Rasterization

Scan Conversion
Antialiasing
[Ch 7.8-7.11, 8.9-8.12]
Rasterization (scan conversion)

- Final step in pipeline: rasterization
- From screen coordinates (float) to pixels (int)
- Writing pixels into frame buffer
- Separate buffers:
  - depth (z-buffer),
  - display (frame buffer),
  - shadows (stencil buffer),
  - blending (accumulation buffer)
Rasterizing a line
**Digital Differential Analyzer (DDA)**

- Represent line as

\[ y = mx + h \quad \text{where} \quad m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{\Delta y}{\Delta x} \]

- Then, if \( \Delta x = 1 \) pixel, we have \( \Delta y = m \Delta x = m \)
Digital Differential Analyzer

• Assume `write_pixel(int x, int y, int value)`
  
  ```
  for (i = x1; i <= x2; i++)
  {
      y += m;
      write_pixel(i, round(y), color);
  }
  ```

• Problems:
  – Requires floating point addition
  – Missing pixels with steep slopes: slope restriction needed
Digital Differential Analyzer (DDA)

- Assume $0 \leq m \leq 1$
- Exploit symmetry
- Distinguish special cases

But still requires floating point additions!
Bresenham’s Algorithm I

- Eliminate floating point addition from DDA
- Assume again $0 \leq m \leq 1$
- Assume pixel centers halfway between integers
Bresenham’s Algorithm II

• Decision variable $a - b$
  – If $a - b > 0$ choose lower pixel
  – If $a - b \leq 0$ choose higher pixel

• Goal: avoid explicit computation of $a - b$

• Step 1: re-scale $d = (x_2 - x_1)(a - b) = \Delta x(a - b)$

• $d$ is always integer
Bresenham’s Algorithm III

• Compute $d$ at step $k+1$ from $d$ at step $k$!
• Case: $j$ did not change ($d_k > 0$)
  – $a$ decreases by $m$, $b$ increases by $m$
  – $(a - b)$ decreases by $2m = 2(\Delta y/\Delta x)$
  – $\Delta x(a-b)$ decreases by $2\Delta y$
Bresenham’s Algorithm IV

- Case: j did change ($d_k \leq 0$)
  - $a$ decreases by $m-1$, $b$ increases by $m-1$
  - $(a - b)$ decreases by $2m - 2 = 2(\Delta y/\Delta x - 1)$
  - $\Delta x(a-b)$ decreases by $2(\Delta y - \Delta x)$
Bresenham’s Algorithm V

• So \( d_{k+1} = d_k - 2\Delta y \) if \( d_k > 0 \)
• And \( d_{k+1} = d_k - 2(\Delta y - \Delta x) \) if \( d_k \leq 0 \)
• Final (efficient) implementation:

```c
void draw_line(int x1, int y1, int x2, int y2) {
    int x, y = y0;
    int dx = 2*(x2-x1), dy = 2*(y2-y1);
    int dydx = dy-dx, D = (dy-dx)/2;

    for (x = x1 ; x <= x2 ; x++) {
        write_pixel(x, y, color);
        if (D > 0) D -= dy;
        else {y++; D -= dydx;}
    }
}
```
Bresenham’s Algorithm VI

• Need different cases to handle $m > 1$
• Highly efficient
• Easy to implement in hardware and software
• Widely used
Outline

• Scan Conversion for Lines
• Scan Conversion for Polygons
• Antialiasing
Scan Conversion of Polygons

• Multiple tasks:
  – **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 is 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 bottom to top
  - Find left and right endpoints of span, $x_l$ and $x_r$
  - Fill pixels between $x_l$ and $x_r$
  - Can use Bresenham’s alg. to update $x_l$ and $x_r$
**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**
Edge vs Scan Line Intersections

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

- Approach 2: 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
- Most modern GPUs scan-convert only triangles
Flood Fill

- Draw outline of polygon
- Pick 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 Lines
• Scan Conversion for Polygons
• Antialiasing
Aliasing

- Artifacts 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

No antialiasing
Antialiasing for Line Segments

• Use area averaging at boundary

(a) (b) (c) (d)

• (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)
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 with real hardware (photo and video cameras)
  - Exposure time (shutter speed)
  - Memory persistence (video camera)
  - Effect is motion blur
Wagon Wheel Effect

Source: YouTube
Motion Blur Example

Achieve by stochastic sampling in time

T. Porter, Pixar, 1984
16 samples / pixel / timestep
Summary

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

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