Scan line algorithm

• The scan line algorithm is an alternative to the seed fill algorithm.
• It does not require scan conversion of the edges before filling the polygons
• It can be applied simultaneously to a set of polygons rather than filling each polygon individually.
Scan line algorithm

• Use a horizontal scan line that traverses the scene top-down.
• Stop at each pixel row
• For each pixel row
  - Compute the intersections of the polygon edges with the scan line.
  - Sort the intersections from left to right.
  - Start filling at first intersection.
  - Stop filling at second intersection
  - Start filling again at third intersection.
...
Scan line algorithm

- Horizontal scan lines scan scene top-down and stop at each row:

\[(x_k, y_k)\]
\[(x_{k+1}, y_{k+1})\]
Scan line algorithm

- Filling pixels between intersections in an alternating fashion:
Scan line algorithm

Example:
Scan line algorithm

Example:
Data structure for scan line algorithm

- The scan line algorithm is supported by a linked list data structure that stores all the edges that are intersected by the current position of the scan line in the correct order.
- Linked list exploits spatial coherence between successive scan line positions.
- The order of the edges stored in the linked list only changes at distinct points (event points).
Data structure for scan line algorithm

• Event points are:
  - start/end of edge: insert/remove edge into sorted list.
  - edge crossings: flip order of successive edges in sorted list.
    (Only in case of multiple polygons!)
• Start/end points are known from the beginning.
• Edge crossings are checked only between neighboring edges in the sorted list. It is necessary whenever a new neighborhood emerges, i.e., at event points.
**Caveats**

- **Scan line crossing vertex of polygon**
  - Distinguish two cases
  - Case 1: 2 edges start -> remove intersection points
  - Case 2: 1 edge starts & 1 ends -> treat as 1 intersection

- **Horizontal edges**
  - Eliminate horizontal edge and treat neighboring vertices as above.
3.4 Anti-aliasing
Aliasing

Observation:

- Scan converted objects exhibit discretization artifacts such as jagged edges (*staircase effect*).
- These artifacts are called aliasing.
Anti-aliasing

- Anti-aliasing is a technique to alleviate the aliasing problem.

- Idea:
  Replace binary fill/no-fill decision with saturation-based filling.
Anti-aliasing

for each fragment of the framebuffer

• Compute the percentage, by which the fragment is covered by the edge.
• Fill the fragment with a color, whose saturation reflects the coverage percentage.

Remark:
• In order to compute the coverage, one must assign a certain thickness to the edge.
Anti-aliasing

Example:

- When scan converting a black edge in front of a white background, the black edge will be drawn using grayscale colors.
- When scan converting a blue edge in front of a red background, the edge will be drawn using a color palette that successively mixes red and blue.
Aliasing and anti-aliasing
Anti-aliasing of polygonal meshes
Anti-aliasing of entire scenes
Moiré effect and anti-aliasing
Coverage computation

• How do we compute the coverage?
• Geometrically:

\[
\text{coverage} = \frac{\text{edge area}}{\text{fragment area}}
\]

- precise
- expensive

• Speed is more important than accuracy
\[\rightarrow\text{compute an approximate solution!}\]
Supersampling anti-aliasing

Idea:

• Sample each fragment multiple times.
• Determine at each sample position, whether the sample lies inside or outside the object.
• The percentage of samples lying inside the object approximates the coverage of that fragment by the object.
Subpixel supersampling

- Divide each fragment into a number of subpixels.
- Apply standard scan conversion to subpixel image.
- Determine coverage as percentage of colored subpixels of each fragment.
- Fill the fragment with the edge color using the respective percentage as saturation.
Subpixel supersampling

Remarks:
• 2x2 or 4x4 subpixel supersampling are common.
• Sometimes a rotated subpixel grid is used (rotation by 45°) to eliminate artifacts occurring for regular patterns (cf. Moiré effect).
Stochastic supersampling

- Select a number of sample locations (pseudo-)randomly within each fragment.
- Perform an inside-outside test for each sample.
- Determine coverage as percentage of samples detected as lying inside the object.
- Fill the fragment with the edge color using the respective percentage as saturation.
Stochastic supersampling

Remarks:

• Stochastic supersampling further decreases the risk of artifacts due to regular pattern.
• Its performance depends on the computation time for the inside/outside test.
• It is suitable for implicit surfaces.
  - Example: For spheres one would only have to compare the distance of the sample to the sphere's center with its radius.
Supersampling on shaded objects

- Average of inside/outside property at samples to obtain coverage percentage (as before).
- Average color property at samples to obtain color for fragment filling.
Supersampling hardware implementations

- **Subpixel supersampling (GeForce 2):**
  - Increased memory requirements.

- **Multisampling anti-aliasing (GeForce 3):**
  - Use a mask with 5 or 9 samples to compute coverage.
  - The mask determines the sampling pattern.
  - Use only 2 or 4 of these samples to compute shaded color.

- **Coverage sampling anti-aliasing (GeForce 8):**
  - Introducing coverage samples to compute coverage.
  - Decouple shaded color samples and coverage samples.
Supersampling hardware implementations

- no anti-aliasing
- multisampling (4x)
- coverage sampling (16x)
Postfiltering anti-aliasing

• Alternative to supersampling anti-aliasing.

• Idea:
Signal processing uses low-pass filtering for smoothing, which corresponds to local averaging.

• Approach:
Scan convert many projected scenes, which are subpixel transformations of the original scene, and average the resulting images.
Postfiltering anti-aliasing

Scan conversion after subpixel transformation:

\[
\frac{(5 \times \text{original} + 1 \times \text{shifted vertically} + 1 \times \text{shifted horizontally})}{7} = \text{post-filtered anti-aliased image}
\]
Supersampling vs. post-filtering

• Post-filtering is applied to the entire scene equally, while supersampling can be applied locally to certain fragments.
  For example, supersampling can be restricted to only the silhouettes of an object to not blur surface parts.
• Post-filtering can deal with any geometry, i.e., explicit or implicit object representation, while supersampling is typically targeted at polygonal mesh representation.
Anti-aliasing in OpenGL

• Removing the Jaggies
  - `glEnable( mode )`
    • `GL_POINT_SMOOTH`
    • `GL_LINE_SMOOTH`
    • `GL_POLYGON_SMOOTH`
  - coverage computed by computing supersampling (subpixel approach).
  - coverage stored in alpha channel of RGBA representation.
Full scene antialiasing

- Implementation of post-filtering anti-aliasing
- Each time we move the viewer, the image shifts
  - Different aliasing artifacts in each image
  - Averaging images using accumulation buffer averages out these artifacts
Blending

- Combine pixels with what’s already in the framebuffer
  - `glBlendFunc(src, dst)`

\[
\tilde{C}_r = src \tilde{C}_f + dst \tilde{C}_p
\]
Multi-pass rendering

- Blending allows results from multiple drawing passes to be combined together
  - enables more complex rendering algorithms

Example of bump-mapping done with a multi-pass OpenGL algorithm
Accumulation Buffer

- Problems of compositing into color buffers
  - limited color resolution
    - clamping
    - loss of accuracy
  - Accumulation buffer acts as a "floating point" color buffer
    - accumulate into accumulation buffer
    - transfer results to frame buffer
Accumulation Buffer

- **glAccum(op, value)**
  - operations
    - within the accumulation buffer: **GL_ADD, GL_MULT**
    - from read buffer: **GL_ACCUM, GL_LOAD**
    - transfer back to write buffer: **GL_RETURN**
  - **glAccum(GL_ACCUM, 0.5)** multiplies each value in read buffer (specified by glReadBuffer()) by 0.5 and adds to accumulation buffer