Filtering Approaches for Real-Time Anti-Aliasing

www.iryoku.com/aacourse

Jorge Jimenez[1] Diego Gutierrez[1] Jason Yang[2] Alexander Reshetov[3] Pete Demoreuille[4]
Tobias Berghoff[5] Cedric Perthuis[6] Henry Yu[7] Morgan McGuire[8,9] Timothy Lottes[8]
Hugh Malan[10] Emil Persson[11] Dmitry Andreev[12] Tiago Sousa[13]

[1] Universidad de Zaragoza [2] AMD [3] Intel Labs [4] Double Fine
[5] Sony Computer Entertainment WWS ATG [6] Sony Computer Entertainment WWS Santa Monica [7] Kalloc Studios
[8] NVIDIA [9] Williams College [10] CCP [11] Avalanche Studios [12] Lucas Arts [13] Crytek

Figure 1: *Examples from Star Wars: The Force Unleashed 2 (DLAA) and God of War III (MLAA). Original and anti-aliased images are split horizontally; zoom-in to appreciate the details.*

We encourage downloading the latest version of these course notes in `http://www.iryoku.com/aacourse`.

1 Intro

For more than a decade, Supersample Anti-Aliasing (SSAA) and Multisample Anti-Aliasing (MSAA) have been the gold standard anti-aliasing solution in games. However, these techniques are not well suited for deferred shading or fixed environments like the current generation of consoles. Recently, Industry and Academia have begun to explore alternative approaches, where anti-aliasing is performed as a post-processing step. The original, CPU-based Morphological Anti-Aliasing (MLAA) method gave birth to an explosion of real-time anti-aliasing techniques that rival MSAA. This course will cover the most relevant ones, from the original MLAA to the latest cutting edge advancements. In the following text, we will describe each technique, and establish the conceptual links between them to give the rationale behind the selection of techniques.

One of the first works that awakened the interest in filter-based approaches may be that of Jason Yang and colleagues (*A Directionally Adaptive Edge Anti-Aliasing Filter*), given its implementation at driver level. Using a custom, weighted resolve it increases the number of steps in the gradients generated by MSAA, increasing the perceived anti-aliasing quality. The resolve is done by using the length of the isolines that cross a pixel as subpixel weights. This allows quality equal to a standard hardware resolve with 2 to 3 times less the number of samples. *Jason Yang* (AMD) will present the details of this technique.

Unfortunately, many popular rendering techniques such as deferred shading cannot directly take advantage of MSAA, and are best suited to non-multisampled framebuffers. The advent of MLAA demonstrated that high-quality antialiasing results can be obtained by inferring subpixel coverage and forming a plausible antialiased edge from such a buffer. *Alex Reshetov* (Intel Labs) will explain the core idea of MLAA, and describe the problems that following implementations solve.

Although MLAA was aimed at offline ray-tracers, this changed in late 2009, when the *Playstation 3* game *God of War III* started using MLAA as its AA solution. The algorithm was moved to the PS3's Synergistic Processing Units, freeing up significant amounts of GPU time over MSAA, while delivering very high image quality. The code was subsequently made available to all PS3 developers and is being deployed in games such as *Killzone 3*, *LittleBigPlanet 2*, as well as a host of other titles. *Tobias Berghoff* (SCE WWS ATG) will for the first time reveal the inner workings of this method and share recent improvements and lessons learned.

Cedric Perthuis (SCE WWS Santa Monica) will discuss the integration into God of *War III*, showing the benefits and pitfalls of the technique live on a special build of the game.

However, SPU hardware is only available on the PS3 platform. The work of Jorge Jimenez and colleagues addressed this issue, being the first GPU adaptation that performed in practical execution times for the PC platform, by transforming the algorithm to use texture structures and making extensive use of hardware facilities and precomputation. *Jorge Jimenez* (Universidad de Zaragoza) will explain the mapping of the original algorithm to a GPU, as well as all the details behind Jimenez's MLAA. In concurrent work, Demoreuille devised a very efficient Hybrid CPU/GPU implementation for the Xbox 360, that was deployed with the game *Costume Quest*, the first known game to ship with MLAA in the Xbox 360. *Pete Demoreuille* (Double Fine) will describe this hybrid approach, including edge detection routines and integration issues for Double Fine games.

Conceptually, MLAA-like algorithms consist of: a) detecting edges; b) calculating neighborhood weights; and c) blending with the neighborhood. From now on, a series of alternative techniques will be presented that replace, improve or approximate various of these components, but that are in essence, closely related.

When working with a non-multisampled image, undersampling problems with subpixel features are unavoidable. Chajdas et al. introduced Subpixel Reconstruction Anti-Aliasing (SRAA), which extends MLAA with additional buffers that provide this information at low cost. The key difference with MLAA is that SRAA calculates blending weights from a multisampled depth buffer by using a *continous* edge detection, which allows blending weights to be determined without an expensive pattern search. *Morgan McGuire* (Williams College and NVIDIA) will give an insight into this technique.

Fast approXimate Anti-Aliasing (FXAA) is a cutting-edge, unreleased technique being developed in NVIDIA by Timothy Lottes. FXAA deals with edge aliasing in similar way to MLAA, but to maximize performance it adapts the algorithm to take advantage of specific hardware features. For example, FXAA takes advantage of anisotropic texture fetch for an approximate end of edge search. It can be considered a dual algorithm, as it is able to detect and reduce both sub-pixel and edge aliasing problems. *Timothy Lottes* (NVIDIA) will describe GPU optimizations and FXAA's method to handle sub-pixel aliasing.

In concurrent work with MLAA, Hugh Malan devised Distance-to-Edge Anti-Aliasing (DEAA), which is very similar in spirit. However, DEAA departs from the MLAA pattern search, eliminating it from the pipeline: instead, the forward rendering pass pixel shaders calculate the distance to each triangle edge with subpixel precision, and stores them in a separate rendertarget. The postprocess pass uses this information to derive blend coefficients. *Hugh Malan* (CCP) will present this technique, giving additional results from the initial work to integrate this anti-aliasing technique with the MMO *APB*.

In the middle of 2010, Dmitry Andreev introduced Directionally Localized Anti-Aliasing (DLAA), which was the method of choice for *Star Wars: The Force Unleashed 2*. This technique makes a further simplification over MLAA, by working in a perceptual space: exact gradients produced by the exact covered areas may not be required. Instead of calculating weights, it makes use of vertical and horizontal blurs to produce gradients in the aliased edges. Gradients with different number of steps are used for the two kind of edges considered: short and long. DLAA doesn't replace pattern search but totally eliminates it. *Dmitry Andreev* (Lucas Arts) will present an in-detail description of DLAA.

All the techniques presented so far use spatial filters to prevent aliasing. However, there is also a growing trend towards the use of temporal filters to attenuate aliasing in games, *Halo: Reach* and *Crysis 2* being two pioneers in this respect. In the *Crysis 2* approach, anti-aliasing is calculated by accumulating frames over time, by using temporal reprojection. *Tiago Sousa* (Crytek) will provide all the details of the *Crysis 2* approach, finishing the course with a slightly different approach to make sure the audience knows that the sky is the limit.

The course is aimed at all attendees, from casual users who just want to better understand post-processing anti-aliasing techniques to researchers and game developers, for whom we provide implementation details and code-snippets to allow them to quickly explore their own ideas in this explosively growing area. We believe this course may lead to more coordinated efforts in the following years, and serve as a solid base for future research.

2 About the Lecturers

Jorge Jimenez
Universidad de Zaragoza
http://www.iryoku.com/

Jorge Jimenez is a real-time graphics researcher at the Universidad de Zaragoza, in Spain, where he received his BSc and MSc degrees, and where he is pursuing a PhD in real-time graphics. His interests include real-time photorealistic rendering, special effects, and squeezing rendering algorithms to be practical in game environments. He has various contributions in books and journals, including Transaction on Graphics, where our skin renderings made the front cover of the SIGGRAPH Asia 2010 issue.

Diego Gutierrez
Universidad de Zaragoza
http://giga.cps.unizar.es/~diegog/

Diego Gutierrez is an Associate Professor at the Universidad de Zaragoza, where he got his PhD in computer graphics in 2005. He now leads his group's research on graphics, perception and computational photography. He is an associate editor of three journals, has chaired and organized several conferences and has served on numerous committees including the SIGGRAPH and Eurographics conferences.

Jason Yang
AMD

Jason Yang is a Principal Member of Technical Staff for Advanced Technology Initiatives at AMD where he has contributed to graphics, physics, video, encryption, and GPGPU technologies. Recently he worked on the Radeon 6000 series demo titled "HK2207". He received his BS and PhD from MIT.

Alexander Reshetov
Intel Labs
http://visual-computing.intel-research.net/people/alex.htm

Alex Reshetov received his Ph.D. degree from Keldysh Institute for Applied Mathematics (in Russia). He joined Intel Labs in 1997 as a senior staff researcher after working for two years at the Super-Conducting Super-Collider Laboratory in Texas. His research interests span 3D graphics algorithms and applications,

and physically based simulation.

Pete Demoreuille
Double Fine

Pete Demoreuille is a Lead Programmer at Double Fine Productions, working on all aspects of the engine and graphics technology. Prior to Double Fine he was at Pixar Animation Studios, working on the internal animation and modeling systems, and developing new rendering and lighting tools for Cars.

Tobias Berghoff
Sony Computer Entertainment WWS ATG

Tobias Berghoff first got paid for making games when he joined Ascaron Entertainment in 2005. Later, he spent a year making GPUs stare at the sun for the Royal Observatory of Belgium, before moving on to the Advanced Technology Group of Sony Computer Entertainment's World Wide Studios. He now works on graphics related technologies for PS3 and NGP. Tobias holds an Informatik Diplom from the University of Paderborn.

Cedric Perthuis
Sony Computer Entertainment WWS Santa Monica

Cedric Perthuis started his work in the graphics community at Intrinsic Graphics in 2002. He quickly joined Sony Computer Entertainment to build the original Playstation Cross Media Bar. After some work on embedded devices in the startup Emdigo, he co-developed Cg for PS3 in the PSGL group. He then became lead PS3 engineer for the Swedish studio Grin, before joining Sony Santa Monica Studio where he has been actively researching, developing and optimizing new rendering techniques. He recently shipped God of War 3. Cedric has a Master's degree in Applied Mathematics and Computer Science from ENSEEIHT, Toulouse, France.

Henry Yu
Kalloc Studios

Henry Yu is the founder and technical director of Kalloc Studios, in developing game engine technology and tool pipeline for consoles and PC platforms.

Morgan McGuire
NVIDIA and Williams College
`http://www.cs.williams.edu/~morgan/`

Morgan McGuire is a professor at Williams College and visiting professor at NVIDIA Research. He co-chaired the I3D 2008, 2009, and NPAR 2010 conferences, is a member of the Journal of Graphics, Game, and GPU Tools editorial board, and the lead author of Creating Games: Mechanics, Content, and Technology. He has contributed to many commercial products including the E-Ink display for the Amazon Kindle, the PeakStream high-performance computing infrastructure acquired by Google, the Titan Quest role playing game, and the Marvel Ultimate Alliance 2 video game for Xbox 360. His current research is in high-performance parallel algorithms and sampling techniques.

Timothy Lottes
NVIDIA
`http://timothylottes.blogspot.com/`

Timothy Lottes is an Engineer in the Developer Technology group at NVIDIA. Prior Timothy was a console game developer at Human Head Studios, in Systems Research and Development at Industrial Light and Magic, and owner/photographer/developer of Farrar Focus, a fine art landscape photography and digital photo development tools business.

Hugh Malan
CCP

Hugh Malan is a graphics programmer working on Dust 514 at CCP, in Newcastle. Previously he worked as graphics lead for Crackdown and MyWorld for Realtime Worlds. Hugh is a graduate of Victoria University and Otago University, New Zealand.

Emil Persson
Avalanche Studios
`http://www.humus.name/`

Emil Persson is a graphics programmer at Avalanche Studios where he is working on advanced rendering techniques and optimizations. Previously he worked as an ISV Engineer at ATI/AMD developer relations where he assisted the top game developers with rendering techniques and optimizations, in addition to R&D and SDK development. Emil also runs the site www.humus.name where he provides open source graphics samples to the community.

Dmitry Andreev
Lucas Arts
`http://and.intercon.ru/`

Dmitry Andreev is a rendering engineer and video games developer with more than 10 years of experience. He started his journey from 8-bit machines and got famous for his 64k intros in the demoscene, pushing the bar of competition at the time. Worked on all variety of game genres including FPS, RPG and Action games, leading development of core technology and graphics. Over past years he has been pushing real-time graphics in high-end AAA titles to its limits on PlayStation3 and XBox360 consoles with elegant and simple high-performance solutions. B.S. in Applied Mathematics and Mechanics.

Tiago Sousa
Crytek

Tiago Sousa is Crytek's Principal R&D Graphics Engineer, where he has worked for past 8 years, on all Crytek numerous demos, shipped game titles and engines, including Far Cry, Crysis and more recently finished Crysis 2 - Crytek's first multiplatform game. He is a self-taught graphics programmer, who before joining Crytek army on the cause of world domination, cofounded a pioneering game development team in Portugal and very briefly studied computer science at Instituto Superior Técnico, which he still has hopes to finish one day.

3 Morphological Antialiasing

Speaker: Alexander Reshetov (Intel Labs)

Morphological AntiAliasing (MLAA) algorithm belongs to a family of data-dependent filters allowing efficient anti-aliasing at a post-processing step. The algorithm infers sub-pixel coverage by

Figure 2: *Fairy Forest model: morphological antialiasing improves the quality of the rendered image without having a significant impact on performance. The algorithm uses separation lines falling between perceptually different pixels to infer silhouette lines and then blend colors around such silhouettes.*

estimating plausible silhouettes from a collection of axis-aligned separation lines that fall between perceptually different pixels (see Figure 2).

The algorithm consists of the following steps:

1. Noticeably different pixels are identified. Figure 3 shows a sample image with solid axis-aligned lines separating different pixels.
2. Piecewise-linear silhouette lines are derived from the separation lines. This is illustrated in Figure 3 with a Z-shape formed by **b**-**c**-**d** lines and a U-shape defined by **d**-**e**-**f** lines. In MLAA, silhouette segments originate at the edges of pixels that have both horizontal and vertical separation lines (all such pixels are shown with stripped shading). Not all potential end-points are used. The exact placement of end-points is somewhat arbitrary, with half-edge points producing satisfactory results.
3. Color filtering is performed for all pixels intersected by silhouette lines. Essentially, this is done by propagating colors on opposite sides of separation lines into polygons formed by silhouettes and separation lines, as illustrated at the bottom of Figure 3. The areas of these polygons are used for color mixing.

Any data that helps quantify differences between pixels can be used as an input at the first step of the algorithm (z-depth, normals, material ids, etc.) At the same time, the smallest amount of data to use is, obviously, color data itself. Since similarly-colored pixels tend to group together, this information can be used to infer plausible silhouettes between such groups. We use luminance according to ITU-R recommendations.

To identify all valid silhouettes (at the second step of the algorithm), we find all possible horizontal and vertical separation lines (between different pixels) and then

2.1. look at all start/end points on adjacent orthogonal lines and

2.2. choose the longest segment (preferring Z-shapes over U-shapes if multiple shapes are possible). This is illustrated in Figure 4;

2.3. if both horizontal and vertical silhouette lines intersect a pixel, we process only the longest silhouette line (preferring the horizontal one in the case of a tie). In Figure 3, both light-blue and dark-blue end-points can generate silhouette lines, but we choose only those bounded by dark-blue points.

Super-sampling is the gold standard for antialiasing, since it emulates integration processes in a camera or a human eye by averaging multiple samples per pixel. In MLAA, we assume that samples on the same side of a given silhouette line have the same color (pink and khaki colors in Figure 3). This allows evaluating the integral by computing areas of trapezoids formed by the silhouette line and the corresponding separation line. These areas vary linearly from pixel to pixel, allowing an efficient implementation. Of course, this approximation breaks when multiple silhouette lines cross the pixel, resulting in overblurring.

Among MLAA advantages,

- it is a universal algorithm, which can operate on the least possible amount of data — pixel colors;
- it is embarrassingly parallel and independent from the rendering pipeline, allowing efficient implementation on modern hardware;
- the quality is comparable with 4X supersampling.

At the same time,

- by relaying on a single sample per pixel, MLAA is predisposed to creating image artifacts in areas where image features are commeasurable with pixel sizes (Nyquist limit);
- it is susceptible to temporal artifacts;
- varying lighting can trigger silhouette changes in static scenes (if a color-based edge detector is used);
- it can mangle small text;
- there could be additional artifacts near image border as for any other screen-space technique.

We refer the reader to *Morphological Antialiasing* [Reshetov 2009] for full details.

4 A Directionally Adaptive Edge Anti-Aliasing Filter

Speaker: Jason Yang (AMD)

Authors: Konstantine Iourcha and Jason Yang and Andrew Pomianowski

In current GPU multisample anti-aliasing (MSAA), a pixel is limited by the number of available hardware samples. So, with $8x$ AA, meaning eight hardware samples, with traditional MSAA you can achieve at most eight levels of color gradation along an edge. Given the horsepower of today's GPUs, can we do better?

Our filter uses existing MSAA hardware and improves filtering by going outside the pixel. The main idea is that in areas of low edge

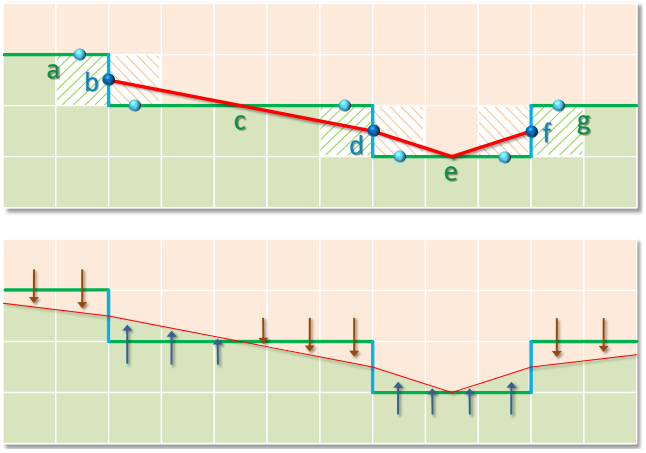

Figure 3: *A sample picture illustrates the main MLAA concepts. Solid axis-aligned lines separate different pixels. Top: reconstructed silhouettes are shown as red lines. Bottom: filtering through color propagation.*

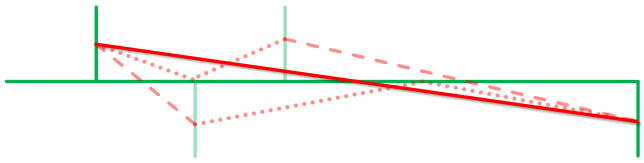

Figure 4: *Ambiguity resolution: even though multiple silhouette lines are possible, the longest one is used.*

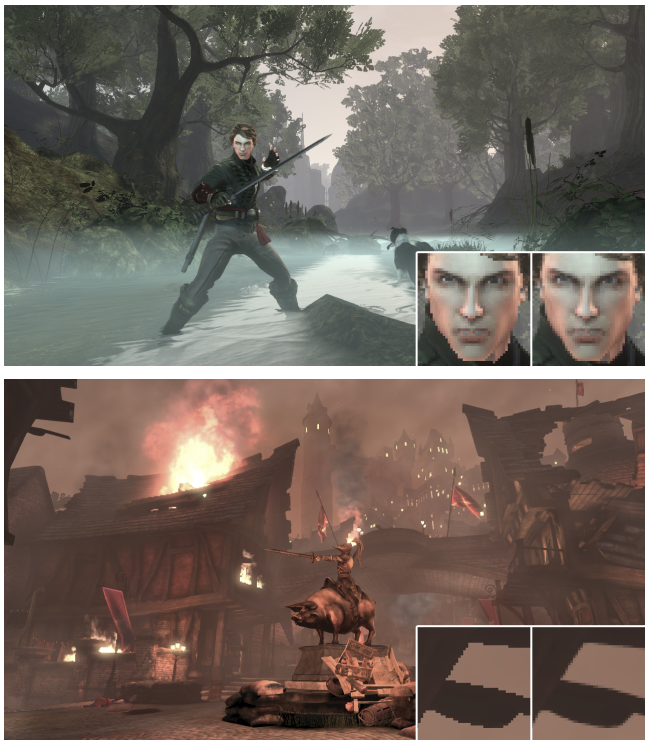

Figure 5: *Images obtained with Jimenez's MLAA. Insets show close-ups with no anti-aliasing at all (left) and processed with our technique (right). Images from Fable® III courtesy of Lionhead Studios.*

curvature, you can imagine isolines of the image function. You can then use samples outside of a pixel on these isolines to reconstruct a pixel.

How do you find the isolines? You can use a linear approximation by reducing this to a least squares problem.

Basically fitting a plane to the color samples. Or think of it as a linear regression. This should be the A-ha moment to solving the problem.

So, once this problem you know the gradient, then you use the integration method from the paper. The basic idea is you use samples both inside and outside of the pixel where the weight of the sample is determined by how much the isoline intersects with the pixel.

See *A directionally adaptive edge anti-aliasing filter* [Iourcha et al. 2009] for full details.

5 Practical Morphological Anti-Aliasing (Jimenez's MLAA)

Speaker: Jorge Jimenez (Universidad de Zaragoza)

Authors: Jorge Jimenez and Belen Masia and Jose I. Echevarria and Fernando Navarro and Diego Gutierrez

Multisample anti-aliasing (MSAA) remains the most extended solution to deal with aliasing, crucial when rendering high quality graphics. Even though it offers superior results in real time, it has a high memory footprint, posing a problem for the current generation of consoles, and it implies a non-negligible time consumption. Further, there are many platforms where MSAA and MRT (multiple render targets, required for fundamental techniques such as deferred shading) cannot coexist. The majority of alternatives to MSAA which have been developed, usually implemented in shader units, cannot compete in quality with MSAA, which remains the gold standard solution. This work introduces an alternative anti-aliasing method offering results whose quality averages at $16x$ MSAA (from a gradients quality perspective) at a fraction of its memory and time consumption (see Figure 5 for some examples). Besides, the technique works as a post-process, and can therefore be easily integrated in the rendering pipeline of any game architecture.

The technique is an evolution of the work "Morphological Antialiasing", which is designed for the CPU and unable to run in real time. The method presented here departs from the same underlying idea, but was developed to run in a GPU, resulting in a completely different, extremely optimized, implementation. We shift the paradigm to use texture structures instead of lists (See Figure 6), which in turn allows to handle all pattern types in a symmetric way, thus avoiding the need to decompose them into simpler ones, as done in previous approaches. In addition, pre-computation of certain values into textures allows for an even faster implementation.

The algorithm detects borders (using color, depth, normals or instance id's information) and then finds specific patterns in these. Anti-aliasing is achieved by blending pixels in the borders intelligently, according to the type of pattern they belong to and their position within the pattern. Pre-computed textures (see Figure 7) and extensive use of hardware bilinear interpolation (see Figures 8 and 9) to smartly fetch multiple values in a single query are some of the key factors for keeping processing times at a minimum.

Typical execution times are 1.3 ms on Xbox 360 and 0.44 ms on a nVIDIA GeForce 9800 GTX+, for a resolution of 720p. Memory

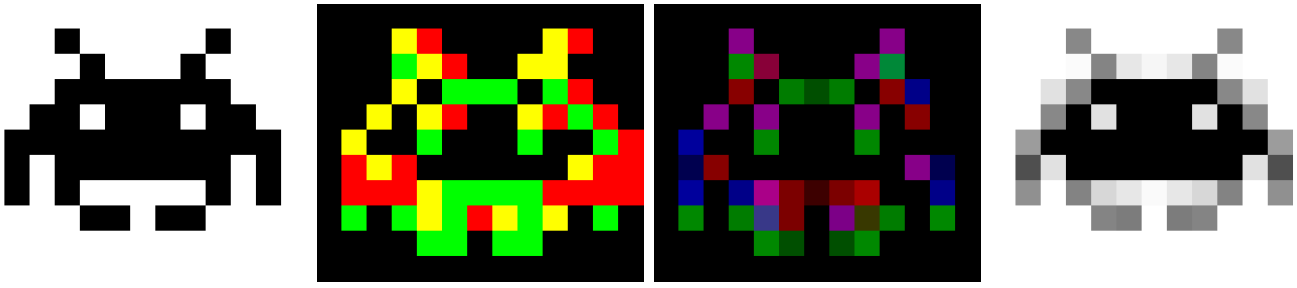

Figure 6: *In Jimenez's MLAA, and starting from an aliased image (left), edges are detected and stored in the edges texture (center left). The color of each pixel depicts where edges are: green pixels have an edge at their top boundary, red pixels at their left boundary, and yellow pixels have edges at both boundaries. The edges texture is then used in conjunction with the precomputed area texture to produce the blending weights texture (center right) in the second pass. This texture stores the weights for the pixels at each side of an edgel in the RGBA channels. In the third pass, blending is performed to obtain the final anti-aliased image (right).*

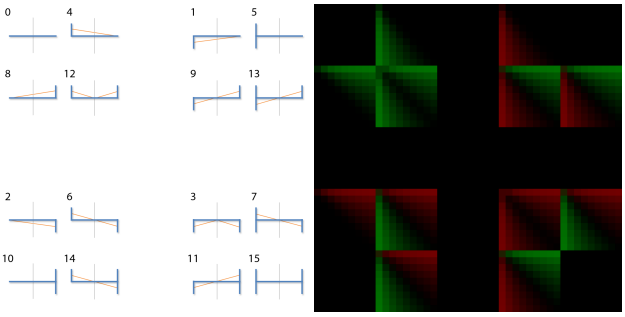

Figure 7: *Patterns processed by Jimenez's MLAA (left) and their corresponding pre-calculated weights depending on their size (right). Each 9×9 subtexture corresponds to a pattern type. Inside each of these subtextures (u, v) coordinates encode distances to the left and to the right, respectively.*

Figure 8: *In Jimenez's MLAA, hardware bilinear filtering is used when searching for distances from each pixel to the end of the line. The color of the dot at the center of each pixel represents the value of that pixel in the edges texture. In the case shown here, distance search of the left end of the line is performed for the pixel marked with a star. Positions where the edges texture is accessed, fetching pairs of pixels, are marked with rhombuses. This allows us to travel double the distance with the same number of accesses.*

footprint is $2x$ the size of the backbuffer on Xbox 360 and 1.5x on the 9800 GTX+. Meanwhile, $8x$ MSAA takes an average of 5 ms per image on the same GPU at the same resolution, 1180% longer for the PC case (i.e. processing times differ by an order of magnitude). The method presented can therefore challenge the current gold standard in real time anti-aliasing.

See *Practical Morphological Anti-Aliasing* [Jimenez et al. 2011] for full details. Visit http://www.iryoku.com/mlaa for latest news, source code releases, a showcase gallery, performance tables, a

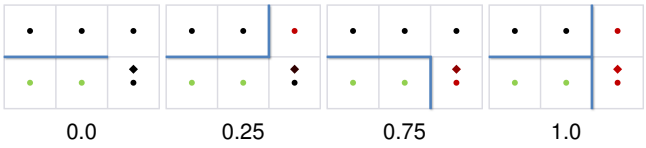

Figure 9: *Examples of the four possible types of crossing edge and corresponding value returned by the bilinear query of the edges texture (Jimenez's MLAA). The color of the dot at the center of each pixel represents the value of that pixel in the edges texture. The rhombuses, at a distance of 0.25 from the center of the pixel, indicate the sampling position, while their color represents the value returned by the bilinear access.*

F.A.Q., and more.

6 Hybrid Morphological Anti-Aliasing

Speaker: Peter Demoreuille (Double Fine)

Despite the original *Morphological Anti-Aliasing* (Reshetov) approach being purely CPU-based and seemingly unsuitable for realtime use, the attractive properties of a high-quality post-process antialiasing solution have driven the development of several implementations suitable for real-time applications. *Practical Morphological Anti-Aliasing* (Jimenez et al) presents a optimized GPU adaptation, including implementation details for PC and the Xbox 360. In their approach, all three fundamental stages of the algorithm (edge detection, blend weight calculation and image filtering) are performed on the GPU in separate passes. Memory requirements are low, and performance is very good (0.44 ms on a nVIDIA GeForce 9800 GTX+ at 720p), though GPU utilization may be unpredictable, as it is a function of the number of edges found in the scene. *Low latency MLAA in God of War III* and *PlayStation EDGE MLAA* (Perthuis and Berghoff) present an optimized implementation for the PlatStation3 where the entire algorithm is performed in parallel on several SPUS, offloading all work from the GPU.

Hybrid Morphological Anti-Aliasing presents an implementation of MLAA for the Xbox 360 where the edge detection and filtering stages of the algorithm are performed on a GPU, and the blend weight calculations are performed on CPUs. This approach shares the low memory requirements of *Practical Morphological Anti-Aliasing*, and offloads a great deal of work from the GPU just as *PlatStation EDGE MLAA*. However, a hybrid CPU/GPU algorithm has several advantages when compared to a pure CPU or GPU algo-

rithm. Utilizing the GPU trivially allows the use of non-color data when detecting edges (such as depth, material and normal data), allowing more stable edges and avoiding some jittering artifacts common when using only color. Use of a GPU also allows the use of linear-intensity colors, both when detecting edges and when blending. Lastly, using the CPU to compute blend weights saves a great deal of GPU time, and leaves the GPU performing only those passes that have small, fixed costs, allowing GPU performance of the algorithm to be strictly bound.

While using the GPU for some stages has advantages, using the CPUs for others has tradeoffs, including non-trivial CPU/GPU communication and potentially heavy utilization of CPU time. We present techniques to optimize calculations by reducing the bandwidth requirements of the algorithm, including packing data and performing a very fast transpose of the input data to optimize cache utilization during vertical edge calculations. CPU utilization is further reduced by adapting calculations to be better suited to the Xbox's PowerPC architecture. Additional controller latency may be added due to overhead in CPU/GPU communication, and we describe how our implementation rearranges work to avoid adding latency to our titles. As each of our titles has unique visual styles and different average scene construction, several edge detection techniques were used to ensure filtering produced acceptable results. We will present these techniques and various tips to avoid overblurring, missing edges, and maintaining stable filtering.

As each stage of the MLAA algorithm may be tuned for individual applications, we hope to provide additional options for developers creating their own anti-aliasing solutions, so that they may best fit the performance, quality and memory restrictions placed upon them.

Figure 10: *Example results of Hybrid Morphological Anti-Aliasing from* Costume Quest*,* Stacking *and* Once Upon a Monster.

7 PlayStation®Edge MLAA

Speakers: Tobias Berghoff and Cedric Perthuis

Authors: Tobias Berghoff, Cedric Perthuis and Matteo Scapuzzi

PlayStation®Edge MLAA is an implementation of the Morphological Anti-Aliasing (MLAA) algorithm for the Sony PlayStation®3 console.

It is used in a number of high-profile games and has undergone a series of improvements and optimizations since its inception. In this talk, we will provide an overview of this process and its results, focusing on how to fit the algorithm to the PS3™'s unique architecture and significantly improve the image quality provided by the method. Both successful and unsuccessful approaches will be examined, to provide the listener with a well rounded understanding of the problem space.

7.1 MLAA on SPUs

Our implementation runs on the main processing units of the Cell Broadband Engine™, the Synergistic Processing Units, or SPUs. These units are interconnected through a high speed ring bus and have each 256KB of private fast low-latency memory (*Local Store*, or LS). The processing elements themselves consist of a high-performance dual-issue SIMD processor coupled with a DMA engine. Each of these features had significant influence on the design of the system.

Initially, we experimented with a tile-based implementation, where the image would be divided into rectangular regions, each small enough to fit into LS. This would allow the processing of arbitrarily sized images and an easy balancing of LS budgets, as the tile

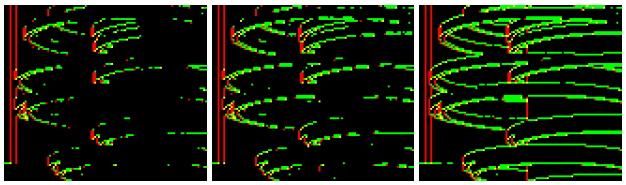

Figure 11: *From left to right: Edges detected with basic thresholding, relative thresholding, predicated thresholding.*

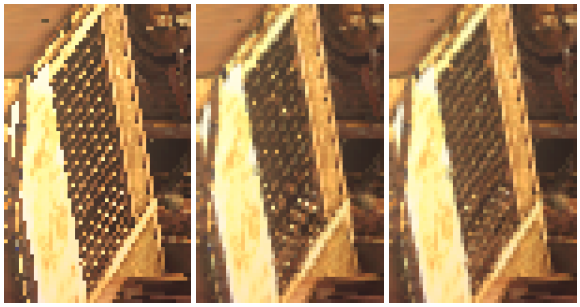

Figure 12: *From left to right: Source image, relative thresholding, relative thresholding with feature splitting.*

size could be changed depending on other needs for memory. A library of post-process effects implemented in this fashion is already available in the form of PlayStation®Edge Post, making a unified approach attractive.

Due to artifacts caused by the tiling, this implementation was discarded early on, and a more classical scan-line approach was chosen and later shipped in *God of War®III*. This scan-line based approach only implements a horizontal MLAA step and uses an in-place pseudo-transpose to implement the vertical step. The algorithm itself begins with an edge detection step, followed by a number of sequence algorithms on the edges found. The goal of this approach is to quickly reduce the length of the sequence using comparatively cheap loops before executing more expensive loops. This approach is viable, as LS accesses are very fast and are executed by a different pipe (the *odd* pipe) in the processor as the ALU operations (which are executed in the *even* pipe). We can thus use the *odd* pipe to load and prepare data for consumption by the *even* pipe, as well as pack and store it for the next processing step.

7.2 Image Quality Improvements

For the original MLAA paper, a very simple edge detection system was intentionally used for reasons of simplicity. While this was intended as a placeholder for more sophisticated edge detection algorithms, the method is still widely used today. While working on *God of War®III* we quickly realized that the quality of the edge detection was one of the most important ingredients for image quality and temporal stability. As a first step, the basic edge detection system was replaced with a method that is better suited for images with varying levels of brightness along features and over time.

Furthermore, we introduced the ability to cut an edge into multiple parts, thus allowing a better handling of adjacent edges. This is primarily an improvement in areas with a very high edge density, where it make the edge cases approach a slight blur instead of adding additional noise to them, as shown in figure 12.

As there are limits to how well we can detect an edge in color data alone, we introduced the concept of a predicated edge detection,

Figure 13: *Comparison between the Xbox 360 and the PS3 versions of* The Saboteur*, without anti-aliasing and with SPUAA respectively. Images from Digital Foundry:*
`http://www.eurogamer.net/articles/digitalfoundry-saboteur-aa-blog-entry`

which first performs edge detection on non-color data and then controls the sensitivity of the color data edge detection based on this. The non-color data can be generated for this specific purpose, like object IDs, or be a byproduct of the rendering process like depth. Of course, the availability of suitable data is highly dependent on the rendering systems of individual games, and as such this method is less generic than a purely color based one.

Figure 11 shows a comparison of the thresholding methods. Predicated thresholding has important advantages over approaches that do not base the final decision of processing a ”edge” on the color data.

8 Using SPUs for Anti-Aliasing in The Saboteur

Speaker: Henry Yu (Kalloc Studios)

Authors: Henry Yu, Ian Elliot and David Clamage

Kalloc Studios was contracted to adapt *The Saboteur* to function on PS3 concurrent to Pandemic Studios' development of *The Saboteur's* content and Xbox 360 version. As a result, we had many unique challenges to overcome in altering a continuously changing product to a platform known to be powerful but difficult to develop for, amongst the most difficult of which was posed when Pandemic Studios requested that the PS3's release of The Saboteur contain anti-aliasing (see Figure 13). *The Saboteur's* final SPU anti-aliasing method (internally dubbed 'SPUAA') was created independently of knowledge of Morphological Anti-Aliasing (MLAA) techniques, although it resembles them in many ways.

Several standard anti-aliasing techniques were initially investi-

gated, including using MSAA and using a full-screen post-processing shader on the GPU. MSAA was quickly eliminated as an option due to the high cost for both graphics memory and GPU time when rendering to a larger buffer, as well as the technical difficulty of using a deferred rendering pipeline with MSAA. The full-screen post-processing shader was rejected due to underwhelming results along with a 2.4ms cost on the GPU, which was already over-budget without an anti-aliasing techniques. Due to similar limitations, the Xbox 360 version of *The Saboteur* used no anti-aliasing solution. The only resource on the PS3 that had time to spare were the SPUs, so we investigated the possibility of using the SPUs for anti-aliasing by analyzing the final color buffer, which at the time was a method we had not heard of from other PS3 games.

After all rendering has been performed by the GPU, the following conceptual steps are taken by the SPUs to perform the Anti-Aliasing. Start by determining Edges, Calculate luminance based on the final 24-bit RGB output color buffer and threshold the absolute difference between directly adjacent pixel luminance to determine edges. Then find edge streaks, a run of adjacent edges which terminate with a normal oriented edge. Each pixel is populated with the type and length of the streak in each screen space direction (up, down, left, right). The type of the streak is whether its terminating edge is toward the negative side, toward the positive side, or on both sides. The length of the streak is up to 63 pixels, saturated. Next, population of streak occurs in two passes. A diagonal line of 80x80 pixel tiles moving down and right simultaneously from the top left corner, and another moving up and left simultaneously from the bottom right corner. Finally, tween edges. Select the longest combined edge streak, with vertical or horizontal streaks combined. Compute the 'shape' of the streak, based on which side the pair of terminating edge sets reside on, determining the exact tweening shape. Then blend the pixel with appropriate adjacent pixels based on the determined tweening shape and location upon the edge streak.

The weaknesses exhibited by this technique are similar to observed weaknesses in other MLAA family algorithms. Most notably, it has poor handling of sub-pixel detail. Also, the simple edge reconstruction model fails to account for 'hitches' in aliased results as a result of arbitrary line slopes. (With SPUAA, reconstructed slopes only take the form of $1/t$ or $t/1$ with t as an integer). Additionally, Moiré patterns are evident in texture sampled surfaces which can cause spurious and flicker-like results especially with minor camera angle changes due to spatiotemporally discontinuous edge detection of visually relevant features.

For the implementation in *The Saboteur*, there were a few details that, given more time, could have been even more optimal. SPUAA was performed in three rounds. The first round computed luminance and stored the result into the unused alpha channel of the final GPU render target buffer. The second round marked edges and computed right / down edge streaks across diagonal tile sets. The third round computed left / up edge streaks diagonal tile sets and performed edge tweening.

In the future, better performance could be gained through a few changes to the implementation in *The Saboteur*. First, the luminance computation, edge marking, and right/down edge streak detection could occur in the same round, saving redundant data transfer. Second, in the last phase only top left and top right relevant streak information needs to be written out for the next diagonal set, which would also reduce data transfer requirements. Finally, though SPUs are very fast at floating point mathematics, it would have been faster for computation to perform tweening by using software ratio approximations instead of hardware division approximations to compute the main tweening lerp coefficients.

The only input to SPUAA is the final color buffer of the scene, and it spends the next frame processing the scene. Because of the low requirements on how the existing graphics pipeline is implemented, the technology is portable and can be quickly integrated into any PS3 game. Despite potential for improvement, the resulting method used in *The Saboteur* both looked better than our expectations, as well as being within budget for SPU time, GPU time, and memory use.

9 Subpixel Reconstruction Anti-Aliasing (SRAA)

Speaker: Morgan McGuire (NVIDIA and Williams College)

Authors: Matthäus G. Chajdas, Morgan McGuire, David Luebke

Using the color/depth buffer at pixel-resolution has the problem that edges may flicker as the edge flips back and forth between adjacent pixels. Without using additional information, for instance back-projection, this is an issue which makes pure post-processing algorithms unstable.

SRAA tries to circumvent the problem by taking sub-pixel samples to find edges while keeping the shading still at 1 sample/pixel to guarantee good coherence and minimal changes to the existing pipeline (see Figure 14). In particular, SRAA requires some edge detector similar to Jimenez' MLAA. The edge detector is used to determine sample similarity in order to reconstruct the color at each sample. Once that is done, the samples get filtered – typically using a box filter – and can be sent to the post-processing pipeline.

SRAA works on geometric edges only. That is, no shader or texture aliasing can be resolved. Discontinuities are found by investigating a super-resolution buffer, so both pixel as well as sub-pixel aliasing can be resolved. This makes SRAA more stable under animation, in particular, slow-moving edges are less prone to swimming artifacts. SRAA works best on deferred renderers, as all information required for SRAA is readily available in the G-Buffers. However, SRAA can be also used with forward rendering as long as the information necessary to determine edges is generated. This would be typically done as part of a z-prepass.

The most basic implementation of SRAA requires the following inputs:

- Depth and normal buffers with MSAA enabled
- Colour buffer without any MSAA (i.e. one shaded sample per pixel)

One important requirement is that the colour buffer sample must line up with the MSAA samples; the easiest way to guarantee is to simply only shade the very first MSAA sample, for instance by resolving the buffer with a pixel shader. SRAA works in two steps:

- For each sample from the MSAA input:
  - If the sample is shaded, continue
  - Otherwise: Compare the geometric information against the neighboring shaded samples.
  - Copy the color from the shaded sample, weighted by similarity.
- For each pixel: Filter all samples inside the filter support. In order to minimize blurring, a Gaussian with strong falloff should be used, or a comparatively sharp filter.

It conceptually reconstructs color at each MSAA position and then resolves, but since the computation of each happens simultaneously, the high-resolution color is never actually stored. That is, there's a

Figure 14: *Comparison between regular sampled input, SRAA with 1 shaded sample per pixel and 4 MSAA depth/normal samples, and 16x Super-Sampling.*

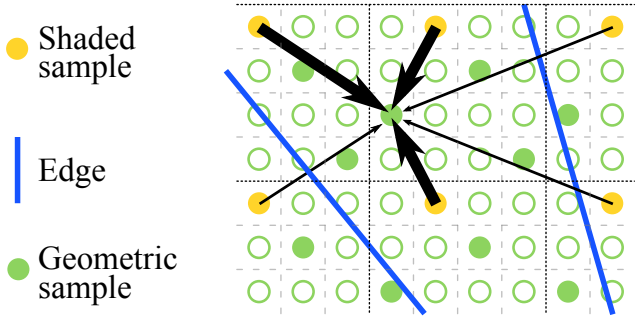

Figure 15: *The color at each geometric sample is reconstructed by using the neighboring shaded samples.*

single register that *accumulates* the result of the resolve within an unrolled loop.

The key part of the algorithm is the similarity metric, which determines whether the value should be copied across from the shaded sample or not. A simple metric uses the distance between the plane defined by the fragment position and normal and the position of the other fragment. Let $P(x, \vec{n})$ be the plane at point x with normal $\vec{n}$. We can define a distance between $(x_0, \vec{n}_0)$ and $(x_1, \vec{n}_1)$ as $\max(d(P(x_0, \vec{n}_0), x_1), d(P(x_1, \vec{n}_1), x_0)$ with d being the distance from the point to the plane. This works reliable in practice, but requires MSAA'ed depth and normal buffers. A simple optimization is to use the `SV_PrimitiveID`, which is unique for each triangle: simply comparing each sample using the primitive ID provides a good approximation and is very fast. The primitive ID can be hashed down to 8 bit and is thus comparatively cheap.

See *Subpixel Reconstruction Anti-Aliasing* [Chajdas et al. 2011] for full details.

10 Fast approXimate Anti-Aliasing (FXAA)

Speaker: Timothy Lottes (NVIDIA)

The fastest of three algorithms which make up FXAA will be covered, FXAA Console: Local Contrast Adaptive Directional Edge Blur (see Figure 16 for an example).

Algorithm applies a 2 or 4-tap variable length box filter 90 degrees to luma gradient, with an adjustment on filter length and direction to help remove sub-pixel aliasing. Works as one full screen pixel shader pass taking color as input and writing color as output. Extremely fast algorithm, averaging just 0.11 ms per million pixels on NVIDIA GTX480. Details:

- Optimized early exit on pixels which do not need anti-aliasing. Early exit test is based on a comparison of local luma contrast to local maximum luma.
- Estimates filter direction as perpendicular to local luma gradient. Maximum gradient is 1, which corresponds to one pixel in width.
- Estimates filter width by scaling direction such that the shortest axis magnitude is 1. This has the effect of lengthening the filter on nearly horizontal or vertical edges, while keeping a very short filter on the diagonal. Length is decreased as local contrast decreases, and finally maximum filter width is clamped to 9 pixels.
- Estimate single pixel detail using the difference between local luma contrast and 2x2 box filtered local luma contrast. Modify filter vector by increasing length in both x and y by this estimation of single pixel detail. If length was zero, then add in the positive direction. This extends the filter diagonally to increase the amount of blur.
- Calculate a full-width 4-tap box filter, and a half-width 2-tap box filter along the filter vector. If the luma of the full-width filter result exceeds the range of local luma, discard the full-width result and return the half-width result. Otherwise return the full-width result. This step removes noise associated with bad filter vector estimates.

11 Distance-to-edge Anti-Aliasing (DEAA)

Speaker: Hugh Malan (CCP)

Distance-to-edge AA (DEAA) simulates antialiasing by selective blurring, similar to MLAA. The main difference with respect to

Figure 16: *Comparison between the original 1x input (top) and FXAA Console (bottom).*

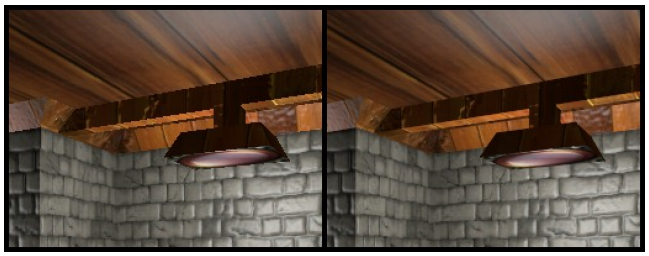

Figure 17: *Comparison between the original 1x input (left) and GBAA (right).*

MLAA is that the pixel coverage values are derived from distance-to-edge values calculated during the forward pass.

Scalar texture coordinates have been up so each edge of each triangle has a scalar value taking the value 0 along that edge, and 1 on the opposite vertex. During the forward pass pixel shader, the value and screen-space derivatives of each scalar are used to estimate the horizontal and vertical distance onscreen to the corresponding edge. The process is repeated for each of the interpolated scalars, yielding distances to each edge of the triangle. Distances to triangle edges in the four directions up, down, left and right are found, and these four distances are written out to a separate rendertarget.

The postprocess set uses this information to calculate pixel coverage. By considering the distance to each edge, the overlap area for each neighbouring pixel is estimated, and the final pixel color is a corresponding blend of the neighbouring pixel colors.

This method has somewhat different strengths and weaknesses to MLAA. Since the distance-to-edge values are calculated with subpixel precision, the postprocess blur can simulate subpixel effects that are not possible by simple inspection of a framebuffer. (For instance, imagine a vertical edge sweeping across the image: with DEAA columns of pixels will fade in and out to indicate the subpixel location of the edge. MLAA and other methods that are based on the framebuffer can only provide an edge profile that advances in single-pixel steps.)

Realtime MLAA is limited to a localized neighbourhood, so it is unable to provide plausible antialiasing for edges with pixel steps longer than this neighbourhood. In comparison, DEAA is able to provide plausible antialiasing effects for edges regardless of gradient.

Conversely, DEAA is unable to provide antialiasing in several situations where MLAA can. DEAA requires all edges to have verts on them, so the distance-to-edge logic can estimate coverage. Interpenetrating geometry will produce an edge with no verts; shadow edges and texture edges are other situations where DEAA cannot antialias the edge but MLAA can.

DEAA will fail to antialias edges correctly in a few other cases too. If there is a subpixel triangle beside the edge - perhaps due to foreshortening, or due to a very thin piece of geometry - then the distance-to-edge information is incorrect, and this tends to produce an aliased region. Very thin gaps are another problem case.

See *Edge Anti-aliasing by Post-Processing* [Malan 2010] for full details.

12 Geometry Buffer Anti-Aliasing (GBAA)

Speaker: Emil Persson (Avalanche Studios)

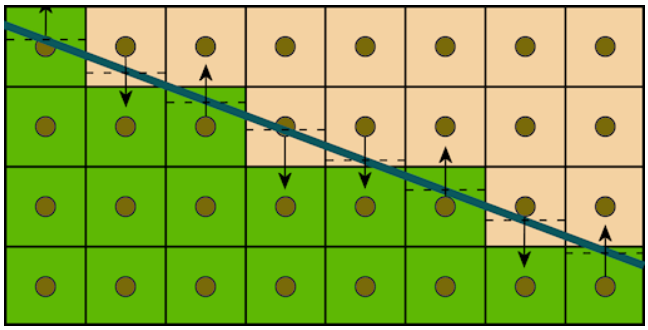

Figure 18: *Neighborhood sampling and coverage computation in GBAA.*

Anti-aliasing in real-time applications (such as games) has in recent years primarily been addressed with multisampling (MSAA) or similar approaches such as CSAA. Aside from substantial memory consumption, these methods are practical and general for traditional forward rendering approaches, and the characteristics are widely known and understood. As more and more game engines switch to a deferred shading model the appeal of MSAA has been significantly reduced with added complexity and the memory consumption further multiplied. With limited memory on current game consoles alternative approaches get more interesting. Recently a number of approaches for performing anti-aliasing as a post-processing step have been proposed. This includes MLAA, FXAA, SRAA and DLAA, all with different pros and cons. The unifying idea behind all these techniques is that the image is analyzed to find discontinuities to smooth, optionally using the depth buffer to aid the process. In contrast, we propose that the game engine use its knowledge of the underlying geometry instead to smooth edges.

The author has previously proposed GPAA [Persson 2011] as a geometric post-process anti-aliasing solution. This technique also has a pre-process step and potential scaling issues which may reduce its attractiveness for real-time applications such as games. To address these problems an alternative approach has been implemented that store geometric edge information to a fullscreen render target during the main rendering pass (see Figure 17 for an example). This is easiest accomplished with a geometry shader, although a vertex shader implementation is also possible. Edge orientation is passed down to the pixel shader that writes the information to an edge buffer. In the end the buffer is analyzed and resolved in a fullscreen pass, similar in concept to how traditional MSAA works. In the resolve stage each pixel is checked for information about any edge intersecting it, and if so, what coverage the primitive that shaded the pixel had (see Figure 18). Depending on the edge orientation relative the pixel center a suitable neighbor pixel is selected for blending. For pixels missing intersection information the immediate neighborhood is searched for edges that might apply. This is because silhouette edges will only have edge information on pixels on one side of the edge, whereas pixels on both sides need to be anti-aliased. Final blending is performed by simply shifting the texture coordinate such that the texture unit can blend using a regular linear filter. As such, only a single sample is required from the color buffer.

13 Directionally Localized Anti-Aliasing (DLAA)

Speaker: Dmitry Andreev (Lucas Arts)

Multisample Anti-Aliasing has been the holy grail solution in games for many years. But unfortunately, it's not always applicable. The more multi-pass and deferred techniques we put in place, to keep increasing visual complexity, the more costly it becomes. Especially on consoles, directly and indirectly, when adjusting for all the post-processing effects.

Anti-Aliasing From a Different Perspective is a story behind Directionally Localized Anti-Aliasing. It shows how a technical problem of anti-aliasing could be solved in an artistic way. By going through a series of prototypes, failures and successes, geting to the final idea of the algorithm and its console specific implementation details.

The proposed solution is a novel anti-aliasing technique which was used in The Force Unleashed 2. It is designed with simplicity in mind, that makes it GPU and CPU friendly and allows to have efficient implementations on modern gaming consoles such as the PlayStation3 and XBox360. It is temporally stable and very effective, offering very high quality to performance ratio.

See *Anti-Aliasing From a Different Perspective* [Andreev 2011] for full details.

14 Anti-Aliasing Methods in CryENGINE 3.0

Speaker: Tiago Sousa (Crytek)

14.1 Introduction

Real time rendering Anti-Aliasing has been for many years depending on a set of strict rules imposed by graphics hardware. When rendering technology deviates from the golden standard, for which hardware was originally designed, issues arise, which require effort and workarounds, introducing complexity, increased memory footprint and performance penalties.

Such has motivated exploration of alternative and robust solutions, where the most popular trend is becoming Anti-Aliasing by post processing, popularized by the CPU-based Morphological Anti-Aliasing (MLAA) method.

For CryENGINE 3.0 iterations, we are exploring different alternatives allowing sub-pixel accuracy and which are orthogonal to a multiplatform environment, from performance, video memory and implementation perspective.

Our presented work is heavily inspired by real time rendering approach to temporal anti-aliasing and by OpenGL's Accumulation Buffer and its many distributed ray tracing effects usages, like temporal super-sampling among others (see Figure 19).

14.2 Background

The introduction of the consoles PS3 and Xbox 360, opened a big opportunity window for Deferred Rendering and its derivatives to become the new standard for real time rendering.

Decoupling shading or lighting from primitives rendering, brings many benefits for all platforms: decreasing amount of drawcalls; many light sources can be used, resulting in improved lighting quality and flexibility for lighting artists; decreased amount of shader permutations; and an immense set of algorithms approximated in screen space, from Screen Space Ambient Occlusion, Screen Space Skin Surface Scattering to Stereo Rendering, are few of the examples.

This console generation also allowed for High Dynamic Range Rendering to become a wider option for real time rendering. Unfortunately render target formats across multiple platforms are not

Figure 19: *Crysis 2, Crytek's first game using CryENGINE 3.0.*

standardized and their performance varies significantly, requiring additional effort.

Such adds implications as well for hardware anti-aliasing due to the aforementioned increased memory and performance requirement.

14.3 OpenGL Accumulation Buffer SSAA

OpenGL Accumulation Buffer SSAA is achieved by adding a sub-pixel jittering to the view frustum and a linear combination of sub-samples. For example, 4x SSAA means rendering the scene 4 times, which is not yet very practical for realtime purposes.

14.4 Real Time Rendering Temporal AA

Real time temporal anti-aliasing, popularly denominated as Motion Blur, has become very popular in video games and extensively used in games like Crytek's Crysis 1 and 2, Killzone 2 and Halo 3. Such TAA approximation is done in screen space, by applying a directional blur from previous frame to current frame using the screen space velocity vector. This velocity vector is computed by projecting previous frame pixel, or vertices, world space position to screen space and computing respective delta from current and previous frame screen-space position.

14.5 Combining Concepts

By combining both concepts, we can achieve $2x$ SSAA or higher, at cost of higher latency for final result.

This is done by interleaving non-uniform sub-pixel jittering to the view frustum for different frames, storing previous frame (s). At end of frame, post-tone mapping, previous frame sub samples are fetched by means of velocity vector and all sub samples are then blended selectively, based on depth, velocity or color.

A Quincunx sampling pattern was also used for the $2x$ SSAA version for approximating 4x SSAA results just with 2 available sub-samples.

14.6 Results

Orthogonal multiplatform SSAA solution with sub-pixel accuracy, costing 1ms, at 720p, on PS3/X360 hardware. At 1080p on PC hardware, costs 0.2 ms.

References

ANDREEV, D. 2011. Anti-aliasing from a different perspective. In *Game Developers Conference 2011*.

CHAJDAS, M., MCGUIRE, M., AND LUEBKE, D. 2011. Sub-pixel reconstruction antialiasing. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM Press.

IOURCHA, K., YANG, J. C., AND POMIANOWSKI, A. 2009. A directionally adaptive edge anti-aliasing filter. In *Proceedings of the Conference on High Performance Graphics 2009*, ACM, New York, NY, USA, HPG '09, 127–133.

JIMENEZ, J., MASIA, B., ECHEVARRIA, J. I., NAVARRO, F., AND GUTIERREZ, D. 2011. *GPU Pro 2*. AK Peters Ltd., ch. Practical Morphological Anti-Aliasing.

MALAN, H. 2010. *GPU Pro: Advanced Rendering Techniques*. AK Peters Ltd., ch. Edge Anti-aliasing by Post-Processing, 265–289.

PERSSON, E., 2011. Geometric post-process anti-aliasing. http://www.humus.name/index.php?page=3D&ID=86.

RESHETOV, A. 2009. Morphological antialiasing. In *Proceedings of High Performance Graphics*, 109–116.