Super-sampling Anti-aliasing Analyzed

Kristof Beets Dave Barron
Beyond3D
whitepaper@Beyond3D.com

Abstract - *This paper examines two varieties of super-sample anti-aliasing: Rotated Grid Super-Sampling (RGSS) and Ordered Grid Super-Sampling (OGSS). RGSS employs a sub-sampling grid that is rotated around the standard horizontal and vertical offset axes used in OGSS by (typically) 20 to 30°. RGSS is seen to have one basic advantage over OGSS: More effective anti-aliasing near the horizontal and vertical axes, where the human eye can most easily detect screen aliasing (jaggies). This advantage also permits the use of fewer sub-samples to achieve approximately the same visual effect as OGSS.*
In addition, this paper examines the fill-rate, memory, and bandwidth usage of both anti-aliasing techniques. Super-sampling anti-aliasing is found to be a costly process that inevitably reduces graphics processing performance, typically by a substantial margin. However, anti-aliasing's positive impact on image quality is significant and is seen to be very important to an improved gaming experience and worth the performance cost.

What is Aliasing?

Computers have always strived to achieve a higher-level of quality in graphics, with the goal in mind of eventually being able to create an accurate representation of reality. Of course, to achieve reality itself is impossible, as reality is infinitely detailed. People deal with computer systems that have a finite, or set amount of memory, bandwidth and processing ability. Because of this, it is impossible to deal with infinite detail. The closer you look at something, the more you see and this remains true down to the sub-atomic level. So, computers (at least for the foreseeable future) must work around the problem of infinite detail by taking shortcuts. For instance, they can use sampling to approximate the character of extremely complex source data.

A sample is a measurement of a very specific point in time and/or a location in space. To understand this, consider sound waves. Sound is nothing more than a pressure wave: air compressing and decompressing. This physical event is infinite in detail and it moves through space, evolving with time. A CD is a digital medium of storing sound; it stores numbers equivalent to the amount of sound for specific points in time. This translation from a pressure wave to a number is done through a microphone and AD-converter. A microphone can translate the infinitely detailed pressure wave into an infinitely detailed electrical signal. The AD-converter then measures this electrical signal at specific points in time. Each such measurement is a sample. So an infinitely detailed event is translated into a discrete sample version that can be processed by a CPU, or stored on a digital medium like a CD. This translates to graphics in that a sample represents a specific moment as well as a specific area. A pixel represents each area and a frame represents each moment.

At our current level of consumer technology, it simply is not possible to render enough samples for anything close to an acceptable representation of reality. Because of this lack of samples, artifacts are introduced, artifacts known as "aliasing." Aliasing brings to the table a number of rendering problems that can seriously detract from the quality of an image. These problems are there every day in current 3D accelerators, taking away that immersiveness that computer games and 3D applications strive to deliver. Jagged, crawling edges and flickering objects are all symptoms of aliasing. Look at the edges of an object on a PC 3D accelerator and there you will find jagged edges. Now start moving away from the object and you'll see a "crawling" effect. These aliasing artifacts substantially reduce the overall quality of the rendered display.

To better understand aliasing, shown on the next page are two sample images. In Part A of the Figure 1, we see what would potentially be the edge of a rendered image, mountains if you will. Now we see that this surface is on a grid, with each section of the grid representing a pixel. Because we generally render on a per-polygon and per-pixel basis, we either color the pixel or we don't. It becomes like binary, either 1 (on) or 0 (off). There can be no partially filled pixels. On or off is decided based on the sampling of the center of the pixel zone (represented by a small circle in the image). So looking to Part

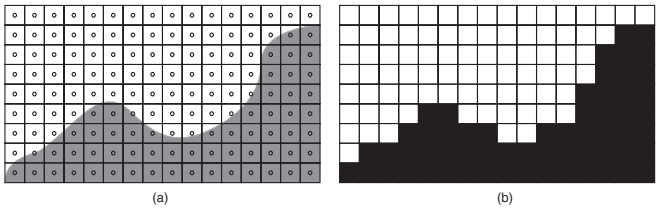

Figure 1: Illustration of jaggies caused by sampling
(a) Infinitely detailed curve
(b) Jagged sampled representation

B we see the result, a group of pixels that try and represent the surface we wanted to render. Unfortunately, this can result in something of a mess. These staircase-edges are often referred to as "jaggies".

Another example of aliasing in computer graphics is polygon "popping" (also referred to as pixel popping). This problem presents itself when dealing with thin polygons less than a pixel wide or tall. Sometimes, a thin polygon can be positioned in a spot where it cannot be sampled at all. From there, either all or part of the polygon disappears. Figure 2 (top right) illustrates various polygon popping effects. On the left we see the desired result, and on the right the rendered and sampled result, the latter suffering from polygon popping artifacts. Notice, for example, the green bar in (a). This bar covers the pixel centers in (a) and is thus visible, but now imagine a downward animation where the green bar ends up in position (b). In position (b) no pixel centers are covered by the green bar so nothing actually is rendered. Notice how the green bar went from full visibility to being completely invisible, all because of slight downward animation. If the animation continued downward, the bar would again cover pixel centers and become fully visible. Thus, as the bar moves down, the rendering would "flash" off and on - this is a perfect example of polygon popping aliasing. Furthermore, notice how the yellow shapes change between frame (a) and (b) of the animation. The yellow shapes are rendered differently from (a) to (b) due to the lack of sufficient rendering samples. The red, elongated triangles are exactly the same size but transposed, yet in both frames (a) and (b) they are shown covering a different amount of pixels. The blue square in frame (a) illustrates how a single equal polygon can end up looking very different when rendered just by changing its position or orientation. [1] All of these artifacts, characterized by different rendering of the same object as a function of different location or orientation, are symptoms of polygon popping aliasing.

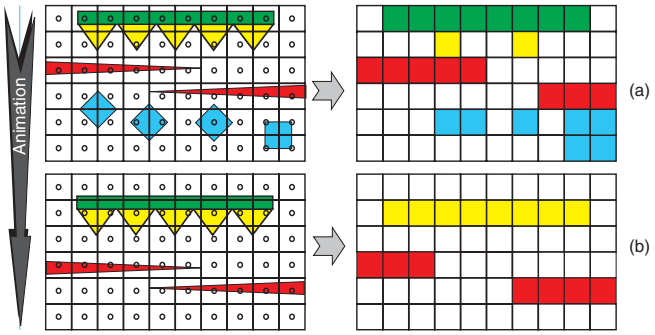

Figure 2: Illustration of Polygon Popping
(a) Finely detailed representation on the left, sampled version with missing polygons and details on the right.
(b) Animation frame after (a) different polygons are visible creating a flickering effect.

To reduce the artifacts associated with aliasing, both jagged triangle edges and polygon popping, we utilize anti-aliasing algorithms. Anti-aliasing is basically the process of removing the unwanted artifacts. The problem with anti-aliasing algorithms and techniques has long been the required high bandwidth and fill-rates. This has caused AA to long be stuck in the CAD and high-end computer imagery markets. However, consumer level products are now reaching a point where real-time anti-aliasing is possible. In the next section we'll describe the working of several anti-aliasing methods.

Practical Implementations of Anti-aliasing

Although several hardware and software implementations of anti-aliasing have been developed, they have not been successful in the mainstream consumer market to date. High CPU processing overhead, memory bandwidth constraints, and memory costs have made implementing anti-aliasing capability impractical for the consumer market. Even the less demanding "edge" anti-aliasing (which attempts to address the jagged edges artifacts, but not improving the polygon popping artifacts) has not been successful due to its relatively poor quality and negative performance impact. Anti-aliasing until very recently has remained exclusive to the high-end CAD-CAM and off-line rendering markets.

With the recent generation of CPUs and graphics processors, however, consumer systems finally have the processing power, memory bandwidth, and memory capacity to make anti-aliasing practical. This article explains the basic types of super-sample anti-aliasing and examines two specific implementations.

Super-Sampling Techniques

As explained earlier, aliasing is a result of sampling, or more specifically, the lack of a sufficient sampling rate. Super-sampling, as the name suggests, solves the aliasing problem by taking more samples than would normally be the case. By taking more samples, we are able to more accurately capture the visual quality of the infinitely detailed natural world. So the first essential element of super-sampling is that extra samples are used to increase the density of image information. You can see this as taking sub-samples at the pixel level. Thus, instead of one single central sample per-pixel, super-sampling techniques utilize several samples per pixel. It is the location of these sub-samples within a pixel that defines two different types of super-sampling anti-aliasing.

Ordered Grid Super-Sampling is the first and most common type of super-sampling anti-aliasing. The name "Ordered Grid" is descriptive of the sub-sample positions within a given pixel. The extra samples are positioned in an ordered grid shape. The sub-samples are aligned horizontally and vertically, creating a matrix of points. These sub-samples are thus located inside the original pixel in a regular pattern as shown in Figure 3a.

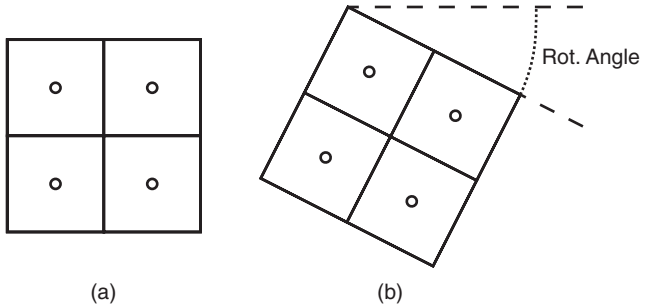

Figure 3: Illustration of different sampling grids
(a) Ordered Grid as used by the OGSS method.
(b) Rotated Grid as used by RGSS method.

The second type of super-sampling anti-aliasing is known as Jittered Grid Super-Sampling (JGSS). JGSS is similar to Ordered Grid Super-Sampling in that extra samples are stored per pixel, but the difference between the two is the position of the sub-samples. With OGSS, the sub-sample grid is parallel and aligned to the horizontal and vertical axis. However with JGSS, the sub-sample grid is "jittered," or shifted, off of the axis. An example of a sub-sample pattern used with JGSS is shown in Figure 3b. There are a variety of different ways of "jittering" the sub-sample positions, and we will now investigate the two most common implementations.

The first implementation, Fully Random Jittered Super-Sampling, is better known as Stochastic Sampling. Essentially this means that within a pixel, the sub-samples are positioned in random locations. The key here is that the sub-sample pattern is random for every pixel on screen. The basic idea behind this technique is that the randomized locations of the sub-samples is actually seen as "white noise" by the human eye. This technique hides the aliasing effect in noise, which is based on the known fact that the human eye is less sensitive to random noise than regular patterns. Our eyes are very good at recognizing patterns, but the introduction of random noise makes the recognition of patterns - in this case aliasing artifacts- significantly more difficult. While the quality of this technique is excellent, it is very expensive to implement. Approximately 16 (or more) randomly distributed sub-samples per-pixel are needed to attain the necessary level of white noise. Lower sample counts also work, but these are more susceptible to artifacts. Fully random positions also tend to be difficult to generate. Improvements in this technique exist to avoid accidental patterns, or samples being taken too

closely to each other. One example is the use of a Poisson Distribution of sample points [5], but these techniques are extremely complex and not yet available in consumer level hardware.

A second implementation of sample positions of JGSS is a simplified form of the first. Instead of using completely random patterns, a predefined pattern that approximates the effect of random sampling is re-used for every pixel of the screen. So we again get a grid (as in a repeating pattern), but the repeated pattern is more random and less uniform. Again, various random patterns exist. A pattern suggested by the OpenGL documentation is a "shear" transformed grid [2]. The ordered grid is thus shifted sideways. This shift removes some of the regularities that exist in the OGSS pattern. However, this pattern removes only the vertical regularity. The horizontal regularity is maintained, so the column structure of OGSS is destroyed but the row-like structure remains. Also, quite often used is a rotated ordered grid. We'll call this technique Rotated Grid Super-Sampling, or RGSS for short. With RGSS, the sub-sample locations are the same as those used in OGSS, but they are rotated by a certain angle. The sub-sample positions for an RGSS implementation are illustrated in Figure 3b on the previous page. Note that both the horizontal and vertical regularities are destroyed, so there are no definite columns or rows.

With the theory and definitions of OGSS and RGSS behind us, we will now focus on practical implementations using today's hardware for both techniques. We examine one hardware implementation for each super-sampling technique, but keep in mind that there are numerous ways for implementing the super-sampling anti-aliasing techniques described.

An OGSS Implementation

Ordered Grid Super-sampling (OGSS) is a technique that can be implemented on almost all 3D accelerators, given that they support rendering to an off-screen buffer. An off-screen buffer has room to store the frame's pixel colors, as well as the Z and Stencil values. However, it differs from a conventional front- or back-buffer in that it is never displayed directly on screen.

As was explained before, the OGSS method uses a regularly-patterned ordered grid of sub-samples for each pixel. Below we detail a step-by-step description of how OGSS anti-aliasing can be performed on today's 3D hardware.

1. The game engine creates the 3D environment using a 3D API like Direct3D or OpenGL. Both of these APIs use triangles as their basic building block to create 3D objects. Each triangle has coordinates in 3D space. These coordinates are transmitted, transformed and lit , through the API via the 3D card's specific driver. We'll assume in this simple example that our screen has a resolution of 10-by-10 pixels, and that we have a triangle with vertices at positions (5,5), (10,10) and (10,0) - forming a triangle on the right hand side of the screen.

2. The coordinates supplied by the API target a specific screen resolution. When the vertices are transformed and lit, they are provided screen-space coordinates (unlike the world-coordinates used by the 3D application). These coordinates are thus linked to the final screen resolution. To reach our anti-aliasing goal, we need to up-sample these coordinates by at least a factor of two in both the horizontal and vertical direction to create a sufficient number of sub-samples for effective anti-aliasing. This up-sampling is a simple multiplication by two of all screen coordinates. Up-sampling by a factor of two increases the screen resolution of our example to 20-by-20 (10 multiplied by 2). Our vertex positions will be up-sampled to (10,10), (20,20) and (20,0). Notice that the vertices and thus the triangle remain at the same relative position in screen-space.

3. The result of the previous operation is that all geometry is zoomed by a factor of two in both the horizontal and vertical directions. Simply put, everything is twice as big. We thus have four times the number of pixels drawn as compared to no up-sampling being performed. Our original screen of 10-by-10 pixels is now 20-by-20 pixels.

4. We render all the up-sampled geometry of this frame as we normally would, but to an off-screen (invisible) buffer. The reason for using an off-screen buffer is that our goal is to have a 10-by-10 anti-aliased image, not a 20-by-20 up-sampled one. Note that our example assumes only one triangle. A real world application, of course, has many more.

5. When the whole scene of this frame is rendered, we have a high-resolution picture of the 3D world. We now need to down-sample this high-resolution picture into an anti-aliased lower resolution version. We thus need to go from 20-by-20 "super-sampled" (double resolution) image to a 10-by-10 anti-aliased image. This down-sampling is achieved by mixing pixel colors together in groups of two-by-two. Essentially, we take the color values of four neighboring pixels (square shaped), add them together and then divide by a factor of four. This means that the resulting color is an equal mix of the colors of the four high-resolution pixels. These four pixels in the high-resolution image are really the sub-samples of the anti-aliased pixels. Combined together, these sub-samples form a final anti-aliased pixel for the rendered image. By sampling at a higher resolution and then filtering down using an averaging filter, the high frequency components are smoothed out, which reduces the aliasing considerably. For a simple example, assume our scene contains dark bars (0%) on a bright background (100%), arranged in the vertical direction (striped effect) with a height of one sub-sample. The sub-sample would contain two dark and two bright sub-samples. The filtered down result of this is a half bright, half dark pixel. So the high frequency effect (0% and 100% alternating) causing aliasing is reduced to a continuous 50% blend.

6. The end result is of this process is again a 10-by-10 image, but anti-aliased via an OGSS super-sampling technique. We will discuss another important aspect of this process - the quality of anti-aliasing - later in the article.

A schematic overview of this OGSS technique can be seen in Figure 4 on the right.

This practical implementation achieves Ordered Grid Super-Sampling by up-sampling the scene by a factor of 2, both horizontally and vertically. By increasing the horizontal and vertical resolution during the up-sampling, extra sample positions are introduced in an ordered grid shape. These sub-samples are located inside the original pixel, as illustrated by Figure 3a.

In our example, we used an up-sampling rate of two in both the horizontal and the vertical direction. Nothing prevents an implementation where other up-sampling ratios are used, such as four or even more. Usually this factor is used to identify the type of OGSS. Thus,

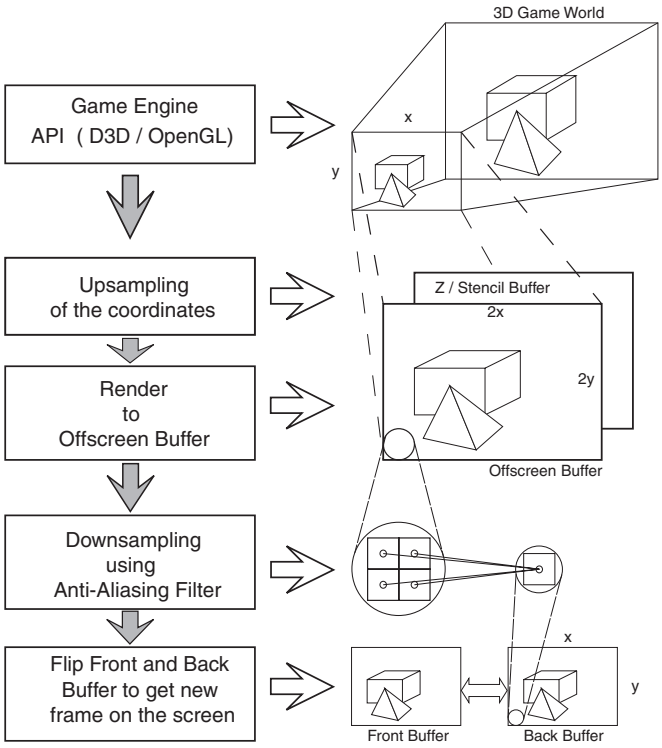

Figure 4: Overview of the Ordered Grid Super-sampling Method (OGSS).

the terms 2X OGSS and 4X OGSS are often used to describe the number of sub-samples used in an OGSS implementation. Sometimes the number of sub-samples is identified by the word "tap", thus leading to the names 4-tap or 16-tap OGSS. So with a 16-tap OGSS, we have 4 sub-samples in the vertical and 4 in the horizontal. A point worth mentioning is that the names Full-Scene Anti-aliasing and Full-Screen Anti-Aliasing are often used out of place. Generally, this term is used to represent OGSS, but what it truly means is that an entire image undergoes an anti-aliasing process (whether it be OGSS or RGSS or some other technique for anti-aliasing). However, it is important that we note what type of anti-aliasing is implemented in hardware because it has an impact on quality, as will shortly be investigated.

Finally, it should be noted that this implementation of OGSS using existing 3D hardware suffers from potential incompatibility with applications that use Linear Frame Buffer access. Linear Frame Buffer access is a technique where an application writes values directly into the buffers (Frame and/or Z/Stencil). The problem with this is that internally the render target is up-sampled to a higher resolution. The application is not aware of this and, as a result, the Linear Frame Buffer access goes wrong. A simple example will illustrate the point. Assume a game wants to use Linear Frame Buffering

to overlay the screen with a cockpit image. This overlay image has the resolution of the final image, not of the up-sampled, internal, higher-resolution off-screen buffer. As a result, an overlay with the size of the final image is written to the up-sampled buffer (so 1/4th the correct size), which of course results in an incorrect result. The overlay would end up occupying only ¼th of the proper dimensions in the final anti-aliased screen image. Applications that use Linear Frame Buffer access, therefore, must be handled with special care. The problem must be managed with special tricks at the driver level, as solutions are rarely supported in hardware. Needless to say, this has a nasty performance impact but without this special handling, vulnerable applications will show an incorrect end result.

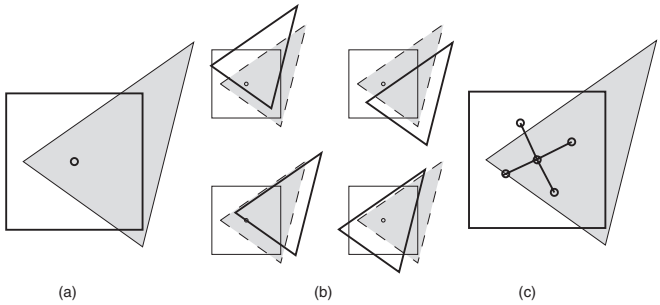

Figure 5: Illustration linking jittered geometry to different sample positions.
(a) The normal non-AAed grid with sample point.
(b) 4 Jittered versions of the scene geometry. Note that the triangle with the dashed line is the original triangle while the triangle with the solid edge is the jittered position. Sample point remains at the same spot inside the pixel.
(c) Equivalent Sub-sample positions. Compare the resulting sample points shown in (b) with these equivalent Sub-sample positions indicated by small circles.

A RGSS Implementation

Rotated Grid Super-sampling (RGSS) can be implemented using an Accumulation Buffer technique and also by the 3dfx T-Buffer technology. We will detail step-by-step the implementation of RGSS utilizing the VSA-100 T-Buffer capability found in 3dfx's recent Voodoo5 product offering:

1. The game engine creates the 3D environment using a 3D API such as Direct3D or OpenGL. Both these APIs use triangles as their basic building block to create 3D objects. Each triangle has coordinates in 3D space. These coordinates are transmitted, transformed and lit. If hardware T&L is supported, of course, the data is sent directly to the video card's T&L unit for transformation and lighting.

2. The 3dfx VSA-100 T-Buffer implementation uses a multi-chip solution where each chip calculates 2 sub-samples (it is safe to assume that in the future chips will allow for more sub-samples per chip). We thus need at least 2 VSA-100 chips to implement 4 sub-sample anti-aliasing. We'll assume a 2-chip configuration, such as the Voodoo5 family, in our explanation. As said before, the sub-samples are jittered or, more specifically, rotated. These jittered sample positions are obtained by shifting the geometry's vertices. So for each sub-sample, the vertices receive a precise sub-pixel level perturbation that matches the targeted sub-sample positions. Figure 5, right top of this page, illustrates this. The sample position and resolution stays equal, but by moving the geometry at the sub-pixel level we get different equivalent sub-samples. These geometry shifts are handled in hardware in the VSA-100 chip, so there is no software overhead required for RGSS anti-aliasing.

3. Now all the shifted geometry is rendered. Each shifted version is sent to its own T-Buffer. Each T-Buffer has the same resolution as the final anti-aliased image. The number of buffers is equal to the number of sub-samples taken. Each VSA-100 chip manages 2 sub-samples and thus writes to two T-Buffers. The writing is done to the invisible "back" T-Buffer, which is similar to the front- and back-buffers normally found on 3D accelerators. The front buffer is written to the monitor while rendering is done in the invisible back buffer. This avoids artifacts like tearing.

4. Once all the geometry for this frame is jittered and rendered to the T-Buffers, we end up with each T-Buffer containing the pixel-colors for each jittered scene. Each buffer contains a sub-sample of the final image, as illustrated in Figure 5. We now flip back and front T-Buffers.

5. The front T-Buffers now contains the sub-samples of the scene we just rendered. The sub-samples now need to be combined to form the final anti-aliased image. This combining is done just before the RAMDAC by special video circuitry that mixes the various buffers together at the pixel level. The RAMDAC is a special component of a 2D/3D chip that translates the contents

of the buffers into a signal that can be displayed by your monitor. Most monitors take analog signals as input, which explains the DAC part of the name: Digital to Analogue Converter. The RAM refers to the fact that the AD conversion is done using a table contained in RAM (this has to do with Gamma Correction). The main advantage of this approach is that no down-sampled version of the image has to be stored and the color depth at the output level is higher than the color depth of an individual buffer. The sub-sample T-Buffers can contain, for example, 16-bit color, but the combining operation (mixing of the colors) is done at a higher accuracy by the video circuitry which leads to a final anti-aliased image with a color depth higher than the color depth of the individual buffers. This principle is similar to that of the post-filter technology found in the Voodoo2 and 3 designs [3].

A schematic overview of this technique can be seen in Figure 6, below.

This same technique can also be implemented in hardware that supports an Accumulation Buffer [2] [4]. However, the traditional Accumulation Buffer technique has some disadvantages in implementing RGSS. The jittering has to be done using software and the geometry thus has to be sent several times to the hardware. The T-Buffer capability of the VSA-100 does this jittering at the hardware level, internally saving valuable bandwidth (the geometry data only needs to be sent once to the VSA-100, as the chip itself automatically jitters the geometry and renders into the T-Buffers). Traditional hardware T&L accelerators can calculate and apply the shift in hardware, but the geometry still has to be sent to the rendering core several times. Another disadvantage of the Accumulation Buffer lies in the recombining of the samples. T-Buffer does this just before the RAMDAC level while traditional systems require a costly copy and combine operation that merges the Accumulation Buffer contents with the frame-buffer contents after every sub-sample is calculated. More details about the Accumulation Buffer technique can be found in [2] and [7].

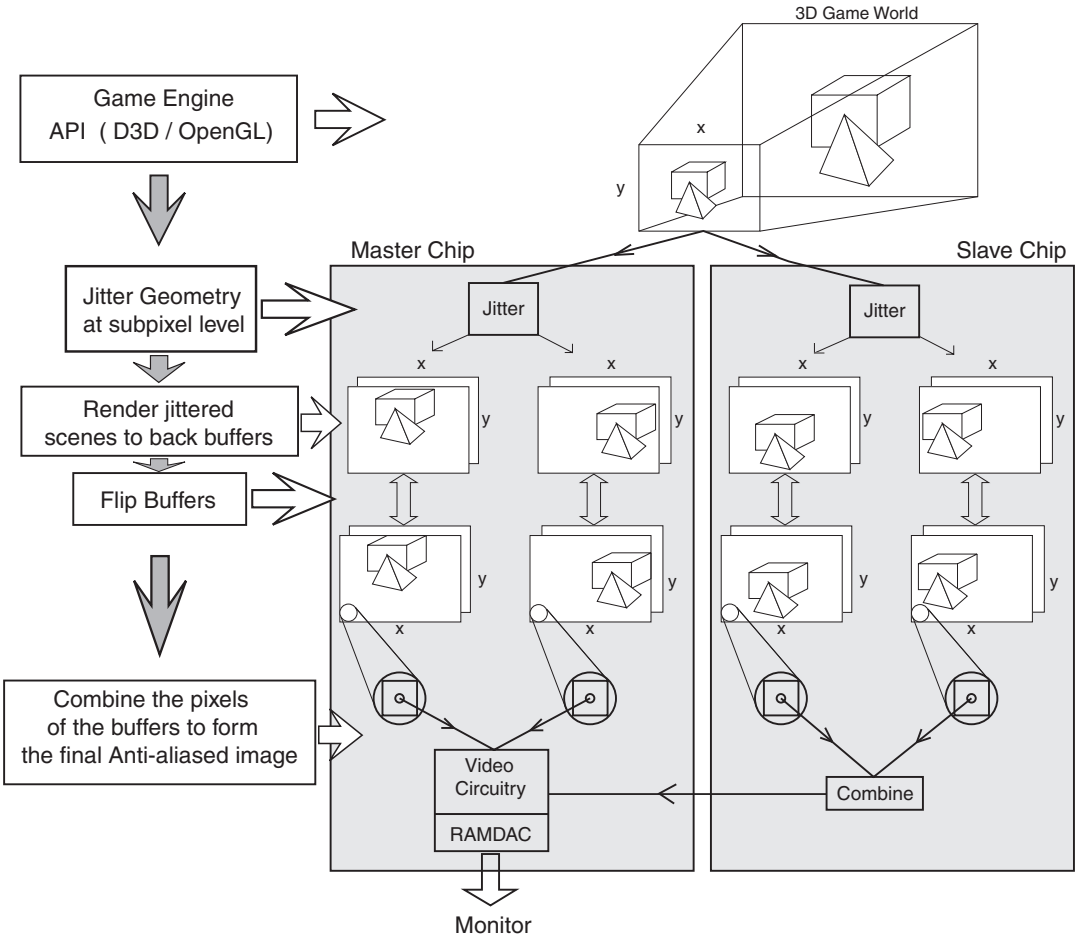

Figure 6: Overview of the Rotated Grid Super-sampling Method.

Note that the Linear Frame Buffer issue raised in the above section describing the OGSS implementation is not a problem for the VSA-100. This is because all of the T-Buffers have the same resolution as the final image. Instead of writing to just one buffer, the VSA-100 hardware writes the data to all T-Buffers automatically. Furthermore, for Linear Frame Buffer reads, the VSA-100 architecture merges the sub-samples together to form the anti-aliased pixel result before the data is returned to the CPU. This allows screen captures done by the host CPU to look identical to what the user sees on his monitor. These techniques allow the VSA-100 to implement RGSS anti-aliasing in a manner completely compatible with all 3D APIs.

Also note that OGSS can also be implemented on the VSA-100 architecture as the sub-sample position offsets are actually completely programmable by software. Implementing OGSS on the VSA-100 would simply entail using different sub-sample positions (in the case of OGSS, a regular ordered grid).

Summary

We've thus discussed the different implementations of super-sampling, focusing on Ordered Grid Super-Sampling and Rotated Grid Super-Sampling. The primary difference identified between the two methods is the location of the sub-samples within the anti-aliased pixel. We also discussed the implementations of these methods using PC 3D accelerators. The next part of this white paper will discuss the difference in image quality between the two super-sampling two methods, as well as several other key points.

Theoretical Image Quality

In the previous section we introduced two practical ways to do anti-aliasing. The first was Ordered Grid Super-Sampling (OGSS) and the second was Rotated Grid Super-Sampling (RGSS). We found that the big difference between these methods is the location of the sub-sample positions. Figure 3 on page 3 showed the different sub-sample patterns of the two techniques.

We will concentrate on comparing 4 sub-sample OGSS with 4 sub-sample RGSS. In particular, we will concentrate on what happens at the edges of polygons. The aliasing inside polygons is mainly solved by texture filtering, but that goes beyond the scope of this paper. Super-sample anti-aliasing methods help out by providing a higher sample resolution, but this is not enough to solve severe texture mapping aliasing. Both methods use 4 sub-samples and at an edge, this results in five possible cases depending on the number of sub-samples that fall inside and outside the polygon. For simplicity, we'll assume a uniformly black colored polygon on a white background. The various possible situations per-pixel are as follows:

Case 1: All sub-samples fall outside the polygon. This means the outcome is 0%. Zero of the 4 sub-samples are inside the polygon and the resulting color of this case is pure white.

Case 2: A single sub-sample falls inside the polygon. We thus have one black and three white sub-samples. The mixed down result of this is a 25% gray colored pixel. This is the 25% case.

Case 3: Two sub-samples fall inside the polygon. We thus have two black and two white sub-samples. The mixed down result of this is a 50% gray colored pixel, and this is called the 50% case.

Case 4: Three sub-samples fall inside the polygon, so we have three black sub-samples and one white sub-sample. The mixed down result is thus a 75% gray colored pixel. This case is the 75% case.

Case 5: All samples fall inside the polygon. The mixed-down result is thus a completely black colored pixel, and this is known as the 100% case.

In summary, 4 sub-sample anti-aliasing can have one of five different edge effects: 0%, 25%, 50%, 75% and 100%, depending on the number of sub-samples that are inside the polygon. These different values have varying impact on the contrast between the pixel(s) inside the polygon and the pixel(s) outside the polygon. Reduced contrast results in less obvious jagged edges.

Knowing that theoretically there are five different outcomes per-pixel and knowing that we have two techniques with different sub-sample positions, it is reasonable to conclude that edges will be anti-aliased differently when using the different anti-aliasing algorithms. Here is where the concept of Critical Edge Angles comes into play. A Critical Edge Angle is a special case where the edge passes though or nearly through 2 sub-samples at the same time. The easiest way to understand this is as follows. Take the sub-sample positions of the OGSS technique (see figure 3a) and imagine a horizontal edge moving though a single pixel with sub-samples from below to above the pixel. Notice that the sub-samples are aligned in the horizontal direction. This means that the bottom two samples will be passed by the edge at the same time. The same is true for the two samples at the top, which are also aligned in the horizontal direction. Essentially, this means that the moving horizontal edge moves from the 0% case, to the 50% case and then to the 100% case. Notice that the 25% and 75% cases are lost due to the fact that 2 sub-samples are passed by the edge at the same time twice! Losing 2 shade levels from the 5 is an example of a "Bad Angle Case" for OGSS.

When we look at the same sub-sample pattern we notice that exactly the same effect pops up when the edge is at a 90° (or vertical) angle. So essentially the OGSS algorithm has 2 Bad Angle Cases: horizontal edges (0°), and vertical edges (90°).

The same effect, but to a lesser degree, also occurs with 45° and 135° angle edges. When you imagine an edge moving though the sub-sample pattern at 45° from left to right, you'll notice one sub-sample enters the polygon and then the next 2 sub-samples, which are aligned in the diagonal direction, pass the edge at the same time.

The final sub-sample is passed later by the edge. Again we lose a sub-sample value, the 50% case. Since we only lose one shade level, this is a "Mediocre Angle Case." There are no other special cases.

The RGSS method has its sub-sample points aligned in special directions, too (see Figure 3b). Essentially, the sub-sample positions are equal to those of the OGSS method, but rotated around an angle. This means that the RGSS method also has Critical Edge Angles. Just as with OGSS approach, there are two Bad Angle Cases and two Mediocre Angle Cases. Essentially, the bad cases are the same as with the OGSS method, but perturbed by the Grid Rotation Angle. So if this angle is 20°, for instance, then the Bad Angle Cases are 0°+20°=20° and 20°+90°=110° and the Mediocre Angle Cases are 20°+45°=65° and 20°+135°=155°.

Based on this information you might get the incorrect impression that both techniques are equally good and bad. After all, the not-so-ideal anti-aliasing cases (where you lose shade levels) are just shifted to different edge angles. We'll soon discover that some angles require better anti-aliasing than others. We will begin by looking at theoretical examples of all the essential Critical Edge Angles.

Theoretical images analyzed

The first example edge is close to a Critical Edge Angle of the OGSS method. More specifically, we'll look at a Near Horizontal Edge, thus close to the 0° Bad Angle Case. Figure 7a and 7b (on the next page) show the same Near Horizontal Edge overlaid on the OGSS and RGSS sub-sample patterns. Beneath each grid example is the resulting anti-aliased edge created by the sub-sampling and blending processes.

The first thing to notice is that the OGSS method clearly has encountered a bad case. Only 0%, 50% and 100% shades are available to smooth the edge and decrease the contrast. On the other hand, the RGSS method has access to all shades to smooth the edge. Better still, they are nicely and evenly spaced. The rotation angle of the RGSS method in all these examples is 27°. Note that specific implementations such as 3dfx's T-Buffer might use a different rotation angle. The angle and possibly even the sub-sample positions might also be programmable. This example teaches us that not just the exact Critical Edge Angles are problematic. The edge angles near them can experience the loss of shade levels as well. Of course, it would be helpful to actually define what "near" is. Unfortunately, that is much more diffi-

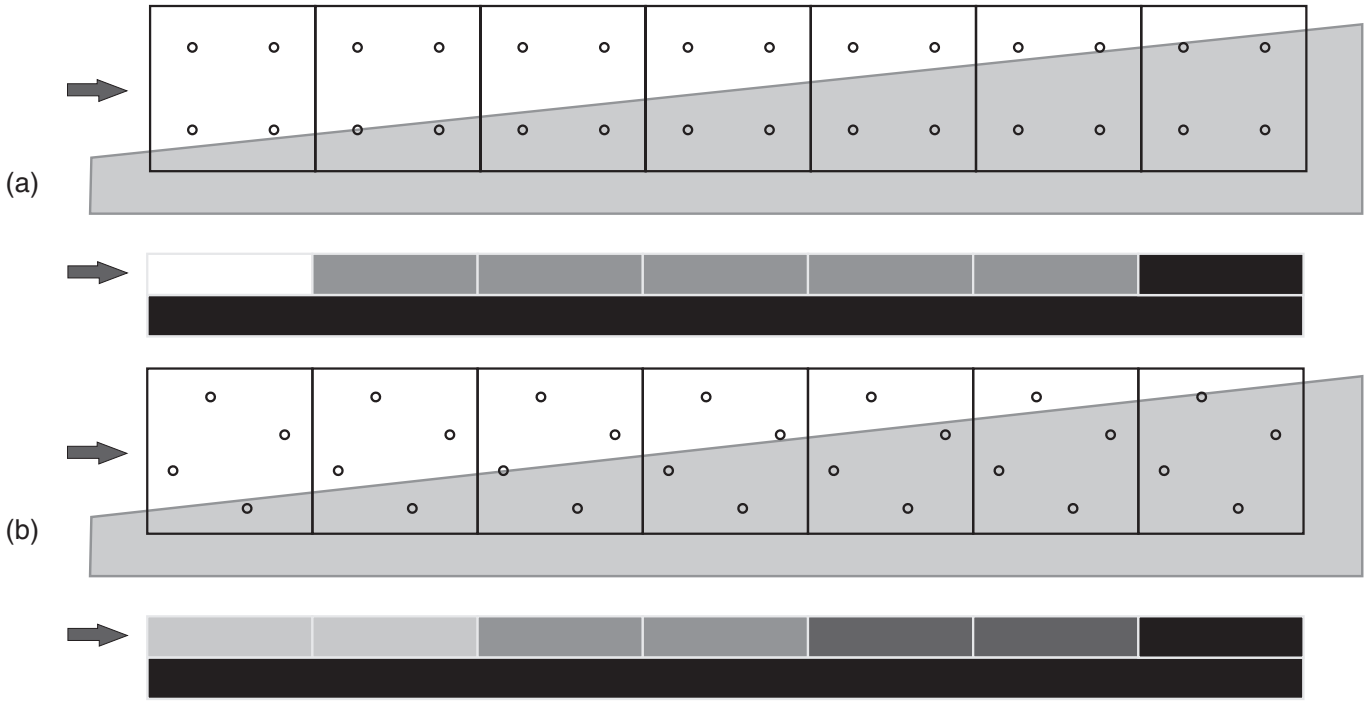

Figure 7: Near Horizontal Edge Case Comparison
(a) OGSS (b) RGSS

cult than it seems at first glance. However, we can identify the factor that determines when an angle is near a Bad Edge Case. The main influencing factor is the screen resolution. The higher it is, the closer together the sample and sub-sample points are. When they are closer together, the risk of jumping directly over more than one sub-sample becomes smaller. Essentially, the "near-factor" will change depending on the screen resolution used. The basic rule here is this: The higher the resolution, the better the anti-aliasing results. In this specific example it is very obvious that the RGSS method is superior to the OGSS method. So for now we can conclude that for near-horizontal and near-vertical edges (just rotate your page 90° and look at the same examples), the RGSS method is superior.

It really isn't surprising in the example above that the RGSS method is superior. After all, we selected an angle that was near critical for the OGSS method while it was nowhere near a critical angle for the RGSS method. Now, let's examine a Bad Angle Case for the RGSS method. More specifically, lets look at the case of an edge with the same angle as the rotation angle. This is similar to the horizontal edge case using the OGSS method. In Figures 8 and 9 (next page) we have a polygon that slowly moves along a line perpendicular to the bad angle, so the edge will pass though all possible shading levels.

When we compare the Bad Angle Cases of these two methods, we notice that both suffer from the same thing: lost shading levels. But when looking more closely, we notice that in the Near Horizontal Case the end result of the OGSS method comes nowhere near the end result of the RGSS method. Moving the edge up and down leaves the situation just as bad as it was. However, when we look at the Bad Case Angle for RGSS, we notice that some of the final results of both techniques look very similar. Indeed, they are close to identical. Notice that result (b) from the OGSS method is virtually equal to (d) from the RGSS method. The same is true for (a) and (b). They are not equal, they just look much more similar than the near horizontal example. The reason for this reduced difference is that under the near horizontal edge the staircase effect of the edge is very wide. The steps are elongated and with only one shade level it is nearly impossible to approach that smoothness. However, in the critical angle for the RGSS case, the staircase is very narrow and jaggy. As a result, there isn't much room to place different shade values along the edge. Actually, in a static case you only notice one major shade-level being used. Because of this, the RGSS method can come close to mimicking the results of the OGSS method. A cautious conclusion is suggested by these examples. It would appear that the near horizontal edge benefits more from multiple shade levels than does the edge paralleling the RGSS method's rotation angle. Although each edge represents the worst case for one of the anti-

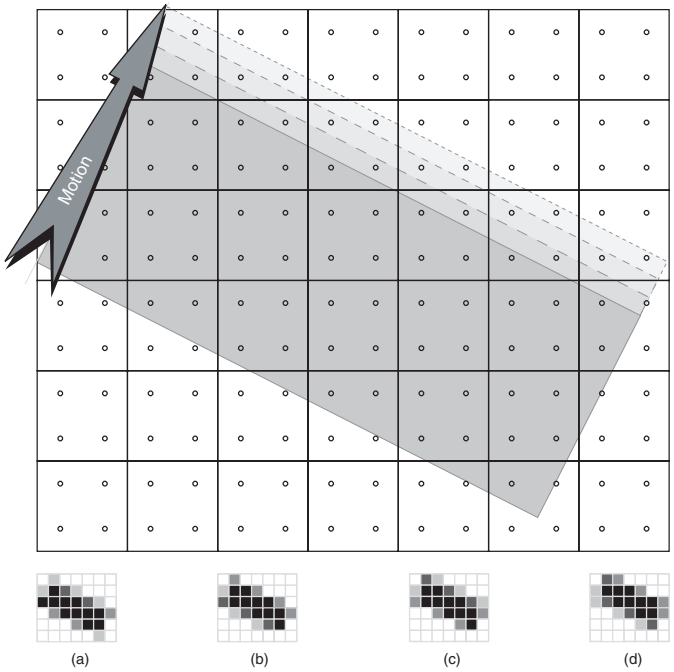

Figure 8: Bad Angle case for RGSS
Results with OGSS shown.

(a),(b),(c) and (d) show the sampled result of the animation

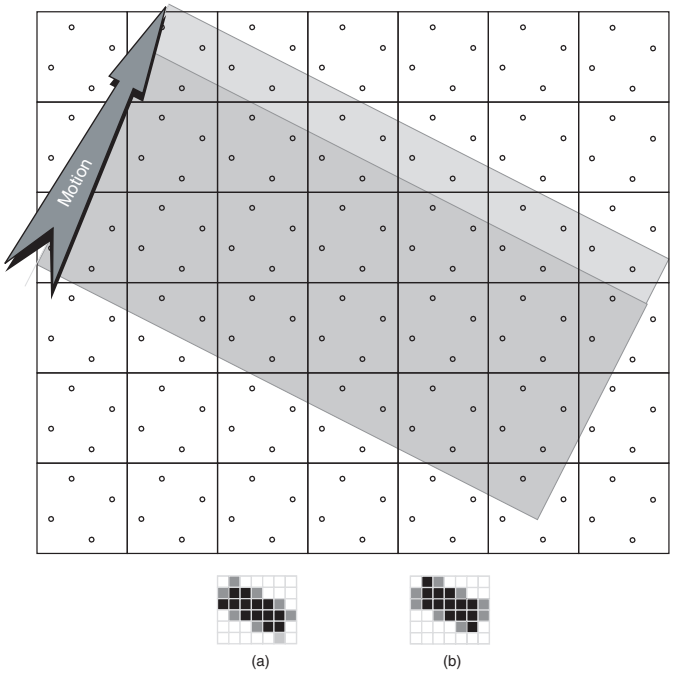

Figure 9: Bad Angle Case for RGSS
Results with RGSS shown.

(a), and (b) show the sampled result of the animation

aliasing methods, the worst case of the OGSS method (Near-Horizontal and Near-Vertical) is visibly worse than the worst case of the RGSS method (an edge parallel to the rotation angle). The impact of fewer shade levels is far greater on the horizontal edge, as anti-aliased by the OGSS method. The results of the two methods in their handling of the RGSS Bad Case Angle are really only distinguishable when the edge is moving around. Even then, the two look very similar. If you know that contrast has a major influence on the visibility of edges, it might very well be that if the contrast isn't too high you won't even notice the difference between the two methods.

Now that we have had a look at the Bad Edge Angle Cases, let us look at the Mediocre Edge Angle Cases (45° for OGSS and 45°+27° for RGSS). These are illustrated in Figures 10 and 11 (next page).

The 45° case is influenced by an effect very similar to that seen in the 27° case discussed before. Simply said, the edge has a very small stair size that makes it impossible to place multiple shade levels. Again, it becomes very difficult to see the difference between both methods and it is only during motion of the edges that you notice that a shade level is lost. The final results look very similar. A cautious conclusion based on this example is that in a game situation it will be very difficult to even notice this lost shade level. Theoretically, the RGSS method is slightly better and will create smoother looking animation, but the difference is minimal, especially when reduced contrast comes into play (white/black is the worst contrast case you can have and is not very realistic in games).

The 45°+27° mediocre case for RGSS is very similar to the previous one and is illustrated in Figures 12 and 13 (next page). Again, the staircase itself is relatively small which leaves little room to use different shades. You lose only one shade level and it's probably not noticeable under more realistic contrast levels. One thing to notice in this angle case is that animation speed also has an impact on the final result. In this OGSS example, you'll see that not every pixel is updated during each animation step. Some pixels keep the same color value. This reveals another factor that can influence quality: frame rate. The higher the frame rate is, the smaller the animation steps are, and the risk is smaller that you will jump over a smoothing level. Simply said, while OGSS might have an extra shading level available it might very well end up looking worse than RGSS when the frame

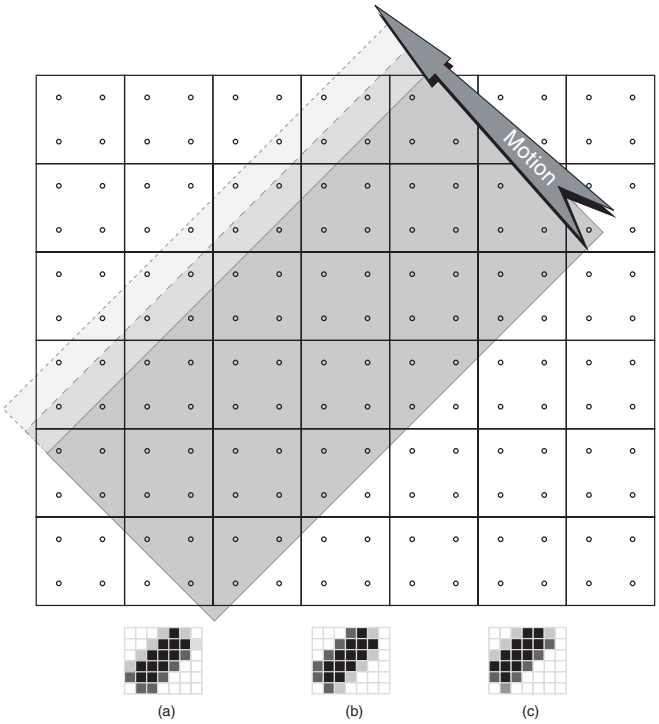

Figure 10: Mediocre Angle Case for OGSS (45°)
Results with OGSS shown.

(a), (b), and (c) show the sampled result of the animation

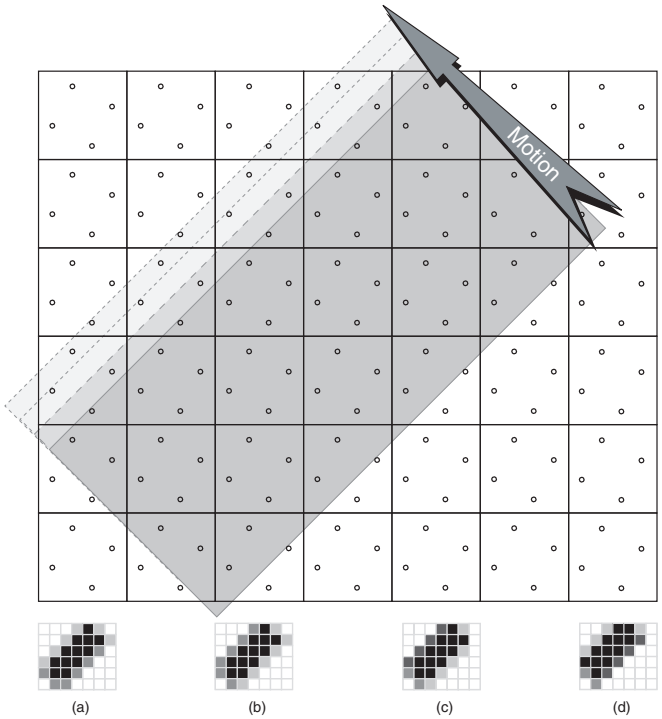

Figure 11: Mediocre Angle Case for OGSS (45°)
Results with RGSS shown.

(a), (b), (c) and (d) show the sampled result of the animation.

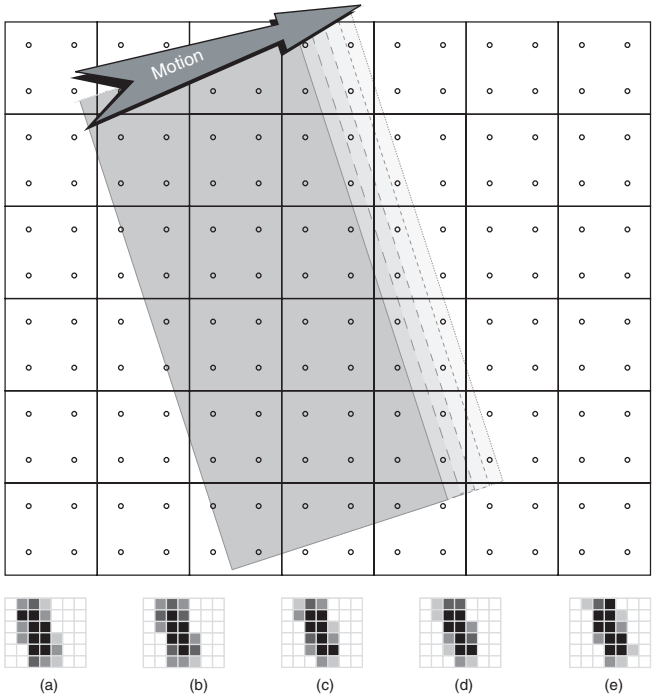

Figure 12: Mediocre Angle Case for RGSS (45°+angle)
Results with OGSS shown.

(a), (b), (c), (d) and (e) show the sampled result of the animation.

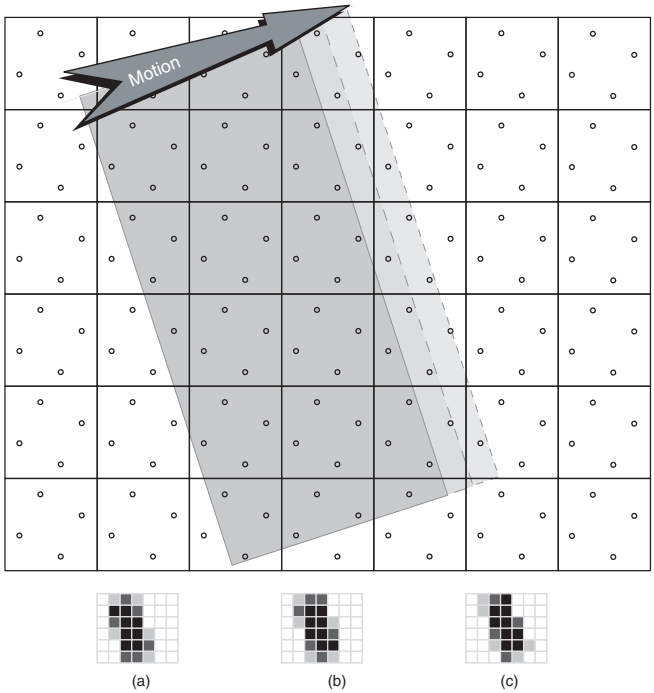

Figure 13: Mediocre Angle Case for RGSS (45°+angle)
Results with RGSS shown.

(a), (b), and (c) show the sampled result of the animation.

rate is lower. What would happen is that the low frame rate makes you lose in-between shade positions. Based on this, we can say that a high enough frame-rate combined with as many shade levels as possible is essential for image quality.

Up until now we have concentrated on 4 sub-sample super-sampling. As mentioned before nothing stops us from using more, or even fewer, sub-samples. Using a regular ordered grid with only 2 sub-samples would result in aliasing in a single direction only. Simply said, if you up-sample the image in the vertical direction, so that a 10-by-10 image turns into a 10-by-20 image, you end up with 2 sub-samples vertically aligned. This situation would give you 3 shade levels: 0 , 50 and 100% in the vertical direction only. The anti-aliasing quality of this setup is very poor. Rotating this grid solves the alignment problem and 3 levels become available in both the horizontal and vertical direction. Figure 14 below illustrates that 2 sub-sample RGSS for near horizontal (and automatically also for near vertical) edges results in an output close to that of 4 sub-sample OGSS.

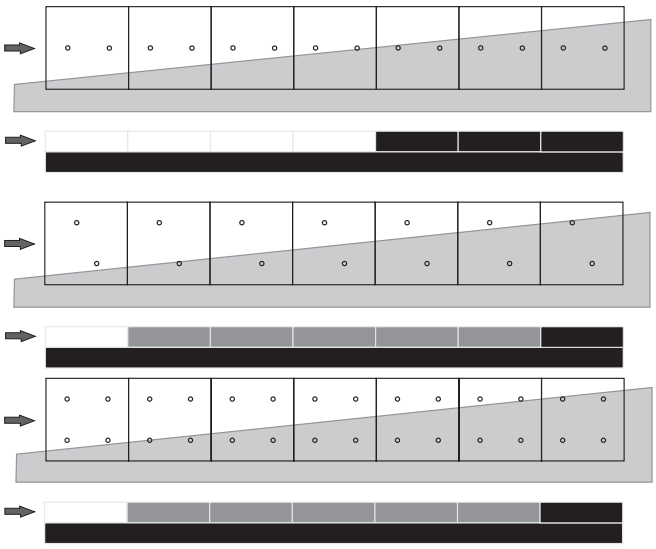

Figure 14: Quality of 2 Sub-sample Super-sampling
From top to bottom we have 2 Sub-sample OGSS, 2 Sub-sample RGSS and 4 Sub-sample OGSS.

A similar result can be found when we compare 16 sub-sample OGSS with 4 sub-sample RGSS for near horizontal and vertical edges. We'll leave the drawing of those images to the reader. Based on these images we can conclude that for near horizontal and near vertical edges the RGSS method delivers quality that is equivalent to the OGSS method, but with fewer sub-samples (sometimes many fewer). In short, 2 sub-sample RGSS is equivalent to 4 sub-sample OGSS and 4 sub-sample RGSS is equivalent to 16 sub-sample OGSS, but only for near-horizontal and near-vertical edges. Over all angles the quality levels are somewhere in-between. From lowest quality to highest quality (based on the theoretical analysis) we get: 2 sub-sample OGSS, 2 sub-sample RGSS, 4 sub-sample OGSS, 4 sub-sample RGSS, and 16 sub-sample OGSS.

Based on these theoretical images, we can come to the following conclusions. Near-horizontal and near-vertical edges show an elongated, staircase jaggie that leaves a lot of room to place shade-levels to make the edge look smooth. The OGSS technique fails to deliver many shade levels for those angles, losing both the 25% and 75% smoothing level. RGSS, on the other hand, supplies the full range of smoothing levels for the near-horizontal and near-vertical edges. The other potentially problematic angles turned out to be not so problematic due to very small staircase effects. The real difference only shows up during animation and even then the differences between the two techniques are minimal. Based on this we can conclude that the difference between OGSS and RGSS is mainly concentrated around the near horizontal and vertical lines.

Which edge angles most need Anti-aliasing?

In the previous part we discovered that the main difference between the two competing techniques lie around the near-horizontal and near-vertical edges, while the difference for other edge angles is close to minimal. Now, it is important to ask whether near horizontal and near vertical edges matter.

There are several arguments to support the idea that the effective anti-aliasing of near-horizontal and near-vertical edges is critical. The first argument is gravity and the second argument has to do with the way our eyes work.

Now you're probably wondering what gravity has to do with anti-aliasing. The key to understanding its impact lies in understanding its nature. When you look around you'll notice that most objects have a lot of near vertical and horizontal edges. Sitting at a desk, you find that it stands horizontally over the floor, the edges of the computer screen and computer are perpendicular to the floor and thus pretty much vertical. A simple look around

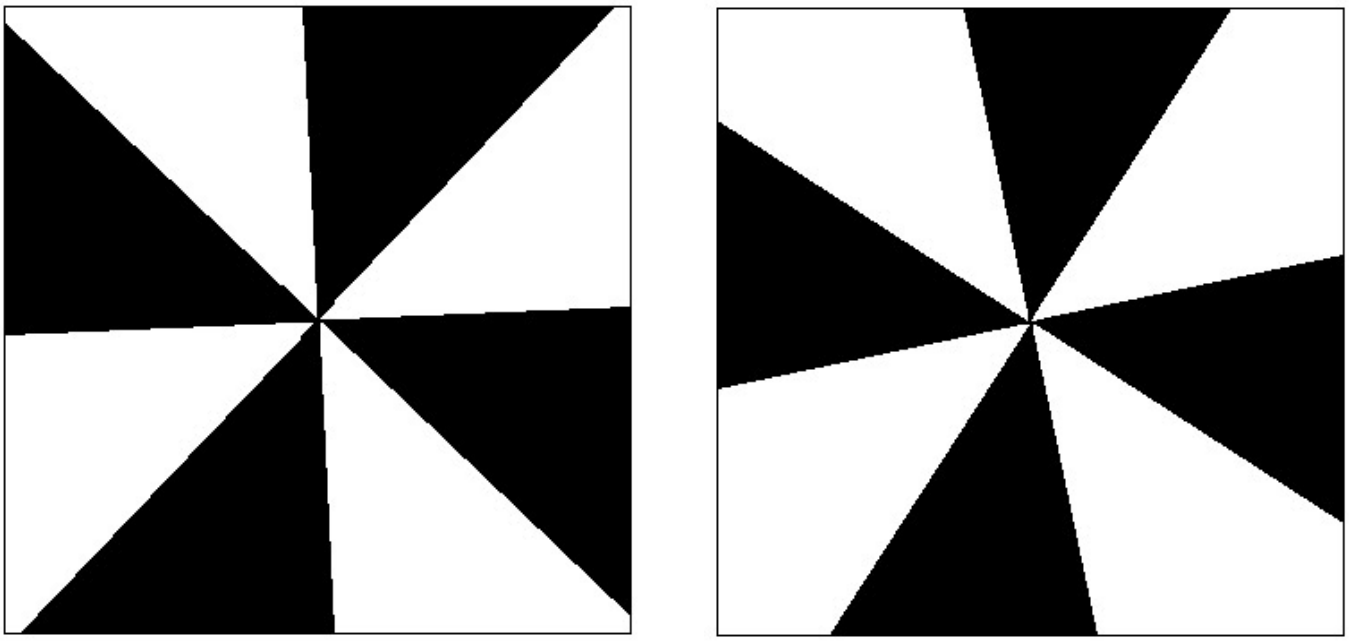

Figure 15: Illustration to show varying eye sensitivity for different angles

should convince you that a lot of things are oriented nearly horizontally and nearly vertically. The reason for this is gravity. If your desk isn't horizontal, then everything will come crashing down due to that little thing called gravity. Of course, there is no rule that says that there will be more horizontal and vertical edges, its perfectly possible to create a game that avoids them, but overall games try to approach reality and that's where gravity comes in. Just think about it. When you play some first person shooter game or simulation, you run or drive or fly around and you act just as you would in the real world. Many of the features in the simulated world are horizontal or vertical. The most striking example is of course a flight simulator. Most of time, you fly in a straight line and the horizon appears as a horizontal line in the distance. The same thing is seen in racing games. In a first-person shooter, you walk around buildings and they typically have numerous horizontal and vertical surfaces. Yes, you can hold your head at an angle to the side but do you really do that while playing?

The point is this: near-horizontal and near-vertical edges, especially as border edges with a high contrast level, appear a lot in reality and in games. While there is no rule that guarantees this, it does seem to be true in most cases.

Our second argument involves the operation of the human eye. Instead of going into an in-depth discussion of how neurons work, we decided to just let you experience the effect. At the top of this page, you can find two images (Figure 15) that show edges under varying angles. The object is the same in both images. The only difference is that they are rotated relative to each other. The image on the left shows you near-horizontal, near-vertical lines and near-diagonal lines. The image on the right shows the same, but shifted with an angle of more-or-less 27°.

You will probably remember from the previous text that those are the Critical Edge Angles of the OGSS method (left) and the RGSS method (right). Hold the image in front of you and keep looking at the edges. Now slowly start to move the paper further away from your eyes, but keep checking out the relative smoothness of the various edges. If you are looking at the image on a computer monitor, position your chair so that you can slowly move back while closely watching the edges of the image. Just keep moving further away until all lines look smooth. Do this before continuing to read this text.

OK, what you should have seen in action is an effect called Vernier Acuity [6]. Vernier Acuity refers to the minimum resolution at which the eye can detect discontinuities. People involved with vision research have been measuring the Vernier Acuity for a long time and they have found that somehow our eyes are very sensitive to discontinuities. In other words, they are very sensitive to the "jaggies." This sensitivity is obviously

influenced by contrast and the test images you have seen have a very high contrast. Now what you should have noticed is that the edges in the right image look smoother and un-jagged much sooner than the near horizontal and near vertical edges in the left image. The edges around 45° also should have appeared smooth sooner. Now what this teaches us is that the human eye is sensitive to discontinuities, especially in the horizontal and vertical directions. Discontinuities at angles in between do not seem to register as sharply. From this we can conclude that special care is needed when doing anti-aliasing for edges near the horizontal and vertical angles since they are the most noticeable and disturbing for our eye.

Real World Image Quality

Image quality within computer games has always been a very touchy subject in that it is such a personal thing. Each person has their own opinion on what looks the best. Many debates have even taken place on the subject. With anti-aliasing, however, there is little room for disputing image quality. Whatever removes the unwanted artifacts brings with it the higher quality. Theoretical image quality is very important because it shows us what should be able to deliver the best quality in a given situation. However, it is also important to look at what real world quality looks like.

Understanding the theory of image quality can mean a lot, but it never compares to real world results. We understand how an image should look, where it should look best and where it should look the worst. However, to really grasp this, we need to see it. To do this, we need to use some very obvious examples of aliasing. For this, we'll use Relic's Homeworld.

Why Homeworld? Homeworld shows off anti-aliasing very well because it is an extremely high contrast game. Like most any game taking place in outer space, Homeworld has dark backgrounds (space) and the light surfaces (ships). Besides that, because the game allows for free rotation, so we get to view nearly every possible angle. Of course all games do not have aliasing nearly as badly as Homeworld, but Homeworld provides a good reference point and allows us to easily distinguish the quality of different anti-aliasing types.

In Figure 16 (next page) we see an image of Homeworld using 4X OGSS. Looking at image 16-A, we see a less than optimal condition for ordered grids. The problem here, as discussed in the theoretical image quality, is there is a lack of additional color samples. However, looking at figure 16-B, we see a considerably more optimal situation for ordered grids bring very good quality, looking near perfect in this particular case.

Figure 17 (next page) shows us a Homeworld image using 2 sub-sample RGSS. Looking closely at Figure 17-A, we see anti-aliased image quality that is very similar to that produced by 4 sub-sample OGSS. This is because the edges in the image are at a near-optimal angle for RGSS. Now, switching to Figure 17-B, we see a near-worst-case for RGSS anti-aliasing. Even so, the anti-aliased image quality is not too bad because the steps are very small. Still, it is not perfect

Figure 18 (page 17) is anti-aliased using a 4 sub-sample RGSS technique. Figure 18-A highlights a near-optimal edge angle situation. The anti-aliased image quality is considerably better compared against both 2 sub-sample RGSS and 4 sub-sample OGSS, coming very close to looking perfect. Figure 18-B illustrates a near-worst-case for RGSS anti-aliasing.

What can we conclude having examined these screenshots illustrating OGSS and RGSS anti-aliasing? On average, it appears that 4 sub-sample OGSS is not noticeably better than 2 sub-sample RGSS, in all but the worst RGSS cases. Four sub-sample RGSS produces superior anti-aliasing image quality during optimal conditions, clearing up almost all traces of aliasing. When it encounters worst-case edge angles, 4 Sub-sample RGSS still manages to hold its own, closely matching the anti-aliasing image quality of 4 Sub-sample OGSS under what are, for it, optimal edge angle conditions.

The theoretical discussion earlier and now these screenshots illustrate the advantages of RGSS anti-aliasing. In all but the worst-case situation, 2 sub-sample RGSS produces image quality that is similar to that produced by 4 sub-sample OGSS. Switching to 4 sub-sample RGSS, the anti-aliasing image quality becomes superior to both and, in the worst-case situation for RGSS, can still be considered nearly identical to the results of best-case 4 sub-sample OGSS.

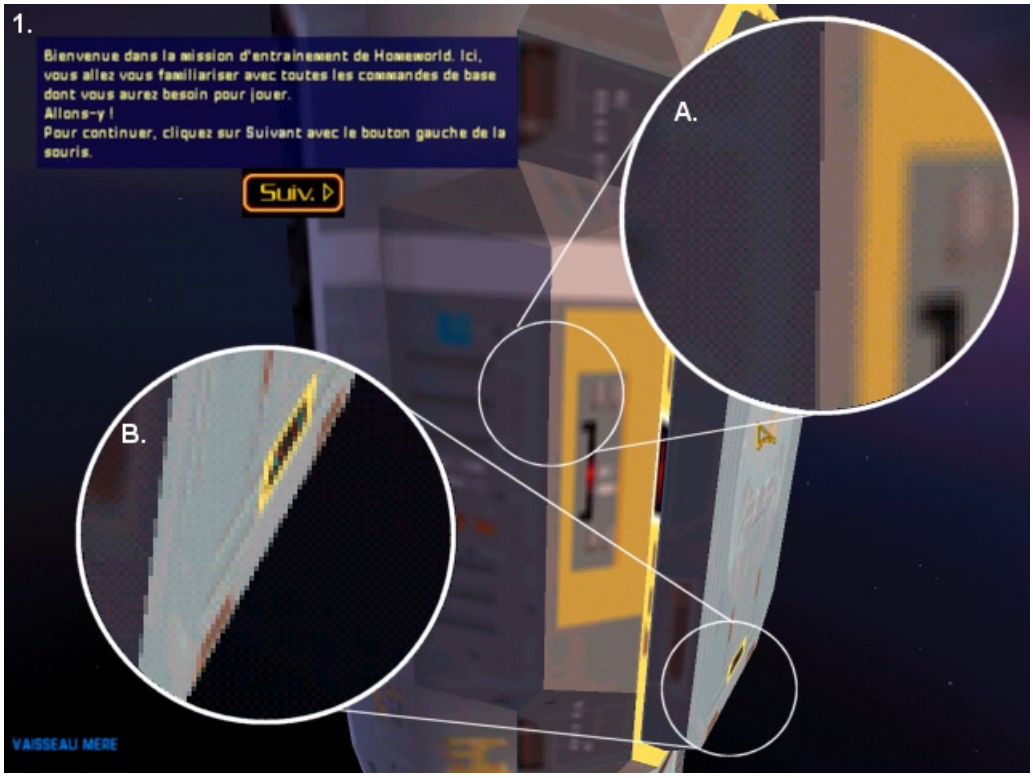

Figure 16: Real World Example of 4 sub-sample OGSS
(a) shows a Bad Case Angle resulting in poor quality
(b) shows a normal Edge Angle resulting in acceptable quality

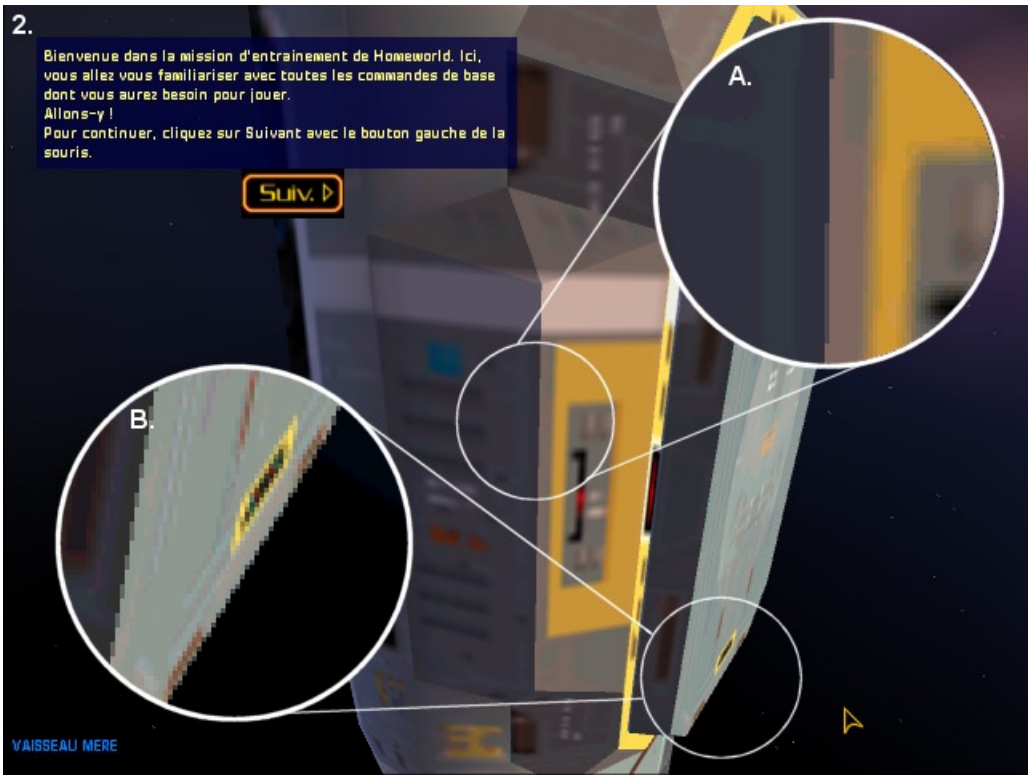

Figure 17: Real World Example of 2 sub-sample RGSS
(a) shows a shows a near vertical edge.
(b) shows a near 45° edge.

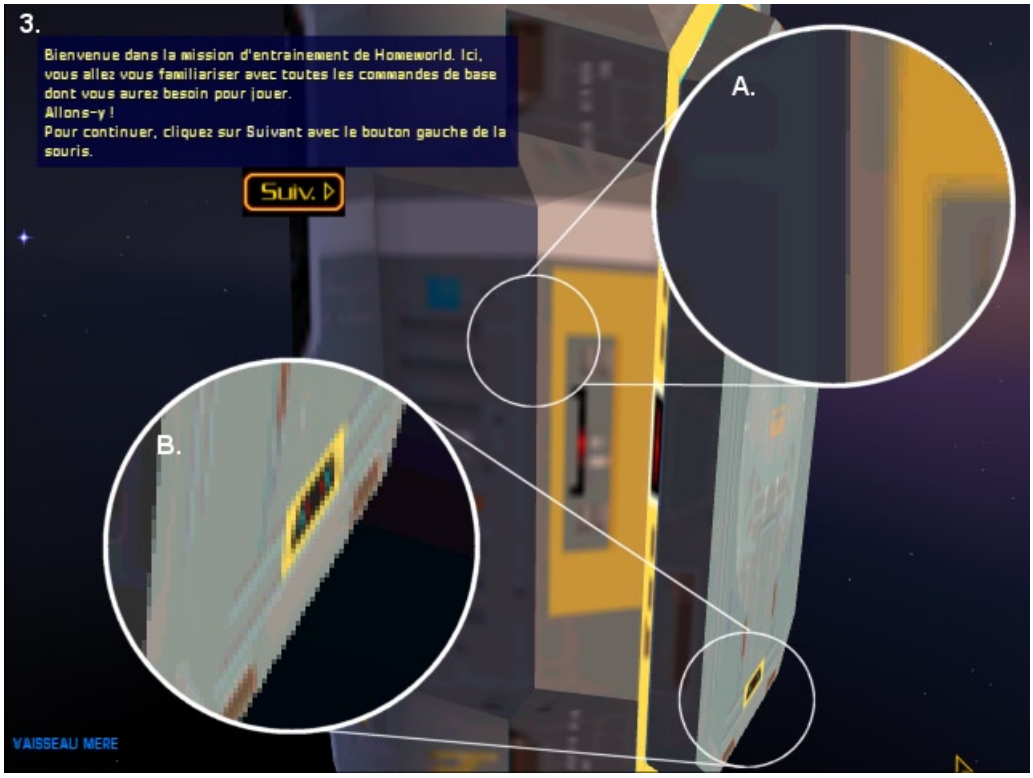

Figure 18: Real World Example of 4 sub-sample RGSS
(a) shows a near vertical edge.
(b) shows a near 45° edge.

Theoretical costs of Anti-aliasing

While anti-aliasing is a great feature to have, it is also one of the most expensive features to activate. In this part of the paper, a brief look will be taken at the various performance penalties incurred when anti-aliasing is enabled using the aforementioned super-sampling methods.

The first factor influencing performance is fill-rate. Super-sampling, as was explained previously, takes sub-samples to execute its anti-aliasing job. These extra samples eat up fill-rate (fill-rate is the peak number of pixels you can determine the final color of). More specifically, a super-sampling method that uses 4 sub-samples has a total fill-rate need that is four times higher than when not doing FSAA. Now what this means for performance is that if fill-rate is the limit, then you will see the frame-rate drop by a factor of four when turning on super-sampling. This fact holds up for all hardware doing true super-sampling no matter how the sub-samples are positioned. Note that multi-sampling is different in this respect and does not impose the same fill-rate penalty. Multi-sampling, however, is beyond the scope of this paper so we will not explore its potential virtues any further. It is important to remember that super-sampling and multi-sampling aren't the same thing.

The second factor influencing performance is memory bandwidth. Bandwidth is very often confused with fill-rate. Actually, the reason for this is that bandwidth and fill-rate are very closely related and linked. In 99% of the cases you'll never reach your fill-rate limit because you'll hit the memory bandwidth wall first. Determining a pixel's color requires information. You need to know what textures are used, you need to know where the polygons are, etc. This information stream is limited by memory bandwidth. When we compare the T-Buffer method of RGSS with OGSS performed using an accumulation buffer, we notice several important differences. The first difference is that the T-Buffer uses a segmented memory pool. Each chip has its own memory pool to store and access textures and buffers. Traditional renderers that implement the OGSS method usually have a single unified memory pool (although there are some exceptions). The impact of this difference is simple. Segmented memory allows more flexibility. Each chip can access the data it needs independently; so one chip can be fetching texture data while the other chip is writing out a final pixel. On top of that, the bandwidth

is effectively doubled since in the case of a two-chip set-up you have two independent memory interfaces (though this memory advanced does come at a cost in the VSA-100 implementation, in that texture data must be repeated in each chip's local memory). Even if a traditional accelerator has memory running at twice the clock speed, it cannot keep up with the VSA-100 since the flexibility is not present in other designs.

The third factor is effectively available clock cycles. The OGSS method implemented on traditional accelerators has to execute a copy and down-sample phase in which the high-resolution image is read, down-sampled and written to the back buffer. This operation stalls the whole 3D pipeline since the 3D part has to wait until this phase is finished before it can continue. The main reason for this is memory bandwidth use. The copy operation takes priority and leaves the 3D core with no memory access. Furthermore, there is no buffer to render to.

The fourth and final factor is memory use. The T-Buffer implementation of RGSS uses four T-Buffers when doing 4 sub-sample super-sampling. These T-Buffers have the same size as the final finished anti-aliased image. To maintain full speed, the system also uses front and back versions of the T-Buffers. This means the total buffer use of the T-Buffer implementation equals: Final Horizontal Screen Resolution x Final Vertical Screen Resolution x 4 Buffers x 3 (one front/back buffer and one Z/Stencil buffer) x bit depth (16- or 32-bit color and Z/Stencil). The traditional OGSS method requires a high-resolution off-screen buffer with four times the resolution of the final image containing a color buffer and a Z/Stencil buffer. This method also requires final back and front buffers to down-sample too (with or without Z/Stencil buffer depending on the implementation). So the total memory use equals: Final Horizontal Screen Resolution x Final Vertical Screen Resolution x 4 (up-sampled resolution) x 2 (color and Z-Stencil Buffer) x bit depth (16 or 32 color and Z/Stencil Depth) + Final Horizontal Screen Resolution x Final Vertical Screen Resolution x 2 (front and back buffers) x bit depth (16 or 32) x (1 or 2) (depending on whether or not a Z/Stencil Buffer is needed at this level - we'll assume that this is not needed). Based on these formula we can calculate how much memory is left for storing textures. It is important to note that due to the segmented memory structure of the VSA-100 T-Buffer implementation, you don't have access to the full texture memory. The left-over texture memory has to be divided equally over the number of chips since each chip has its own bank for textures. All these banks contain the same textures, as these banks are unshared. The buffers (frame and Z/Stencil) are shared. So to obtain the final memory amount available for textures on a T-buffer board, one needs to divide the left over memory (after subtracting the memory needed for the buffers) by the number of VSA-100 chips (memory banks), which is a factor 2 for the Voodoo5 5000 64 MB boards and a factor 4 for the Voodoo5 6000 128 MB boards.

	T Buffer		Traditional		Difference	
	16	32	16	32	16	32
640x480	7.372.800	14.745.600	6.144.000	12.288.000	1.228.800	2.457.600
800x600	11.520.000	23.040.000	9.600.000	19.200.000	1.920.000	3.840.000
1024x768	18.874.368	37.748.736	15.728.640	31.457.280	3.145.728	6.291.456
1280x1024	31.457.280	62.914.560	26.214.400	52.428.800	5.242.880	10.485.760
1600x1200	46.080.000	92.160.000	38.400.000	76.800.000	7.680.000	15.360.000

Table 1: Overview of memory
used for the buffers of the different techniques.

16 bit					
	640	800	1024	1280	1600
64Mb T-buffer Board	28.313.600	26.240.000	22.562.816	16.271.360	8.960.000
128 Mb T-Buffer Board	30.156.800	29.120.000	27.281.408	24.135.680	20.480.000
32 Mb Traditional Board	25.856.000	22.400.000	16.271.360	5.785.600	Underflow
64 Mb Traditional Board	57.856.000	54.400.000	48.271.360	37.785.600	25.600.000
	32 bit				
	640	800	1024	1280	1600
64Mb T-buffer Board	24.627.200	20.480.000	13.125.632	542.720	Underflow
128 Mb T-Buffer Board	28.313.600	26.240.000	22.562.816	16.271.360	8.960.000
32 Mb Traditional Board	19.712.000	12.800.000	542.720	Underflow	Underflow
64 Mb Traditional Board	51.712.000	44.800.000	32.542.720	11.571.200	Underflow

Table 2: Overview of memory
left for the textures of the different techniques.

The tables above (1 and 2) should make it obvious that super-sampling, no matter which implementation, is not a cheap feature to activate. The amount of texture memory remaining decreases enormously with increasing final image resolution. The VSA-100 T-Buffer method requires 12 samples (4 front, 4 back, 4 Z) to store in its buffers but due to the segmented nature of its memory, it needs even more due to the fact that it needs to store the same textures in all memory banks (the segmented structure needed for bandwidth does not allow sharing). Because of these duplicated textures and the fact that only 10 samples are needed (1 front, 1 back, 4 color and 4 Z) the traditional architecture has more memory left when using a 64MB board, and unfortunately, at least in the case of 64 MB DDR boards, they are not mainstream yet due to high costs. The 64MB T-buffer product always has more texture memory left than the similarly priced 32MB traditional boards. Notice that some resolutions and bit depths are not available due to memory under-flow. This is when absolutely no texture memory is left, or when part of the buffers ends up in AGP memory, which delivers unacceptable performance levels. Having enough texture memory left is essential for smooth performance. Otherwise, texture thrashing occurs and this seriously degrades performance. Solid texture compression support can help to reduce this problem.

In the final analysis, we can conclude that super-sampling anti-aliasing methods are very expensive and use up lots of resources. It is close to impossible to predict real world performance based on the theoretical factors discussed here. Due to many complex factors, the difference between theory and reality tends to be great. The true efficiency of a 3D accelerator is influenced by a huge number of factors. This tends to make predictions based on theoretical considerations very hazardous, if not useless. The only conclusion we can safely draw is that turning on anti-aliasing will almost certainly result in a performance drop.

Frequent Misconceptions about Anti-aliasing

In this part of this paper we want to address two myths that are often used as an excuse for not supporting anti-aliasing. The first argument heard quite frequently is this:

"*When you run your game at 1024x768 or higher, you don't see those jaggies and artifacts anymore, so why bother with anti-aliasing?*"

While it is true that some aliasing artifacts are reduced, it does not mean that they are gone. Polygon popping is one of the artifacts that indeed gets reduced by running at a higher resolution, simply because the sample points are closer together due to the higher resolution. This reduces the risk of completely missing a polygon. What doesn't disappear though is the presence of jaggies. Every edge under an angle will remain a staircase no matter how high the resolution of your monitor.

Again, refer to the Vernier Acuity, which refers to the smallest misalignment of two lines an observer can detect, or how easy it is to detect a jaggy. Researchers have run tests where they discovered that the average Vernier Acuity is about 10 arc-seconds. At an 18 inch viewing distance, this is about 1/1200 inch and is barely detectable on a 600 dpi printer. On a typical video display, a one pixel offset is 10-12 times the acuity limit and always noticeable to viewers with normal vision. On the monitors of today, you always will see jaggies no matter how high your resolution gets. One scientific paper reports that a monitor with a resolution of 4000x4000 pixels is required to reduce the jaggies. For people with better than average eyesight (young children for example), you might even need an 18000x18000 display to hide the jaggies. So very simply said, our eyes are very sensitive to jaggies and today's mainstream monitors do not have a high enough resolution to hide them. Do note that contrast has a large impact on the visibility of jaggies and since many games use very low contrast environments (read dark and spooky) it might very well be that you have problems detecting the jaggies at higher resolutions. What is important to understand, however, is that a low-resolution anti-aliased image can look just as good or even better than a high-resolution aliased image. The reason: fewer distractions by un-natural artifacts. [6]

The second misconception is:

"*Anti Aliasing is just blurring…*"

To a certain extent, this is true. When you compare a high-resolution image with one having half the resolution but using anti-aliasing, the latter does indeed look more blurred. But the point is the reduction of artifacts. You should actually be comparing the overall quality and the feel of realism of the image. Actually, go a step further: Our eyes also do nothing but "blur" what we see. It's true. The world around us has infinite detail, yet we do not see it due to the limited resolution of our eye. Nevertheless, the detail is out there. Just move closer to an object and you'll discover small details you did not notice before. Now, would you say that the world around you looks blurry? I don't think so. The basic principle behind anti-aliasing and the way our eyes work is blurring. The things you cannot identify because they are too small are blurred together. So while anti-aliasing is based on blurring, you should not interpret it as something bad. It is blurring, but not over-blurring.

Conclusion

Anti-aliasing is certainly an important factor in image quality and will continue to be important on into the future. At least, until our monitors are capable of displaying incredibly high resolutions, and that type of thing just isn't anywhere in sight. It certainly offers a considerable image quality improvement in nearly every 3D game or application. Of course, it does come at a performance cost. Anti-aliasing, especially when 4 sub-samples come into play, requires approximately 4 times the fill-rate and memory bandwidth to deliver.

It is a common misconception that anti-aliasing is no longer needed with high resolutions such as 1280x1024 and 1600x1200. This is simply not the case. In many situations, a game will actually look better in a lower resolution with anti-aliasing when compared to a higher resolution image with no anti-aliasing. The reason behind this is visual realism, fewer un-natural artifacts. Of course, the eventual goal is to be able to use anti-aliasing and high resolution.

When it comes down to implementation, using a rotated grid is without question better than an ordered grid. Both theoretical and real-world images demonstrate this. How much better depends on the situation. It is clear that a rotated grid takes care of the worst aliasing considerably better than ordered grids and, when using 4 sub-samples, delivers similar results. More significantly, a two sub-sample rotated grid, in most situations, will produce anti-aliasing results similar to those produced by four sub-sample ordered grid anti-aliasing.

The main thing to remember, however, is that *seeing is believing*. The image quality improvement delivered by anti-aliasing must be seen in action to be fully appreciated.

About this White paper

This white paper was commissioned by 3dfx. The project was specifically and explicitly intended to produce an independent technical analysis of various super-sampling implementations and algorithms. The editorial contents were not influenced or altered by 3dfx in any way.

The goal of this white paper is to provide an objective description and analysis of super-sampling techniques and to assess their relative quality.

Contacts

Questions, comments directly related to this white paper should be addressed to Beyond3D by email using the following email-address:

Whitepaper@Beyond3D.com

Questions regarding products from 3dfx should be addressed to 3dfx directly. Contact information can be found on their web site at the following URL:

http://www.3dfx.com

Corrections and addenda to this white paper will be published on the Beyond3D web site and, when necessary, the original paper will be updated. Discussions concerning this white paper will be conducted in the Beyond3D Hardware Forum. The Beyond3D Web site and Forum can be found at the following URL:

http://www.Beyond3D.com

Acknowledgements

The authors would like to thank the following people (in no specific order) for their assistance:

Brian Burke, Bubba Wolford, Marla Kertzman, Gary Tarolli, Scott Sellers, Peter Wicher, Luc Van Gool, Simon Fenney and Derek Perez

Special thanks also go to *Tim Smith* and *Jim Macintosh* from the Beyond3D Team for proofreading, editing and being critical.

References

[1] Aliasing Problems and Anti-aliasing Techniques
http://www.education.siggraph.org/materials/HyperGraph/aliasing/alias0.htm

[2] Programming with OpenGL: Advanced Rendering (Anti-aliasing)
http://toolbox.sgi.com/TasteOfDT/documents/OpenGL/advanced97/node58.html#SECTION00090000000000000000

[3] 22 bit colour Analysed
http://www.beyond3d.com/articles/22bit/
http://www.beyond3d.com/articles/22bitfu/
http://www.beyond3d.com/articles/22bitfu2/

[4] 3dfx T-Buffer
http://www.3dfx.com/3dfxTechnology/tbuffer/tbuffer_whitepaper.pdf

[5] Sampling, Aliasing and Anti-aliasing
http://www-graphics.stanford.edu/courses/cs248-95/samp/samp3.html

[6] Human Vision, Anti-aliasing, and the cheap 4000 Line Display, by William J Leler, 1980 ACM

[7] The Accumulation Buffer: Hardware Support for High-Quality Rendering, Paul Haeberli and Kurt Akeley, Computer Graphics, Vol 24, No. 4, Aug'90, Siggraph '90 Proceedings