Correction: W2V vs. V2W

- M_{W2V}=TR
- we derived position of camera in world
- invert for world with respect to camera
- M_{W2V}^{-1}\cdot R\cdot T^{-1}
- more effort to break into triangles
- perspective correction affects Q8
- use flood-fill to draw interior
- other octants: different tests from bilinear interpolation of point P on simple convex polygons

News
- P2 due date extended to Tue Mar 2 5pm
- V2W correction affects Q1 and thus cascades to Q4-Q7
- perspective correction affects Q8
- TA office hours in lab for P2/H2 questions Fri 2-4 (Garrett)

Review: Flood Fill
- simple algorithm
- draw edges of polygon
- use flood-fill to draw interior

Clariation/Correction II: Midpoint
- we're moving horizontally along x direction (first octant)
- only two choices: draw at current y value, or move up vertically to top
- check if midpoint between two possible pixel centers above or below line
- candidates:
  - top pixel (x+1, y)
  - bottom pixel (x+1, y)
  - midpoint (x+1, y)
- check if midpoint above or below line
  - below: pick top pixel
  - above: pick bottom pixel
  - other octants: different tests
  - octant II: x-loop, check if left/right?

Review: Barycentric Coordinates
- non-orthogonal coordinate system based on triangle itself
  - origin: P1, basis vectors: (P2-P1) and (P3-P1)

Using Barycentric Coordinates
- weighted combination of vertices
  - smooth mixing
  - speedup
    - compute once per triangle
  - \beta = 0
  - \alpha + \beta + \gamma = 1
  - \alpha, \beta, \gamma \geq 0

Computing Barycentric Coordinates
- 2D triangle area
  - \alpha = |A_{01} + A_{02} + A_{03}|
  - \beta = |A_{12} + A_{13} + A_{10}|
  - \gamma = |A_{20} + A_{23} + A_{21}|

Deriving Barycentric From Bilinear
- from bilinear interpolation of point P on scanline
  - A = A_{00} + A_{02} + A_{03}
  - B = A_{12} + A_{13} + A_{10}
  - C = A_{20} + A_{23} + A_{21}
  - P1 - P2 = d1 + d2
  - P1 - P3 = d1 + d3
  - P2 - P3 = d2 + d3

Correction: Perspective Derivation
- z axis flip
  - x' = x/z
  - y' = y/z
  - z' = z
  - z = -far \rightarrow z'/z' = -1
  - z = -near \rightarrow z'/z' = -1

Review: Triangulating Polygons
- simple convex polygons
  - trial to break into triangles
  - pick one vertex, draw lines to all others not immediately adjacent
  - OpenGL supports automatically
  - glBegin(GL_POLYGON) ... glEnd()
- concave or non-simple polygons
  - more effort to break into triangles
  - simple approach may not work
  - OpenGL can support at extra cost
  - glNewList(), glUseList()
Deriving Barycentric From Bilinear

- similarly

\[ \begin{align*}
P_1 &= P_1 + \frac{b}{b_1 + b_2} (P_2 - P_1) \\
&= \frac{b_1}{b_1 + b_2} P_1 + \frac{b_2}{b_1 + b_2} P_2
\end{align*} \]

Deriving Barycentric From Bilinear

- combining

\[ \begin{align*}
P_1 &= \frac{c_2}{c_1 + c_2} P_1 + \frac{c_1}{c_1 + c_2} P_2 \\
P_2 &= \frac{d_2}{d_1 + d_2} P_1 + \frac{d_1}{d_1 + d_2} P_2
\end{align*} \]

- gives

\[ \begin{align*}
P &= \frac{c_1 + c_2}{c_1 + c_2} P_1 + \frac{c_1}{c_1 + c_2} P_2
\end{align*} \]

Deriving Barycentric From Bilinear

- thus \( \mathbf{P} = \alpha \mathbf{P}_1 + \beta \mathbf{P}_2 + \gamma \mathbf{P}_3 \) with

\[ \begin{align*}
\alpha &= \frac{c_1}{c_1 + c_2 + c_2 + b_1 + b_2} \\
\beta &= \frac{c_2}{c_1 + c_2 + d_1 + d_2 + b_1 + b_2} \\
\gamma &= \frac{c_1 + c_2 + b_1 + b_2}{c_1 + c_2 + b_1 + b_2}
\end{align*} \]

- can verify barycentric properties

\[ \alpha + \beta + \gamma = 1, \quad 0 \leq \alpha, \beta, \gamma \leq 1 \]

Lighting I

- simulate interaction of light and objects
- fast: fake it!
- approximate the look, ignore real physics
- get the physics (more) right

Goal

- transport of energy from light sources to surfaces & points
- global includes direct and indirect illumination
- more later

Photorealistic Illumination

- area lights
- light sources with a finite area
- more realistic model of many light sources
- not available with projective rendering pipeline (i.e., not available with OpenGL)

Light Sources

- ambient lights
- no identifiable source or direction
- hack for replacing true global illumination
- (diffuse interreflection: light bouncing off from other objects)

Illumination in the Pipeline

- local illumination
- only models light arriving directly from light source
- no interreflections or shadows
- can be added through tricks, multiple rendering passes
- light sources
- simple shapes
- materials
- simple, non-physical reflection models

Light Sources

- types of light sources
- area lights
- light sources with a finite area
- more realistic model of many light sources
- not available with projective rendering pipeline (i.e., not available with OpenGL)

Rendering Pipeline

- Geometry Database
- Model/View Transform.
- Lighting
- Perspective Transform.
- Clipping
- Scan Conversion
- Depth Test
- Texturing
- Blending
- Frame-buffer

Lighting

- perspective transformation
- view-related
- device transformation

Point Light Sources

- scene lit with ambient and point light source

Diffuse Interreflection

- scene lit only with an ambient light source

Light Sources

- scene lit with directional and ambient light
**Light Sources**
- geometry: positions and directions
- standard: world coordinate system
- effect: lights fixed wrt world geometry
- alternative: camera coordinate system
- points and directions undergo normal model/view transformation
- illumination calculations: camera coords

**Types of Reflection**
- **specular** (a.k.a. mirror or regular) reflection causes light to propagate without scattering.
- diffuse reflection sends light in all directions with equal energy.
- mixed reflection is a weighted combination of specular and diffuse.

**Lambert's Cosine Law**
- ideal diffuse surface reflection
  - the energy reflected by a small portion of a surface from a light source is proportional to the cosine of the angle between that direction and the surface normal
- reflected intensity
- independent of viewing direction
- depends on surface orientation wrt light
- often called Lambertian surfaces

**Reflectance Distribution Model**
- most surfaces exhibit complex reflectances
  - vary with incident and reflected directions.
  - model with combination

**Surface Roughness**
- at a microscopic scale, all real surfaces are rough
- cast shadows on themselves
- “mask” reflected light:

**Lambert’s Law**
- depends on angle of incidence: angle between surface normal and incoming light
  - \( I_{\text{diffuse}} = k_d I_{\text{light}} \theta \)
- in practice use vector arithmetic
  - \( I_{\text{diffuse}} = k_d I_{\text{light}} \cdot \mathbf{n} \)
- always normalize vectors used in lighting!!!
  - \( \mathbf{n} \) should be unit vectors
  - scalar (B/W intensity) or 3-tuple or 4-tuple (color)
  - \( k_d \): diffuse coefficient, surface color
  - \( I_{\text{light}} \): incoming light intensity
  - \( I_{\text{diffuse}} \): outgoing light intensity (for diffuse reflection)

**Specular Highlights**
- retro-reflection occurs when incident energy reflects in directions close to the incident direction, for a wide range of incident directions.
- gloss is the property of a material surface that involves mixed reflection and is responsible for the mirror like appearance of rough surfaces.

**Types of Reflection**
- retro-reflection occurs when incident energy reflects in directions close to the incident direction, for a wide range of incident directions.

**Diffuse Lighting Examples**
- Lambertian sphere from several lighting angles:
  - need only consider angles from 0° to 90°
  - why?
  - demo: Brown exploratory on reflection