Morphological Antialiasing

Alexander Reshetov
Intel Labs

Figure 1. Fairy Forest model: morphological antialiasing improves the quality of the rendered image without having a noticeable impact on performance.

ABSTRACT

We present a new algorithm that creates plausibly antialiased images by looking for certain patterns in an original image and then blending colors in the neighborhood of these patterns according to a set of simple rules. We construct these rules to work as a post-processing step in ray tracing applications, allowing approximate, yet fast and robust antialiasing. The algorithm works for any rendering technique and scene complexity. It does not require casting any additional rays and handles all possible effects, including reflections and refractions.

CR Categories: I.4.3 [Computer Graphics]: Image Processing And Computer Vision — Enhancement

Keywords: antialiasing, image enhancement, morphological analysis, rendering

1 INTRODUCTION

A human eye has a discrete number of light receptors, yet we do not discern any pixels, even in peripheral vision. What is even more amazing is that the number of color-sensitive cones in the human retina differs dramatically among people – by up to 40 times [Hofer et al. 2005]. In spite of this, people appear to perceive colors the same way – we essentially see with our brains. The human vision system also has an ability to ascertain alignment of objects at a fraction of a cone width (hyperacuity). This explains why spatial aliasing artifacts are more noticeable than color errors.

Realizing this fact, graphics hardware vendors put significant efforts in compensating for aliasing artifacts by trading color accuracy for spatial continuity. Multiple techniques are supported in hardware, based on mixing weighted color samples, similar to the integrating property of digital cameras [ATI; NVIDIA].

---

e-mail: alexander.reshetov@intel.com

Of course, any aliasing artifact will eventually disappear with an increase in display resolution and sampling rates. It can also be handled at lower resolutions, by computing and averaging multiple samples per pixel. Still, for ray tracing algorithms this might not be very practical, dramatically decreasing overall performance by computing color samples that are eventually discarded through averaging. Whitted [1980] was the first who proposed to minimize the number of additional rays by casting more rays only if neighboring pixels have a significant color variation. Modification of this adaptive technique is based on comparing distance-related terms instead of color [Keller 1998], which is better suited for models with high-frequency textures. Other researchers were estimating color variance *inside* a pixel, by tracing beams [Heckbert and Hanrahan 1984], cones [Amanatides 1984], pencils [Shinya et al. 1989], bounds [Ohta and Maekawa 1990], covers [Thomas et al. 1989], or pyramidal rays [Genetti et al. 1998]. Mostly, researchers were trying to improve still images, but there were also efforts to utilize – and improve – the temporal coherence [Martin et al. 2002].

Another line of research is based on an observation that color discontinuities are usually most noticeable at silhouette boundaries. For rasterization-based rendering, it is possible to draw wide prefiltered lines at object silhouettes [Chan and Durand 2005], which are composited into final image using alpha-blending. To avoid blurriness due to the blending, [Sander et al. 2001] proposed eliminating jagged edges by overdrawing such edges as antialiased lines. The ray tracing implementation, described by Bala et al. [2003], constrains color interpolation by discontinuity edges, found by projecting silhouettes onto image plane.

Among the advantages of algorithms which use object silhouettes is that such algorithms provide a way to reliably find discontinuities inside pixels. These discontinuities could be missed by adaptive techniques that initially cast a single ray per pixel. At the same time, discontinuities due to different materials will not be found through silhouette tracing. This might result in artifacts unless additional processing is implemented. Furthermore, silhouette-based techniques cannot remove aliasing in reflections or refractions, as these algorithms require a single center of projection (camera or light source) to determine what a silhouette is.

Handling of pixel-size features in a scene is not a trivial problem, even if such features could be reliably identified. If a large number of primitives are projected onto a single pixel, integrating color over these primitives in many cases will yield some average grayish color. The same effect can be approximated by averaging single color samples obtained for neighboring pixels or using fog or haze-like effects. This resembles Leonardo da Vinci's *sfumato* painting technique, who described it as "without lines or borders, in the manner of smoke or beyond the focus plane" [Earls 1987].

In this paper, we propose a new antialiasing algorithm which is based on recognizing certain patterns in an image. We took our inspiration in computer vision algorithms, in particular based on morphological image analysis [Soille and Rivest 1993]. Once these patterns are found, we blend colors around these patterns, aiming at achieving the most probable a posteriori estimation. As such, this new approach aspires to simulate human vision processing.

Morphological antialiasing (MLAA) has a set of unique characteristics distinguishing it from other algorithms. It is completely independent from the rendering pipeline. In effect, it can be used for both rasterization and ray tracing applications, even though we consider it naturally aligned with ray tracing algorithms, for which there is no hardware acceleration available. It represents a single post-processing kernel, which can be used in any ray tracing application without any modifications and, in fact, can be implemented on the GPU even if the main algorithm runs on the CPU.

MLAA, even in its current un-optimized implementation, is reasonably fast, processing about 20M pixels per second on a single 3GHz core. It is embarrassingly parallel and on a multi-core machine can be used to achieve better load balancing by processing the final output image in idle threads (either ones that finish rendering or ones that finish building their part of acceleration structure). Even though ray tracing rendering is highly parallelizable as such, creating or updating accelerating structures for dynamic models has certain scalability issues [Wald et al. 2007]. MLAA is well positioned to use these underutilized cycles to achieve zero impact on overall performance.

Among the chief shortcomings of the proposed algorithm is its inability to handle features smaller than the Nyquist limit. This is similar to all other techniques that rely on a single sample per pixel to find features inside an image. We discuss this problem at length in section 3 and also propose a palliative solution which aims at reducing these under-sampling artifacts.

2 MORPHOLOGICAL ANTIALIASING

MLAA is designed to reduce aliasing artifacts in displayed images without casting any additional rays. It consists of three main steps:

1. Find discontinuities between pixels in a given image.
2. Identify predefined patterns.
3. Blend colors in the neighborhood of these patterns.

For the sake of simplicity, we first describe the MLAA technique for (binary) black-and-white images, for which these three steps are trivial, and generalize it for color images later on.

2.1 BLENDING HARD SHADOWS

The practical application for black-and-white implementation of MLAA is antialiasing of hard shadows. As such, it is close to shadow silhouette maps [Sen and Cammarano 2003], which allow constructing a piecewise-linear approximation to the true shadow. The main difference is that our implementation is always pixel-accurate, as it approximates ray traced shadow rays (see also discussion at the end of this section).

Figure 2 shows a sample black-and-white image on the left. Informally, we would like to create a piecewise-linear

approximation of borders between black and white pixels (shown as red contour) in order to obtain antialiased image on the right.

We will use a chess notation for pixels on this picture: the top-left pixel will be **a8** and so on. Suffices **l**, **r**, **t**, and **b** will define edges of a particular pixel, so the right edge of **a8** is **a8r**, its bottom edge is **a8b**, etc. A single edge can have up to two different tags (**a8b = a7t**). A simple shorthand notation will also be used for groups of pixels, e.g. **{cd}4** for **{c4,d4}**.

At the first step of the algorithm, we search for all lines separating black pixels on one side and white pixels on another, keeping the longest ones. It is achieved by comparing each two adjacent rows and two adjacent columns. As an example, for rows 4 and 5 we will find a single separation line **{c4u,d4u,e4u}**, while columns **g** and **h** will have 2 different separation lines **g3r** and **g7r**. We will also assume that border pixels are extended into additional imaginary rows and columns, so pixel **c9** will be black and so on. Each separation line is defined by its two farthest vertices.

At the second step of the algorithm, we look for *other* separation lines which are orthogonal to the current one at its farthest vertices. Most separation lines will have two orthogonal lines, with an exception of lines originating at image edges. This observation allows us to classify separation lines into the following three categories:

1. A Z-shaped pattern which occurs if two orthogonal lines cross both half-planes formed by the current separation line. One example is the pattern formed by **b5r**, **{cde}4u**, **e4r** on Figure 2.

2. A U-shaped pattern for which orthogonal lines are on the same side, for example **c2r**, **{de}2b**, **e2r**. (1)

3. An L-shaped pattern which could only happen if separation line originates at image border, for example **{ab}5t**, **b5r** .

A single separation line can be a part of multiple patterns (up to four). This would happen, for example, if pixel **h4** was instead black. The algorithm does not take this into consideration and immediately processes any pattern, as soon as it is identified. This results in a slight blurring, which further dampens down unwarranted high frequencies in the image. Similarly, a single pixel can be bounded by multiple separation lines, resulting in additional blending for such pixels (e.g. **c8** and **g7** on Figure 2).
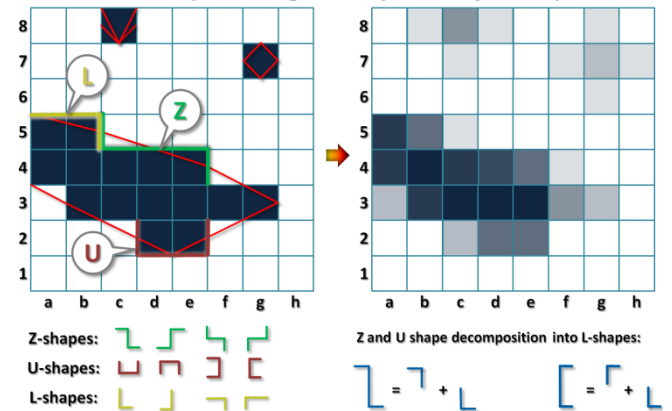

Figure 2. MLAA processing of a black-and-white image with some Z, U, and L shapes shown on the original image on the left.

Z and U shapes can be split into two L-shapes, so it will be sufficient to describe processing of L-shapes at **the third step of the algorithm**. We consider each L-shape to be formed by a primary edge found at the first step, and a secondary edge found at the second step. The length of the primary edge is 0.5 pixels or more (fraction due to the splitting of Z or U shapes), while each

secondary edge has length equal to 1 (even if it can be a part of a longer separation line, we are interested only in the fragment immediately attached to the primary edge). Any L-shape is formed by the three vertices. In order to find blending weights, we connect the middle point of the secondary edge with the remaining vertex of the primary edge (shown as the red line on Figure 3).
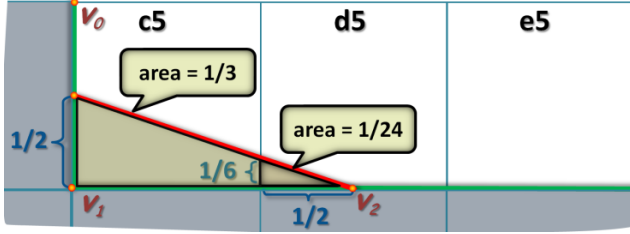

Figure 3. Computing blending weights. L-shape is formed by the secondary edge $v_0 v_1$ which is split in the middle, and the primary edge $v_1 v_2$. Blending weights are computed as trapezoid areas.

This connection line splits each cell attached to L-shape into two trapezoids. We calculate the area a of each trapezoid attached to the primary edge and use it to compute new color of these cells as follows:

$$c_{new} = (1 - a)\, c_{old} + a\, c_{opposite} \qquad (2)$$

Here c_{old} is the old color of the cell, and $c_{opposite}$ is the color of the cell on the opposite side of the primary edge (assigning 0 to black and 1 to white colors). In particular, for the cell **c5**, $a = (1/2 + 1/6)/2 = 1/3$. Its new color will be computed as $2/3 \cdot 1 + 1/3 \cdot 0$. Similarly, cell **d5** will have its new color equal to $23/24 \cdot 1 + 1/24 \cdot 0$ (almost white).

It is important to note that this one-size-fits-all approach (secondary edges are always split in half) creates a systematic error in area and color estimation. However, variance is reduced, resulting in smoother, better-looking image (see Figure 7 for further considerations).

Applied to hard shadows, black-and-white version of MLAA infers piecewise-linear approximation of a shadow silhouette from an image data. This makes it different from the silhouette maps [Sen and Cammarano 2003], in which the silhouette edges are identified using any of the techniques for shadow volumes (at a preliminary stage of the algorithm). By itself, it results in better approximation of the silhouette than using sampled image data alone. The accuracy of approximation, arguably, might suffer though at the second stage of the silhouette map algorithm when the silhouette edges are rasterized into the silhouette map at a resolution different from one of the final image. Both algorithms will exhibit artifacts when multiple disjoint silhouette edges are projected onto either single pixel (MLAA) or texel (the silhouette maps). Sen [2004] aimed at reducing these artifacts at a pre-processing step by storing boundary status, and also extending the base technique to RGB color values, suitable for the improved texture magnification.

2.2   PROCESSING COLOR IMAGES

Processing color images requires certain adjustments in basic MLAA steps. Though challenging, it also opens an opportunity to use additional data to fine-tune the algorithm and avoid the one-size-fits-all situation described at the end of the previous section.

2.2.1   DISCONTINUITIES IN COLOR IMAGE

To find discontinuities in an image, one can rely either on differences in color between neighboring pixels or on some additional information, geometric or discrete (different materials). The biggest advantage of the first approach (using color) is that this approach does not process pixels with similar color, thus avoiding extra work that would likely not change the color of these pixels in any meaningful way. On the downside, this approach may also trigger superfluous processing of pixels that get significant difference in color only through texture mapping.

MLAA as such can be used with *any* method that tells whether two pixels are different. For the remainder of this paper, we will use a threshold-based color differences without any additional information, because this approach is the most simple and universal one. Since MLAA does not cast any additional rays, it mitigates performance impediments caused by choosing low threshold value. We will also use an additional rule to decide when to actually blend pixels, as will be shown in the next section.

In all results, presented in this paper, we use only the 4 most significant bits as a similarity measure for each 8-bit red, green, and blue channel (this is comparable with DXT1 texture compression, in which all texels that have the same 5 leading bits are considered equal). If there is a difference in these bits in either channel, two colors are assumed to be different. Using either 3, 5, or 6 bits does not change results in any significant way, though for some images 3 bits are not sufficient to find all aliased pixels that are visible by a human eye.

2.2.2   PATTERN SEARCH

We will start deriving rules for searching patterns in color images by noticing that the color blending expression (2) does not depend on numerical values for black (0) and white (1) colors. It could be used for RGBA colors as well, separately for each channel. For black-and-white (B&W) images, colors computed according to this equation are changing smoothly when different Z, U, or L shapes are adjacent to each other. It is a consequence of using middle points of the secondary edges as it defines continuous piecewise-linear approximation of silhouette edges. For color images, we will relax this restriction, but will still require continuous silhouette approximation.
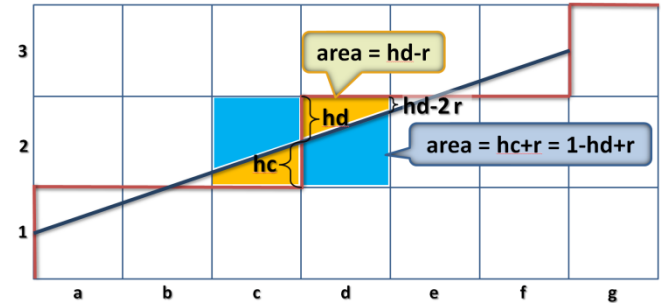

Figure 4. Stitching two shapes together. We choose unknown heights *hc* and *hd* to minimize color differences at stitching position.

Figure 4 shows two adjacent Z-shapes formed, in this particular case, by a linear silhouette edge. Colors of all pixels below the red separation line are similar (or the same for B&W images), as well as colors of pixels above the red line. For B&W images, $hc = hd = 1/2$ and $r = hd/l$, where l is the length of the Z-shape. The ratio r characterizes the relation between the two trapezoid bases, and the trapezoid's area is the average of its two bases. If the lengths of two connected shapes are the same, then two blue (bigger) trapezoids are congruent, as are the two yellow (smaller) trapezoids. We will use cell names to represent their colors. To state the requirements for smooth transition between two shapes, we compute the same blended color twice, using parameters of each shape:

$$(hc - r)\, c1 + (1 - hc + r)\, c2 \; = \; (hd - r)\, d2 + (1 - hd + r)\, d3$$
$$hc + hd \; = \; 1 \qquad (3)$$

For images with $c1 = d2$ (two pixels below the red line) and $c2 = d3$ (above), the first equation is simplified to $hc = hd$. The solution is $hc = hd = 1/2$, which is in the perfect agreement with

B&W version of MLAA. For color images, we would like to solve these equations for unknown split variables hc and hd. We will need a definition for $c1, c2, d2,$ and $d3$ variables in these equations. One possibility is to solve the equations (3) separately for each channel and then average the solution. We adopted another, simpler, approach by summing all three red, green, and blue values to form a single numerical value, representing a given pixel. To avoid mixing integer and float values, for the CPU version of MLAA we did not assign different weights to different channels, deferring more advanced luminance processing until GPU implementation. These aggregate values are used only to find split values hc and hd, color blending will be done separately for each channel.

To find patterns which result in smooth color blending, we will test all possible candidates (shapes bounded by color discontinuities) by solving linear equations (3). If the found values of hc and hd are in the [0, 1] interval, we process the tested shape, otherwise ignore it. There still could be a situation, when one separation line forms multiple valid patterns, especially in textured areas. For simplicity, we process only the first found shape of a certain kind (two possible Z-shapes and two U-shapes) and then move on.

By providing an additional criterion for accepting or rejecting tested patterns, we avoid unnecessary computations due to setting the color discontinuity threshold too low. Moreover, we assure smooth transition between connected shapes.

2.3 TEXTURES

By design, we are not paying any particular attention to textures, considering them as an integral part of a given image. For textures with big color differences between neighboring pixels, this will result in blending of these pixels. Perhaps, this should have been done in the first place by using mipmapping. In any case, it does not affect the output image adversely.

Figure 9 shows morphological antialiasing of the Edgar model, which is a part of the digital content provided with the 3D Poser editor from Smith Micro. On the three bottom images, processed pixels are shown with a different color. When the figure is viewed from a distance (left image), pixels in Edgar's tie have significant discontinuities and are chosen for MLAA processing. This does not adversely affect the quality of the output image (top right) and is roughly equivalent to mipmapping of the corresponding texels. When zooming in on the model (bottom middle and right), the number of processed pixels decreases as texel interpolation kicks in. At any distance to the model, no pixels corresponding to Edgar's shirt are chosen for blending, despite the high-frequency texture which is used for it.

2.4 OPTIMIZATION

The first step of MLAA (searching for color discontinuities) is executed for every pixel. Fortunately, it allows efficient SSE® implementation. By using 8 bits for color channel, each RGBA color requires 32 bits, so 4 RGBA pixels will fit into one SSE register. The next C++ code fragment shows our implementation of **ssedif()** function, which compares 4 RGBA values using SSE2 instruction set:

```
#define most_significant_color_bits char(0xff ^ ((1<<4) - 1))
unsigned short int ssedif(const __m128i& c0, const __m128i& c1) {
    // return 16 bits set to 1 for each channel where c0,c1 are different.
    __m128i hibits = _mm_set1_epi8(most_significant_color_bits);
    __m128i d = _mm_sub_epi8(_mm_max_epu8(c0, c1),
                             _mm_min_epu8 (c0, c1));
    d = _mm_and_si128(d, hibits);
    d = _mm_cmpeq_epi8(d, _mm_setzero_si128());
    return 0xffff ^ _mm_movemask_epi8(d);
}
```

The implementation is pretty straightforward, allowing simultaneous processing of sixteen 8-bit values. We compute the absolute differences of 4 RGBA colors (variable **d**), keep only the most significant bits, and compare the result with 0. Note that we cannot compare the variable **d** directly with a threshold value since there are no comparison operations for unsigned 8-bit values in SSE® set. If an arbitrary threshold is required, this function could be easily re-implemented with available instructions, albeit less efficiently.

We use this function for finding discontinuities between each two adjacent rows and each two columns in the original image. Since pixels are stored in a framebuffer row-wise, accessing 4 colors in a row (sixteen 8-bit channels) can be achieved with one **_mm_load_si128** instruction. For columns, we use **_MM_TRANSPOSE4_PS** macro [Intel] to convert data into SSE-friendly format.

We did not try to implement other MLAA steps using SSE® instructions (though it might be possible to do it using SSE4 operations), opting instead for preserving the universal nature of the algorithm. The upcoming Larrabee chip [Seiler et al. 2008], as well as modern GPU cards, are capable of handling 8-bit data extremely efficiently, so our algorithm will benefit from porting to these architectures.
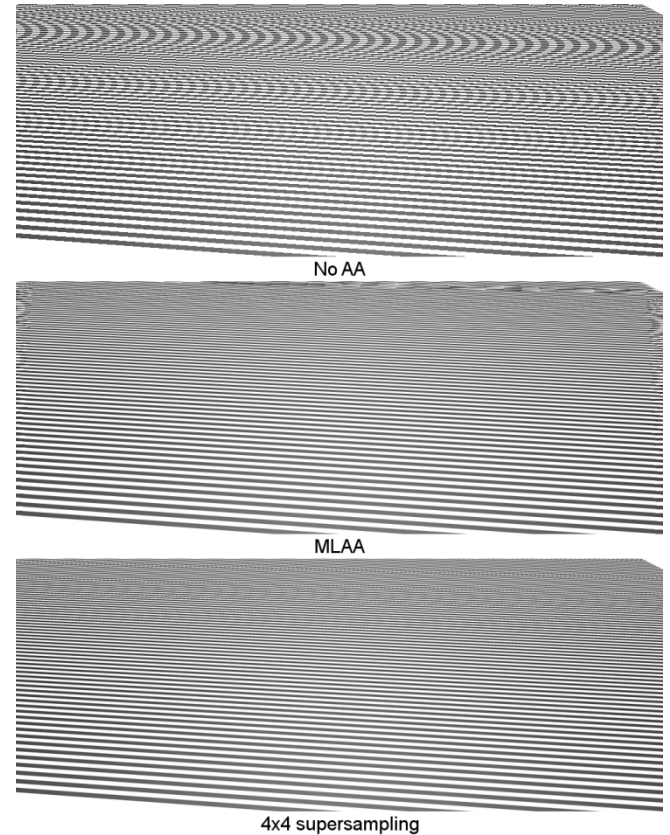

No AA

MLAA

4x4 supersampling

Figure 5. A model consisting of 100 elongated quads (no texturing).

3 MLAA LIMITATIONS

Arguably the biggest MLAA limitation is its inability to handle pixel-size features. This problem is illustrated in Figure 5, which shows 100 elongated quads at a resolution of 1024x512 pixels. MLAA takes the aliased image shown on the top and produces the one in the middle. Despite being very narrow, quads, which are closer to the camera, span multiple pixels. For these quads, MLAA is working just fine, removing higher frequencies from the image. Closer to the top of the image though, there are no identifiable

features, resulting in *moiré* pattern, which is a typical example of geometrical aliasing. Similarly, if presented with unfiltered texture, MLAA will try to remove higher frequencies where possible and then give up. Of course, textures could be reliably filtered with mipmapping.

Geometrical aliasing could happen not only for very small or thin objects, but also when just few rays penetrate space between bigger objects. Another type of aliasing, which is characteristic for ray tracing applications, may happen when shading normals are used. Indeed, when an angle between incoming rays and *geometrical* normal is close to $\pi/2$, for *shading* normals the angle could become bigger than $\pi/2$, causing reflected rays to go in wrong direction (inside the object). This usually happens just for few pixels resulting in an aliasing effect.

Isolated pixels are naturally blended with neighboring ones after MLAA pass, since these pixels belong to multiple shapes. It is even possible to identify all such pixels or small groups of pixels and then apply additional blending to them. However, this will not help for certain moiré patterns, for which aliased images appear to have some structure (caused by interference of multiple frequencies).

A somewhat less noticeable drawback of MLAA is its handling of border pixels, for which no full information about neighborhood is available. This is similar to all adaptive techniques that rely on a single sample per pixel. Another type of problem is specific for MLAA. Because MLAA uses only image data for the reconstruction, very slowly changing scenes might produce non-smooth transition between frames. Images at different time steps might have identical sets of morphological shapes if camera or objects are moving at sub-pixel speed (see discussion related to Figure 7). In practice, it is not perceptible if the output resolution is higher than 200x200 pixels. For color images, even if two images have identical sets of shapes, MLAA output might be different, since interpolation coefficients depend on color values, to certain degree allowing reconstruction of sub-pixel motion.

MLAA processing of text could result in additional distortion, since MLAA does not distinguish between foreground and background. It is especially noticeable if original font was already antialiased (see Figure 6).
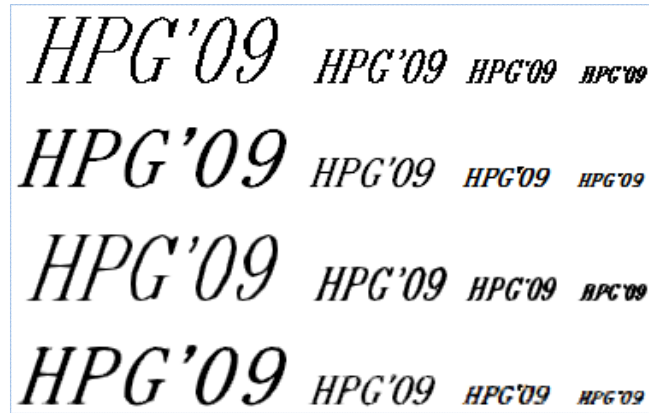

Figure 6. Top row: bitmap font; second row: TrueType font (antialiased), third row: MLAA processing of bitmap font; fourth row: MLAA processing of TrueType font. Font size from left to right: 24, 12, 8, and 6.

4 COMPARISON WITH SUPERSAMPLING

To understand similarities and differences between MLAA and other antialiasing techniques, we created a very simple example by rendering a model consisting of a single half-plane. Figure 7 shows six pixels separating white (top) and black (bottom) areas.

We compute pixel coverage analytically (black line on the top chart) and compare it with MLAA (blue) and two supersampling approaches, using 4 and 16 samples per pixel. All techniques estimate area with 15% accuracy, while MLAA provides a linear estimation, without any unwarranted frequencies. At the same time, MLAA estimation will be the same for all images in which separation line intersects two black arrow-ended intervals on the bottom chart. MLAA sacrifices accuracy of local estimation in favor of maintaining smoother global silhouette approximation, stitching it at different morphological shapes.
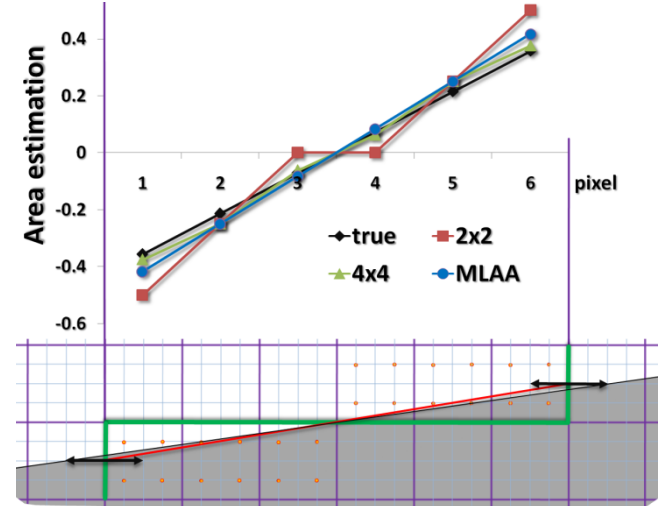

Figure 7. Sampling 6 pixels separating black-and-white areas (2x2 samples per pixel are shown). Bottom image: Z-shape is bounded by the green line, while red line shows MLAA reconstruction of "true" silhouette edge. MLAA reconstruction will be the same for all "true" curves intersecting two black arrow-ended intervals.

On Figure 8, the half-plane silhouette is almost horizontal, straddling 24 pixels per Z-shape. For this view, 2x2 sampling results in significant errors in area estimation. For MLAA, actually the opposite is true since a longer Z-shape allows for smaller variability in the true image. Accordingly, to match MLAA quality, 8x8 sampling is required. These examples also illustrate the well-known fact that uniform sampling is not the best approach to reduce variance [Laine and Aila 2006], and supersampling or adaptive antialiasing techniques would benefit from using different sampling patterns. While very common in global illumination algorithms, low variance sampling patterns are rarely used for antialiasing in ray tracing applications (though it is a common practice in hardware-accelerated rasterization). One early example of using stochastic sampling, particularly for ray tracing, is given in [Dippe and Wold 1985].
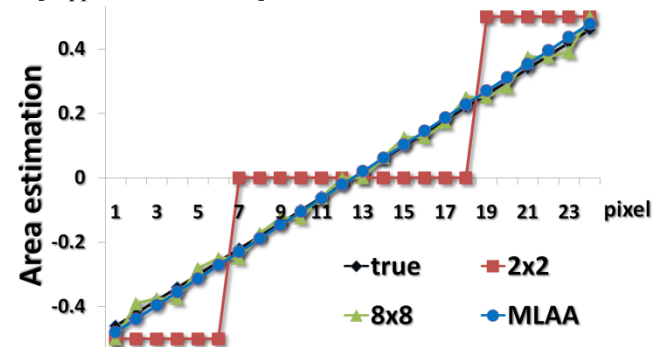

Figure 8. Sampling 24 pixels.

5 SUMMARY AND FUTURE WORK

We have proposed a new approach to antialiasing based on searching for certain shapes in images and processing pixels adjacent to these shapes according to a set of simple rules. This removes artificial high frequencies due to a discrete sampling. The new approach has unique characteristics that, we hope, will make it worthwhile considering, along with other algorithms:

– MLAA can be used for any image processing task and does not use any data besides color values. It is very simple and does not require any modifications to the rendering pipeline. It does not need casting any additional rays.

– The algorithm is embarrassingly parallel and can be executed concurrently with rendering threads (using double buffering), allowing for better processor utilization.

– The algorithm achieves reasonable performance, in many cases without any noticeable impact.

– The quality is comparable with supersampling. MLAA has very different approximation characteristics, achieving better results in situations in which uniform sampling suffers from significant errors and undesired frequencies.

In order to illustrate our technique's universal nature, we have also applied it to images from "Call of Duty®: World at War" (see Figure 10). There wasn't any particular reason for choosing this game, except that it came free with a new GPU card. We turned hardware antialiasing off and ran the game at lower resolution to make artifacts visible in printed form.

If additional information is available (besides color values), MLAA could use it to fine-tune the first step (search of discontinuities) or implement additional effects. In particular, we are planning to use it for computing approximate soft shadows by expanding blended area proportional to a distance to a light source, similar to smoothies described in [Chan and Durand 2003].

Another possibility is to use our technique to enhance the resolution of output images by using constraint interpolation in spirit of [Bala et al. 2003], or utilize sub-pixel capabilities of modern LCDs.

We are also interested in using color spaces other than RGB, and studying non-linear color comparison and blending techniques.

ACKNOWLEDGEMENTS

REFERENCES

AMANATIDES J. 1984. Ray Tracing with Cones. In *Computer Graphics*, vol. 18(3), 129-135.

ATI CrossFire™ Technology White Paper, http://ati.amd.com/technology/crossfire/CrossFireWhitePaperweb2.pdf

BALA K., WALTER B., GREENBERG D. 2003. Combining Edges and Points for Interactive High-Quality Rendering. *ACM Transactions on Graphics* (Proceedings of ACM SIGGRAPH 2003), 22(3), 631-640.

CHAN E. and DURAND F. 2003. Rendering Fake Soft Shadows With Smoothies. In *Proceedings of the 14th Eurographics workshop on Rendering*, 208-218.

CHAN E. and DURAND F. 2005. Fast Prefiltered Lines. In Matt Pharr, ed., *GPU Gems 2*. ISBN 0321335597. Addison-Wesley.

DIPPE M. and WOLD E.H. 1985. Antialiasing Through Stochastic Sampling. *ACM Transactions on Graphics* (Proceedings of ACM SIGGRAPH 1985), 19(3), 69-78.

EARLS I. 1987. Renaissance Art: A Topical Dictionary. ISBN 0313246580. Greenwood Press. 263-263.

GENETTI J., GORDON D., and WILLIAM G. 1998. Adaptive Super-sampling in Object Space using Pyramidal Rays, *Computer Graphics Forum*, vol. 17, 29-54.

HECKBERT P. and HANRAHAN P. 1984. Beam Tracing Polygonal Objects, In *Computer Graphics*, vol. 18(3), 119-127.

HOFER H., CARROLL J., NEITZ J., NEITZ M., and WILLIAMS D.R. 2005. Organization of the Human Trichromatic Cone Mosaic. *The Journal of Neuroscience*, 25(42), 9669-79.

Intel® C++ Compiler User and Reference Guides, software.intel.com/en-us/intel-compilers

KELLER A. 1998. *Quasi-Monte Carlo Methods for Realistic Image Synthesis*. PhD thesis, University of Kaiserslautern.

LAINE S. and AILA T. 2006. A Weighted Error Metric and Optimization Method for Antialiasing Patterns. *Computer Graphics Forum* 25(1), 83-94.

MARTIN W., REINHARD E., SHIRLEY P., PARKER P., and THOMPSON W. 2002. Temporally Coherent Interactive Ray Tracing. *Journal of Graphics Tools*, vol. 2, 41-48.

NVIDIA. Coverage-Sampled Antialiasing, developer.download.nvidia.com/whitepapers/2007/SDK10/CSAATutorial.pdf

OHTA M. and MAEKAWA M. 1990. Ray-Bound Tracing For Perfect And Efficient Anti-Aliasing. *The Visual Computer*, 6(3), 125-133.

SANDER P. V., HOPPE H., SNYDER J. and GORTLER S. J. 2001. Discontinuity Edge Overdraw. In *Proceedings 2001 Symposium on Interactive 3D Graphics*, 167-174.

SEILER L., CARMEAN D., SPRANGLE E., FORSYTH T., ABRASH M., DUBEY P., JUNKINS S., LAKE A., SUGERMAN J., CAVIN R., ESPASA R., GROCHOWSKI E., JUAN T., and HANRAHAN P. 2008. Larrabee: A Many-Core x86 Architecture For Visual Computing. In *ACM Transactions on Graphics* (Proceedings of ACM SIGGRAPH 2008), 27(3), 1-15.

SEN P. and CAMMARANO M. 2003. Shadow Silhouette Maps. In *ACM Transactions on Graphics* (Proceedings of ACM SIGGRAPH 2003), 22(3), 521-526.

SEN P. 2004. Silhouette Maps For Improved Texture Magnification. *Graphics Hardware* (Proceedings of ACM SIGGRAPH/EURO-GRAPHICS 2004), 65-73.

SHINYA M., TAKAHASHI T., and NAITO S. 1987. Principles and Applications of Pencil Tracing, In *Computer Graphics*, vol. 21(4), 45-54.

SOILLE P. and RIVEST J.-F. 1993. Principles And Applications Of Morphological Image Analysis. ISBN 0646128450.

THOMAS D., NETRAVALI A.N., and FOX D.S. 1989. Anti-aliased Ray Tracing with Covers, *Computer Graphics Forum*, vol. 8(4), 325-336.

WALD I., MARK W., GÜNTHER J., BOULOS S., IZE T., HUNT W., PARKER S.G., and SHIRLEY P. 2007. State of the Art in Ray Tracing Animated Scenes. *Eurographics 2007 State of the Art Reports*, 89-116.

WHITTED T. 1980. An Improved Illumination Model for Shaded Display, *Commun. ACM*, vol. 23(6), 343-349.
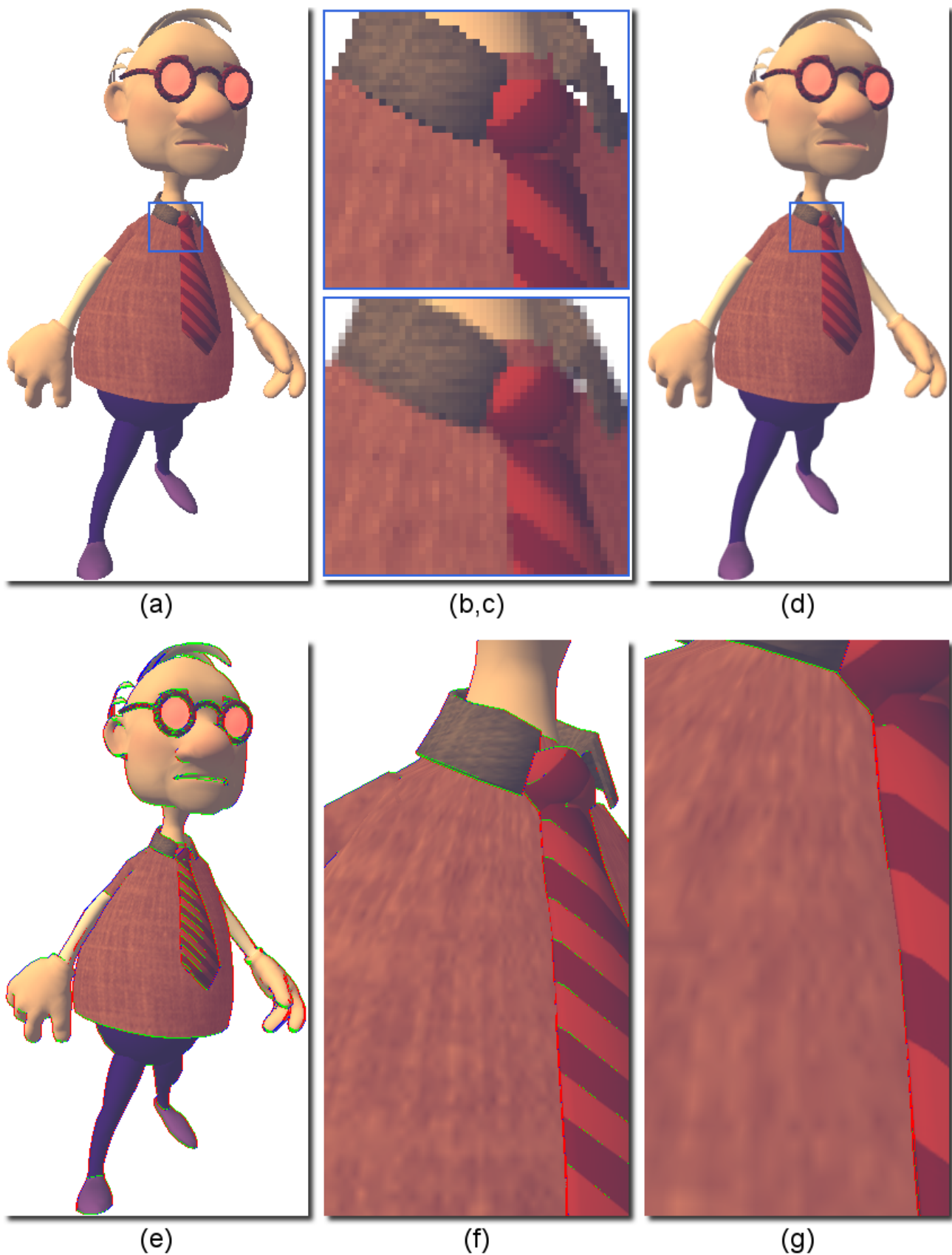
Figure 9. Processing textures for the Edgar model: (a) – original aliased image; (d) – antialiased image, processed with MLAA; (b,c) – enlarged regions of left and right images; (e,f,g) – visualization of pixels processed with MLAA at different zoom levels. Pixels belonging to horizontal shapes are marked with green, vertical – with red color. Pixels, which are included into horizontal and vertical shapes simultaneously, are shown as blue. Note that aliased pixels (top left) were unintentionally blurred when this paper was created (both in electronic and paper form).

Figure 10. A jungle scene from **"Call of Duty®: World at War"** game. This screenshot was captured at 640x480 resolution with geometric antialiasing turned off but texture antialiasing set at best quality. Note that some of the vegetation is implemented with transparent textures and, therefore, antialiased in hardware. There are also far-away billboards. MLAA handles all these cases uniformly by analyzing color discontinuities.