Accumulative Anti-Aliasing

Eric Enderton Eric Lum Christian Rouet Oleg Kuznetsov

NVIDIA

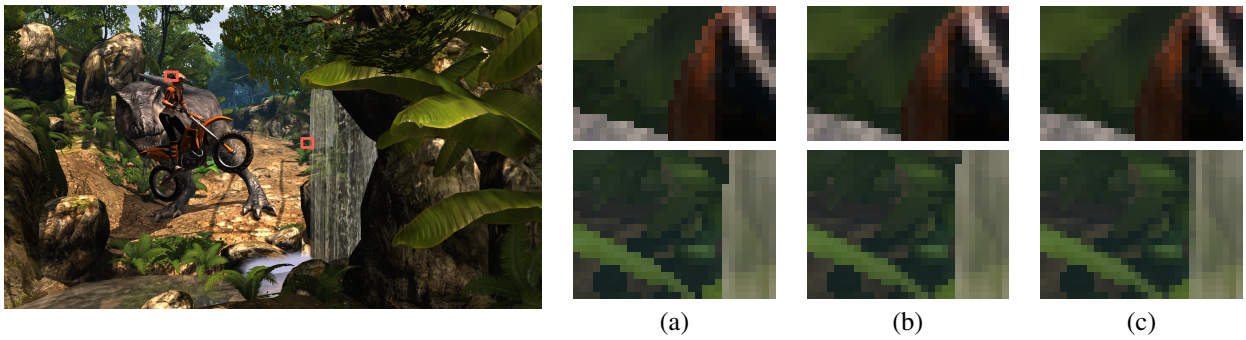

Figure 1: *An ACAA frame, with close-ups comparing (a) Single-sampled image. (b) 8x ACAA image. (c) 8x MSAA image. ACAA renders opaque surfaces nearly identically to MSAA but uses half the framebuffer memory. Scene courtesy of Kishonti Informatics.*

Abstract

Accumulative anti-aliasing (ACAA) is a simple modification of forward-rendered multi-sample anti-aliasing (MSAA). It produces the same image quality but consumes half as much multi-sample framebuffer memory, and reduces both render time and off-chip bandwidth by 20% to 30%. ACAA stores multiple depth samples, computed by a depth-only pre-pass, but stores only one color sample per pixel, which is used to accumulate final color as the sum of shaded fragment colors weighted by visibility. ACAA makes higher sample rates practical, improving image quality.

1 Algorithm and Results

Many anti-aliasing schemes track color and coverage information for multiple visible fragments per pixel, resolving this data to a final color after all scene geometry has been processed. We observe that if visibility is computed and stored in an earlier pass, then color storage can be reduced to one color accumulator per pixel. Each pixel's final color is summed on the fly: As each visible fragment is shaded, its contribution is weighted by the stored visibility.

The ubiquitous example is multi-sample anti-aliasing (MSAA), which stores both depth and color at each of several samples within a pixel. While modern hardware graphics pipelines support MSAA with up to 8 samples per pixel, the memory and performance penalties are generally considered prohibitive for consumer applications such as computer games, and even 4 samples is considered a high-end setting. For forward-rendered game engines, we propose accumulative anti-aliasing (ACAA), which computes multi-sampled color using one color per pixel. First, multi-sampled depth information is rendered during a z pre-pass; this pass is already a typical game engine optimization, to avoid overshading. Second, during the shading pass, each visible fragment is tested against that z-buffer. The resulting post-z coverage indicates the weight of the fragment's color in the accumulated pixel color.

The result is improved performance. Memory savings are half or more the size of the MSAA framebuffer, which can be quite substantial: at 4K resolution, an 8x MSAA framebuffer consumes 759 MB, while 8x ACAA requires only 316 MB. ACAA is particularly efficient on recent GPUs that include hardware support for both post-z coverage and target-independent rasterization (TIR). Here, ACAA boosts performance significantly, recovering 50% of the added frame time of MSAA rendering vs single-sampled rendering, a savings of 20-30% of total render time in our tests. We also measured total bytes read and written to the framebuffer by the GPU during one frame. ACAA recovers 50% of the MSAA penalty here as well, which was 30% of total bandwidth used. This is significant because GPU performance is limited by total power dissipation, and off-chip memory access is one of the highest-power operations.

2 Limitations and Extensions

Without adequate bit precision, intermediate sums of colors can introduce numerical errors. More significantly, if two fragments have numerically coincident z values at a sample, the sample may be double counted. We describe two efficient solutions, one using stencil tests and one using saturated alpha blending.

It is not clear how to anti-alias transparent surfaces with ACAA, as it does not readily distinguish between fragments that abut and fragments that overlap. The water effects in Figure 1 are rendered single-sampled. ACAA does not work well for deferred rendering, as it would merge shader inputs rather than shader outputs.

For mobile GPUs that use tiled rendering to combat MSAA penalties, ACAA lets larger tiles fit in memory, improving efficiency.

ACAA extends naturally to higher image quality. Better, wider filters can be implemented by allowing each fragment to contribute to a neighborhood of pixels. High sample rates can be achieved without extra memory, either with hardware support, or by accumulating multiple ACAA passes with custom sampling patterns.

SIGGRAPH 2015 Talks, August 09 – 13, 2015, Los Angeles, CA.
ACM 978-1-4503-3636-9/15/08.
http://dx.doi.org/10.1145/2775280.2792515

Accumulative Anti-Aliasing

Eric Enderton Eric Lum Christian Rouet Oleg Kuznetsov

NVIDIA

Appendix: Source Code

Source code for the basic ACAA algorithm is publicly available as part of NVIDIA's GameWorks SDK. The code sample is named `BlendedAA`, and is described here: `http://docs.nvidia.com/gameworks/index.html#gameworkslibrary/graphicssamples/opengl_samples/blendedaasample.htm`.

To download, click on "OpenGL Graphics and Compute Samples" here: `http://developer.nvidia.com/gameworksdownload`.