Transparency and Anti-Aliasing Techniques for Real-Time Rendering

Marilena Maule*, João L. D. Comba* Rafael Torchelsen† Rui Bastos‡
*UFRGS
{mmaule,comba}@inf.ufrgs.br
†UFFS
rafael.torchelsen@uffs.edu.br
‡NVIDIA
rbastos@nvidia.com

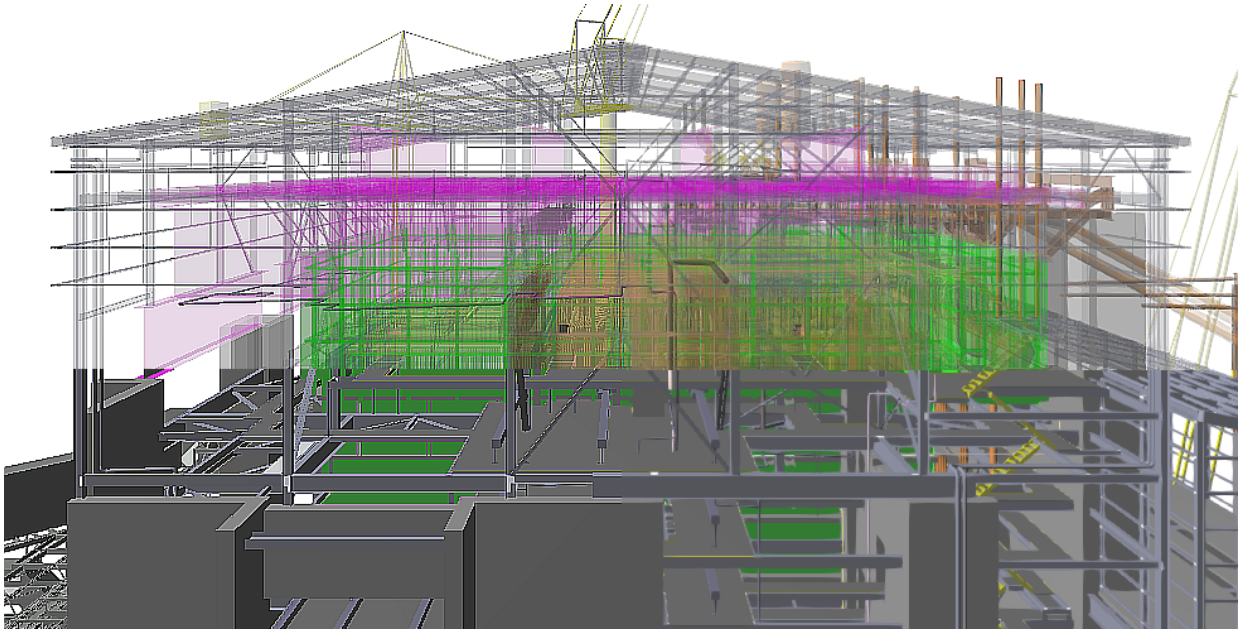

Fig. 1. **Transparency and Anti-Aliasing** play an important role in visual cues and image quality of computer-generated images. Achieving real-time rendering with high-quality images is challenging. In this tutorial, we summarize state-of-the-art techniques to handle both problems.

***Abstract*—Transparency and anti-aliasing are crucial to enhance realism in computer-generated images, which have a high demand for such effects. Transparency is largely used to denote relationships among objects in a scene, and to render several structures, such as particles and foliage. Anti-aliasing (AA) is also important, since jagged edges can be easily spotted and create disruptive distractions during a scene walkthrough, which are unacceptable in real-time applications. Figure 1 illustrates both effects. In common, they have the fact that they rely on processing discrete samples from a given function, but using the samples for different purposes. In this tutorial we review state-of-the-art techniques for transparency and anti-aliasing effects, their initial ideas and subsequent GPU accelerations. We support our presentation with a discussion on their strengths and limitations.**

***Keywords*-Transparency, Anti-Aliasing, Real-time.**

I. Introduction

Transparency is an important rendering effect, used often without being noticed by the viewer, as in hair, grass, smoke, fire, snow and rain. Its presence is obvious only on surfaces of transparent materials, such as glass and plastic. Anti-aliasing, in turn, is noticed the most when it is not applied, since its absence causes the unpleasant sensation of discontinuity in computed generated images. On the other hand, when applied, transitions at the boundaries of objects in images become smooth.

The key to render transparency effects is the integration of the contributions of several fragments, which are generated along the viewing ray that passes through a pixel center. Figure 2 illustrates the transparency problem. For the correct computation of colors associated with a given pixel, fragments must be processed in visibility ordering (either back-to-front or front-to-back) using the blending equations described in [1]. There are different ways to achieve the proper ordering of fragments. For example, sorting primitives in object-space before they are rendered and blended is one approach. Another approach is to defer the sorting to be performed at the fragment

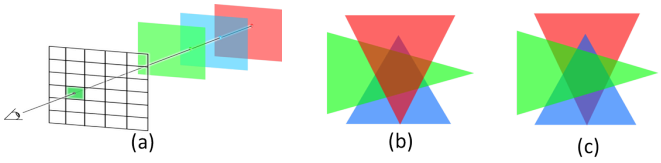

Fig. 2. The transparency problem: (a) the view ray for a pixel traversing the scene and hiting transparent surfaces. (b) out-of-order composition of fragments leading to incorrect colors. (c) depth-sorted composition of fragments generating correct colors.

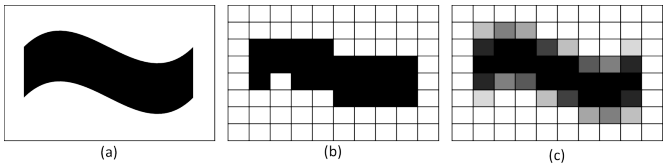

Fig. 3. The aliasing problem: (a) the curved shape to be drawn. (b) the aliased result from sampling the pixels centers. (c) the anti-aliased solution: each pixel receives an amount of color corresponding to the percentage of its area covered by the shape.

level, which does not impose an ordering on the primitives sent to the graphics pipeline. The techniques that produce correct and invariant transparency results are classified as Order-Independent Transparency (OIT). Fragment-level approaches have finer computational costs due to the high granularity of samples compared to primitives, and the variable and unbound memory usage. Detailed techniques and analysis are provided in our companion survey about transparency rendering [2].

Transparency and anti-aliasing techniques both manipulate multiple samples per pixel, but with a different purposes. While the main goal of anti-aliasing is also to integrate the contributions of fragments, these come from portions of the scene that are projected into different locations within the pixel area. The different fragment contributions must be weighted by their coverage over the pixel area. Figure 3 illustrates the anti-aliasing problem. As can be seen, aliasing can create disruptive transitions (Figure 3 (b)), which can be reduced by anti-aliasing techniques (Figure 3 (c)), which soften edges to better capture details that have higher frequency than the sampling rate. AA techniques can benefit from dedicated hardware such as GPUs, with some GPUs already having support for AA in hardware. While good results with respect to quality and performance are obtained for static scenes, to maintain temporal coherence (coherence between frames) is still difficult in real-time applications, especially when relying only on color information.

In this tutorial we review different OIT and AA approaches, which serves to understand the techniques separately, and builds the foundation for algorithms that combine both effects. Section II briefly reviews recent transparency techniques, and we refer the reader to the companion tutorial on transparency techniques for the remaining algorithms [2]. Our emphasis lies mostly on AA techniques, described in Section III. Section IV discusses the combination of these effects.

II. Transparency Rendering Techniques

Transparency rendering relies on sorting transparent primitives with respect to their distance to the viewer. In object space it is done by sorting meshes and triangles, but this approach can lead to artifacts when dealing with interpenetrating primitives. To avoid such problems, the sorting must be done at fragment level.

Sorting of fragments can be performed by storing them into buffers during rendering, as proposed by Carpenter in the A-Buffer algorithm [3]. In the A-buffer approach, once all fragments of a pixel are computed, they are sorted and blended to produce the final pixel color. Another way to achieve the same result is called depth peeling, which uses geometry step to incrementally compute depth layers in visibility order [4]. A third way, which achieves approximate results, is to first estimate the visibility of each fragment, and use this information in the blending step, directly [5] or stochastically [6].

We defer the reader to the companion tutorial on transparency rendering [2]. To complement that survey, we describe below three techniques that rely in buffers to store information before sorting.

A. Adaptive Transparency (AT)

Adaptive Transparency [7] is an OIT technique that combines fragments out-of depth-sorted order by processing the geometry twice. The first geometry pass computes and stores a visibility function, which is used to weigh the composition of shaded fragments in the second pass.

First, the geometry is rendered without shading. The depth value of the fragments is used to build a visibility function inside a fixed size buffer per pixel, which corresponds to storing the visibility and depth of some fragments in depth-sorted order. During the insertion of the fragment as a node of the list, visibility is computed by combining the fragment alpha and the visibility list using the principles described by Porter and Duff [1]. In case of overflow, the node removed is the one that causes the smallest modification in the area below the function. As shown in Figure 4, visibility may be over or underestimated. In the second geometry pass, the depth of the shaded fragment is used to evaluate its visibility according to the previously computed function. This can be achieved by searching the list for the depth interval containing the fragment z, and getting the visibility at that point.

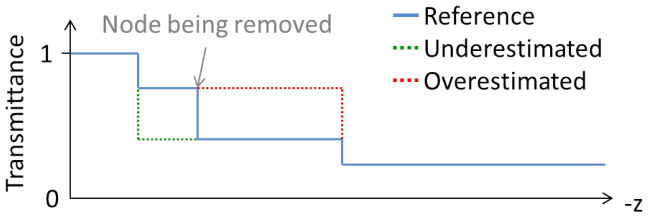

Fig. 4. Adaptive Transparency visibility compression: A node removal may lead to an overestimation (red), or to an underestimation (green), of the remaining visibility, indicated in the transmittance axis. Image adapted from [7].

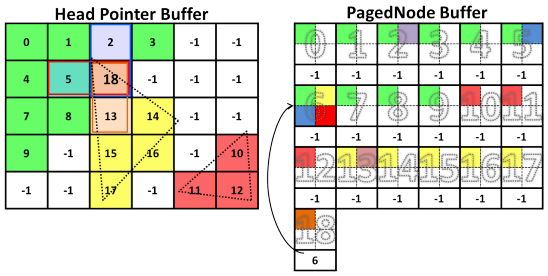

Fig. 5. Per-Pixel Paged Linked Lists: example showing three transparent triangles and three transparent lines, which are illustrated over the Head Pointer Buffer. The **Head Pointer Buffer** keeps a pointer to the PagedNode Buffer, which indexes the page of the last received fragment from the respective pixel. For example, number 18 in the Head Pointer Buffer indicates that the last fragment for that pixel entry is stored in page 18 of the PagedNode Buffer. The **PagedNode Buffer** keeps fragment attributes in pages of size four per node, with one index to the next page and if there is no more pages, it indicates with -1. For example, in page 18, the number 6 indicates that the continuation of the list is in page 6. Image modified from [2].

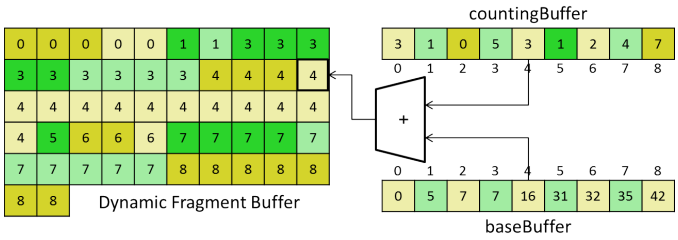

Fig. 6. DFB storage. The pixel-correspondent value from the *countingBuffer* is added to the associated index from the *baseBuffer*. This composed address stores the incoming fragment into a free position in the shared buffer.

Since the technique only stores depth and visibility values, it uses less memory than methods that also store fragment colors. The price of using less memory is an extra geometry pass to shade the fragments.

B. Per Pixel Paged Linked Lists (PPPLL)

PPPLL [8] is an extension of the Per Pixel Linked Lists (PPLL), proposed by Yang et al. [9]. While PPLL builds a linked list of nodes with one fragment per node, PPPLL allows storing more fragments per node, favoring memory spatial locality.

The technique works in a single geometry pass, by storing all incoming fragments in a shared buffer. These fragments are blended in a post-processing phase. To emulate the linked list per pixel, it requires two buffers. The first one has one entry per pixel, which points to the head of the pixel list into the shared buffer. The shared buffer has nodes composed by a pointer to the next node and a page with up to k fragments. When the page of a node is full, a new fragment causes the allocation of a next node, which is indexed by the previous node pointer. Figure 5 illustrates the technique.

Node allocation is protected by critical sections, which reduces parallelism. Since it does not count fragments, in overflow cases the re-allocation of the shared buffer is heuristical and can be repeated until the storage contains the required size.

C. Dynamic Fragment Buffer (DFB)

DFB[10] is a two-pass OIT technique. It uses a first geometry pass to count fragments and allocate the exact memory to store them. In the second pass, it stores the fragments, which are sorted and blended in a post-processing phase.

First, a geometry pass is performed only to count the number of fragments generated per pixel into a buffer. An intermediate step uses the number of fragments to compute a base+displacement index scheme, which is used to allocate a consecutive amount of memory per pixel inside a shared buffer. A second geometry pass shades and stores the fragments in the reserved space for the respective pixel. At the end, the lists of each pixel are individually sorted and blended to compose the final pixel color. Figure 6 illustrates this memory scheme.

DFB is a memory-efficient algorithm, able to allocate the precise amount of storage required by each frame. In order to do that, it needs to process the geometry twice.

III. Anti-Aliasing Techniques

Anti-aliasing is important because it increases the perceptual quality of computer-generated images. This feature is desirable, but high-quality anti-aliasing is costly (both in terms of processing and memory). The approaches described here assume tradeoffs between computational cost and image quality, often ignoring certain kinds of aliasing in order to provide better performance.

We present techniques classified into three main classes: (i) Full-Scene Anti-Aliasing (FSAA), (ii) Image Post-Processing Anti-Aliasing (IAA) and (iii) Geometric Anti-Aliasing (GAA). Each one presents a singular general approach to the aliasing problem, with different scope and limitations.

The first and most intuitive idea for solving undersampling artifacts is to take more samples. That is precisely what the FSAA approaches do. With more samples, fragment to pixel coverage is better approximated, producing more pleasant edges with smoother transitions.

The result of eliminating aliasing from high frequencies is blurred edges. Minor blur is less objectionable to the human eye than aliasing is. That is the basic idea of the AA approaches, which search for aliasing directly in the final image, and remove it by fading high frequencies.

Most of the aliasing problems originate from the edges and silhouettes of geometric objects. With the intent to solve them, ignoring the other sources of aliasing, the GAA approaches concentrate their efforts in processing the geometric edges, weighting their coverage, so they will appear smooth in the final image.

Below we describe a collection of techniques that follow these three approaches.

A. Full-Scene Anti-Aliasing

Full scene means that all parts of the scene get some processing for AA—even if they are not visible. This is the

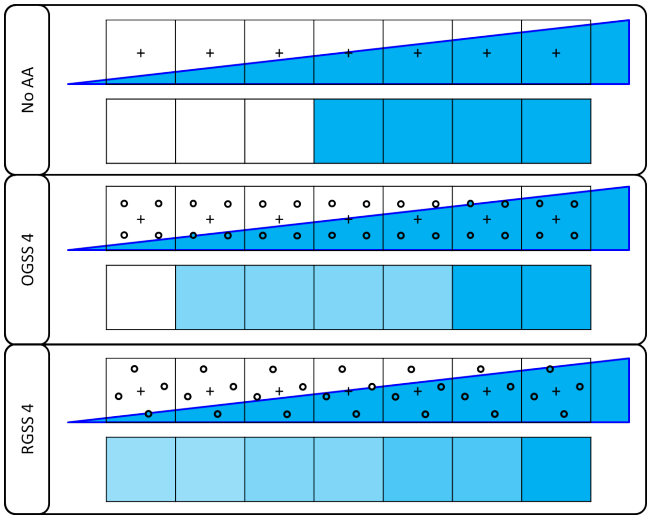

Fig. 7. Critical edge angle for ordered and rotated grid sample techniques: The worst case for OGSS are nearly-vertical/horizontal edges. In such cases OGSS loses the ability to provide all its shading levels, reducing smoothness. In the same scenario, RGSS provides more shading levels due to its rotated samples distribution. Image modified from [11].

first, simplest and most intuitive anti-aliasing approach. The key idea is to sample the scene at higher frequency than what is needed for display, by rendering to a higher resolution and, then, applying a filter to downsample to the desired resolution.

There are variations among the FSAA techniques that optimize the distribution of the samples inside the pixel area [11]. The simplest is the ordered grid (OGSS), in which the samples are taken from a regular subpixel distribution. One implementation for this technique is to render to a screen n times larger than the desired resolution. For example, for a screen with size WxH, a 4-OGSS takes four samples per pixel by rendering to a framebuffer $2W$x$2H$, and downsampling with a box filter to combine each four pixels into one.

However, OGSS has problems to solve the most visible aliasing cases: nearly-vertical and nearly-horizontal edges, as you can see in Figure 7. This figure also shows how the rotated grid technique (RGSS) does not share this limitation, providing more shading levels and, consequently, smoother edges for the nearly-vertical and nearly-horizontal edge cases.

RGSS consists in rotating the grid of samples inside each pixel. A rotation of around 30 degrees leads to good results. This simple rotation changes the critical angle (in which less shading levels can be provided) to a less disturbing edge angle, providing more pleasing images.

Regular patterns are easily perceived by our eyes, so, other distributions were proposed to alleviate the critical angle cases: (i) the random distribution, (ii) the Poisson distribution and (iii) the jittered sampling. Figure 8 illustrates these approaches.

Increasing the number of samples taken for each output (or displayable) pixel always reduces the artifacts of all distributions. To halve the aliasing, four times more samples are needed. Below, we discuss techniques that use more than one sample per output pixel with different approaches.

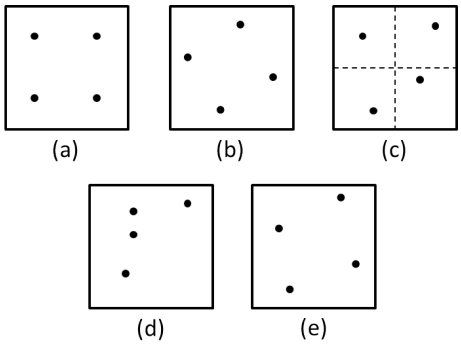

Fig. 8. Super Sampling distributions: (a) Ordered Grid. (b) Rotated Grid. (c) Jittered Grid. (d) Random. (e) Poisson.

1) Super Sampling Anti-Aliasing (SSAA): In SSAA[11] the number of input pixels is increased with respect to the number of output pixels and there is a sample for each input pixel. This means that all the attributes are evaluated and stored for each sample (ex. depth, color, normal and texture coordinates. See Fig. 9). This approach gives the best image quality, at the cost of fully computing these attributes per sample.

2) Multi-Sampling Anti-Aliasing (MSAA): The key difference between MSAA[11] and SSAA is the amount of information super-sampled. In other words, MSAA is an SSAA where some pixel attributes are not evaluated for every sample, commonly this attribute is the shaded color. In MSAA, these attributes are evaluated at the center of the pixel and copied to each sample of the pixel (see Fig. 9).

One example of MSAA is to take four samples per pixel, computing for each sample its own depth and stencil values, but each of those samples receiving the same shaded color sampled at the pixel center. The difference from SSAA is that the shading computation is performed only once per output pixel, saving processing time; while the memory requirements are the same for both approaches.

3) Coverage-Sampled Anti-Aliasing (CSAA): This technique, called CSAA (by NVIDIA[12]) or Enhanced-Quality Anti-Aliasing (EQAA) (by AMD[13]), uses the standard MSAA approach combined with extra samples per pixel to better capture pixel coverage, as shown in Figure 9. Some of the samples (not necessary half) capture color, depth, and location within the pixel. The remaining samples do not receive any of these attributes. They are only used to capture the fragment coverage at some location, in order to weight the contribution of the fragment to the final pixel color. In this web page [14] you can find details and image comparisons for CSAA and EQAA.

The image quality is improved by the extra coverage information, which better approximates a contribution of a fragment over the pixel area. However, the results are order-dependent since the coverage samples are not directly subject to the depth test (so, may be overwritten).

4) Directionally-Adaptive Edge AA (DAEAA): DAEAA [15][16] consists of an MSAA process with improved image filtering in the final stage. The hardware-optimized MSAA is

used to take samples of each pixel. These samples approximate the value of isolines passing inside a pixel, which estimate the primitive coverage and provide better color weighting.

Pixels are selected to be processed when they present different MSAA sample colors inside them, which means that they are from more than one fragment and the pixel may present aliasing (or, at least, a pixel that needs processing). Some regions, e.g. corners, may be excessively blurred, so they are masked before processing. For the remaining pixels, the gradient of primitive edges is computed by sampled isolines (see Fig. 9).

The isolines are calculated as straight lines, assuming low curvature along the pixel area. So, the function values, which the isolines represent, are approximated by the tangent plane at the pixel center, and resolved by least squares with the pixel samples. The last step is a stochastic integration, using a 3x3 box filter over the samples, which are weighted by the isoline length inside the pixel.

The image quality of 4xDAEAA is comparable with a 16xMSAA [15], at the cost of four full-screen shader passes.

5) Subpixel-Reconstruction Anti-Aliasing (SRAA): [16][17] this technique was developed to work with deferred shading[18](DS). It is inspired by MLAA and MSAA techniques, and operates as a post-process over more information than the simple color buffer. The technique super samples depth and geometry normal attributes, and uses them to fill gaps among pixels of a normal size color buffer, in other words, it builds a virtual super-sampled image, then downsamples to the output resolution.

A separate forward geometry pass is performed to construct the super-sampled depth and geometry normal buffers, by using the hardware MSAA. The DS pass is performed at final resolution and, after fragment shading, a super-sampled image is reconstructed by a cross-bilateral filter. The reconstruction takes depth and normal samples from a neighborhood, weighted by their distance to the pixel center. This process produces subpixel information, which is combined by a box filter to produce the final image with lower resolution.

To take advantage of this technique, a high shading costs is required in order to hide the AA poor performance, since the techniques needs an extra geometry pass to collect MSAA depths and normals, and the final filters are computationally intensive.

6) Enhanced Subpixel Morphological AA (SMAA): [19] was built on top of the PMLAA pipeline, with improvements in edge detection, subpixel-feature management and temporal stability. It uses local luma contrast and a wider neighborhood search to detect edges, MSAA to improve subpixel features and temporal SSAA re-projection to provide temporal stability. The technique may operate in four different modes:

SMAA 1x: works only in the final image, so it cannot solve subpixels and temporal instability. It searches for aliased pixels by comparing the local luma contrast of the neighborhood with the PMLAA pre-defined patterns, and a new diagonal pattern. Only if the diagonal search fails the remaining patterns are verified. Sharpness of corners is improved by a wider search in the direction indicated by the detected edge, which identifies a corner and applies a reduced blur. To save processing time, only the top and left neighbors of q pixel are analyzed, the bottom and left ones are covered by other pixels.

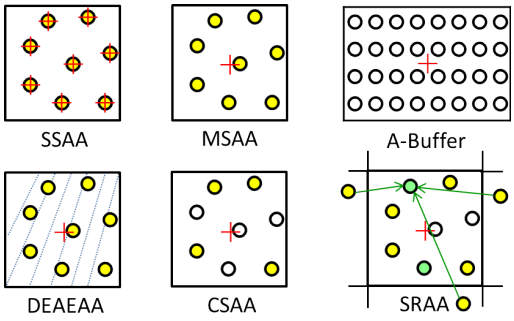

Fig. 9. FSAA sampling approaches comparison: SSAA with full shading of all 8 samples per **pixel**. MSAA with 8 color+coverage samples and central shading per **pixel**. DAEAA with 8 MSAA samples to estimate isolines passing through the **pixel**. CSAA with 4 color+coverage samples and 4 coverage-only samples per **pixel**. SRAA with 4 color+coverage samples, 2 geometry samples, and the color reconstruction from other samples. A-Buffer has a bitmask with 8x4 coverage samples per **fragment**.

SMAA S2x: operates over the SMAA 1x with addition of MSAA to solve subpixel aliasing. The pre-computed textures are modified to provide correct coverage for each subpixel position.

SMAA T2x: operates over the SMAA 1x with addition of temporal supersampling to solve temporal instability. The subsamples of the previous frame are projected into the current frame, weighted by their relative velocity with the current subsamples.

SMAA 4x: operates over the SMAA 1x with MSAA and temporal supersampling. This mode helps solving aliasing patterns detection, subpixel features and temporal instability.

7) Accumulation, Area-averaged and Anti-aliased buffer (A-Buffer): This technique, besides the correct OIT computation, was mainly developed to solve anti-aliasing. If an opaque fragment does not cover a pixel entirely, then a list of visible fragments is built for that pixel. During the post-processing phase, the fragments in the list are combined.

Each fragment coverage is encoded in a bitmask (see Fig. 9), and all visible fragments are stored in a per-pixel linked list. When the geometry pass is over, the algorithm traverses the per-pixel lists, sorts them in front-to-back order and weights the fragments by their coverage when there is no interception. For interpenetrating fragments, the z_{min} and z_{max} values are used to approximate the visibility of the coverages, and weight the fragment contributions.

A-buffer produces high-quality images at the expense of unbounded memory and extra processing time.

B. Image Post-Processing

This approach to handle aliasing is based in the processing of the final image. Differently from the FSAA approach, aliased pixels are detected and selected, so their processing does not

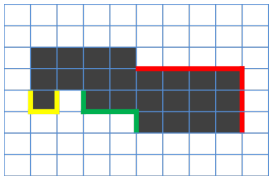

Fig. 10. Examples of L (red), Z (green) and U (yellow) shapes detected in an aliased image by the MLAA technique.

interfer with the entire image. The aliasing removal of those pixels consists in attenuate the neighborhood high frequencies.

One pixel is considered aliased by the analysis of its neighborhood in the color buffer, with or without the aid of extra information, such as depth and normal. This exploration identifies high frequencies and builds a mask to select the pixels which need processing. The last step is the filtering of the selected pixels, which are combined with their neighbors in order to reduce the high frequency by smoothing them. In this phase, the absence of the selection mask would cause the entire image to be excessively blurred. As the result is approximated by blurring aliased pixels in the final image, this approach cannot handle subpixel issues.

1) Morphological Anti-Aliasing (MLAA): [20] [16], is an image-based technique, which aims to minimize aliasing from edges and silhouettes. This technique was developed with the intent of removing aliasing from ray-tracing-generated images without considering more samples, which in ray-tracing are especially costly.

The working flow is simple: (i) find visible discontinuities between pixels by difference thresholds, (ii) identify aliasing patterns from these discontinuities and (iii) blend them with the neighborhood. The discontinuity can be determined by any metric, the first proposal used the sum of the 4 most significant bits of each color channel.

The image is scanned for discontinuities by comparing segments of different luminance between neighbor columns and lines, creating lists of vertical and orthogonal segments. These are classified as L, U or Z shapes, as shown in Figure 10. The U and Z shapes can be decomposed as two L shapes and processed separately.

From the L shapes, the longest edge is first selected, and it forms a triangle with the middle of the shortest edge. The triangle area is used to weight the blending with the neighboring pixels.

MLAA, when proposed, was able to significantly reduce the aliasing of ray-traced images very fast, because it only requires the color buffer to produce good results for nearly-vertical/horizontal edges, which are the most noticeacle kind of aliasing.

Its limitations are, the inability to handle subpixel features, blur of border pixels even when there is no aliasing, and it is not well suited to animation due to temporal instability (because each frame is processed individually).

2) Practical Morphological Anti-Aliasing (PMLAA): [16] is a modification of the original sequentially processed MLAA, which leverage the GPU features. The technique works in the same three steps, each one was improved with relation to the original MLAA. The edge detection can make use of more information, such as depth, ids, normal and combinations of them to improve aliasing detection. The coverage estimation counts with bilinear filter and pre-computed areas acceleration. And the final blending also makes use of the fast filtering offered by the GPU hardware.

The edge detection phase masks the pixels requiring anti-aliasing, avoiding unnecessary processing. After that, the edge reconstruction takes place to estimate the coverage area. For each pixel, the algorithm searches for the end of the edge it belongs to in the top and left borders. This is done by bilinear filtering the pre-processed image, which makes possible analyze more than one pixel at time. Once the end of the edge is found, the crossing edges patterns also are established by bilinear filtering. With a small offset, the filtering is able to recognize four different types of crossing edges.

With identified edge width and crossing edges patterns, the algorithm uses these information to access a texture with pre-computed area coverage patterns. These values are summed into an accumulation buffer and used to blend the pixel neighborhood. The final phase uses the accumulated areas to weight the blending of the 4 neighbors of each pixel by bilinear filtering in sRGB space.

When compared to MLAA, PMLAA presents great improvement in terms of processing time, due to the efficient parallelization in GPU, and in terms of image quality, by using more information to correct select pixels. However, it maintains the main limitations. It may cause excessive blur in sharp edges and presents temporal instability.

3) Directionally Localized Anti-Aliasing (DLAA): [16][21] was developed for the PS3 console to handle the most disturbing aliasing type, which are the nearly vertical and horizontal edges. It was prototyped in Photoshop® [22], using high-pass filters, blur filters, contrast modifiers, thresholds and masks. The main goal was, receiving only the final image, produce a better looking image as fast as possible.

The technique workflow is straightforward. First, a Sobel-like filter is applied to detect only vertical edges. A curved threshold function selects the desired edges by ignoring grayish values, so masking the vertical edges. A vertical blur is applied to the entire image, and the previously created mask is used to select the edges regions to be blended with the original image, producing anti-aliased vertical edges. All these filters are rotated by 90 degrees and the same process is repeated for horizontal edges. In short, the technique process can be resumed to:

1) Vertical blur filtering.
2) Vertical edge detection.
3) Threshold application over edge detection to mask vertical edges.
4) Mask use to blend vertical anti-aliased edges with the original image.
5) Repeat the process to anti-alias horizontal edges.

Different kernel sizes may be required, depending on the

width of the edge. In order to detect long edges, the rate of blur is increased, then the high-pass filter is applied, followed by a contrast adjustment. In this process only the long edges will survive, creating a long edges mask, which are blurred with a bigger kernel.

The results are comparable with MLAA, without the need of search for specific patterns, neither compute coverage estimations. As the filter was designed for nearly vertical and horizontal edges, as the MLAA, it does not performs well along diagonal aliased edges. Temporal instability and loss of subpixel features are also limitations of this technique.

4) Fast approXimate Anti-Aliasing (FXAA): As the name says, FXAA [16][23] does not aim to acquire correct anti-aliasing. The proposal is a very fast algorithm which reduces some aliasing artifacts, improving image quality. It works by determining the need of anti-aliasing by local contrast examination. The selected pixels are processed by a directional edge blur filter.

The algorithm receives as input a color buffer only. The first step of the algorithm is to determine which pixels actually need anti-aliasing. In order to do that, each pixel is tested with a 4-neighbors (neighboring edges), which compares the luminance of the neighbors to verify if the contrast is higher than a user defined threshold. The local contrast is determined by the difference between the maximum and minimum values of luminance among the current pixel and its four neighbors. If the contrast is low, the pixel is discarded from further processing.

For the pixels classified as needing anti-aliasing, a local luma gradient is computed. The directions perpendicular to the gradient are used to sample the neighborhood and blur the pixel. The user can define a scale factor, which controls how many neighbors are considered, varying from 2 to 8 samples. After performing the blur, the local contrast is tested again and, if it is too high, the default 2-samples blur is applied, ignoring the scale.

As the pixels are processed individually, this algorithm does not present good results for long nearly vertical and nearly horizontal edges, which are the most disturbing artifacts. A palliative solution is to use fractional super sampling (FSS) as input image, so, when FXAA downsamples to the target resolution, these artifacts are reduced. As the other IAA techniques, this one also presents temporal instability.

C. Geometric Anti-Aliasing

Most of the aliasing in computed generated images comes from geometric edges, so, this approach works with these edges to select and weight contributions of pixels in the color buffer.

During the geometry rasterization, the line equations of the edges are passed to the fragments. Analytically, the distances from the pixel center to the actual edge are encoded in the fragments. In the post-processing stage, the distances are used to identify aliased pixels and weight their contributions.

The post-processing stage resembles the IAA approach with distance information. Since this approach only work with geometric edges, it is not capable of handle aliasing from other sources, such as alpha texture and interpenetractions.

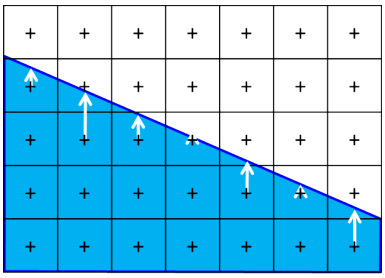

Fig. 11. Extra information is associated to edge pixels: the distance from its center to the geometric edge. This data is used to weight the blur in a post-processing phase. Image modified from [24].

1) Distance-to-Edge Anti-Aliasing (DEAA): [16][24] is a post-processing anti-aliasing technique. It encodes in each pixel the distance to the edges of the triangle it belongs to. After rendering, these distances are analyzed to verify if the pixel needs anti-aliasing.

During the rendering, in the vertex shader, each vertex of the triangle receives either the R, G or B color. The rasterization will generate the fragments with these colors interpolated and the distance to the RGB base approximates the fragment distance to the triangle edge. Four distances are encoded in a RGBA texture of each fragment: up, down, left and right directions, each one in an 8 bits channel.

In the post-processing stage, the smallest distance of each pixel is verified, if it is less than one pixel, the current pixel is marked as belonging to a border, as shown in Figure 11. Only border pixels are processed. If two neighbor pixels have competing distances, which means, both indicate primitive coverage in the neighbor area, the smallest one is chosen and blended into the neighbor, pondered by the coverage area indicated by the distance.

This solution does not solve sub-pixels problems, because no extra samples are taken. Other cases untreatable by this technique are interpenetrations (because the geometric edge is not the limit of the primitive in screen space), and cases where edges are not present, such as shadows and textures.

2) Geometric PostProcessing AA (GPAA): [16][25] gets the silhouettes information from the pipeline. A preprocessing stage evaluates only the silhouettes edges, computing their equations in screen space and passing this information to the pixel shader. As the DEAA, the blur is weighted by the distance of the pixel center to the edges.

This technique presents high-quality results, with accurate coverage even in nearly vertical and horizontal edges, and temporal stability. However, the edge extraction increases memory consumption and processing time, degrading fast with geometry augment.

3) Geometry Buffer Anti-Aliasing (GBAA): [16][25], differently from GPAA, stores the geometric information during the rendering, without the need for a preprocessing stage. The distances are calculated for each vertex in the vertex shader, and interpolated by the rastering process. In this sense, it is similar to DEAA, the final step after rendering checks the distances in each pixel, if smaller than half a pixel, the pixel

receives anti-aliasing, so the coverage of neighboring pixels is computed and they are blended.

The memory requirement is smaller than for GPAA and the overall quality is maintained. However, the entire processing is still expensive for high demands of FPS.

IV. Discussion

Current GPUs have special components to handle transparency, and others to anti-aliasing. Often, the transparency components are used to support anti-aliasing and vice-versa. Here, we will comment the usage of these resources, along with rendering issues related to the application of OIT and AA effects.

The hardware standard to support the transparency effect is the fourth color channel, called alpha channel. This channel is used to encode the opacity attribute, and it weights the contribution of the fragments to the final pixel color. The hardware also supports the blending equations proposed by Porter and Duff [1].

In order to provide anti-aliasing, the classical hardware support is for the MSAA. The components required are a multi-sampled texture, to store the samples information (depth, color, etc), and a multi-sampled rasterization algorithm. Recently, other techniques have gain support, such as CSAA, EQAA, FXAA, and others.

An intriguing problem nowadays is how to combine AA and/or OIT with deferred shading, because of the high memory consumption and technicalities involving the decoupling of samples from their originating surfaces.

Table I summarizes the main features discussed for each technique.

A. AA with the aid of transparency hardware

AA techniques can make use of the opacity channel to weight the fragment visibility by its coverage over the pixel area (ex.: [26]). The blending of such fragments become order-dependent, which means that artifacts may be generated if they are combined out-of depth-sorted order.

To obtain and store information for more than one sample per pixel is costly. The storage space can be saved by encoding the coverage of the fragments into the alpha channel. For example, if 1 of 4 samples is covered, the opacity of the fragment will be multiplied by 0.25; if it was opaque, it will become transparent with 0.25% of opacity. This can be compared to what is done by coverage AA techniques, where the fragment contribution is weighted by the amount of spatial samples it covers.

This approach inserts the transparency order-dependency into the AA problem. If the fragments are combined out-of-order, their visibility will be incorrect due to the blending equation.

Rendering of billboards often use this approach by drawing the opaque background first. For particles effects, which present alpha textures and similar colors, out-of-order algorithms may be used to approximate the result. The performance of this approach is quite superior to other anti-aliasing techniques, but the set of cases to which it is applicable is restricted.

B. OIT with the aid of anti-aliasing hardware

Since the graphic hardware is prepared to store more than one sample per pixel to handle AA, some OIT techniques use this memory to store transparent fragments. The storage capacity implemented for AA is adequate to compute transparency, in case the number of layers per pixels does not exceed the limit of slots for samples.

The Stencil Routed technique [27] is one usage proposal of the MSAA textures to handle OIT, with multiple geometry steps if necessary. This technique basically uses the high-performance of these textures to store and combine up to n fragments per geometry pass, combined with the stencil buffer for fragment counting.

Stochastic Transparency [6] also make use of the MSAA hardware, but with a different approach. It requires three geometry passes to estimate visibility and generate, for each fragment, a coverage probability. When the fragment arrives, this probability is used to fill the MSAA samples individually. Higher probabilities tend to fill more samples, which mean that the fragment is more visible. As the pixels are processed individually with the computed probabilities, the generated image contains large amounts of noise.

For low amounts of transparent layers, the MSAA hardware is a fast approach. The AA of the transparent scene could be performed with an IAA technique, without the benefit of temporal stability.

C. AA for transparent scenes

Apply anti-aliasing to a transparent scene is an interesting problem, mainly because of the treatment for AA samples and the order dependency. A GAA technique would remove the samples management, but not the ordering problem. And an IAA approach, applied only to the final image, brings reasonable results; however, it is unable to solve temporal instability.

Buffer-based OIT needs high amounts of memory, and to take more samples to perform AA for this approach may be impractical. For example, a 4xMSAA algorithm for opaque scene in a 640x480 screen needs 9.4MB (RGBA + depth), if storing five transparent layers per pixel with their samples, the memory requirement goes up to 46.87 MB. Now, think about a bigger screen and you will see the problem. Coverage-only techniques have being used with the recent OIT algorithms proposals; however it relies in sorting by the central z, not having the ability to solve interpenetration cases.

For depth peeling approaches, a NxSSAA would cost n times more processing at each geometry step, and the other multi-sample techniques can easily degenerate to SSAA brute-force. Again, the solution relies in applying IAA and losing temporal stability.

The GAA techniques was thought to blur surviving fragments in the color buffer, this means that it does not resolve thin primitives neither temporal aliasing. Its application for different layers of OIT was never properly explored.

Class	Technique	Samples Per Pixel	Shadings Per Pixel	Geometry Passes	Pattern Detection	Pre Process	On the fly	Solve Subpixel	Order Dependent
FSAA	SSAA	N	N	1			•	•	
	MSAA	N	1	1			•	•	
	CSAA	N+K	1	1			•	•	•
	DAEAA	N	1	1			•	•	
	SRAA	N	1	2		•		•	
	SMAA	N	N	1	•		•	•	
	A-buffer	N	1	1			•	•	•
IAA	MLAA	1	1	1	•				
	PMLAA	1	1	1	•				
	DLAA	1	1	1					
	FXAA	1	1	1	•				
GAA	DEAA	1	1	1			•		
	GPAA	1	1	2		•			
	GBAA	1	1	1			•		

TABLE I

ANTI-ALIASING COMPARISON TABLE: TECHNIQUES ARE CLUSTERED BY CLASS, SAMPLES PER PIXEL COLUMN INDICATES THE SAMPLING RATE PER PIXEL, SHADINGS PER PIXEL INDICATES HOW MANY TIMES THE COLOR IS EVALUATED PER PIXEL, GEOMETRY PASSES INDICATES HOW MANY TIMES THE GEOMETRY MUST BE PROCESSED, PATTERN DETECTION INDICATES THE PRESENCE OF A MODULE TO DETECT ALIASING PATTERNS IN THE FINAL IMAGE, PRE-PROCESS INDICATES THE NEED FOA A PRE-PROCESSING STEP, ON THE FLY INDICATES ANTI-ALIASING PROCESSING DURING THE RENDERING, SOLVE SUBPIXEL INDICATES IF THE TECHNIQUE IS ABLE TO SOLVE SUB PIXEL ALIASING, AS THIN PRIMITIVES AND TEMPORAL ALIASING, ORDER DEPENDENT INDICATES IF THE ORDER IN WHICH THE FRAGMENTS ARRIVE IMPACTS IN THE RESULT.

D. The deferred shading issue

Deferred shading (DS) is a rendering pipeline which supports expensive shading computations. A classic forward pipeline shades the fragments which pass the depth test and stores only their color, which is replaced when a new fragment passes the depth test. The DS strategy consists in storing the geometric information, into the so called G-Buffers, when a fragment passes the depth test. Only the final visible fragments are shaded at the end of the rendering process, saving expensive computations for fragments which are not visible.

The G-Buffers represent a big memory budget, because they store all the information needed to shade the fragment. To apply SSAA in the DS pipeline would be necessary to store all the information for all the samples, implying prohibitive memory consumption. The benefits of the MSAA are lost due to the decoupling of samples from their original surface, degenerating to SSAA brute-force.

IAA techniques, due to their detachment of the pipeline, are compatible with any rendering process, including deferred shading. However, the price to use such AA approach is given up the ability to handle subpixel features and temporal instability. GAA techniques may be used efficiently.

DS pipeline is incompatible with OIT because its main idea is to avoid processing occluded fragments, while OIT implies in partial occlusion, requesting the processing of more than one fragment per pixel in depth-sorted order. The simplest way to combine them is rendering OIT in a second stage, when the opaque depth buffer is set, with a forward rendering pipeline.

Combine order-independent transparency with high-quality anti-aliasing in a deferred shading pipeline is a challenge. The issues involved are: (i) high shading costs, which motivates the use of DS pipeline, (ii) depth-sorted order-dependency, intrinsic to the OIT problem, and (iii) high memory requirements, associated to all the three.

V. CONCLUSIONS

We presented two rendering effects which have in common the primarily need for multi-samples per pixel, either spatially or in depth. Both transparency and anti-aliasing problems were exposed, along with the approaches and main techniques developed, during the last three decades, to solve them. Each technique is most suited for the specific set of features it was developed to, being able to acquire real time performance.

Transparency relies in depth-sort the fragments, in order to correct accumulate their contributions to the pixel color. For the OIT effect be part of a real time application, the techniques must balance image quality with resources consumption, which often involves the storage and sorting of transparent fragments.

Anti-aliasing, in the turn, handles the spatial sampling problem over the projection plane. Generally, it involves the acquisition and management of more samples per pixel to acquire supixel treatment and temporal stability.

We also discussed the hardware features to support the two effects, how AA can make use of the OIT feature, and how OIT can also make use of the AA hardware. At last, we exposed the implications of using them combined, and of combine them with deferred shading pipelines.

This tutorial was written to provide the basic knowledge to guide the reader towards approaches to real time rendering of transparency and anti-aliasing effects. It highlights the issues related to both effects and the combination of them, aiming to provide the reader the required information to choose and combine the specific techniques best suited for her/his application.

VI. INSTRUCTORS BIOGRAPHY

- **Marilena Maule**
  Marilena Maule received the Bachelors degree in computer science from the Federal University of Rio Grande do Sul, Brazil. She is currently a second-year

PhD student at Federal University of Rio Grande do Sul. Her research interests include graphics, visualization, rendering and GPGPU computing.
Contact:
Universidade Federal do Rio Grande do Sul
Instituto de Informatica
Av. Bento Goncalves, 9500
Campus do Vale - Bloco IV - Prdio 43425
Porto Alegre RS 91501-970
e-mail: mmaule@inf.ufrgs.br

- **João L. D. Comba**
João L. D. Comba received the Bachelor's degree in computer science from the Federal University of Rio Grande do Sul, Brazil, a MS degree in computer science from the Federal University of Rio de Janeiro, Brazil and a PhD degree in computer science from Stanford University. He is an associate professor of computer science at the Federal University of Rio Grande do Sul, Brazil. His main research interests are in visualization, geometric algorithms, spatial data structures, high-performance computing, graphics hardware and games.
Contact: e-mail: comba@inf.ufrgs.br

- **Rafael Torchelsen**
Rafael P. Torchelsen has a Ph.D. from the Federal University of Rio Grande do Sul (UFRGS), Brazil, has a B.S in computer science from Catholic University of Pelotas and a M.S. in computer science from University of Vale do Rio dos Sinos (UNISINOS). He also worked in game engine development in several games. Currently he is a professor at the Universidade Federal da Fronteira Sul, UFFS. Research interests include computer graphics, geometric algorithms, geometry modelling, GPUs and game programming.
Contact: e-mail: rafael.torchelsen@uffs.edu.br

- **Rui Bastos** Rui Bastos is a member of the GPU architecture group at NVIDIA, where he has contributed to the design of GPUs since 1999. He received a Ph.D. (1999) and an M.S. (1997) in computer science from the University of North Carolina at Chapel Hill, and an M.S. in computer science (1992) and a B.S. in physics (1988) from the Federal University of Rio Grande do Sul (Brazil).
Contact: e-mail: rbastos@nvidia.com

REFERENCES

[1] T. Porter and T. Duff, "Compositing digital images," SIGGRAPH Comput. Graph., vol. 18, no. 3, pp. 253–259, 1984.

[2] M. Maule, J. L. Comba, R. P. Torchelsen, and R. Bastos, "A survey of raster-based transparency techniques," Computers & Graphics, vol. 35, no. 6, pp. 1023 – 1034, 2011.

[3] L. Carpenter, "The a -buffer, an antialiased hidden surface method," in SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques. New York, NY, USA: ACM, 1984, pp. 103–108.

[4] A. Mammen, "Transparency and antialiasing algorithms implemented with the virtual pixel maps technique," IEEE Comput. Graph. Appl., vol. 9, no. 4, pp. 43–55, 1989.

[5] L. Bavoil and K. Myers, "Order independent transparency with dual depth peeling," 2008. [Online]. Available: http://developer.download.nvidia.com/SDK/10/opengl/src/dual_depth_peeling/doc/DualDepthPeeling.pdf

[6] E. Enderton, E. Sintorn, P. Shirley, and D. Luebke, "Stochastic transparency," in I3D '10: Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games. New York, NY, USA: ACM, 2010, pp. 157–164. [Online]. Available: http://doi.acm.org/10.1145/1730804.1730830

[7] M. Salvi, J. Montgomery, and A. Lefohn, "Adaptive transparency," in Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics, ser. HPG '11. New York, NY, USA: ACM, 2011, pp. 119–126. [Online]. Available: http://doi.acm.org/10.1145/2018323.2018342

[8] C. Crassin, "Opengl 4.0+ abuffer v2.0: Linked lists of fragment pages," http://blog.icare3d.org/2010/07/opengl-40-abuffer-v20-linked-lists-of.html, 2010.

[9] J. C. Yang, J. Hensley, H. Grn, and N. Thibieroz, "Real-time concurrent linked list construction on the gpu," Computer Graphics Forum, vol. 29, no. 4, pp. 1297–1304, 2010.

[10] M. Maule, J. Comba, R. Torchelsen, and R. Bastos, "Memory-efficient order-independent transparency with dynamic fragment buffer." SIBGRAPI Brazilian Symposium on Computer Graphics and Image Processing, 2012.

[11] K. Beets, "Super-sampling anti-aliasing analyzed," Beyond 3D, 2000.

[12] nVidia, "Coverage sampled antialiasing," http://developer.download.nvidia.com/SDK/10/direct3d/Source/CSAATutorial/doc/CSAATutorial.pdf, 2007.

[13] AMD, "Eqaa modes for amd 6900 series graphics cards," http://developer.amd.com/sdks/radeon/assets/EQAA%20Modes%20for%20AMD%20HD%206900%20Series%20Cards.pdf, 2011.

[14] D. Woligroski, "Coverage sampling modes: Nvidias csaa and amds eqaa," http://www.tomshardware.com/reviews/anti-aliasing-nvidia-geforce-amd-radeon,2868-4.html, 2011.

[15] K. Iourcha, J. C. Yang, and A. Pomianowski, "A directionally adaptive edge anti-aliasing filter," in Proceedings of the Conference on High Performance Graphics, 2009. [Online]. Available: http://doi.acm.org/10.1145/1572769.1572789

[16] J. Jimenez, D. Gutierrez, J. Yang, A. Reshetov, P. Demoreuille, T. Berghoff, C. Perthuis, H. Yu, M. McGuire, T. Lottes, H. Malan, E. Persson, D. Andreev, and T. Sousa, "Filtering approaches for real-time anti-aliasing," in ACM SIGGRAPH Courses, 2011.

[17] M. G. Chajdas, M. McGuire, and D. Luebke, "Subpixel reconstruction antialiasing for deferred shading," in Symposium on Interactive 3D Graphics and Games, 2011. [Online]. Available: http://doi.acm.org/10.1145/1944745.1944748

[18] M. Deering, S. Winner, B. Schediwy, C. Duffy, and N. Hunt, "The triangle processor and normal vector shader: a vlsi system for high performance graphics," in Proceedings of the 15th annual conference on Computer graphics and interactive techniques, ser. SIGGRAPH '88. New York, NY, USA: ACM, 1988, pp. 21–30. [Online]. Available: http://doi.acm.org/10.1145/54852.378468

[19] J. Jimenez, J. I. Echevarria, T. Sousa, and D. Gutierrez, "Smaa: Enhanced morphological antialiasing," Computer Graphics Forum (Proc. EUROGRAPHICS 2012), 2012.

[20] A. Reshetov, "Morphological antialiasing," Intal Labs, 2009.

[21] D. Andreev, "Directionally localized anti-aliasing," ACM SIGGRAPH Courses Presentation, 2011.

[22] Adobe, "Photoshop," http://www.photoshop.com.

[23] T. Lottes, "Fxaa," http://developer.download.nvidia.com/assets/gamedev/files/sdk/11/FXAA_WhitePaper.pdf, 2009.

[24] H. Malan, "Distance-to-edge anti-aliasing," ACM SIGGRAPH Courses Presentation, 2011.

[25] E. Persson, "Geometry buffer antialiasing," ACM SIGGRAPH Courses Presentation, 2011.

[26] nVidia, "Antialiasing with transparency," Technical Report, 2004.

[27] K. Myers and L. Bavoil, "Stencil routed a-buffer," in SIGGRAPH '07: ACM SIGGRAPH 2007 sketches. New York, NY, USA: ACM, 2007, p. 21. [Online]. Available: http://doi.acm.org/10.1145/1278780.1278806