CS248 Review Session 2

Friday, 1/29/10

TA: Kevin Montag
Outline

- Transforming normals
- Perspective-correct attribute interpolation
- Coordinate spaces and Assignment 2: a love story
Slide Theft Confession

• Many slides taken from Kurt Akeley’s lectures for CS248 in 2007

Outline

- Transforming normals
- Perspective-correct attribute interpolation
- Coordinate spaces and Assignment 2: a love story
Transforming Normals

• To get to eye coordinates, we transform vertices by the modelview matrix
• However, this doesn’t work for normals
• Why not?
Non-rigid transformations warp normals

Example: scaling transformation

\[(1 \ 1 \ 0) \times S^{-1}(1,2,1) = (1 \ 1 \ 0) \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & \frac{1}{2} & 0 \end{pmatrix}\]

\[S(1,2,1)\]
Correct way to perform transformation

• Multiply normals by inverse transpose of modelview matrix, rather than by the modelview matrix itself

• To compute inverse transpose, we can keep track of the matrix’s decomposition into primitives
Elemental inverse transforms are simple

\[ T^{-1}(t_x, t_y, t_z) = T(-t_x, -t_y, -t_z) \]
\[ S^{-1}(s_x, s_y, s_z) = S(1/s_x, 1/s_y, 1/s_z) \]
\[ R^{-1}(q, a) = R(-q, a) \]
\[ R_x^{-1}(q) = R_x(-q) \]
\[ R_y^{-1}(q) = R_y(-q) \]
\[ R_z^{-1}(q) = R_z(-q) \]
Outline

• Transforming normals
• **Perspective-correct attribute interpolation**
• Coordinate spaces and Assignment 2: a love story
The need for perspective-correct interpolation

• “Attributes” of vertices (e.g. normals, colors, etc.) are in a sense specified in eye space

• Once we apply the projection matrix to vertices, attributes no longer vary linearly between the transformed vertices
Incorrect attribute interpolation

Linear interpolation

\[ F \leftarrow f \]
Linear (barycentric) attribute evaluation

\[ f = c_0 f_0 + c_1 f_1 + c_2 f_2 \]

\[ a = (y_0 x_1 - x_0 y_1) + (y_1 x_2 - x_1 y_2) + (y_2 x_0 - x_2 y_0) \]

\[ a_0 = (y x_1 - xy_1) + (y_1 x_2 - x_1 y_2) + (y_2 x - x_2 y), \quad c_0 = a_0 / a \]

\[ a_1 = (y x_2 - xy_2) + (y_2 x_0 - x_2 y_0) + (y_0 x - x_0 y), \quad c_1 = a_1 / a \]

\[ a_2 = (y x_0 - xy_0) + (y_0 x_1 - x_0 y_1) + (y_1 x - x_1 y), \quad c_2 = a_2 / a \]
Perspective-correct attribute evaluation

\[
f = \frac{c_0 \frac{f_0}{w_0} + c_1 \frac{f_1}{w_1} + c_2 \frac{f_2}{w_2}}{c_0 \frac{1}{w_0} + c_1 \frac{1}{w_1} + c_2 \frac{1}{w_2}}
\]

All \( w \)'s are \( w_c \)'s

\[
a = (y_0x_1 - x_0y_1) + (y_1x_2 - x_1y_2) + (y_2x_0 - x_2y_0)
\]

\[
a_0 = (yx_1 - xy_1) + (y_1x_2 - x_1y_2) + (y_2x - x_2y), \quad c_0 = a_0 / a
\]

\[
a_1 = (yx_2 - xy_2) + (y_2x_0 - x_2y_0) + (y_0x - x_0y), \quad c_1 = a_1 / a
\]

\[
a_2 = (yx_0 - xy_0) + (y_0x_1 - x_0y_1) + (y_1x - x_1y), \quad c_2 = a_2 / a
\]

Requires a division for each fragment
Rasterization requires access to $w_c$

```
struct {
    float x, y, z, w;
    float nx, ny, nz;
    float r, g, b, a;
} vertex;

struct {
    float x, y, z, w;
    float r, g, b, a;
} clipvertex;

struct {
    winvertex v0, v1, v2
} triangle;
```
OpenGL coordinate systems

Vertex operations

Object coordinates

Model-view transformation

Eye coordinates
( lighting calculations )

Projection transformation

Clip coordinates
( clip to unit cube )

Homogenize

Normalized device coordinates

Viewport transformation

Window coordinates

Primitive assembly

Transform $v_o$ by $M$

Transform $n_o$ by $M^{-1}$

$\mathbf{n}_e \rightarrow \mathbf{v}_e$

Transform $v_e$ by $P$

$\mathbf{v}_c$

Divide by $w_c$

$\mathbf{v}_d, w_c$

Map to Window

$\mathbf{v}_w, w_c$
Where you’ll need to do this in Assignment 2

- Normal interpolation (for lighting computations)
- Texture mapping
Example: Perspective-correct texture coordinate evaluation

\[
s_w = \frac{c_0 \frac{s_0}{w_0} + c_1 \frac{s_1}{w_1} + c_2 \frac{s_2}{w_2}}{c_0 \frac{q_0}{w_0} + c_1 \frac{q_1}{w_1} + c_2 \frac{q_2}{w_2}}
\]

Texture coordinates homogenized here.
Supports image remapping (from q-specified warp to render-specified warp).

\[\begin{align*}
w_c's, S_c's, q_c's \\
x_w's, y_w's \\
(x_0, y_0, w_0, s_0)
\end{align*}\]

\[\begin{align*}
a &= (y_0x_1 - x_0y_1) + (y_1x_2 - x_1y_2) + (y_2x_0 - x_2y_0) \\
a_0 &= (yx_1 - xy_1) + (y_1x_2 - x_1y_2) + (y_2x - x_2y), \quad c_0 = a_0/a \\
a_1 &= (yx_2 - xy_2) + (y_2x_0 - x_2y_0) + (y_0x - x_0y), \quad c_1 = a_1/a \\
a_2 &= (yx_0 - xy_0) + (y_0x_1 - x_0y_1) + (y_1x - x_1y), \quad c_2 = a_2/a
\end{align*}\]
Outline

• Transforming normals
• Perspective-correct attribute interpolation
• Coordinate spaces and Assignment 2: a love story
Coordinate spaces are important

• There are lots of abstractions in assignment 2 (lights, normals, vertices, texture coordinates, etc.)

• It’s important to keep track of the space in which each of them is currently “living”
Lighting

• To perform lighting computations, we need vertex normals

• However, we don’t want to transform these normals past eye space, since this would require computing the inverse of the projection matrix

• So, lighting computations happen \textit{in eye space}
Lighting

• How should we transform specified light positions?
• By modelview matrix, not by projection matrix
• This is also why we need to use perspective-correct interpolation for normal vectors
Thanks for coming!

Good luck on the assignment!

Questions?