Antialiasing

Outline
- Aliasing and antialiasing
- Taxonomy of antialiasing approaches
- Some details

Emphasis is on geometry antialiasing
- Will not cover image resampling explicitly
- But good texture filtering may be more important than geometry antialiasing:
  - Crawling textures are really horrible
  - Users sometimes experience texture filtering as full-scene antialiasing
What is “aliasing”? 

Result of sampling below the Nyquist rate = aliasing
For example:

More generally it is imperfect signal reconstruction
- Reconstruction without sinc basis
- Low-precision arithmetic
- Other sloppiness

Prefiltering

Graphics primitives have infinite frequency content
For example, the cross section of a line is a box

The filtered cross section is much smoother

And could be sampled correctly, but how to do it?
Integration

We implement prefiltering with an integration filter

So antialiased frame buffer contents are high-quality samples, to be reconstructed by the visual system

Unfortunately the ideal filter has infinite extent ☹

So we approximate with imperfect filters

- Finite domain
- Incomplete sampling

Antialiasing = no more jaggies

CS448 Lecture 8
Kurt Akeley, Pat Hanrahan, Spring 2007
Antialiasing system goals

Good static and dynamic image quality
- Avoid sudden frame-to-frame changes
- Avoid false motion (crawling)
- Avoid negative-training (e.g., pulsing aircraft on horizon)
Reasonable ...
- Hardware and performance costs
- Implementation and application complexity
Integration with other GPU features
- Depth buffer for occlusion computation
- Stencil buffer
- Transparency?

Taxonomy of antialiasing methods

Two fundamental approaches, based on what coverage info is
- Contained in each fragment, and
- Stored in each pixel in the frame buffer
Fractional
- No geometric information, just “coverage” percentages
  - Note: coverage may be weighted
- OpenGL “smooth” antialiasing
Geometric
- Some geometric information
  - Point sampling
  - Area sampling
- OpenGL “multisample” antialiasing
Each approach has strengths and weaknesses
Ideal jaggie removal - integration

Define a 2-D spatial filter function for a pixel

- Probably not a box filter (though may be)
- Finite, otherwise unworkable
- Empirical, depends on display properties

Render image into an infinite-precision shapes buffer

- Hidden geometry is somehow eliminated,
- Leaving exact geometry and color information

Integrate filter function with geometry/color info for each pixel
Fractional antialiased points

Compute weighted coverage by integrating:
- Point “geometry”
- With the filter function of each pixel that intersects the point geometry

Pixel filter function
**Fractional antialiased points**

Compute weighted coverage by integrating:

- Point “geometry”
- With the filter function of each pixel that intersects the point geometry

---

**Pixel filter function**
Hardware implementation

For each point size and sub-pixel point location

- Pre-integrate for each affected pixel
- Store results in a hardware table

![Diagram](image)

Hardware implementation

Reduce table size by

- Limiting the number of supported point sizes
  - Small maximum point size reduces “pixel” count too
- Limiting $x_{\text{screen}}y_{\text{screen}}$ sub-pixel resolution
- Exploiting symmetry
  - Horizontal
  - Vertical
  - Diagonal

Normalize total point intensity for each point size

- Same sum of pixel weights for each $x_{\text{screen}}y_{\text{screen}}$ offset
- Avoid frame-to-frame strobe effects for moving points
- Gamma correction is critical for optimal image quality!
Frame buffer operations

Point on background
- Blend point color with background color
- Use fragment alpha to specify blend weights
  \[ C'_p = A_f C_f + (1 - A_f) C_p \]

Point intersecting point
- Geometric relationship is unknown
  - Best guess is random relationship
- Use same blend function!
  - Call this blend function “uncorrelated”
- Works recursively for all points
- No need to retain alpha in pixels

Fractional antialiased points

Strengths
- Excellent static and dynamic image quality
  - Point overlaps are stable if not accurate
  - Strobing effects are eliminated by aggregate intensity normalization
- Simple and inexpensive to implement and use
  - Frame buffer gets blend function, no added storage
- Operates with depth and stencil buffers

Weaknesses
- Depth buffer yields non-optimal results
  - Nearer small coverage replaces farther large coverage
Fractional antialiased lines

Table is larger
- Line width, offset to pixel center, slope
- Must exploit symmetry for reasonable table size
- End-of-line filtering can be very expensive

Iterate $1 \times n$ function

Pixel filter function
Fractional antialiased lines

Table is larger

- Line width, offset to pixel center, slope
- Must exploit symmetry for reasonable table size
- End-of-line filtering can be very expensive

Iterate $1 \times n$ function

Pixel filter function
Fractional antialiased lines

Strengths

- Very good static and dynamic image quality
  - Line overlaps are stable if not accurate
  - Roping effects are eliminated by aggregate intensity normalization
- Simple and inexpensive to implement and use
  - Frame buffer gets blend function, no added storage
- (Barely) operates with depth and stencil buffers

Weaknesses

- Depth buffer yields very non-optimal results
  - Nearer small coverage replaces farther large coverage
  - Colors interact badly (especially depth cueing)

Fractional antialiased triangles

Difficult to pre-compute coverage integrations

- Edge slopes OK, but
- Vertexes introduce two edge slopes, and
- Small triangles have all 3 edges in play!

Blending approximation breaks down completely

- Uncorrelated blend leaves visible seams
- Adjacent triangles are anti-correlated, not uncorrelated
Anti-correlated blend functions

Assuming perfect tiling, depth complexity 1.0
- E.g., 2-D rendering (a clock face, for example)
  \[ C'_p = A_f C_f + C_p \]

Assuming nearest-to-farthest primitive sorting
- Special case of 3-D rendering
- Requires addition of alpha channel in frame buffer
  \[ i = \min(A_f, (1 - A_p)) \]
  \[ A'_p = A_p + i \]
  \[ C'_p = i C_f + C_p \]

Fractional antialiased triangles

Strengths
- Produces useful results in very specialized circumstances
- Requires minimal frame buffer additions
  - Anti-correlated (saturation) blend, alpha buffer

Weaknesses
- Filter quality is poor
  - Table is impossible to implement, so
  - Convolution is typically with a box filter
- Difficult and expensive to implement
- Fails entirely with depth buffer or stencil buffer
- Imposes unreasonable sorting burden on application
  - Cannot handle polygon interpenetrations
Demonstration

Fractional antialiasing summary

Great for single-colored dot clouds
Good for lines
  - High-quality filtering, but
  - Problems with line-line intersections
Almost useless for triangles
  - Expensive to implement
  - Filtering quality is poor
  - Depth buffer fails completely
  - Application burden is unacceptable
Taxonomy

Antialiasing
  ▪ Fractional
  ▪ Geometric
    ▪ Point Sampled
    ▪ Area Sampled

Multi-pass accumulation buffer AA

Advantages
▪ Linear performance/quality ratio
▪ Simple to implement
▪ Simple to use (no sort, interpenetration OK, ...)
▪ Point sample pattern is arbitrary
▪ “Free” anisotropic texture filtering ....

Disadvantages
▪ Solves “jaggies”, but not “aliasing”
▪ Shading is too expensive
  ▪ REYES shades just once or twice per pixel
  ▪ Perception: NTSC chroma vs. luminance bandwidth
Multi-pass accumulation buffer AA

Key Disadvantage
- Computation and bandwidth are replicated
  - Computation: application, geometry, esp. shading
  - Bandwidth
    Application \(\rightarrow\) GPU
    GPU \(\rightarrow\) frame buffer

Transistors are cheap, communication is expensive

Multisample antialiasing

Specify the location of multiple sample points per pixel
- Patterns may differ spatially, but not temporally

Rasterize fragments that include
- A bitmask of occluded samples
- Appropriate color, depth, and texture coords

Evaluate texture once per fragment (not per sample)

Store color and depth for each sample in frame buffer

Resolve samples to final pixel value either
- Each time the pixel is modified, or
- Once, before the buffer is displayed
Multisample vs. supersample

Terminology is not agreed upon
I prefer

Supersample → Shading computed for each sample
Multisample → Shading computed once per fragment

Sample pattern is implementation dependent

Supersample implementations can be
- Multipass, or
- Single pass, but repeated shading operations

Multisample sample pattern

Trade-off
- Random (pseudo random)
  - Better, more efficient filtering
- Uniform
  - Easier, more efficient rasterization

Compromise: subset(s) of uniform
- Manageable sample count
- Empirically best
  - Optimal for horizontal and vertical
  - Good for everything else
- Pixar owns patent on this 😊

8-rooks pattern

Single Pixel
**Multisample sample pattern**

**Trade-off**

- **Box filter**
  - No sharing of samples
  - Full utilization of each sample
  - Good quality with low sample counts
  - No boundary issues

- **Shaped filter**
  - Best for higher sample count
  - May require pixel-pixel communication

How does sample sharing affect filtering? (e.g., Quincunx)

---

**Rasterization fragment selection**

Box sampled, as in tiled rasterization
- The bitmask is composed of point samples

Pixel’s box must enclose all sample locations
- Might be outside the 1 x 1 ideal pixel area

Look how pixel depth complexity has increased!
- Area is 5.0, but 13 fragments are generated
Rasterization attribute assignment

Sample depth at each occluded sample location
- Depth buffer controls “geometry” of final image

Sample color once per fragment
- Do not sample outside the triangle!
- Choose a sample location in a repeatable manner
  - Occluded sample nearest to pixel center
  - Occluded sample nearest to “fragment” center

Sample texture coordinates once per fragment
- Pixel center - optimizes for adjacent triangles
- Color sample location - optimizes for silhouette

Multisample frame buffer

Store full depth and color values for each sample
- Various clever compressions are possible

Execute full fragment operations for each sample
- Depth buffer
- Stencil buffer
- Blending
- ...

Resolve to final color
- Only final color buffers are double buffered

Examples: high-end SGI machines
8-Sample multisample frame buffer

typedef struct {
    int red, green, blue, alpha;
} Color;

typedef struct {
    Color c;
    int depth;
    int stencil;
} Sample;

typedef struct {
    Color front, back;
    Sample s[8];
} Pixel;

---

Ideal Multisample Summary

Strengths
- Good full-scene antialiasing quality
- Works seamlessly with depth/stencil buffers
  - Even works for interpenetrating triangles
- Operation is predictable and reliable

Weaknesses
- Frame buffer is very expensive
  - Bandwidth
  - Memory requirements
- Point and line filter quality is mediocre
  - Fractional approach has higher filter quality
  - But multisample correctly resolves intersections
Merged multisample and fractional

OpenGL Multisample spec designed to allow this:
- Enable multisample frame buffer
- Render triangles in multisample mode
- Then render points and lines in fractional mode
  - OpenGL “smooth”

Result is
- High-quality points and lines
- Good quality filtering of solids
  - No seams or cracks
- Proper occlusion point/line/solid to solid
  - Point/line to point/line occlusion still poor

Architectural options

Store only mask in frame buffer
- Assume front-to-back rendering
  - no interpenetration
- Intersect and accumulate masks to determine blend function
- Examples: early flight simulation IGs

\[
\begin{align*}
i &= (M_f \ AND \ \bar{M}_{fb}) / n \\
M'_{fb} &= M_{fb} \ OR \ M_f \\
C'_{fb} &= i \ C_f + C_{fb}
\end{align*}
\]

Mask rendering

\[
\begin{align*}
i &= \min(A_f, (1 - A_{fb})) \\
A'_{fb} &= A_{fb} + i \\
C'_{fb} &= i \ C_f + C_{fb}
\end{align*}
\]

Anti-correlated rendering
**Architectural options (continued)**

Tiled rendering
- Reduces frame buffer storage requirements
  - Multisample buffers needed only for active tiles
- But requires region binned geometry
  - Extra frame of latency
  - Lots of data management complexity
- Examples
  - Pixel Planes 5, PixelFlow
  - Talisman
  - GigaPixel, ...  

---

**Architectural options (continued)**

Lossless compression
- One- and two-fragment special cases
  - Saves bandwidth, not memory

Almost-lossless compression
- Reduced color precision per sample
  - E.g., 16 8-bit samples reconstruct a 12-bit color value
  - Saves bandwidth and memory

Lossy compression at pixels
- Examples: countless research papers
- Watch out for failure modes!
System evaluation

Simulate possible algorithms
- Run test patterns
- Run scenes from real applications
- Try to break it - application developers will!

Study individual images carefully
More important: study sequences of images!
- Static images do not tell the whole story

Antialiasing is a well-researched subject
- The standards for publication are high
- Document evaluation procedure when publishing

Jaggies vs. aliasing

Supersample/Multisample
- Handle jaggies well
- Cannot eliminate aliasing
  - Input is not band limited, so
  - Finely detailed geometry will alias

Application LOD management addresses this problem
- Deliver geometry of “appropriate” resolution
- Typically done for performance reasons
  - But provides effective “prefilter” as well
- Frame-to-frame coherence is a significant issue
- Close analogy here to MIPmapping
Area sampling

How can one get the good qualities of point sampling

- Accurate depth comparisons
- Samples taken only within triangle boundaries
- “Perfect” anti-correlated blending
- Robust, reliable algorithms

With the greater filter quality of area-based sampling

- Greater spatial resolution
- High-quality, display-tuned filter function

In one algorithm?
I know of no general solution

But there are “hacks” that get some value

2-fragment area filtering

Optimize for special case of just two visible fragments
For each fragment compute
  - Coverage mask
  - Weighted (filter-function-integrated) coverage value
At each pixel store
  - All multisample values
  - One coverage value, and
  - One extra state bit (tracks 2-fragment case)
When merging fragment colors during resolution
  - Use coverage value in 2-fragment case
  - Use multisample values otherwise
Extend to three or more fragments?
“Schilling” antialiasing

Choose samples based on coverage, not just geometry
Good idea, but must sample outside primitive

- Colors wrap
- T-junctions protrude

Blue triangle occludes ¼ of the pixel’s unit area, but only one sample. Select a second sample to get the best “weight”

A-buffer and relatives

Variable data structure at each pixel

Strengths

- Handle transparency and depth occlusion
- Simple for applications to program

Weaknesses

- Complex to implement
  - Memory allocation / deallocation
- Fragile in operation
  - Failure modes are unpleasant!
Summary

Two fundamental approaches to geometry antialiasing

- Fractional - ignore geometry at frame buffer
  - Simple to implement
  - Good quality filters
  - Good for non-geometric primitives, poor for triangles

- Geometric - maintain geometry at frame buffer
  - Expensive to implement
  - Mediocre filter quality for reasonable sample counts
  - Good for triangles, mediocre for non-geometric primitives

Can mix and match to get the best of both

Remember, we didn’t discuss image filtering!

Suggested readings

- Filtering Edges for Gray-Scale Displays, Gupta and Sproull, SIGGRAPH Proceedings ’81.
- Compositing Digital Images, Porter and Duff, SIGGRAPH Proceedings ’84.
- A New Simple and Efficient Antialiasing with Subpixel Masks, Schilling, SIGGRAPH ’91.
- Multisample extension to OpenGL.
Hardware workshop papers


Prefiltered Antialiased Lines Using Half-Plane Distance Functions, McNamara, McCormack, Jouppi, 2000.


High-Quality Rendering Using the Talisman Architecture, Barkans, 1997

Real-Time Graphics Architecture

Lecture 8: Antialiasing

Kurt Akeley
Pat Hanrahan
http://graphics.stanford.edu/cs448-07-spring/