Review

- Rasterization: from screen coordinates (floats) to frame buffer (ints)
- Scan conversion of lines
  - DDA algorithm
  - Bresenham’s incremental algorithm
Scan Conversion of Polygons

- Multiple tasks for scan conversion
  - Filling polygon (inside/outside)
  - Pixel shading (color interpolation)
  - Blending (accumulation, not just writing)
  - Depth values (z-buffer hidden-surface removal)
  - Texture coordinate interpolation (texture mapping)

- Hardware efficiency critical
- Many algorithms for filling (inside/outside)
- Much fewer that handle all tasks well

Filling Convex Polygons

- Find top and bottom vertices
- List edges along left and right sides
- For each scan line from top to bottom
  - Find left and right endpoints of span, xl and xr
  - Fill pixels between xl and xr
  - Can use Bresenham’s alg. to update xl and xr
Other Operations

• Pixel shading (Gouraud)
  – Bilinear interpolation of vertex colors

• Depth values (z-Buffer)
  – Bilinear interpolation of vertex depth
  – Read, and write only if visible
  – Preserve depth (final orthographic projection)

• Texture coordinates u and v
  – Rational linear interpolation to avoid distortion
  – $u(x,y) = \frac{(Ax+By+C)}{(Dx+Ey+F)}$ similarly for $v(x,y)$
  – Two divisions per pixel for texture mapping
  – Due to perspective transformation

Concave Polygons: Odd-Even Test

• Approach 1: odd-even test

• For each scan line
  – Find all scan line/polygon intersections
  – Sort them left to right
  – Fill the interior spans between intersections

• Parity rule: inside after an odd number of crossings
Concave Polygons: Winding Rule

- Approach 2: winding rule
- Orient the lines in polygon
- For each scan line
  - Winding number = right-hdd – left-hdd crossings
  - Interior if winding number non-zero
- Different only for self-intersecting polygons

Concave Polygons: Tessellation

- Approach 3: divide non-convex, non-flat, or non-simple polygons into triangles
- OpenGL specification
  - Need accept only simple, flat, convex polygons
  - Tessellate explicitly with tessellator objects
  - Implicitly if you are lucky
- GeForce3 scan converts only triangles
Boundary Cases

• Boundaries and special cases require care
  – Cracks between polygons
  – Parity bugs: fill to infinity
• Intersections on pixel: set at beginning, not end
• Shared vertices: count $y_{\min}$ for parity, not $y_{\max}$
• Horizontal edges: don’t change parity

Edge/Scan Line Intersections

• Brute force: calculate intersections explicitly
• Incremental method (Bresenham’s algorithm)
• Caching intersection information
  – Edge table with edges sorted by $y_{\min}$
  – Active edges, sorted by x-intersection, left to right
• Process image from smallest $y_{\min}$ up
Flood Fill

- Draw outline of polygon
- Color seed
- Color surrounding pixels and recurse
- Must be able to test boundary and duplication
- More appropriate for drawing than rendering

Outline

- Scan Conversion for Polygons
- Antialiasing
- Compositing
Aliasing

- Artefacts created during scan conversion
- Inevitable (going from continuous to discrete)
- Aliasing (name from digital signal processing): we sample a continues image at grid points
- Effect
  - Jagged edges
  - Moire patterns

Moire pattern from sandlotscience.com

More Aliasing
Antialiasing for Line Segments

- Use area averaging at boundary

- (c) is aliased, magnified
- (d) is antialiased, magnified
- Warning: these images are sampled on screen!

Antialiasing by Supersampling

- Mostly for off-line rendering (e.g., ray tracing)
- Render, say, 3x3 grid of mini-pixels
- Average results using a filter
- Can be done adaptively
  - Stop if colors are similar
  - Subdivide at discontinuities
Supersampling Example

- Other improvements
  - Stochastic sampling (avoiding repetition)
  - Jittering (perturb a regular grid)

Pixel-Sharing Polygons

- Another aliasing error
- Assign color based on area-weighted average
- Interaction with depth information
- Use accumulation buffer or $\alpha$-blending
Temporal Aliasing

- Sampling rate is frame rate (30 Hz for video)
- Example: spokes of wagon wheel in movie
- Possible to supersample and average
- Fast-moving objects are blurred
- Happens automatically in video and movies
  - Exposure time (shutter speed)
  - Memory persistence (video camera)
  - Effect is motion blur

Motion Blur

- Achieve by stochastic sampling in time
- Still-frame motion blur, but smooth animation
Motion Blur Example

T. Porter, Pixar, 1984
16 samples/pixel

Outline

- Scan Conversion for Polygons
- Antialiasing
- Compositing
Accumulation Buffer

• OpenGL mechanism for supersampling or jitter
• Accumulation buffer parallel to frame buffer
• Superimpose images from frame buffer
• Copy back into frame buffer for display

\[
\gamma_A \chi_{\text{er}}(\Gamma_A \chi_{\text{XM}} \chi_{\text{YPF}} \chi_{\text{EP}} \chi_{\text{BIT}});
\]
\[
\text{for } (i = 0; i < \text{n} \mu \chi_{\text{em}} \mu \chi_{\text{ges}}; i++) \{
\gamma_A \chi_{\text{er}}(\Gamma_A \chi_{\text{FMLP}} \chi_{\text{YPF}} \chi_{\text{EP}} \chi_{\text{BIT}}, \Gamma_A \chi_{\text{OPH}} \chi_{\text{YPF}} \chi_{\text{EP}} \chi_{\text{BIT}});
\delta \chi_{\text{lam}} \mu \chi_{\text{e}}(i);
\gamma_A \chi_{\text{hym}}(\Gamma_A \chi_{\text{XM}}, 1.0/(\delta \chi_{\text{me}})) \text{nu} \mu \chi_{\text{ges}});
\}
\gamma_A \chi_{\text{hym}}(\Gamma_A \chi_{\text{PETYP}}, 1.0);
\]

Filtering and Convolution

• Image transformation at pixel level
• Represent N \times M image as matrix \( A = [a_{ik}] \)
• Process each color component separately
• Linear filter produces matrix \( B = [b_{ik}] \) with

\[
b_{ik} = \sum_{j=-m}^{m} \sum_{l=-n}^{n} a_{ij} h_{i-j,k-l}
\]

• \( B \) is the result of convolving \( A \) with filter \( H \)
• Represent \( H \) by n \times m convolution matrix
Filters for Antialiasing

- Averaging pixels with neighbors

\[ H = \frac{1}{5} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \]

- For antialiasing: weigh center more heavily

\[ H = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \]

Filter for Depth-of-Field

- Simulate camera depth-of-field
  - Keep plane \( z = z_f \) in focus
  - Keep near and far planes unchanged

- Move viewer by \( \Delta x \)

- Compute \( x'_{\min}, x'_{\max}, y'_{\min}, y'_{\max} \) for new frustum
Depth-of-Field Jitter

- Compute

\[ x_{\text{min}}' = x_{\text{min}} + \frac{\Delta x}{z_f} (z_f - z_{\text{min}}) \]

- Blend the two images in accumulation buffer

Blending

- Frame buffer
  - Simple color model: R, G, B; 8 bits each
  - \( \alpha \)-channel A, another 8 bits
- Alpha determines opacity, pixel-by-pixel
  - \( \alpha = 1 \): opaque
  - \( \alpha = 0 \): transparent
- Blend translucent objects during rendering
- Achieve other effects (e.g., shadows)
Image Compositing

• Compositing operation
  – Source: \[ s = [s_r, s_g, s_b, s_a] \]
  – Destination: \[ d = [d_r, d_g, d_b, d_a] \]
  – \( b = [b_r, b_g, b_b, b_a] \) source blending factors
  – \( c = [c_r, c_g, c_b, c_a] \) destination blending factors
  – \( d' = [b_r s_r + c_r d_r, b_g s_g + c_g d_g, b_b s_b + c_b d_b, b_a s_a + c_a d_a] \)

• Overlay n images with equal weight
  – Set \( \alpha \)-value for each pixel in each image to \( 1/n \)
  – Source blending factor is “\( \alpha \)”
  – Destination blending factor is “1”

Blending in OpenGL

• Enable blending
  \( \gamma \)Εναβλε(ΓΛ_ΒΛΕΝΔΩ);

• Set up source and destination factors
  \( \gamma \)ΕλενδΦυνδ(σουρχε_φαχτορ, δεστ_φαχτορ);

• Source and destination choices
  – GL_ONE, GL_ZERO
  – GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA
  – GL_DST_ALPHA, GL_ONE_MINUS_DST_ALPHA
Blending Errors

- Operations are not commutative
- Operations are not idempotent
- Interaction with hidden-surface removal
  - Polygon behind opaque one should be culled
  - Translucent in front of others should be composited
  - Solution: make z-buffer read-only for translucent polygons with `glDepthMask(GL_FALSE);`

Antialiasing Revisited

- Single-polygon case first
- Set $\alpha$-value of each pixel to covered fraction
- Use destination factor of “$1 - \alpha$”
- Use source factor of “$\alpha$”
- This will blend background with foreground
- Overlaps can lead to blending errors
Antialiasing with Multiple Polygons

- Initially, background color $C_0$, $\alpha_0 = 0$
- Render first polygon; color $C_1$, fraction $\alpha_1$
  - $C_d = (1 - \alpha_1)C_0 + \alpha_1C_1$
  - $\alpha_d = \alpha_1$
- Render second polygon; assume fraction $\alpha_2$
- If no overlap (a), then
  - $C'_d = (1 - \alpha_2)C_d + \alpha_2C_2$
  - $\alpha'_d = \alpha_1 + \alpha_2$

Antialiasing with Overlap

- Now assume overlap (b)
- Average overlap is $\alpha_1\alpha_2$
- So $\alpha_d = \alpha_1 + \alpha_2 - \alpha_1\alpha_2$
- Make front/back decision for color as usual
Antialiasing in OpenGL

- Avoid explicit $\alpha$-calculation in program
- Enable both smoothing and blending

\[
\text{γλΕναβλε(ΓΛ_POINT_SMOOTH);}\n\text{γλΕναβλε(ΓΛ_LINE_SMOOTH);}\n\text{γλΕναβλε(ΓΛ_ΒΛΕΝΔ);}\n\text{γλΒλενδΦυνχ(ΓΛ_ΣΡΧ_ΑΛΠΗΑ, ΓΛ_ONE_MINYS_ΣΡΧ_ΑΛΠΗΑ);}\n\]

Depth Cueing and Fog

- Another application of blending
- Use distance-dependent ($z$) blending
  - Linear dependence: depth cueing effect
  - Exponential dependence: fog effect
  - This is not a physically-based model

\[
\text{ΓΛφλοατ φχολορ[4] = {...};}\n\text{γλΕναβλε(ΓΛ_ΦΟΓ);}\n\text{γλΦογψ(ΓΛ_ΦΟΓ_ΜΟΔΕ; ΓΛ_ΕΞΠ);}\n\text{γλΦογψ(ΓΛ_ΦΟΓ_ΔΕΝΣΙΤΨ, 0.5);}\n\text{γλΦογψ(ΓΛ_ΦΟΓ_ΧΟΛΟΡ, φχολορ);}\n\]

[Example: Fog Tutor]
Summary

• Scan Conversion for Polygons
  – Basic scan line algorithm
  – Convex vs concave
  – Odd-even and winding rules, tessellation

• Antialiasing (spatial and temporal)
  – Area averaging
  – Supersampling
  – Stochastic sampling

• Compositing
  – Accumulation buffer
  – Blending and $\alpha$-values

Preview

• Assignment 5 extended to Friday night
• Assignment 6 out tonight, due next Thursday
• Next topics:
  – More on image processing and pixel operations
  – Ray tracing