**Projector Gamuts**

- how to handle colors outside gamut?
  - one way: construct ray to white point, find closest displayable point within gamut

**Device Color Gamuts**

- gamut is polygon, device primaries at corners
  - defines reproducible color range
  - X, Y, and Z are hypothetical light sources, no device can produce entire gamut

**RGB Color Space (Color Cube)**

- define colors with (r, g, b) amounts of red, green, and blue
  - used by OpenGL
  - hardware-centric
  - RGB color cube sits within CIE color space
  - subset of perceivable colors
  - scale, rotate, shear cube

**HSV Color Space**

- more intuitive color space for people
  - H = Hue
  - S = Saturation
  - V = Value
  - X = Value of black/white
  - Y = luminance (same as CIE)
  - RGB color cube sits within CIE color space
  - subset of perceivable colors
  - scale, rotate, shear cube

**HSI/HSV and RGB**

- color model used for color TV
  - Y is luminance (same as CIE)
  - I & Q are color (not same I as HSI)
  - using Y backwards compatible for B/W TVs
  - conversion from RGB is linear
  - expressible with matrix multiply
  - green is much lighter than red, and red lighter than blue

**Color Gamut Mapping**

- how to handle colors outside gamut?
  - one way: construct ray to white point, find closest displayable point within gamut

**Color Gamut Mapping**

- how to handle colors outside gamut?
  - one way: construct ray to white point, find closest displayable point within gamut

**Projector Gamuts**

- how to handle colors outside gamut?
  - one way: construct ray to white point, find closest displayable point within gamut
vischeck.com
• simulates color vision deficiencies
Deuteranope
Protanope
Tritanope
Normal vision

Stroop Effect
• red
• blue
• orange
• purple
• green
• red
• blue
• orange
• purple

Color/Lightness Constancy
• color perception depends on surrounding
• colors in close proximity
• simultaneous contrast effect
• illumination under which the scene is viewed

Rendering Pipeline

Scan Conversion - Rasterization
• convert continuous rendering primitives into discrete fragments/pixels
  • lines
    • midpoint/Bresenham
  • triangles
    • flood fill
    • scanline
    • implicit formulation
    • interpolation

Midpoint Algorithm
• we're moving horizontally along x direction
  • only two choices: draw at current y value, or move up vertically to y+1
  • check if midpoint between two possible pixel centers above or below line
  • candidates
    • top pixel: (x+1, y)
    • bottom pixel: (x+1, y+1)
  • midpoint: (x+1, y.5)
• check if midpoint above or below line
  • below: pick top pixel
  • above: pick bottom pixel
• key idea behind Bresenham
  • [demo]

Making It Fast: Reuse Computation
• midpoint: if f(x+1, y+.5) < 0 then y = y+1
• on previous step evaluated f(x-1, y-.5)
• f(x+1, y) = f(x,y) + (y0 - y1)
• f(x+1, y+.5) = f(x,y) + (y0 + 1) - (y1 + .5)
• d = d + 2(y0 - y1)

Color/Lightness Constancy
• automatic “white balance” from change in illumination
  • vast amount of processing behind the scenes!
  • colorimetry vs. perception

Color/Lightness Constancy
• color perception depends on surrounding
• colors in close proximity
• simultaneous contrast effect
• illumination under which the scene is viewed

Stroop Effect
• red
• blue
• orange
• purple
• green
• blue
• green
• purple
• red
• orange
• interplay between cognition and perception

Rasterization
• convert continuous rendering primitives into discrete fragments/pixels
• lines
• triangles
• flood fill
• scanline
• implicit formulation
• interpolation

Making It Fast: Integer Only
• avoid dealing with non-integer values by doubling both sides

Rasterizing Polygons/Triangles
• basic surface representation in rendering
• why?
  • lowest common denominator
    • can approximate any surface with arbitrary accuracy
    • all polygons can be broken up into triangles
  • guaranteed to be:
    • planar
    • triangles - convex
  • simple to render
  • can implement in hardware

Basic Line Drawing
• line (x0,y0) to (x1,y1)
  • begin
  • float dx, dy, x, y, slope
  • dy = y1 - y0
  • slope = dy / dx
  • x = x0
  • for x from x0 to x1, do
  • y = y0 + i*dy / dx
  • draw Pixel (x,y, Round (y));
  • end

Scan Conversion
• given vertices in DCS, fill in the pixels
• display coordinates required to provide scale for discretization
  • [demo]
Triangulating Polygons
• simple convex polygons
  • trivial to break into triangles
  • pick one vertex, draw lines to all others not immediately adjacent
• concave or non-simple polygons
  • more effort to break into triangles
  • simple approach may not work
  • OpenGL can support at extra cost
    • gluNewTess(), gluTessCallback(), ...

General Polygon Rasterization
• idea: use a parity test for each scanline
  edgeCnt = 0;
  for each pixel on scanline
    if (oldpixel + newpixel crosses edge)
      edgeCnt ++;
    // draw the pixel if edgeCnt odd
    if (edgeCnt % 2)
      setPixel(pixel);
• drawbacks?

Making It Fast: Bounding Box
• smaller set of candidate pixels
  • loop over xmin, xmax and ymin, ymax instead of all x, all y

Interpolation During Scan Conversion
• drawing pixels in polygon requires interpolating many values between vertices
  • r,g,b colour components
  • use for shading
  • z values
  • u,v texture coordinates
  • \( N_x, N_y, N_z \) surface normals
  • equivalent methods (for triangles)
    • bilinear interpolation
    • barycentric coordinates

Bilinear Interpolation
• interpolate quantity along \( L \) and \( R \) edges, as a function of \( y \)
  • then interpolate quantity as a function of \( x \)

Barycentric Coordinates
• non-orthogonal coordinate system based on triangle itself
  • origin: \( P_1 \), basis vectors: \( (P_2-P_1) \) and \( (P_3-P_1) \)
  \[ P = P_1 + \beta(P_2-P_1) + \gamma(P_3-P_1) \]
  \[ \beta(0) \gamma(1) \]

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

Flood Fill Drawbacks
• pixels visited up to 4 times to check if already set
  • need per-pixel flag indicating if set already
  • must clear for every polygon!

Scanline Algorithms
• scanline: a line of pixels in an image
  • set pixels inside polygon boundary along horizontal lines one pixel apart vertically

Triangle Rasterization Issues
• exactly which pixels should be lit?
  • pixels with centers inside triangle edges
• what about pixels exactly on edge?
  • draw them: order of triangles matters (it shouldn’t)
  • don’t draw them: gaps possible between triangles
  • need a consistent (if arbitrary) rule
  • example: draw pixels on left or top edge, but not on right or bottom edge
  • example: check if triangle on same side of edge as offscreen point

Interpolation
• interpolating many values between vertices
  • r,g,b colour components
• z values
• u,v texture coordinates
• \( N_x, N_y, N_z \) surface normals
• equivalent methods (for triangles)
  • bilinear interpolation
  • barycentric coordinates

Problem
• input: closed 2D polygon
  • problem: fill its interior with specified color on graphics display
  • assumptions
    • simple - no self intersections
    • simply connected
  • solutions
    • flood fill
    • edge walking
  • OpenGL supports automatically
    • (GL_POLYGON) ...

Flood Fill
• start with seed point
  • recursively set all neighbors until boundary is hit
Barycentric Coordinates
- non-orthogonal coordinate system based on triangle itself
  - origin: $P_1$, basis vectors: $(P_2-P_1)$ and $(P_3-P_1)$
  
  $P = P_1 + \beta (P_2-P_1) + \gamma (P_3-P_1)$
  $P = \alpha P_1 + \beta P_2 + \gamma P_3$

- with
  
  $\alpha = 0 \rightarrow P = P_1$
  $\alpha = 1 \rightarrow P = P_2$
  $\alpha = 0 \rightarrow P = P_1$
  $\beta = 1 \rightarrow P = P_3$

Using Barycentric Coordinates
- weighted combination of vertices
  - smooth mixing
  - speedup
  - compute once per triangle

$P = \alpha \cdot P_1 + \beta \cdot P_2 + \gamma \cdot P_3$

$0 \leq \alpha, \beta, \gamma \leq 1$ for points inside triangle

"Convex combination of points"

Deriving Barycentric From Bilinear
- from bilinear interpolation of point $P$ on scanline
  
  $P = P_1 + \frac{d_1}{d_1 + d_2} (P_2 - P_1)$

Barycentric Coordinates From Bilinear
- similarly

$P = P_1 + \frac{h_1}{h_1 + h_2} (P_2 - P_1)$

$P = (1 - \frac{h_1}{h_1 + h_2}) P_1 + \frac{h_1}{h_1 + h_2} P_2$

Gives

$P = \frac{c_1 \cdot P_1 + c_2 \cdot P_2}{c_1 + c_2} = \frac{c_1 \cdot P_1 + c_1 \cdot c_2}{c_1 + c_2}$

Thus $P = \alpha P_1 + \beta P_2 + \gamma P_3$ with

$\alpha = \frac{c_1}{c_1 + c_2} \frac{b_1}{b_1 + b_2}$

$\beta = \frac{c_2}{c_1 + c_2} \frac{b_2}{b_1 + b_2}$

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

Can verify barycentric properties

$\alpha + \beta + \gamma = 1$, \hspace{1em} $0 \leq \alpha, \beta, \gamma \leq 1$

Computing Barycentric Coordinates
- 2D triangle area
  - half of parallelogram area
    - from cross product

$A = A_{P_1} + A_{P_2} + A_{P_3}$

$\alpha = A_{P_2} / A$

$\beta = A_{P_3} / A$

$\gamma = A_{P_1} / A$