Program 2 Corrections/Clarifications

- handin 314 proj2 (not 414)
- 'f' not 's' to toggle flat/smooth shading
  - 's' already in use for camera
- add: 't' to toggle between randomly colored and grey terrain
  - makes it easier to check if lighting correct
- add: 'u' to replace terrain with new randomly generated geometry
- consider adding for a bit of extra credit:
  - '+/-' toggle to increment/decrement abs cam speed

- roll/yaw confusion
  - first para. correct, second para. wrong
    - left horiz drag = yaw, right horiz drag = roll
  - image + flying expertise courtesy of Matt Baumann

Program 2 Quick Demo

Review: Midpoint Algorithm

- moving incrementally along x direction
  - draw at current y value, or move up to y+1?
    - check if midpoint between two possible pixel centers above or below line
- candidates
  - top pixel: (x+1,y+1),
  - bottom pixel: (x+1, y)
- midpoint: (x+1, y+.5)
- check if midpoint above or below line
  - below: top pixel
  - above: bottom pixel
- assume \( x_1 < x_2 \), slope \( 0 < \frac{dy}{dx} < 1 \)

Review: Bresenham Algorithm

- all integer arithmetic
- cumulative error function

```c
y=y0; e=0;
for (x=x0; x <= x1; x++) {
    draw(x,y);
    if (2(e+dy) < dx) {
        e = e+dy;
    } else {
        y=y+1;
        e=e+dy-dx;
    }
}
```

```c
y=y0; eps=0
for ( int x = x0; x <= x1; x++ ){
    draw(x,y);
    eps += dy;
    if ( (eps << 1) >= dx ){
        y++;  eps -= dx;
    }
}
Review: Flood Fill
- Draw polygon edges, seed point, recursively set all neighbors until boundary is hit to fill interior.
- Drawbacks: visit pixels up to 4x, per-pixel memory storage needed.

Review: Scanline Algorithms
- Set pixels inside polygon boundary along horizontal lines one pixel apart.
- Use bounding box to speed up.

Review: Edge Walking
- Basic idea:
  - Draw edges vertically.
  - Interpolate colors down edges.
  - Fill in horizontal spans for each scanline.
  - At each scanline, interpolate edge colors across span.

Review: General Polygon Rasterization
- Idea: use a parity test.

But Wait, There's More!
- Nice comment :)!

Hall of Fame
- Two fourth year CG courses await you!
  - 424 Geometric Modelling
  - 426 Animation
Midterm Review

- Midterm Exam
  - Friday Feb 11, 10am-10:50am
  - you may use one handwritten 8.5"x11" sheet
    - one side of page
  - no other notes, no books
  - nonprogrammable calculators OK
  - arrive on time!
  - sit every other seat, ID out in front of you
  - coats and bags in front of room

What’s Covered

- transformations
- viewing and projections
- coordinate systems of rendering pipeline
- lighting and shading
- not scan conversion

Reading

- FCS book, Red book
  - see web page for details
  - you can be tested on material in book but not covered in lecture
  - you can be tested on material covered in lecture but not covered in book

Old Exams Posted

- see course web page

The Rendering Pipeline

- pros and cons of pipeline approach
Transformations

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

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

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

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

Homogeneous Coordinates

\[ \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \]

Composing Transformations

ORDER MATTERS!

\[ T(1,1) \quad R(45) \quad T(1,1) R(45) \]

\[ T(1,1) R(45) = R(45) T(1,1) \]

Composing Transformations

- example: rotation around arbitrary center
- step 1: translate coordinate system to rotation center

Composing Transformations

- example: rotation around arbitrary center
- step 2: perform rotation
Composing Transformations

- example: rotation around arbitrary center
- step 3: back to original coordinate system

Transformation Hierarchies

- hierarchies don’t fall apart when changed
- transforms apply to graph nodes beneath

Matrix Stacks

- push and pop matrix stack
- avoid computing inverses or incremental xforms
- avoid numerical error

Transformation Hierarchies

- example
Display Lists
- reuse block of OpenGL code
- more efficient than immediate mode
  - code reuse, driver optimization
- good for static objects redrawn often
  - can’t change contents
  - not just for multiple instances
    - interactive graphics: objects redrawn every frame
- nest when possible for efficiency

Double Buffering
- two buffers, front and back
  - while front is on display, draw into back
  - when drawing finished, swap the two
- avoid flicker

Projective Rendering Pipeline

Projection
- theoretical pinhole camera
  - image plane
  - eye point
  - image inverted, more convenient equivalent

Projection Taxonomy
- planar projections
  - perspective: 1, 2, 3-point
  - parallel
  - oblique
  - orthographic
  - cabinet
  - cavalier
- axonometric:
  - isometric
  - dimetric
  - trimetric

Projective Transformations
- transformation of space
  - center of projection moves to infinity
  - viewing frustum transformed into a parallelepiped
**Normalized Device Coordinates**

left/right \( x = +/- 1 \), top/bottom \( y = +/- 1 \), near/far \( z = +/- 1 \)

Camera coordinates

![Frustum](image)

NDC

![Frustum](image)

**Projection Normalization**

- distort such that orthographic projection of distorted objects is desired persp projection

![Perspective Projection](image)

**Transforming View Volumes**

- orthographic view volume
  - \( x=left \)
  - \( y=bottom \)
  - \( z=-far \)

- perspective view volume
  - \( x=left \)
  - \( y=top \)
  - \( z=-near \)

**Basic Perspective Projection**

- can express as homogenous 4x4 matrix!

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

\[
\begin{bmatrix}
x \\
y \\
z/d \\
\end{bmatrix}
\rightarrow\begin{bmatrix}
x \cdot d/z \\
y \cdot d/z \\
d \\
\end{bmatrix}
\]

**Projective Transformations**

- determining the matrix representation
  - need to observe 5 points in general position, e.g.
    - \([left,0,0,1]^T\rightarrow[-1,0,0,1]^T\)
    - \([0,top,0,1]^T\rightarrow[0,1,0,1]^T\)
    - \([0,0,-f,1]^T\rightarrow[0,0,1,1]^T\)
    - \([0,0,-n,1]^T\rightarrow[0,0,-1,1]^T\)
    - \([left*f/n,top*f/n,-f,1]^T\rightarrow[-1,1,1,1]^T\)
  - solve resulting equation system to obtain matrix
OpenGL Orthographic Matrix

- scale, translate, reflect for new coord sys
- understand derivation from VCS!

\[
P' = \begin{bmatrix}
\frac{2}{\text{right} - \text{left}} & 0 & 0 & \frac{\text{right} + \text{left}}{\text{right} - \text{left}} \\
0 & \frac{2}{\text{top} - \text{bot}} & 0 & \frac{\text{top} + \text{bot}}{\text{top} - \text{bot}} \\
0 & 0 & \frac{-2}{\text{far} - \text{near}} & \frac{\text{far} + \text{near}}{\text{far} - \text{near}} \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

OpenGL Perspective Matrix

- shear, scale, reflect for new coord sys
- understand derivation from VCS!

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

Viewport Transformation

- onscreen pixels: map from [-1,1] to [0, displaywidth]

Light Sources

- directional/parallel lights
  - point at infinity: \((x,y,z,0)^T\)
- point lights
  - finite position: \((x,y,z,1)^T\)
- spotlights
  - position, direction, angle
- ambient lights

Reflectance

- specular: perfect mirror with no scattering
- gloss: mixed, partial specularity
- diffuse: all directions with equal energy

\[
I_{\text{diffuse}} = k_d I_{\text{light}} (n \cdot l)
\]

\[
I_{\text{specular}} = k_s I_{\text{light}} (v \cdot r)^n_{\text{shiny}}
\]

Review: Reflection Equations

- specular + glossy + diffuse = reflectance distribution

\[
2 (N (N \cdot L)) - L = R
\]
Review: Reflection Equations 2

- Blinn improvement
  \[ I_{\text{specular}} = k_s I_{\text{light}} (h \cdot n)^{n_{\text{shiny}}} \]
  \[ h = (l + v)/2 \]

- full Phong lighting model
  - combine ambient, diffuse, specular components
  \[ I_{\text{total}} = k_s I_{\text{ambient}} + \sum_{i=1}^{\text{lighs}} I_i (k_d (n \cdot l_i) + k_s (v \cdot r_i))^{n_{\text{shiny}}} \]

Lighting vs. Shading

- lighting
  - simulating the interaction of light with surface
- shading
  - deciding pixel color
  - continuum of realism: when do we do lighting calculation?

Shading Models

- flat shading
  - compute Phong lighting once for entire polygon
- Gouraud shading
  - compute Phong lighting at the vertices and interpolate lighting values across polygon
- Phong shading
  - compute averaged vertex normals
  - interpolate normals across polygon and perform Phong lighting across polygon

Transforming Normals

- apply nonuniform scale: stretch along x by 2
  - can’t transform normal by modelling matrix
- solution:
  \[ P \rightarrow P' = MP \]
  \[ N \rightarrow N' = QN \]
  \[ Q = (M^{-1}) \]
  normal to any surface transformed by inverse transpose of modelling transformation

Scan Conversion

- done:
  - how to determine pixels covered by a primitive
- next:
  - how to assign pixel colors
    - interpolation of colors across triangles
    - interpolation of other properties
Computing Barycentric Coordinates

- interpolate values between vertices
  - z values
  - r,g,b colour components
  - use for Gouraud shading
  - u,v texture coordinates
  - surface normals
- equivalent methods (for triangles)
  - bilinear interpolation
  - barycentric coordinates

Interpolation During Scan Conversion

- interpolate quantity along L and R edges, as a function of y
  - then interpolate quantity as a function of x

\[
\begin{align*}
P &= \alpha \cdot P_1 + \beta \cdot P_2 + \gamma \cdot P_3 \\
\alpha + \beta + \gamma &= 1 \\
0 \leq \alpha, \beta, \gamma \leq 1
\end{align*}
\]

"convex combination of points"

Bilinear Interpolation

- interpolate quantity between vertices
  - z values
  - r,g,b colour
  - surface normals
- equivalent methods (for triangles)
  - bilinear interpolation
  - barycentric coordinates

Computing Barycentric Coordinates

- for point P on scanline

\[
\begin{align*}
P &= \alpha \cdot P_1 + \beta \cdot P_2 + \gamma \cdot P_3 \\
\alpha + \beta + \gamma &= 1 \\
0 \leq \alpha, \beta, \gamma \leq 1
\end{align*}
\]

"convex combination of points"

3. Barycentric Coordinates

- weighted combination of vertices

\[
P = \alpha \cdot P_1 + \beta \cdot P_2 + \gamma \cdot P_3
\]

\[
\alpha + \beta + \gamma = 1
\]

\[
0 \leq \alpha, \beta, \gamma \leq 1
\]

"convex combination of points"

Computing Barycentric Coords

- similarly

\[
P_R = P_2 + \frac{b_1}{b_1 + b_2} (P_1 - P_2)
\]

\[
= (1 - \frac{b_1}{b_1 + b_2}) P_2 + \frac{b_1}{b_1 + b_2} P_1
\]

\[
= \frac{b_2}{b_1 + b_2} P_2 + \frac{b_1}{b_1 + b_2} P_1
\]

Combining

\[
P = \frac{c_2}{c_1 + c_2} \cdot P_L + \frac{c_1}{c_1 + c_2} \cdot P_R
\]

\[
P_L = \frac{d_2}{d_1 + d_2} \cdot P_2 + \frac{d_1}{d_1 + d_2} \cdot P_3
\]

\[
P_R = \frac{d_2}{d_1 + d_2} \cdot P_2 + \frac{d_1}{d_1 + d_2} \cdot P_1
\]

\[
P = \frac{c_2}{c_1 + c_2} \left( \frac{d_2}{d_1 + d_2} P_2 + \frac{d_1}{d_1 + d_2} P_3 \right) + \frac{c_1}{c_1 + c_2} \left( \frac{b_2}{b_1 + b_2} P_2 + \frac{b_1}{b_1 + b_2} P_1 \right)
\]
Computing Barycentric Coords

Thus

\[ P = a_1 \cdot P_1 + a_2 \cdot P_2 + a_3 \cdot P_3 \]

with

\[ a_1 = \frac{c_1}{c_1 + c_2} \frac{b_1}{b_1 + b_2} \]
\[ a_2 = \frac{c_2}{c_1 + c_2} \frac{d_2}{d_1 + d_2} + \frac{c_1}{c_1 + c_2} \frac{b_2}{b_1 + b_2} \]
\[ a_3 = \frac{c_2}{c_1 + c_2} \frac{d_1}{d_1 + d_2} \]

Computing Barycentric Coords

Can verify barycentric properties

\[ a_1 + a_2 + a_3 = 1 \]
\[ 0 \leq a_i, a_2, a_3 \leq 1 \]