• Homework 1
  – problem 18
    • x and y transposed in bottom layer
  – problem 4
    • rotate 30 deg around x axis with fixed point of (3.5,12,1)
  – new correct PDF posted
  – reminder: no late work after Fri 17 Oct 9am
• Project 1
  – solution, hall of fame on Friday

News

• Office hours reminder: FSC 2618
  – Mondays 10:30-11:30 or by appointment
  – exceptions: Oct 20, Nov 10

• Readings
  – Chap 8.9-8.11, Fri 10/3 slide notes

Scanline Algorithms

• given vertices, fill in the pixels
  triangles
    • split into two regions
    • fill in between edges
  arbitrary polygons (non-simple, non-convex)
    • build edge table
    • for each scanline
      • obtain list of intersections, i.e., AEL
      • use parity test to determine in/out and fill in the pixels

Edge Equations

• define triangle as intersection of three positive half-spaces:
Edge Equations

• So… simply turn on those pixels for which all edge equations evaluate to > 0:

Edge Equation Consistency

• how to get same +/- state for all equations?
  – consistent counterclockwise vertex traversal
    • good: \([0,1],[1,2],[2,0]\) or \([0,2],[2,1],[1,0]\)
    • bad: \([0,1],[2,1],[2,0]\)

• how to ensure interior is positive?
  – explicit area test: if negative, flip all param signs
    • if \(\text{area} < 0\) \(\{A = -A; B = -B; C = -C;\}\)

Parity for General Case

• use parity for interior test
  – draw pixel if edgecount odd
  – horizontal lines: don’t count
  – vertical max: don’t count
  – vertical min: count

Edge Tables

• edge table (ET)
  – store edges sorted by \(y\) in linked list

• active edge table (AET)
  – active: currently used for computation
  – store active edges sorted by \(x\)
    • update each scanline, store ET values + current_\(x\)
    • for each scanline (from bottom to top)
      – do AET bookkeeping
      – traverse AET (from leftmost \(x\) to rightmost \(x\))
        – draw pixels if parity odd

Edge Table Bookkeeping

• setup: sorting in \(y\)
  – bucket sort, one bucket per pixel
  – add: simple check of \(\text{ET}[\text{current}_y]\)
  – delete edges if \(\text{edge.ymax} > \text{current}_y\)

• main loop: sorting in \(x\)
  – for polygons that do not self-intersect, order of edges does not change between two scanlines
  – so insertion sort while adding new edges suffices
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

Interpolation During Scanconvert

- interpolate values between vertices
  - z values
  - r,g,b colour components
  - u,v texture coordinates
  - \(N_x, N_y, N_z\) surface normals

- three equivalent methods (for triangles)
  1. bilinear interpolation
  2. plane equation
  3. barycentric coordinates

1. Bilinear Interpolation

- interpolate quantity along left-hand and right-hand edges, as a function of y
  - then interpolate quantity as a function of x
- only triangles guarantee orientation-independent interpolation
- compute efficiently by using known values at previous scanline, previous pixel

2. Plane Equation

- implicit plane equation
  - \(z = f(x,y)\)
- parametric plane equation
  - \(a(x-x_0) + b(y-y_0) + c(z-z_0) = 0\)
- explicit plane equation
  - \(Plane = A \cdot x + B \cdot y + C \cdot z + D\)

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”

- how to compute \(\alpha, \beta, \gamma\) ?
  - use bilinear interpolation or plane equations
  \[ \alpha = a \cdot x + b \cdot y + c \cdot z + d \]
  \[ \beta = ... \]
- once computed, use to interpolate any # of parameters from their vertex values
  \[
  z = \alpha \cdot z_1 + \beta \cdot z_2 + \gamma \cdot z_3 \\
  r = \alpha \cdot r_1 + \beta \cdot r_2 + \gamma \cdot r_3 \\
  g = \alpha \cdot g_1 + \beta \cdot g_2 + \gamma \cdot g_3 \\
  \]
  etc.
Interpolation: Gouraud Shading

• need linear function over triangle that yields original vertex colors at vertices
• use barycentric coordinates for this
  – every pixel in interior gets colors resulting from mixing colors of vertices with weights corresponding to barycentric coordinates
  – color at pixels is affine combination of colors at vertices

\[
\text{Color}(\alpha \cdot \mathbf{x}_1 + \beta \cdot \mathbf{x}_2 + \gamma \cdot \mathbf{x}_3) := \\
\alpha \cdot \text{Color}(\mathbf{x}_1) + \beta \cdot \text{Color}(\mathbf{x}_2) + \gamma \cdot \text{Color}(\mathbf{x}_3)
\]

Computing Barycentric Coords

• how do we find barycentric coordinates for every pixel efficiently?
  – look at a point \( \mathbf{x} \) on a scanline:

\[
\mathbf{x}_1 = \mathbf{x} + \frac{a_1}{a_1 + a_2} ( \mathbf{x}_2 - \mathbf{x}_1 )
= (1 - \frac{a_2}{a_1 + a_2}) \cdot \mathbf{x}_1 + \frac{a_1}{a_1 + a_2} \cdot \mathbf{x}_2
= \frac{a_2}{a_1 + a_2} \cdot \mathbf{x}_1 + \frac{a_1}{a_1 + a_2} \cdot \mathbf{x}_2
\]

Computing Barycentric Coords

thus
\[
\mathbf{x} = \frac{c_2}{c_1 + c_2} \cdot \mathbf{x}_4 + \frac{c_1}{c_1 + c_2} \cdot \mathbf{x}_3
\]
gives
\[
\mathbf{x} = \frac{c_1}{c_1 + c_2} \left( \frac{a_2}{a_1 + a_2} \cdot \mathbf{x}_1 + \frac{a_1}{a_1 + a_2} \cdot \mathbf{x}_2 \right) + \frac{c_2}{c_1 + c_2} \left( \frac{b_1}{b_1 + b_2} \cdot \mathbf{x}_1 + \frac{b_2}{b_1 + b_2} \cdot \mathbf{x}_2 \right)
\]

Interpolation: Gouraud Shading

• we know
  – affine combinatons are invariant under affine transformations
• thus
  – does not matter whether colors are interpolated before or after affine transformations!
  – colors do not shift around on the surface with affine transformations, but stay attached to every surface point

Computing Barycentric Coords

• similarly:

\[
\mathbf{x}_3 = \frac{b_1 + b_2}{b_1 + b_2} \cdot \mathbf{x}_1 + \frac{b_1}{b_1 + b_2} \cdot \mathbf{x}_3
\]

Computing Barycentric Coords

\[
\mathbf{x} = \frac{c_1 + c_2}{c_1} \cdot \mathbf{x}_4 + \frac{c_1}{c_1 + c_2} \cdot \mathbf{x}_3
\]
Computing Barycentric Coords

• can prove correct by verifying barycentric properties
  - \( \alpha + \beta + \gamma = 1 \)
  - \( 0 \leq \alpha, \beta, \gamma \leq 1 \)

Gouraud Shading with Bary Coords

• algorithm
  - modify scanline algorithm for polygon scan-conversion as follows:
    • linearly interpolate colors along edges of triangle to obtain colors for endpoints of span of pixels
    • linearly interpolate colors from these endpoints within the scanline

Transforming Normals

– polygon:
  \[ N = (P_2 - P_1) \times (P_3 - P_1) \]

– assume vertices ordered CCW when viewed from visible side of polygon

– normal for a vertex
  • used for lighting
  • supplied by model (i.e., sphere), or computed from neighboring polygons

Interpolation During Scanconvert

– interpolate values between vertices
  • \( z \) values
  • \( r, g, b \) colour components
  • \( u, v \) texture coordinates
  • \( N_x, N_y, N_z \) surface normals

Computing Normals

– so if points transformed by modelview vector \( M \), can we just transform vector by \( M \) too?

\[
\begin{bmatrix}
N_x' \\
N_y' \\
N_z'
\end{bmatrix} =
\begin{bmatrix}
m_{11} & m_{12} & m_{13} & 0 \\
m_{21} & m_{22} & m_{23} & 0 \\
m_{31} & m_{32} & m_{33} & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
N_x \\
N_y \\
N_z
\end{bmatrix}
\]

University of British Columbia
CPSC 414 Computer Graphics

Transforming Normals
Transforming Normals

- translations OK: \(w=0\) means unaffected
- rotations OK
- uniform scaling OK
- these all maintain direction

\[
\begin{bmatrix}
  m_{11} & m_{12} & m_{13} & T_x \\ m_{21} & m_{22} & m_{23} & T_y \\ m_{31} & m_{32} & m_{33} & T_z \\ 0 & 0 & 0 & 1
\end{bmatrix}
\]

Transforming Normals

- nonuniform scaling does not work
- \(x-y=0\) plane
  - line \(x=y\)
  - normal: \([1,-1,0]\)
  - ignore normalization for now

Planes and Normals

- plane is all points such that \(N \cdot P = 0\)
  or \(N^T P = 0\) (must transpose for matrix mult!)

\[
N = \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix}, \quad P = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}
\]

- explicit form

\[
\text{Plane} = A \cdot x + B \cdot y + C \cdot z + D
\]

Finding Correct Normal Transform

- transform a plane

\[
P \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = MP \quad N' = QN
\]

\(N^T P = 0\)

\((QN)^T (MP) = 0\)

\(N^T Q^T MP = 0\)

\(Q^T M = I\)

thus the normal to any surface has to be transformed by the inverse transpose of the modelling transformation

\[
Q = (M^{-1})^T
\]
Jaggy Lines

- rasterized lines were not smooth
  - "stair-stepping", "jaggies"

Smother Lines

- solution
  - set 2 points with intensity proportional to distance from line

Smoothing Bresenham

- can modify Bresenham alg to do this
  - for every column of pixels, set the two pixels between which the line intersects the column
  - means that decision variable has to be shifted down one pixel \( d = F(x + 1, y + 1) \)
  - increments for E and NE can be determined as before (but results slightly different)
  - \( d \) can directly be used to multiply pixel intensities
    - fully integer implementation possible

General Problem

- "jaggies": undesirable artifact
  - name for general problem is "aliasing"
- solving the general problem
  - "antialiasing"
- theoretical framework
  - sampling, signal processing

Samples

- most things in the real world are continuous
- everything in a computer is discrete
- the process of mapping a continuous function to a discrete one is called sampling
- the process of mapping a discrete function to a continuous one is called reconstruction
- the process of mapping a continuous variable to a discrete one is called quantization
- rendering an image requires sampling and quantization
- displaying an image involves reconstruction

What is a pixel?

- a pixel is not...
  - a box
  - a disk
  - a teeny tiny little light
- a pixel is a point
  - it has no dimension
  - it occupies no area
  - it cannot be seen
  - it can have a coordinate
- a pixel is more than a point, it is a sample
**Pixels**

- point samples
  - pixels have no extent!

**Pixel Display**

- reconstruction yields continuous function
  - displays constructed to create reconstruction
  - footprints can overlap! (e.g. Gaussians)

**Pixels**

- square pixel model
  - Just ONE possible reconstruction function
  - and a really bad one – leads to poor quality

**Anti-Aliasing: Example**

- we tried to sample a line segment so it would map to a 2D raster display
- we quantized the pixel values to 0 or 1
- we saw stair steps, or jaggies

**Line Segments**

- we tried to sample a line segment so it would map to a 2D raster display
- we quantized the pixel values to 0 or 1
- we saw stair steps, or jaggies
Line Segments
- instead, quantize to many shades
- but what sampling algorithm is used?

Area Sampling
- shade pixels according to the area covered by thickened line
- this is unweighted area sampling
- a rough approximation formulated by dividing each pixel into a finer grid of pixels

Unweighted Area Sampling
- primitive cannot affect intensity of pixel if it does not intersect the pixel
- equal areas cause equal intensity, regardless of distance from pixel center to area

Weighted Area Sampling
- unweighted sampling colors two pixels identically when the primitive cuts the same area through the two pixels
- intuitively, pixel cut through the center should be more heavily weighted than one cut along corner

Weighted Area Sampling
- weighting function, $W(x, y)$
  - specifies the contribution of primitive passing through the point $(x, y)$ from pixel center

Images
- an image is a 2D function $I(x, y)$ that specifies intensity for each point $(x, y)$
Sampling and Image
- our goal is to convert the continuous image to a discrete set of samples
- the graphics system’s display hardware will attempt to reconvert the samples into a continuous image: reconstruction

Point Sampling an Image
- simplest sampling is on a grid
- sample depends solely on value at grid points

Point Sampling
- multiply sample grid by image intensity to obtain a discrete set of points, or samples.

Sampling Errors
- some objects missed entirely, others poorly sampled

Fixing Sampling Errors
- **supersampling**
  - take more than one sample for each pixel and combine them
    - how many samples is enough?
    - how do we know no features are lost?