Announcements

• Programming Assignment #2:
  – Extending deadline one week
  – Tuesday October 26 (was Tuesday Oct 19)
  – Get cracking since it’s not easy
• Movie next Thursday October 21
  – "The Story of Computer Graphics"
  – Away at Univ. Michigan
• Prelim#1
  – Grades released --- these are unadjusted!
  – Histogram
  – Exams available for pick-up
• Programming Assignment #1:
  – Grades to be released to day
  – Some stragglers still being graded

---

3D Viewing

CS 4620 Lecture 10
Text: Chapter 7 "Viewing"

Viewing, backward and forward

• Ray tracing uses backward approach to viewing
  – start from pixel
  – ask what part of scene projects to pixel
  – explicitly construct the ray corresponding to the pixel
• Rasterization uses forward approach to viewing
  – 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
• 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 \( \mathbf{0} \) and plane perpendicular to \( z \)
- Orthographic case
  - a simple projection: just toss out \( z \)
- Perspective case: scale diminishes with \( z \)
  - and increases with \( d \)

Pipeline of transformations

- Standard sequence of transforms

Parallel projection: orthographic

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

to implement orthographic, just toss out \( z \):
**Viewing a cube**

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

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

**Image-to-pixel mapping**

- To draw in image, need coordinates in pixel units, though
- Exactly the opposite of mapping $(i,j)$ to $(u,v)$ in ray generation

$$
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 \\
\end{pmatrix}
\begin{pmatrix}
-1 & -1 \\
1 & 1 \\
-5 & -5 \\
\end{pmatrix}
\rightarrow
\begin{pmatrix}
n_y & -5 \\
n_x & -5 \\
\end{pmatrix}
$$

Viewing a cube of size 2

---

**Pixel-to-image mapping** (for ray casting)

- $(u,v)$ coords of a pixel

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

---

**Windowing transforms**

- This transformation is worth generalizing: take one axis-aligned rectangle or box to another
  
  - a useful, if mundane, piece of a transformation chain

$$
\begin{pmatrix} x' \\ y' \end{pmatrix} =
\begin{pmatrix} x - x_0 \\ y - y_0 \end{pmatrix}
\begin{pmatrix} x + x_0 \\ y - y_0 \end{pmatrix}
\begin{pmatrix} x' - x \end{pmatrix}
\begin{pmatrix} x_0 \\ y_0 \\ 1 \end{pmatrix}
\begin{pmatrix} x - x_0 \\ y - y_0 \end{pmatrix}
$$

[Shirley03 f. 6.16; eq. 6.6]

© 2009 Doug James • 11

---

© 2010 Doug James • 9

---

© 2010 Doug James • 10

---

© 2010 Doug James • 12
**Viewport transformation**

\[
\begin{bmatrix}
1 \\
-1 \\
-1 \\
1 \\
\end{bmatrix} \rightarrow \begin{bmatrix}
n_y - 5 \\
-5 \\
n_z - 5 \\
-5 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
x_{\text{screen}} \\
y_{\text{screen}} \\
1 \\
\end{bmatrix} = \begin{bmatrix}
\frac{n_x}{2} & 0 & \frac{n_x - 1}{2} \\
0 & \frac{n_y}{2} & \frac{n_y - 1}{2} \\
0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
x_{\text{canonical}} \\
y_{\text{canonical}} \\
1 \\
\end{bmatrix}
\]

**Orthographic projection**

- First generalization: different view rectangle
  - retain the minus-z view direction

  ![Orthographic projection diagram](image)

- specify view by left, right, top, bottom (as in RT)
- also near, far

**Clipping planes**

- In object-order systems we always use at least two *clipping planes* that further constrain the view volume
  - near plane: parallel to view plane; things between it and the viewpoint will not be rendered
  - far plane: also parallel; things behind it will not be rendered

- These planes are:
  - partly to remove unnecessary stuff (e.g., behind the camera)
  - but really to constrain the range of depths
    (we’ll see why later)

\[
M_{\text{vp}} = \begin{bmatrix}
\frac{n_x}{2} & 0 & 0 & \frac{n_x - 1}{2} \\
0 & \frac{n_y}{2} & 0 & \frac{n_y - 1}{2} \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\]
Orthographic projection

- We can implement this by mapping the view volume to the canonical view volume.
- This is just a 3D windowing transformation!

\[
M_{\text{orth}} = \begin{bmatrix}
\frac{x_1 - x'_1}{x_n - x'_n} & 0 & 0 & \frac{x'_1 x_2 - x'_2 x'_1}{x_n - x'_n} \\
0 & \frac{y_1 - y'_1}{y_n - y'_n} & 0 & \frac{y'_1 y_2 - y'_2 y'_1}{y_n - y'_n} \\
0 & 0 & \frac{z_1 - z'_1}{z_n - z'_n} & \frac{z'_1 z_2 - z'_2 z'_1}{z_n - z'_n} \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Camera and modeling matrices

- We worked out all the preceding transforms starting from eye coordinates
  - before we do any of this stuff we need to transform into that space
- Transform from world (canonical) to eye space is traditionally called the viewing matrix
  - it is the canonical-to-frame matrix for the camera frame
  - that is, \( F_c^{-1} \)
- Remember that geometry would originally have been in the object’s local coordinates; transform into world coordinates is called the modeling matrix, \( M_m \)
- Note some systems (e.g. OpenGL) combine the two into a modelview matrix and just skip world coordinates

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., \( M_{\text{cam}} = F_c^{-1} \))
- Orthographic projection, \( M_{\text{orth}} \)
- Viewport transform, \( M_{\text{vp}} \)

\[
P_s = M_{\text{vp}}M_{\text{orth}}M_{\text{cam}}M_{\text{in}}P_o
\]

\[
\begin{bmatrix}
x_a \\
y_a \\
z_a
\end{bmatrix} = \begin{bmatrix}
\frac{2}{n_2} & 0 & 0 & \frac{n_2-1}{n_2} \\
0 & \frac{2}{n_3} & 0 & \frac{n_3-1}{n_3} \\
0 & 0 & \frac{2}{n_5} & \frac{n_5-1}{n_5}
\end{bmatrix}\begin{bmatrix}
\frac{2}{n_7} & 0 & 0 & \frac{n_7-1}{n_7} \\
0 & \frac{2}{n_9} & 0 & \frac{n_9-1}{n_9} \\
0 & 0 & \frac{2}{n_5} & \frac{n_5-1}{n_5}
\end{bmatrix}\begin{bmatrix}
u & v & w & e
\end{bmatrix}^{-1}M_m\begin{bmatrix}
x_o \\
y_o \\
z_o
\end{bmatrix}
\]
Perspective projection

Similar triangles:

\[
\frac{y'}{d} = \frac{y}{-z} \\
y' = -dy/z
\]

Homogeneous coordinates revisited

- Introduced \( w = 1 \) coordinate as a placeholder
  - used as a convenience for unifying translation with linear
- Can also allow arbitrary \( w \)

\[
\begin{bmatrix}
  x \\
  y \\
  z \\
  1
\end{bmatrix} \sim
\begin{bmatrix}
  wx \\
  wy \\
  wz \\
  w
\end{bmatrix}
\]

Homogeneous coordinates revisited

- Perspective requires division
  - that is not part of affine transformations
  - in affine, parallel lines stay parallel
    - therefore not vanishing point
    - therefore no rays converging on viewpoint
- “True” purpose of homogeneous coords: projection

Implications of \( w \)

- All scalar multiples of a 4-vector are equivalent
- When \( w \) is not zero, can divide by \( w \)
  - therefore these points represent “normal” affine points
- When \( w \) is zero, it’s a point at infinity, a.k.a. a direction
  - this is the point where parallel lines intersect
  - can also think of it as the vanishing point
- Digression on projective space
Perspective projection

to implement perspective, just move $z$ to $w$:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -dx/z \\ -dy/z \\ 1 \end{bmatrix} \sim \begin{bmatrix} dx \\ dy \\ -z \end{bmatrix} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

View volume: perspective

View volume: perspective (clipped)

Carrying depth through perspective

- Perspective has a varying denominator—can’t preserve depth!
- Compromise: preserve depth on near and far planes

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} \sim \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ 1 \end{bmatrix} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- that is, choose $a$ and $b$ so that $z'(n) = n$ and $z'(f) = f$.

$$\tilde{z}(z) = az + b$$

$$z'(z) = \frac{\tilde{z}}{-z} = \frac{az + b}{-z}$$

want $z'(n) = n$ and $z'(f) = f$

result: $a = -(n + f)$ and $b = nf$ (try it)
Official perspective matrix

- Use near plane distance as the projection distance
  - i.e., $d = -n$
- Scale by $-1$ to have fewer minus signs
  - scaling the matrix does not change the projective transformation

$$P = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n + f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Perspective projection matrix

- Product of perspective matrix with orth. projection matrix

$$M_{\text{per}} = M_{\text{Orth}}P$$

$$= \begin{bmatrix} \frac{2}{l-r} & 0 & 0 & \frac{r+l}{l-r} \\ 0 & \frac{2}{t-b} & 0 & \frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & \frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n + f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{2n}{l-r} & 0 & \frac{l+r}{l-r} & 0 \\ 0 & \frac{2n}{t-b} & \frac{b+t}{b-t} & 0 \\ 0 & 0 & \frac{f+n}{n-f} & \frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Perspective transformation chain

- Transform into world coords (modeling transform, $M_m$)
- Transform into eye coords (camera xf., $M_{\text{cam}} = F_c^{-1}$)
- Perspective matrix, $P$
- Orthographic projection, $M_{\text{orth}}$
- Viewport transform, $M_{\text{vp}}$

$$P_s = M_{\text{vp}}M_{\text{orth}}PM_{\text{cam}}M_mP_o$$

OpenGL view frustum: orthographic

Note OpenGL puts the near and far planes at $-n$ and $-f$
so that the user can give positive numbers
OpenGL view frustum: perspective

Note OpenGL puts the near and far planes at \(-n\) and \(-f\) so that the user can give positive numbers.

Pipeline of transformations

- Standard sequence of transforms

Viewing Demo

http://www.cs.cornell.edu/courses/cs4620/2009fa/demos/viewexplore.stm