# Comparison of Anti-Aliasing Techniques for Real-Time Applications

Federico Liberatore, Jacob Longazo, Stephen Pettinati, and Devin Weise

# Outline

- Classes of Anti-Aliasing (AA)
- Custom Model Creator
- Super-sampling Anti-Aliasing (SSAA)
- Morphological Anti-Aliasing (MLAA)
- Geometric Post-process Anti-Aliasing (GPAA)
- Comparison of results
- Conclusion

# Classes of Anti-Aliasing

- HW6 implemented super-sampling
- SSAA is a type of Full-Screen Anti-Aliasing
  - Performs AA on all areas of the screen
- Other classes perform AA selectively
  - Clean up artifacts after frame buffer gets rendered (Image Post-processing AA)
  - Use polygon information to find edges where aliasing may appear (Geometric AA)
- Implemented one from each class on top of HW6 code

# Custom Model Creator

- Needed other model files to test with
  - Useful for finding scalability
- Used Maya's FBX exporter and a custom FBX parser
- Wrote secondary application to combine buffer and index files into single .asc file

# Super-sampling Anti-Aliasing (SSAA)

- Implemented other formats of subsampling
  - Ordered Grid
  - Rotated Grid
  - Random Sampling
- Varying amounts of subsamples
- Gaussian Filter

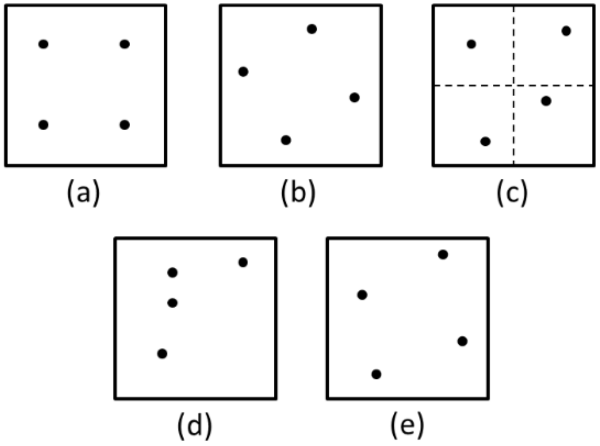
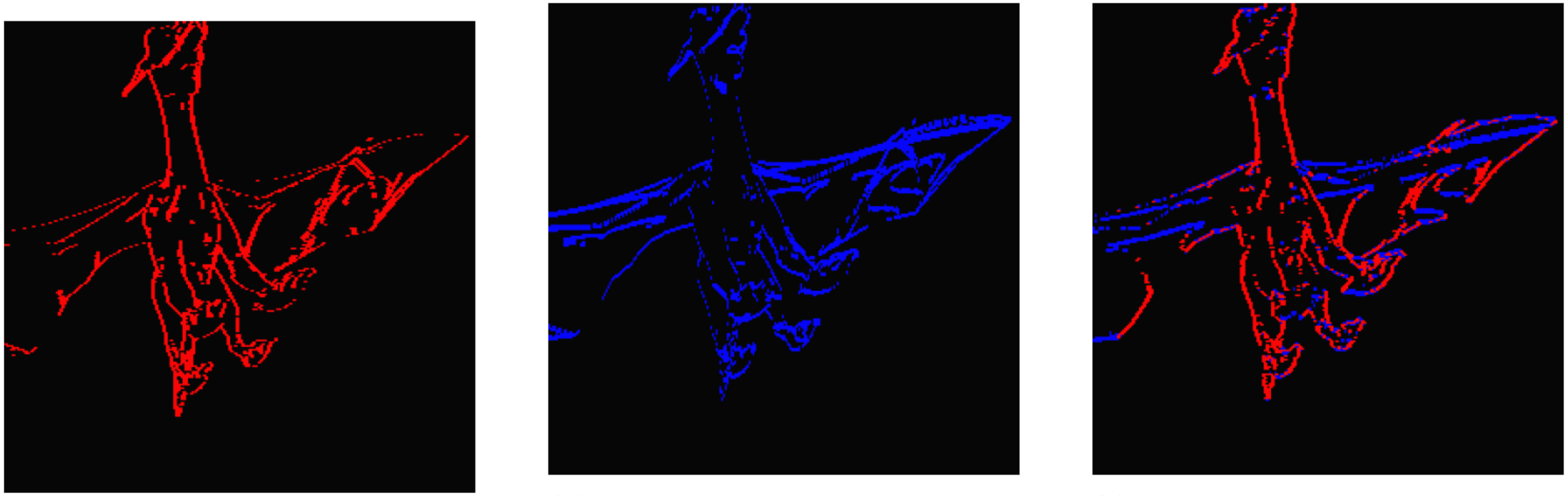


Fig. 8. Super Sampling distributions: (a) Ordered Grid. (b) Rotated Grid. (c) Jittered Grid. (d) Random. (e) Poisson.

# Morphological Anti-Aliasing (MLAA)

- After generating the initial image, look in FB for edges based on threshold value
  - Group edges into shapes (U, Z, or L)
- Deconstruct to L shape, then blur pixels based on approx. area covered by edge



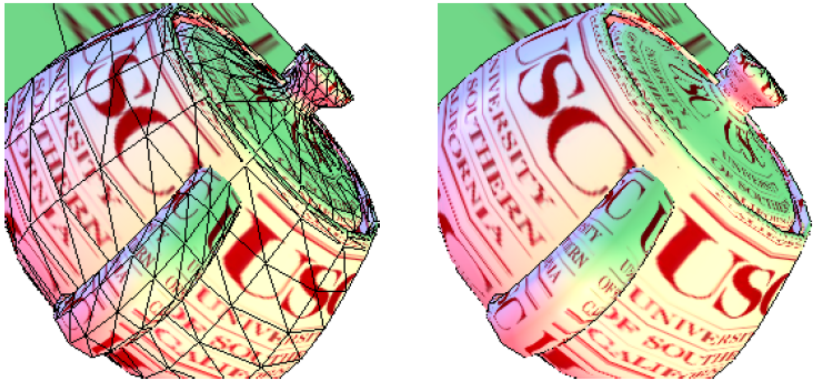
(a) MLAA: Vertical detected edges.

(b) MLAA: Horizontal detected edges.

(c) MLAA: Overlapping vertical and horizontal edges.

# Geometric Post-process Anti-Aliasing (GPAA)

- Use triangle edge information to blur silhouette edges
- Added structures to map pixels to triangle and get edges for rendered triangles.
- Remove edges shared by visible triangles
- Blend pixels based on distance from edge to pixel center



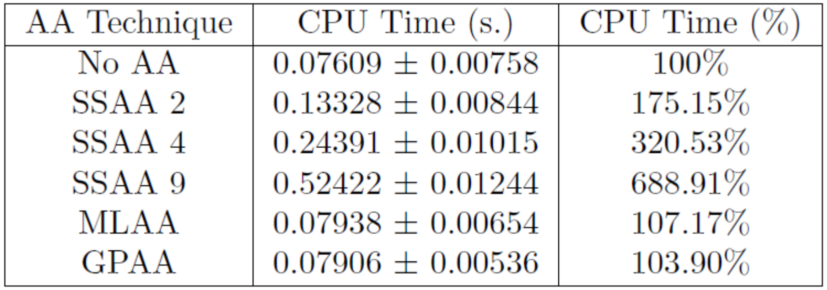
(a) GPAA: Edges for all Rendered Triangles.

(b) GPAA: Silhouette edges, shared edges removed.

Figure 5: GPAA: Edge identification.

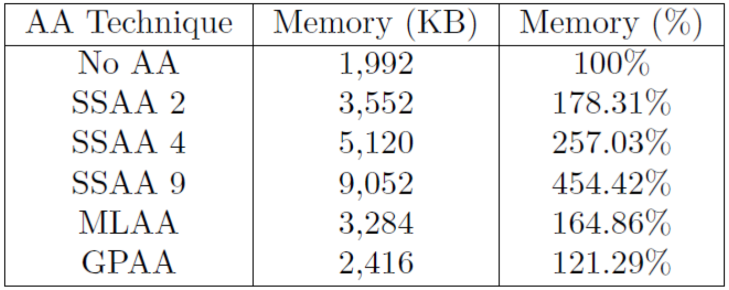
# Technical Comparisons

- Rendered teapot (scene I) and dragon (scene II) and measured CPU and memory usage



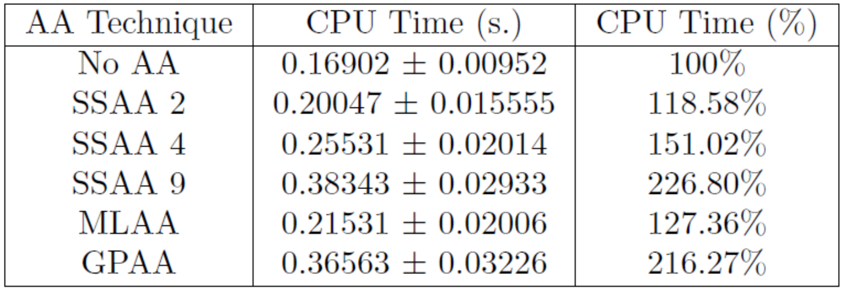
| AA Technique | CPU Time (s.) | CPU Time (%) |
|---|---|---|
| No AA | 0.07609 ± 0.00758 | 100% |
| SSAA 2 | 0.13328 ± 0.00844 | 175.15% |
| SSAA 4 | 0.24391 ± 0.01015 | 320.53% |
| SSAA 9 | 0.52422 ± 0.01244 | 688.91% |
| MLAA | 0.07938 ± 0.00654 | 107.17% |
| GPAA | 0.07906 ± 0.00536 | 103.90% |

Table 3: CPU time for scene I.



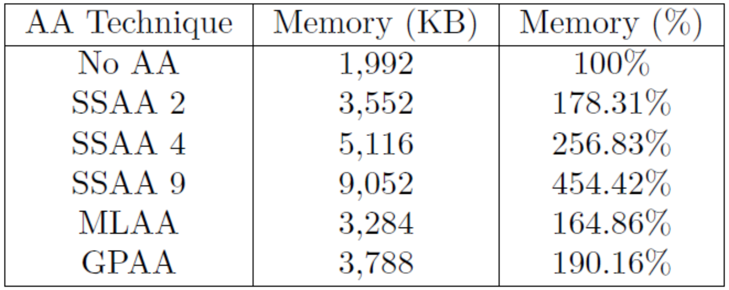
| AA Technique | Memory (KB) | Memory (%) |
|---|---|---|
| No AA | 1,992 | 100% |
| SSAA 2 | 3,552 | 178.31% |
| SSAA 4 | 5,120 | 257.03% |
| SSAA 9 | 9,052 | 454.42% |
| MLAA | 3,284 | 164.86% |
| GPAA | 2,416 | 121.29% |

Table 1: Memory occupation for scene I.



| AA Technique | CPU Time (s.) | CPU Time (%) |
|---|---|---|
| No AA | 0.16902 ± 0.00952 | 100% |
| SSAA 2 | 0.20047 ± 0.015555 | 118.58% |
| SSAA 4 | 0.25531 ± 0.02014 | 151.02% |
| SSAA 9 | 0.38343 ± 0.02933 | 226.80% |
| MLAA | 0.21531 ± 0.02006 | 127.36% |
| GPAA | 0.36563 ± 0.03226 | 216.27% |

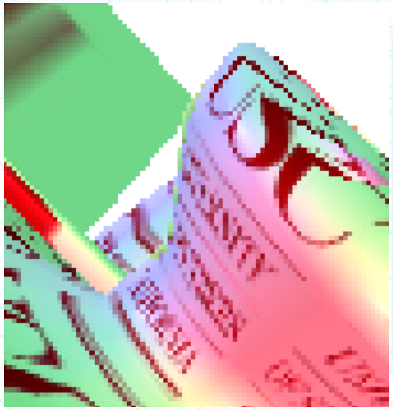
Table 4: CPU time for scene II.



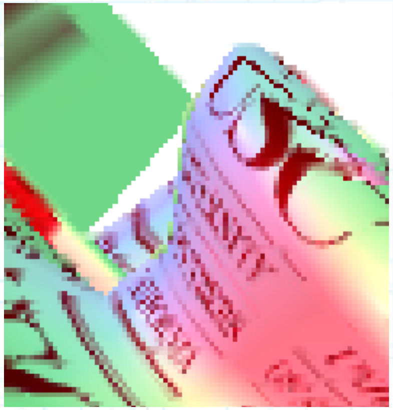
| AA Technique | Memory (KB) | Memory (%) |
|---|---|---|
| No AA | 1,992 | 100% |
| SSAA 2 | 3,552 | 178.31% |
| SSAA 4 | 5,116 | 256.83% |
| SSAA 9 | 9,052 | 454.42% |
| MLAA | 3,284 | 164.86% |
| GPAA | 3,788 | 190.16% |

Table 2: Memory occupation for scene II.

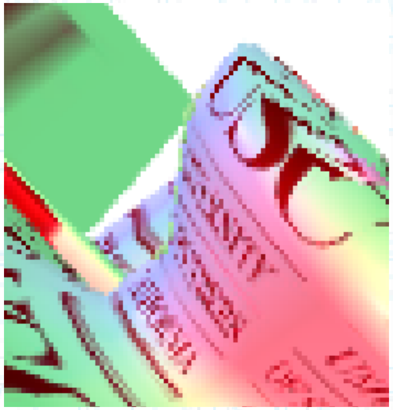
# Visual Comparisons



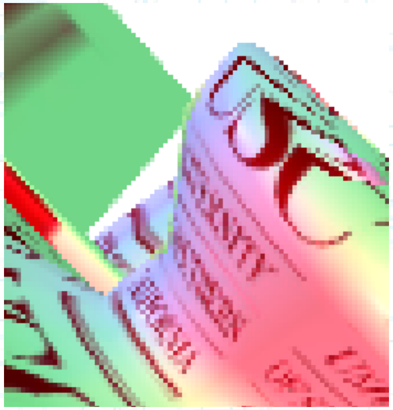
No AA



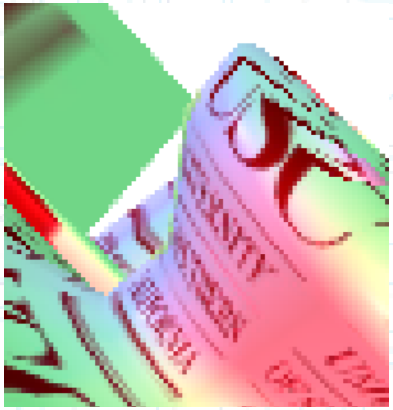
MLAA



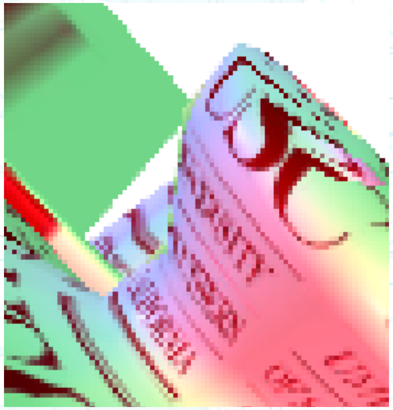
GPAA



OGSS



RGSS



Random SS

# Conclusion

- SSAA can provide good results at the expense of heavy CPU and memory usage.
- MLAA is cheap and scales well with respect to triangle count, but doesn't deliver great image quality.
- GPAA can provide high-quality results, however it scales poorly as scene complexity increases.

# Thank you for your time!