Computer Graphics
Scan Conversion & Anti-Aliasing

Univ.-Prof. Dr. Matthias Harders

Summer semester 2015

Content

- Scan Conversion
- Anti-Aliasing
Scan Conversion

- Central step of rasterization-based rendering
- Turning geometric primitives into sets of fragments (pixels) in screen space

---

Scan Conversion

- Fragment/pixel is point sample of continuous 2D function
- Final coordinates typically rounded to integer grid

Scan Conversion of Lines

- Different strategies to select fragments
- Option: choose all intersected grid squares

Scan Conversion of Lines

- Different strategies to select fragments
- Option: represent line as rectangle and determine overlapped fragment (i.e. centers)
Scan Conversion of Lines

- Different strategies to select fragments
- Option: determine intersections with diamond grid around fragments

Bresenham Algorithm

- Developed by J. Bresenham (1962), originally for drawing lines on plotters
- More general version: Midpoint algorithm (Pitteway); identical for lines and circles
- Key idea: step in $x$- or $y$-major direction and select one of two candidate fragments, based on decision variable
Line Representation

- Line equation
  \[ y = mx + b = \frac{\Delta y}{\Delta x} x + b = \frac{y_1 - y_0}{x_1 - x_0} x + b \]
  \[ y(x_1 - x_0) = (y_1 - y_0)x + (x_1 - x_0)b \]
  \[ p_0 = (x_0, y_0) \quad x_1 - x_0 \]
  \[ p_1 = (x_1, y_1) \quad y_1 - y_0 \]

- Implicit line equation
  \[ f(x, y) = \begin{pmatrix} y_1 - y_0 \\ x_0 - x_1 \end{pmatrix}^T \begin{pmatrix} x \\ y \end{pmatrix} + c = 0 \]
  \[ = \mathbf{n}^T \begin{pmatrix} x \\ y \end{pmatrix} + c = 0 \]
  \[ p_0 = (x_0, y_0) \quad \mathbf{n} = \begin{pmatrix} n_1 \\ n_2 \end{pmatrix} \]
  \[ p_1 = (x_1, y_1) \]

Line Representation

- Implicit line equation

\[ f(x, y) < 0 \]

\[ f(x, y) = 0 \]

\[ f(x, y) > 0 \]

Bresenham (Midpoint) Algorithm

- Limitation to one plane octant and direction

\[ 0 < m \leq 1 \land x_0 < x_i \]

- Arbitrary lines handled similarly (symmetries)
**Bresenham (Midpoint) Algorithm**

- Determine decision variable
  \[ d = f(x_m, y_m) = f(x_p + 1, y_p + 1/2) \]
- Choose fragment NE if \( d \geq 0 \), else choose E

![Bresenham Diagram](image)

**Incremental Update of Decision Variable**

- Determine \( d_{i+1} \) from \( d_i \)
- Update in case of E \( d_{i+1} = d_i + \Delta y \)
- Update in case of NE \( d_{i+1} = d_i + \Delta y - \Delta x \)

![Incremental Update Diagram](image)
Line Drawing Pseudo-Algorithm

- Note: multiplication by 2 to avoid division (integer-only implementation); also only one octant

```c
dx = x1 - x0; dy = y1 - y0; /* Init */
incr_E = 2*dy; incr_NE = 2*(dy - dx);
d = 2*dy - dx;
x = x0; y = y0;
draw (x,y);
while (x < x1): /* Iterate */
    x++;
    if (d > 0):
        y++;
        d += incr_NE;
    else
        d += incr_E;
draw (x,y);
```

Updates in Octants

- y++; x -= 0/1;
- y++; x += 0/1;
- x--; y += 0/1;
- x--; y -= 0/1;
- x++; y += 0/1;
- x++; y -= 0/1;
- y--; x -= 0/1;
- y--; x += 0/1;
Remarks on Bresenham Algorithm

- Resulting line depends on order of endpoints
- Artifacts due to clipping line at screen border
- Intensity of line depends on slope
- Stair-case appearance of lines ("jaggies")
- Line thickness and stippling

\[ \text{length} = 6\sqrt{2} \]

Scan Conversion of Triangles

- Determine fragments covered by polygon
- Focus on triangles simplifies scan conversion
- Possible strategies: inside-outside tests or scanline-based filling
Inside-Outside Tests

- Determine implicit edge (line) equations
- Check fragments inside triangle bounding box
- Straight-forward parallelization
- Inefficient for thin diagonally oriented triangles

Scanline-Based Filling

- Scan convert edges on scanlines
- Fill fragments left to right between edge fragments
- Advantage of using triangles: all spans inside and maximally one edge switch
- Implemented via sorted edge tables
Concerns with Triangle Scan Conversion

- Naïve assignment is drawing order-dependent
- Decision criteria for shared edges and/or vertices
- Possible solution: prioritize “bottom/left” edges
- Fill fragment for edge with \( (n_x = 0 \land n_y > 0) \lor n_x > 0 \)

\[
\mathbf{v} = \begin{pmatrix} n_x \\ n_y \end{pmatrix}
\]

Concerns with Triangle Scan Conversion

- Long thin triangles (slivers) may exhibit holes
- Moving slivers and triangles may flicker
- Attribute interpolation requires consideration of offset from edge to fragment center
**Aliasing Problem**

- **Aliasing** due to undersampling (below Nyquist rate)
- Discrete nature of fragments/pixels introduces (spatial) aliasing artifacts (staircasing, “jaggies”)
- Signal reconstructed based on samples differs from original continuous signal if undersampled

**Handling Aliasing**

- **Anti-aliasing** techniques try to reduce aliasing artifacts
- Typical strategies: pre-filtering or super-sampling
Pre-Filtering

- Key idea: decrease highest frequency before sampling
- Realize by determining coverage in fragment
- Problem: coverage computation time-consuming

Pre-Filtering Example

No anti-aliasing

Pre-filtering
Super-Sampling

- SSAA (Super-Sampling Anti-Aliasing)
- Realize by taking multiple samples per fragment
- Problem: naïve implementation requires larger framebuffer and multiple fragment evaluations

\[ c_p(x, y) = \sum_{i=1}^{N} \omega_i \cdot c(x_i, y_i) \]

E.g. averaging

Samples may be weighted (different levels of importance, e.g. focus on center)
- Different sampling patterns and resolutions possible
- Carefully selected patterns can result in better images in reduced time
Sampling Patterns Results

- 1 sample
- 1 × 2 sample
- 2 × 1 sample
- Quincunx
- 2 × 2 grid
- 2 × 2 RGSS

Multi-Sampling Anti-Aliasing

- Super-Sampling too costly for most applications
- Alternative: Multi-Sampling Anti-Aliasing (MSAA)
- Focus on geometric anti-aliasing at primitive edges
- Only one fragment shader evaluation, but multiple coverage tests at sample locations
Further Anti-Aliasing Strategies

Lecture Schedule

<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
<th>Remark (Proseminar)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3.</td>
<td>Introduction</td>
<td></td>
</tr>
<tr>
<td>11.3.</td>
<td>Rendering pipeline</td>
<td>Introduction to OpenGL</td>
</tr>
<tr>
<td>18.3.</td>
<td>Transformations &amp; projections</td>
<td>Introduction to OpenGL 2</td>
</tr>
<tr>
<td>25.3.</td>
<td>(no lecture)</td>
<td>(no proseminar)</td>
</tr>
<tr>
<td></td>
<td><em>Easter break</em></td>
<td></td>
</tr>
<tr>
<td>15.4.</td>
<td>Light &amp; color</td>
<td>Theory (Transformation &amp; Projection)</td>
</tr>
<tr>
<td>22.4.</td>
<td>Lighting &amp; shading</td>
<td>Programming (Camera control)</td>
</tr>
<tr>
<td>29.4.</td>
<td>Geometry representation</td>
<td>(National celebration day)</td>
</tr>
<tr>
<td>6.5.</td>
<td>(no lecture)</td>
<td>Theory (Color, Lighting &amp; Shading)</td>
</tr>
<tr>
<td>13.5.</td>
<td>Texturing</td>
<td>(Rector’s day)</td>
</tr>
<tr>
<td>20.5.</td>
<td>Visibility</td>
<td>Programming (Using GLM)</td>
</tr>
<tr>
<td>27.5.</td>
<td>Scan conversion &amp; anti-aliasing</td>
<td>Theory (Geometry &amp; Texturing)</td>
</tr>
<tr>
<td>3.6.</td>
<td>Advanced effects</td>
<td>Programming (Shaders)</td>
</tr>
<tr>
<td>10.6.</td>
<td>GPUs &amp; displays</td>
<td>Theory (Visibility &amp; Scan Conversion)</td>
</tr>
<tr>
<td>17.6.</td>
<td>Wrap-up</td>
<td>Programming (Textures)</td>
</tr>
</tbody>
</table>