University of British Columbia

CPSC 414 Computer Graphics

Scan Conversion

Week 6, Wed 8 Oct 2003

- recap: polygon scan conversion
- interpolation
- barycentric coords
- transforming normals
- sampling

News

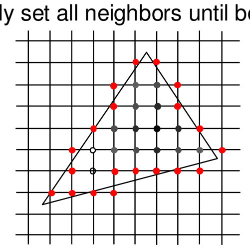
- Homework 1
  - problem 18
    - **x and y transposed** in bottom layer
  - problem 4
    - rotate 30 deg **around x axis** with fixed point of (3,5,12,1)
  - new correct PDF posted
  - reminder: no late work after Fri 17 Oct 9am
    - handin box 18 CICSR basement
- Project 1
  - solution, hall of fame 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

Simple Polygon Scanconversion

- flood fill: simple, slow
  - start with *seed point*
  - recursively set all neighbors until boundary is hit

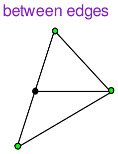


Scanline Algorithms

- given vertices, fill in the pixels

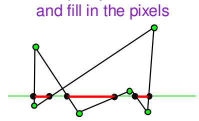
triangles

- split into two regions
- fill in between edges



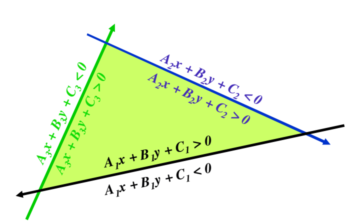
arbitrary polygons (non-simple, non-convex)

- 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



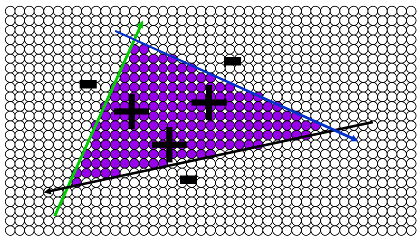
Edge Equations

- define triangle as intersection of three positive half-spaces:



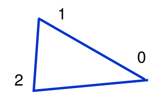
Edge Equations

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



Edge Equation Consistency

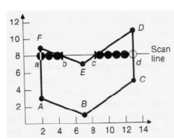
- how to get same +/- state for all equations?
  - consistent counterclockwise vertex traversal
    - good: [0,1], [1,2], [2,0] or [0,2], [2,1], [1,0]
    - bad: [0,1], [2,1], [2,0]



- how to ensure interior is positive?
  - explicit area test: if negative, flip all param signs
    - if (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
    - at ymin, store ymax, xmin, slope
- 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 EAT bookkeeping
    - traverse EAT (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 ET[current_y]
  - delete edges if edge.ymax > 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

University of British Columbia

CPSC 414 Computer Graphics

Interpolation

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

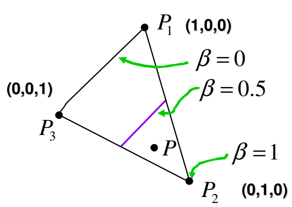
- implicit plane equation
  - z = f(x,y)
- parametric plane equation
  - $a(x-x_0)+ b(y-y_0)+c(z-z_0) = 0$
- explicit plane equation
  - $Plane = A\cdot x + B\cdot y + C\cdot z + D$

3. Barycentric Coordinates

- weighted combination of vertices

$$\begin{cases} P = \alpha\cdot P_1 + \beta\cdot P_2 + \gamma\cdot P_3 \\ \alpha+\beta+\gamma = 1 \\ 0 \le \alpha,\beta,\gamma \le 1 \end{cases}$$

"convex combination of points"



Barycentric Coordinates

- how to compute α,β,γ ?
  - use bilinear interpolation or plane equations

interpolate α,β,γ just like we did for z

$$\alpha = a\cdot x + b\cdot y + c\cdot z + d$$
$$\beta = \ldots$$

  - 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.

Interpolatation: 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

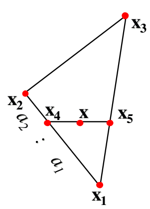
$$Color(\alpha \cdot \mathbf{x}_1 + \beta \cdot \mathbf{x}_2 + \gamma \cdot \mathbf{x}_3) :=$$
$$\alpha \cdot Color(\mathbf{x}_1) + \beta \cdot Color(\mathbf{x}_2) + \gamma \cdot Color(\mathbf{x}_3)$$

Interpolatation: 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

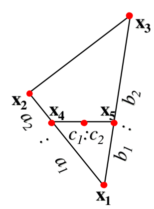
- how do we find barycentric coordinates for every pixel efficiently?
  - look at a point **x** on a scanline:



$$\mathbf{x}_4 = \mathbf{x}_1 + \frac{a_1}{a_1 + a_2}(\mathbf{x}_2 - \mathbf{x}_1)$$
$$= (1 - \frac{a_1}{a_1 + a_2}) \cdot \mathbf{x}_1 + \frac{a_1}{a_1 + a_2} \cdot \mathbf{x}_2$$
$$= \frac{a_2}{a_1 + a_2} \cdot \mathbf{x}_1 + \frac{a_1}{a_1 + a_2} \cdot \mathbf{x}_2$$

Computing Barycentric Coords

- similarly:



$$\mathbf{x}_5 = \frac{b_2}{b_1 + b_2} \cdot \mathbf{x}_1 + \frac{b_1}{b_1 + b_2} \cdot \mathbf{x}_3$$

$$\mathbf{x} = \frac{c_2}{c_1 + c_2} \cdot \mathbf{x}_4 + \frac{c_1}{c_1 + c_2} \cdot \mathbf{x}_5$$

Computing Barycentric Coords

combining

$$\mathbf{x} = \frac{c_2}{c_1 + c_2} \cdot \mathbf{x}_4 + \frac{c_1}{c_1 + c_2} \cdot \mathbf{x}_5 \qquad \mathbf{x}_4 = \frac{a_2}{a_1 + a_2} \cdot \mathbf{x}_1 + \frac{a_1}{a_1 + a_2} \cdot \mathbf{x}_2$$

$$\mathbf{x}_5 = \frac{b_2}{b_1 + b_2} \cdot \mathbf{x}_1 + \frac{b_1}{b_1 + b_2} \cdot \mathbf{x}_3$$

gives

$$\mathbf{x} = \frac{c_2}{c_1 + c_2} \cdot \left(\frac{a_2}{a_1 + a_2} \cdot \mathbf{x}_1 + \frac{a_1}{a_1 + a_2} \cdot \mathbf{x}_2 \right) + \frac{c_1}{c_1 + c_2} \cdot \left(\frac{b_2}{b_1 + b_2} \cdot \mathbf{x}_1 + \frac{b_1}{b_1 + b_2} \cdot \mathbf{x}_3 \right)$$

Computing Barycentric Coords

thus

$$\mathbf{x} = \alpha \cdot \mathbf{x}_1 + \beta \cdot \mathbf{x}_2 + \gamma \cdot \mathbf{x}_3$$

$$\text{with } \alpha = \frac{c_2}{c_1 + c_2} \cdot \frac{a_2}{a_1 + a_2} + \frac{c_1}{c_1 + c_2} \cdot \frac{b_2}{b_1 + b_2},$$

$$\beta = \frac{c_2}{c_1 + c_2} \cdot \frac{a_1}{a_1 + a_2}$$

$$\gamma = \frac{c_1}{c_1 + c_2} \cdot \frac{b_1}{b_1 + b_2}$$

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

University of British Columbia

CPSC 414 Computer Graphics

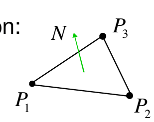
Transforming Normals

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

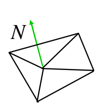
Computing Normals

- 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



Transforming Normals

- what is a normal?
  - a **direction**
  - earlier: vector as direction

$$\begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix}$$

  - so if points transformed by modelview vector M, can we just transform vector by M too?

$$\begin{bmatrix} Nx' \\ Ny' \\ Nz' \\ 0 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & T_x \\ m_{21} & m_{22} & m_{23} & T_y \\ m_{31} & m_{32} & m_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Nx \\ Ny \\ Nz \\ 0 \end{bmatrix}$$

Transforming Normals

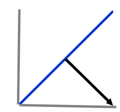
$$\begin{bmatrix} x' \\ y' \\ z' \\ 0 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & T_x \\ m_{21} & m_{22} & m_{23} & T_y \\ m_{31} & m_{32} & m_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix}$$

- translations OK: w=0 means unaffected
- rotations OK
- uniform scaling OK

- these all maintain direction

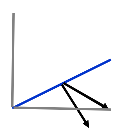
Transforming Normals

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



Transforming Normals

- apply nonuniform scale: stretch along x by 2
  - new plane x = 2y
- transformed normal



$$\begin{bmatrix} 2 \\ -1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0 \end{bmatrix}$$

  - x = -2y or x+2y=0
  - not perpendicular!
  - should be 2x = -y

Planes and Normals

- plane is all points such that $\mathbf{N} \bullet \mathbf{P} = 0$
  or $\mathbf{N}^{\mathrm{T}} \mathbf{P} = 0$ (must transpose for matrix mult!)

$$\mathbf{N} = \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} \mathbf{P} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- explicit form

$$Plane = A \cdot x + B \cdot y + C \cdot z + D$$

Finding Correct Normal Transform

- transform a plane

$$\begin{matrix} P \\ N \end{matrix} \longrightarrow \begin{matrix} P' = MP \\ N' = QN \end{matrix}$$

if we know M, what should Q be?

$N'^{T} P' = 0$ **stay perpendicular**

$(QN)^T (MP) = 0$ **substitute from above**

$N^T \underbrace{Q^T M}P = 0$ $\quad (\mathbf{AB})^{\mathrm{T}} = \mathbf{B}^{\mathrm{T}} \mathbf{A}^{\mathrm{T}}$

$Q^T M = I$ $\quad N^T P = 0$

$\boxed{Q = \left(M^{-1}\right)^T}$

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

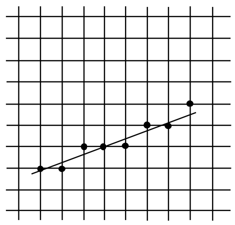
University of British Columbia

CPSC 414 Computer Graphics

Sampling and Antialiasing

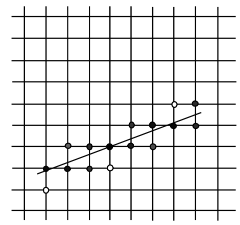
Jaggy Lines

- rasterized lines were not smooth
  - “stair-stepping”, “jaggies”



Smoother 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

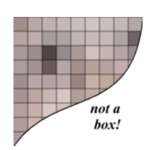
- “jaggies”: undesirable artifact
  – name for general problem is “aliasing”
- solving the general problem
  – “antialiasing”
- theoretical framework
  – sampling, signal processing

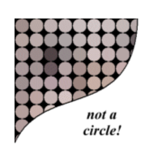
Samples

- 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

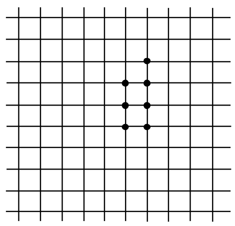




a pixel is *more* than a point, it is a *sample*

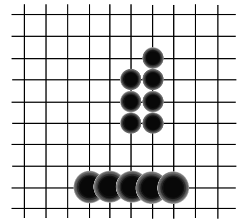
Pixels

- point samples
  – pixels have no extent!



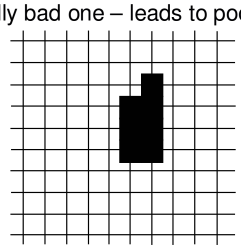
Pixel Display

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

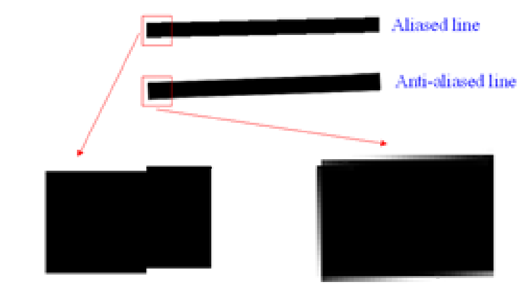


Pixels

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



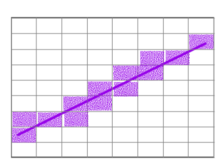
Anti-Aliasing: Example



Samples

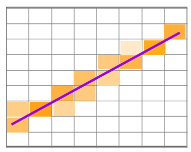
Line Segments

- 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



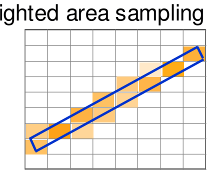
Line Segments

- 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

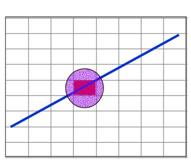
- primitive cannot affect intensity of pixel if it does not intersect the pixel
- equal areas cause equal intensity, regardless of distance from pixel center to area

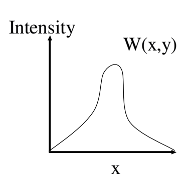
Weighted Area Sampling

- 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

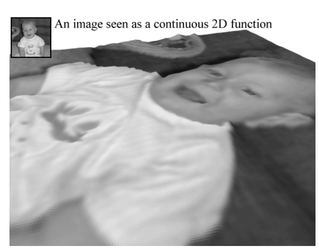
- weighting function, W(x,y)
  - specifies the contribution of primitive passing through the point (x, y) from pixel center





Images

- an image is a 2D function I(x, y) that specifies intensity for each point (x, y)

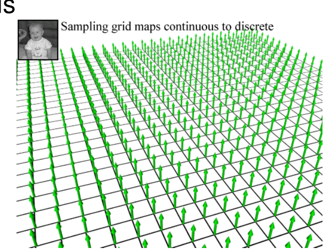


Sampling and Image

- 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*

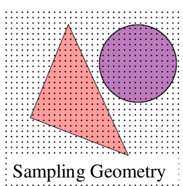
Point Sampling an Image

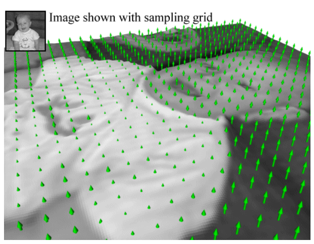
- simplest sampling is on a grid
- sample depends solely on value at grid points



Point Sampling

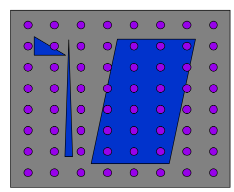
- multiply sample grid by image intensity to obtain a discrete set of points, or samples.





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?