

$$x_{rotated} = \vec{x}_{\parallel} + \cos\theta\ \vec{x}_{\perp} + \sin\theta\ \vec{a} \times \vec{x}$$
$$x_{rotated} = (\vec{a}.\vec{x})\vec{a} + \cos\theta\ (x - (\vec{a}.\vec{x})\vec{a}) + \sin\theta\ \vec{a} \times \vec{x}$$
$$x_{rotated} = (\vec{a}.\vec{x})(1 - \cos\theta)\vec{a} + \cos\theta\ \vec{x} + \sin\theta\ \vec{a} \times \vec{x}$$

$$Sym(\vec{a}) = \begin{bmatrix} a_x \\ a_y \\ a_z \\ 0 \end{bmatrix} \begin{bmatrix} a_x & a_y & a_z & 0 \end{bmatrix} = \begin{bmatrix} a_x^2 & a_x a_y & a_x a_z & 0 \\ a_x a_y & a_y^2 & a_y a_z & 0 \\ a_x a_z & a_y a_z & a_z^2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$Skew(\vec{a}) = \begin{bmatrix} 0 & -a_z & a_y & 0 \\ a_z & 0 & -a_x & 0 \\ -a_y & a_x & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$
$$Skew(\vec{a})\vec{x} = \vec{a} \times \vec{x}$$

$$x_{rotated} = (\vec{a}.\vec{x})(1-\cos\theta)\vec{a} + \cos\theta\ \vec{x} + \sin\theta\ \vec{a}\times\vec{x}$$

$$x_{rotated} = (Sym(\vec{a})(1-\cos\theta) + I\cos\theta\ + Skew(\vec{a})\sin\theta\ )\vec{x}$$

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

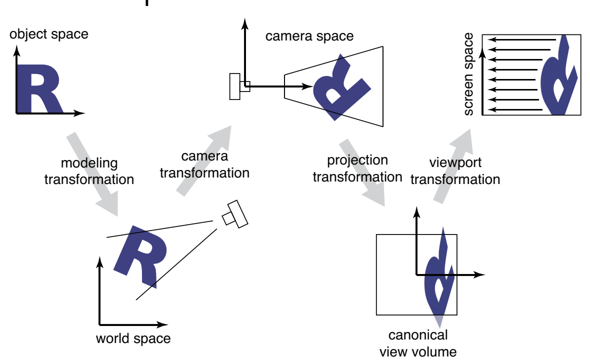
- Would like to just invert the ray generation process
- But ray generation produces rays, not points in scene
- Inverting the ray tracing process requires division for the perspective case

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

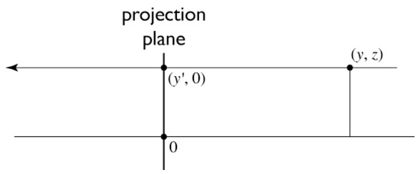


Orthographic transformation chain

- 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}

$$\mathbf{p}_s = \mathbf{M}_{\text{vp}}\mathbf{M}_{\text{orth}}\mathbf{M}_{\text{cam}}\mathbf{M}_{\text{m}}\mathbf{p}_o$$

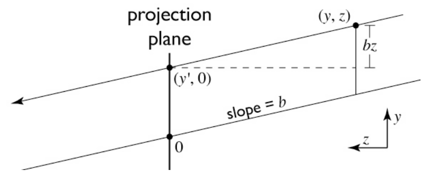
Parallel projection: orthographic



to implement orthographic, just toss out z:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

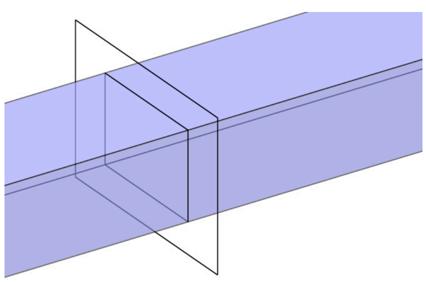
Parallel projection: oblique



to implement oblique, shear then toss out z:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x + az \\ y + bz \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

View volume: orthographic



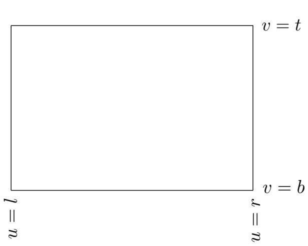
Viewing a cube of size 2

- 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:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Revisiting ray tracing: Pixel-to-image mapping

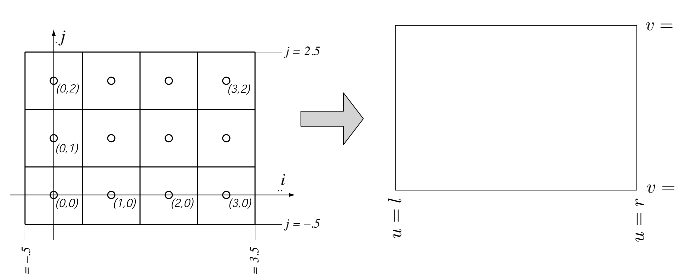
- Pixel center was at (0.5, 0.5) offset from bottom left



$$u = l + (r - l)(i + 0.5)/n_x$$
$$v = b + (t - b)(j + 0.5)/n_y$$

Revisiting ray tracing: Pixel-to-image mapping

- Instead make coordinates go through integers



$$u = l + (r - l)(i + 0.5)/n_x$$
$$v = b + (t - b)(j + 0.5)/n_y$$

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