

GS User's Manual Supplement

Copyright © 2002 Sony Computer Entertainment Inc.
All Rights Reserved.
SCE Confidential

© 2002 Sony Computer Entertainment Inc.

Publication date: April 2002

Sony Computer Entertainment Inc.
1-1, Akasaka 7-chome, Minato-ku
Tokyo 107-0052 Japan

Sony Computer Entertainment America
919 East Hillsdale Blvd.
Foster City, CA 94404, U.S.A.

Sony Computer Entertainment Europe
30 Golden Square
London W1F 9LD, U.K.


The *GS User's Manual Supplement* is supplied pursuant to and subject to the terms of the Sony Computer Entertainment PlayStation® license agreements.

The *GS User's Manual Supplement* is intended for distribution to and use by only Sony Computer Entertainment licensed Developers and Publishers in accordance with the PlayStation® license agreements.

Unauthorized reproduction, distribution, lending, rental or disclosure to any third party, in whole or in part, of this book is expressly prohibited by law and by the terms of the Sony Computer Entertainment PlayStation® license agreements.

Ownership of the physical property of the book is retained by and reserved by Sony Computer Entertainment. Alteration to or deletion, in whole or in part, of the book, its presentation, or its contents is prohibited.

The information in the *GS User's Manual Supplement* is subject to change without notice. The content of this book is Confidential Information of Sony Computer Entertainment.

 and PlayStation® are registered trademarks, and GRAPHICS SYNTHESIZER™ and EMOTION ENGINE™ are trademarks of Sony Computer Entertainment Inc. All other trademarks are property of their respective owners and/or their licensors.

About This Document

This document is a supplement to the "GS User's Manual". It contains additional information about GS internal processing. It mainly describes items that would be useful in obtaining higher drawing performance and quality.

Contents

1. Details of Drawing.....	5
1.1. Pixel Drawing Order and Page Breaks	6
1.2. Parallel Pixel Engines.....	8
1.3. Pipelines of Pixel Engines	9
1.4. Details of Texture Mapping.....	10
1.4.1. Texture Page Buffer.....	10
1.4.2. Reloading Data to Texture Page Buffer.....	10
2. Details of AA1 Processing.....	11
2.1. Overview of AA1	12
2.2. AA1 Drawing for Triangle Primitives	13
2.2.1. AA1 Drawing for Triangle.....	13
2.2.2. AA1 Drawing for Triangle Strip / Triangle Fan	15
2.3. AA1 Drawing for Line Primitives	17
2.3.1. AA1 Drawing for Line	17
2.3.2. AA1 Drawing for Line Strip.....	17
2.4. Notes on AA1 Drawing.....	18
2.4.1. Effect on Drawing by Primitive Drawing Order.....	18
2.4.2. Width of Antialiasing Area	18
2.4.3. Extrapolation Value of Antialiasing Area.....	19
2.4.4. Thickness of AA1 Line	20
3. Host Interface.....	21
3.1. Input FIFO	22

1. Details of Drawing

1.1. Pixel Drawing Order and Page Breaks

When a drawing command is transferred from the EE to the GS, the setup circuit calculates parameters for the DDA (Digital Differential Analyzer) circuit.

The DDA circuit uses these parameters to calculate the RGB and UV values of the pixels on the three sides of a Triangle by applying linear interpolation. The DDA circuit then operates in a horizontal direction, based on the just-obtained values of the pixels on the three sides, to obtain the RGB and UV values of each pixel in the Triangle.

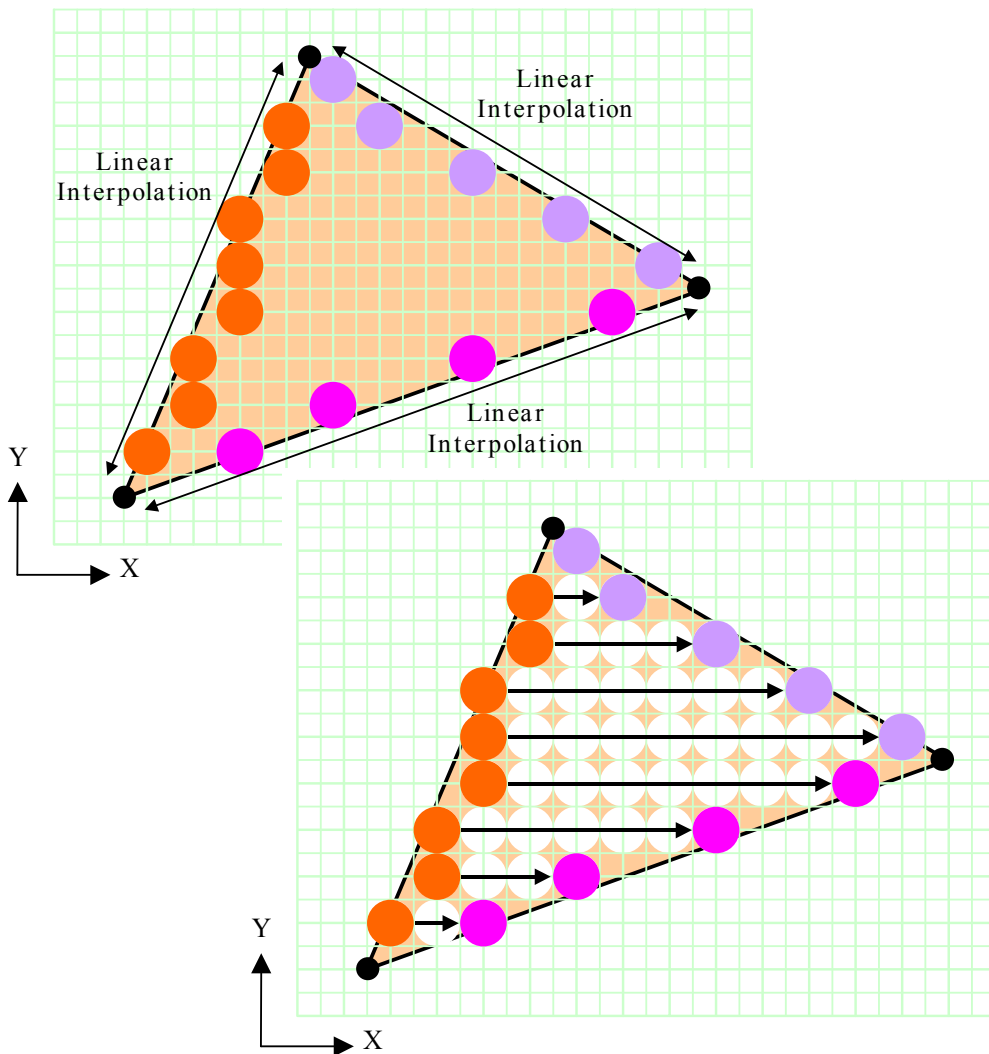


Figure 1-1 Calculation of Pixels by DDA

Since drawing is mainly performed in the horizontal direction, there is a tendency for DRAM page breaks to occur easily when drawing a horizontally lengthened polygon. The frequency of occurrence of a page break varies according to the position where a polygon is located. For example, when drawing Triangles A and B below, 32 page breaks occur to the former, whereas only 1 page break occurs to the latter.

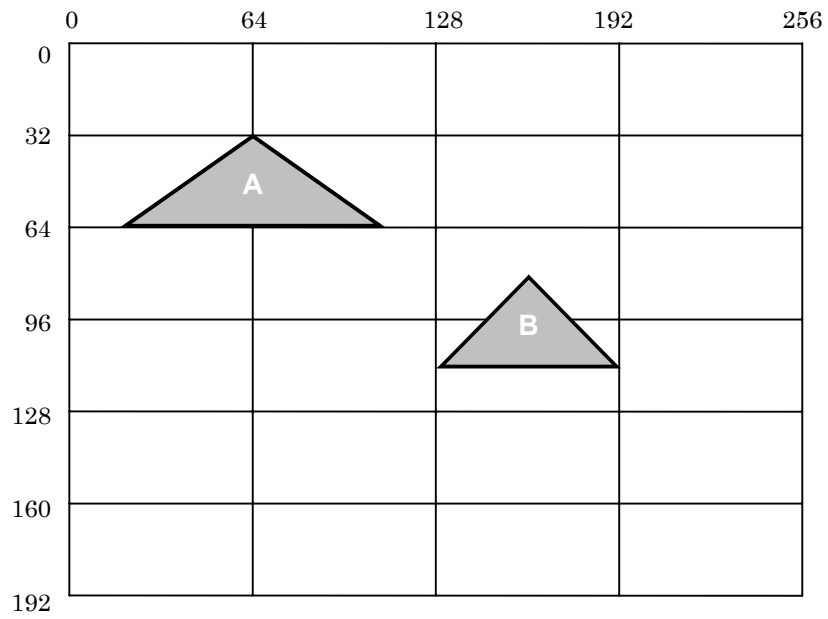


Figure 1-2 Polygons and Page Breaks

1.2. Parallel Pixel Engines

The GS has 16 pixel engines. The DDA circuit described in "1.1. Pixel Drawing Order and Page Breaks" calculates pixel values in parallel every 16 pixels (2 x 8 in height and width) in drawing without texture mapping and every 8 pixels (2 x 4 in height and width) in drawing with texture mapping.

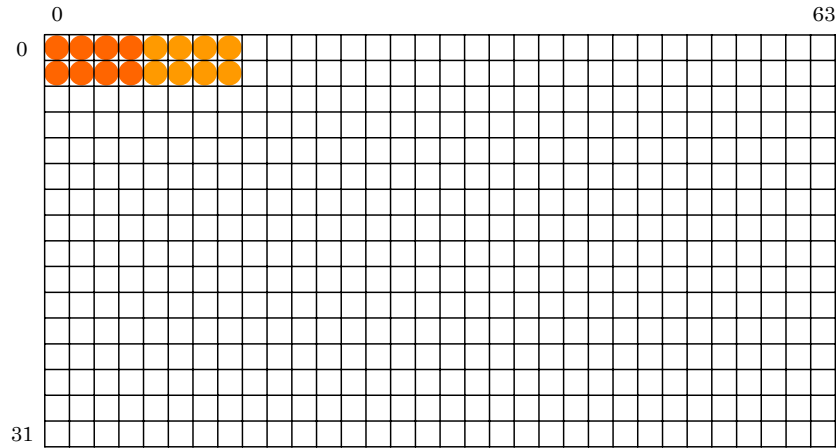


Figure 1-3 Parallel Processing by Pixel Engines

This parallel processing greatly contributes to the high drawing performance of the GS. However, a polygon should be big enough to produce the effect of parallel processing. In most cases, drawing a small polygon has a tendency to enable some pixel engines only and result in performance decrease. Mapping a big texture in smaller sizing may also lose the effect of parallel processing.

1.3. Pipelines of Pixel Engines

RMW (Read Modify Write) is performed in the Z (depth) test and Alpha-blending in the latter half of drawing. RMW reads the color value and Z value from the frame buffer, performs a calculation between them and the color value and Z value obtained in the first half of drawing, then writes the results back to the frame buffer. The time required from the start of the read operation to the end of the write-back operation is seven clocks at the minimum, and will become longer when a penalty is generated by a page break in the frame buffer.

The pixel engines of pipeline structure process multiple pixels continuously. However, while the preceding pixels read an address in the frame buffer and write it back, the succeeding pixels cannot read the same address. Read operation of the succeeding processing is delayed until the end of the write-back operation of the preceding processing, since the results of the preceding processing will be disregarded if the succeeding pixels read the same address during this period. Therefore, if pixels of the same address are drawn successively, the drawing speed decreases for the delay in reading (RMW penalty).

Although an RMW penalty rarely occurs to general drawing, it cannot be avoided completely. In addition, it may extensively decrease performance in special drawing sequences, as shown below.

Drawing Processing	Logical Drawing Performance	Effective Performance by RMW Penalty
Draws a Point to the same coordinates repeatedly.	150 Mpoints/sec	21 Mpoints/sec (14 %)
Draws a Sprite (2 x 8) to the same coordinates repeatedly.	50 Msprites/sec (0.8 Gpixels/sec)	21 Msprites/sec (43 %)
Draws a Sprite (8 x 8) to the same coordinates repeatedly.	37.5 Msprites/sec (2.4 Gpixels/sec)	21 Msprites/sec (60 %)

The RMW penalty occurs regardless of whether Alpha-blending is set on or off. Note that the RMW penalty grows larger when filling the rectangular area with Point primitives or drawing a Sprite repeatedly on the same area for special effects.

Writing to the same rectangular area at an interval of less than 7 cycles will incur the RMW penalty. Also beware of the rectangular area of 2x8 pixels in height and width, i.e. the units in which write operation to local memory is made. However, it is more important not to extend over pages, since the penalty by a page break is larger.

1.4. Details of Texture Mapping

1.4.1. Texture Page Buffer

When performing texture mapping, a necessary texel value is read from the texture buffer, based on the UV value of the pixel for drawing. Texel values are read through the SRAM buffer (called the texture page buffer), not directly from the texture buffer. The texture page buffer is equivalent to the DRAM sense amplifier circuit. It is the same size as one page in local memory (64 x 32 pixels in PSMCT32), and has a function of providing texel values to the pixel engines at high speed. The texture page buffer is like a cache for main memory. If it stores necessary data, the data can be processed at high speed. However, if not, the data should be reloaded from the texture buffer to the texture page buffer. As a result, drawing speed decreases. The data loaded to the texture page buffer corresponds to the pages in the texture buffer, as is. Generally, texture mapping can be performed at high speed with low reload count when using a format with small amount of data per texel, such as PSMT8 and PSMT4.

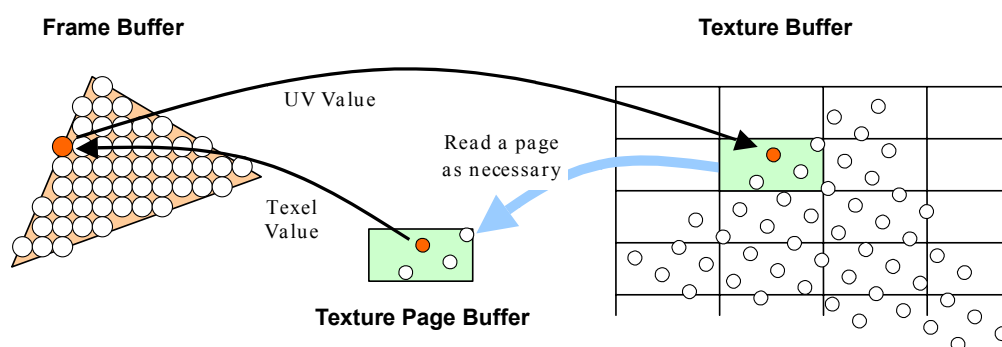


Figure 1-4 Texture Mapping

1.4.2. Reloading Data to Texture Page Buffer

The pixels to be drawn are not mapped simply to the data loaded to the texture page buffer, as shown in Figure 1-4. Especially, reducing a big texture to a small polygon for mapping needs to load the texels corresponding to the pixels in the polygon and reload them into the texture page buffer over and over again. It impairs the parallelism of the pixel engines, and results in drawing speed decrease.

When reduced mapping is required, using MIPMAP enables automatically the use of a small texture that matches the size of the polygon and avoids causing a reload penalty. However, using Trilinear MIPMAP needs two textures to draw the pixels. Therefore, the two textures should exist on the same page, to avoid causing a reload penalty and drawing speed decrease.

The bilinear filter is a technique to use the average of four adjoining texels for mapping. Texels in the adjacent column from where the UV value of the pixel is located are also referred to. Be careful about this, as unexpected reloading may decrease the drawing speed.

You can use the Performance Analyzer to check the frequency of occurrence of a reload operation to the texture page buffer and the degree of its effect on the drawing speed.

2. Details of AA1 Processing

2.1. Overview of AA1

The GS has an Antialiasing drawing function called AA1. It can reduce jaggies on the edges of Triangle, Triangle Strip, Triangle Fan, Line, and Line Strip.

Drawing via AA1 is roughly performed according to the following procedure:

1. Expand the perimeter of the primitive as an Antialiasing area.
2. Calculate the coverage (proportion of the pixel occupied by the original primitive) to each pixel in the Antialiasing area.
3. Perform Alpha-blending to each pixel in the Antialiasing area by using the alpha value obtained based on the coverage.

In other words, by using the Alpha-blending function, AA1 adds the blurred area drawing (which merges into the background) to the perimeter of the primitive.

The following sections provide more details about processing.

2.2. AA1 Drawing for Triangle Primitives

2.2.1. AA1 Drawing for Triangle

This section describes AA1 drawing for a Triangle.

Pixels inside a Triangle are drawn with coverage 1.0. This Triangle is the same as the one drawn with AA1 off.

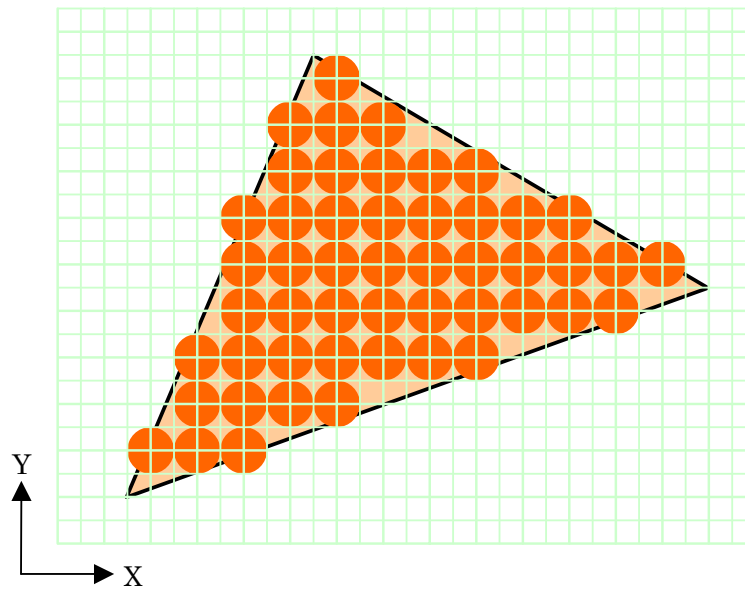


Figure 2-1 Pixels Drawn with AA1 Off

When AA1 is on, the Antialiasing area around the perimeter of the Triangle is also drawn.

First, the three sides of a Triangle are widened for one pixel in the direction of the Y and X axes, at any angle from 0 to 45 degrees and 45 to 90 degrees from the X axis, respectively.

Next, interpolation calculation is performed to pixels included in the widened area to obtain coverage 1.0 for those centered on the original sides and coverage 0.0 for those centered on the new sides after expansion. In addition, the coverage values 0.0 and 1.0 are converted to Alpha values 0 and 0x80 respectively to perform Alpha-blending.

Coverage is calculated with precision of 12 bits. The substantial precision is 7 bits for the gradation for the Alpha value.

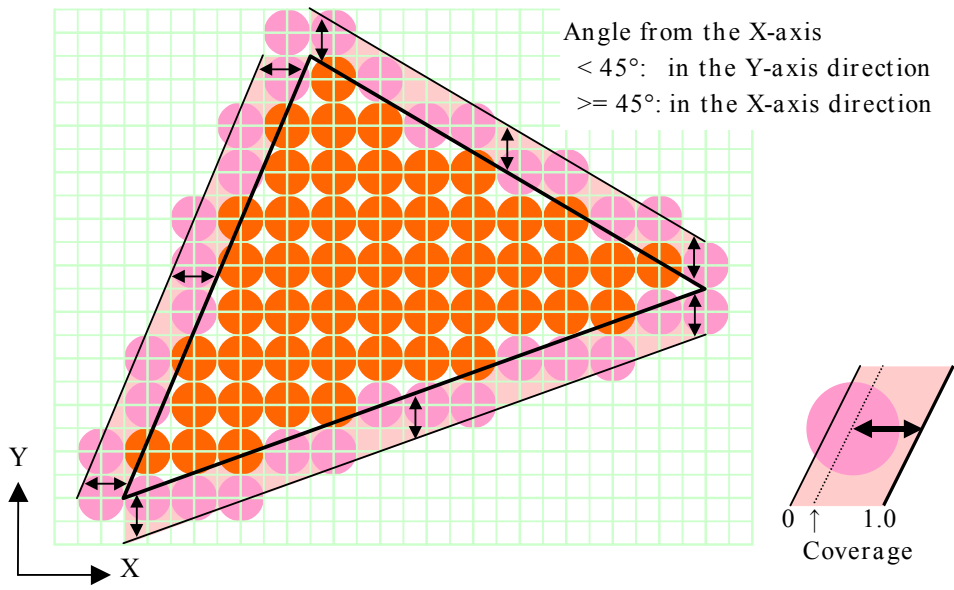


Figure 2-2 Widening the Three Sides

2.2.2. AA1 Drawing for Triangle Strip / Triangle Fan

AA1 drawing for a Triangle Strip and Triangle Fan is basically performed by repeating AA1 drawing for an independent Triangle.

The Antialiasing area expanded by AA1 overlaps with the succeeding Triangle. However, since the Z values of the drawing pixels in the Antialiasing area are not written in the Z buffer, the succeeding Triangle draws over these pixels. Compared to this, the pixels inside the Triangle that overlap the Antialiasing area of the succeeding Triangle are blended by Alpha-blending and result in natural drawing.

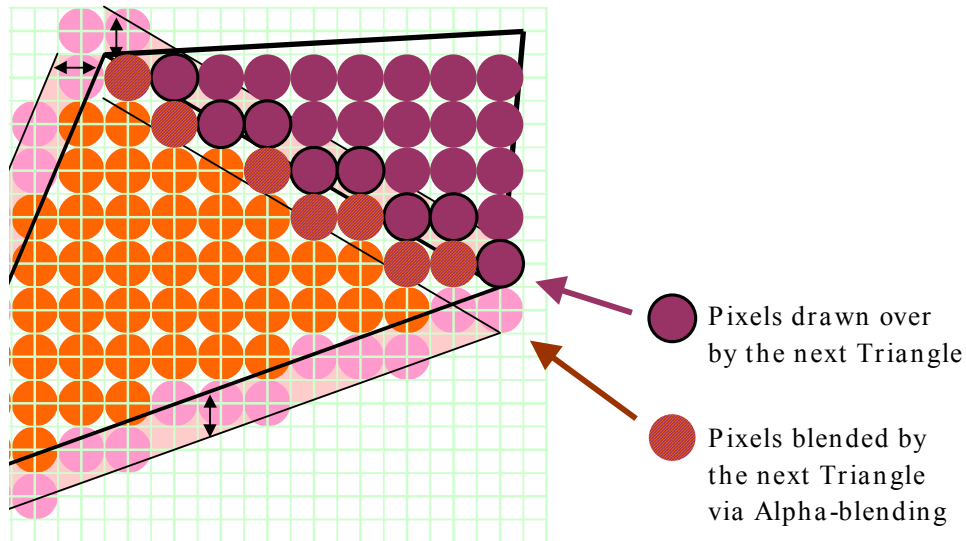


Figure 2-3 Side Shared by Triangles in Triangle Strip

The above figure shows the side shared by two triangles of a Triangle Strip is protruding. When it is dented, the Antialiasing area of both triangles are hidden behind them. Jaggies appear on the side unless ZTEST is set off. Be careful when drawing an object with a dent.

When pixels in the Antialiasing area of a Triangle overlap with those of another Triangle, Alpha-blending is performed twice. The result is the same as the one obtained when the pixels were drawn with alpha values higher than the actual coverage. It may appear as if the vertices were pointed and projected. (A technique to blend AA1 Line is effective at avoiding this phenomenon.)

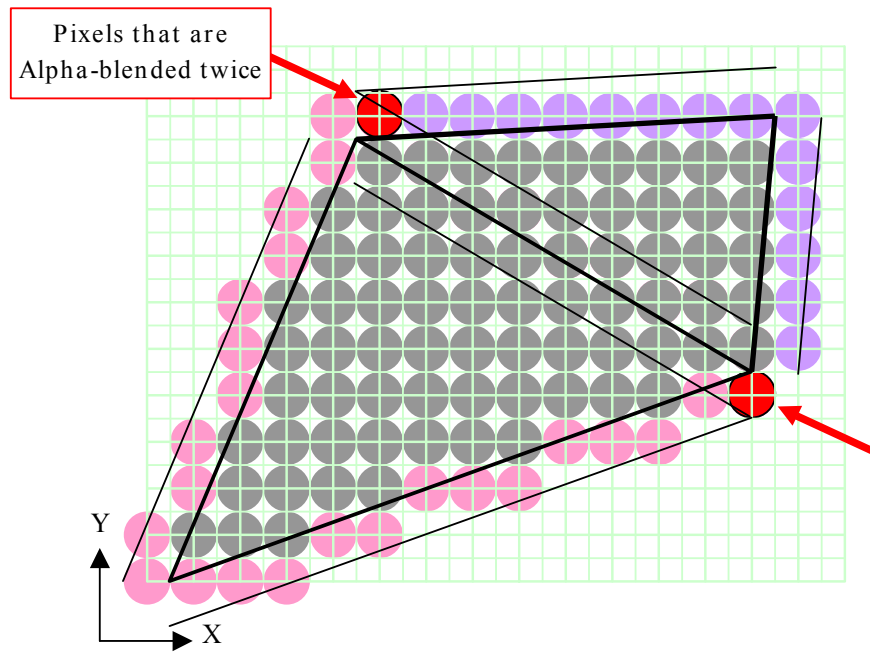


Figure 2-4 Antialiasing Areas Overlapping Each Other

2.3. AA1 Drawing for Line Primitives

2.3.1. AA1 Drawing for Line

AA1 drawing for a Line is a process to perform Alpha-blending to an Antialiasing area that is created by widening the Line.

First, a Line is widened for one pixel on both sides in the direction of the Y and X axes, at any angle from 0 to 45 degrees and 45 to 90 degrees from the X axis, respectively. Next, calculation is performed to the pixels included in the Antialiasing area to obtain coverage 1.0 for those centered on the original Line and coverage 0.0 for those centered on the outer lines of the widened area. In addition, the coverage values 0.0 and 1.0 are converted to Alpha values 0 and 0x80 respectively to perform Alpha-blending.

Note that the original Line itself is not drawn. AA1 Line drawing is performed assuming that all pixels in the Antialiasing area have coverage, and the Z buffer is not written.

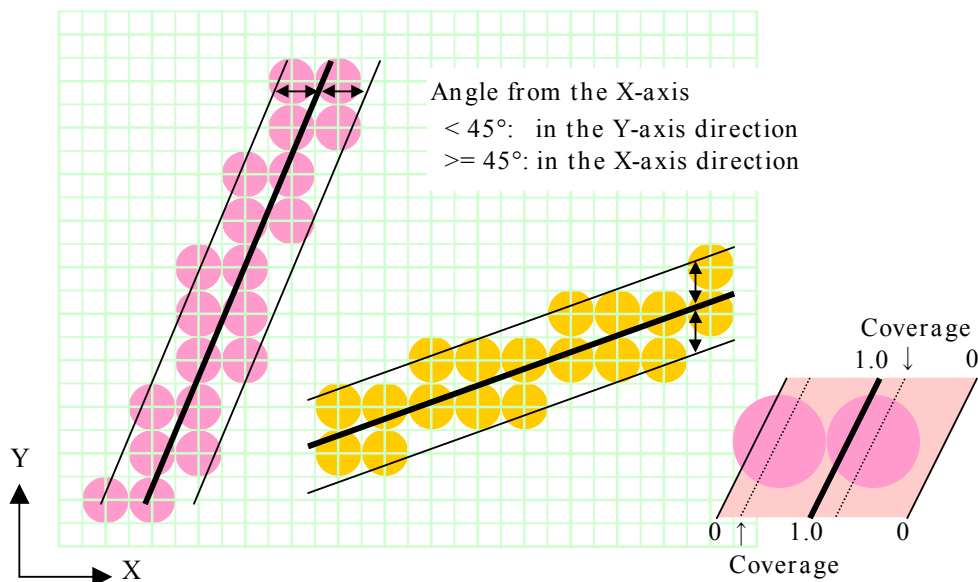


Figure 2-5 AA1 Drawing for Line

2.3.2. AA1 Drawing for Line Strip

AA1 drawing for a Line Strip is performed by repeating AA1 drawing for a Line.

No special processing is applied to endpoints shared by lines.

2.4. Notes on AA1 Drawing

2.4.1. Effect on Drawing by Primitive Drawing Order

The AA1 function performs drawing in the Antialiasing area around the perimeter of the primitive via Alpha-blending. It is necessary to sort the drawing primitives in the direction of the Z axis and draw them in positional order (far to close) to perform drawing appropriately, in the same way as normal Alpha-blending. An example is shown in the following figure.

If Triangle A is drawn before Triangle B when the former is located at a closer position than the latter, drawing is not properly performed to an overlap between them in the Antialiasing area of Triangle A. The overlap is drawn over by Triangle B, and results in jaggies.

Note the drawing order when drawing a spheroidal object. Many primitives are drawn over the perimeter.

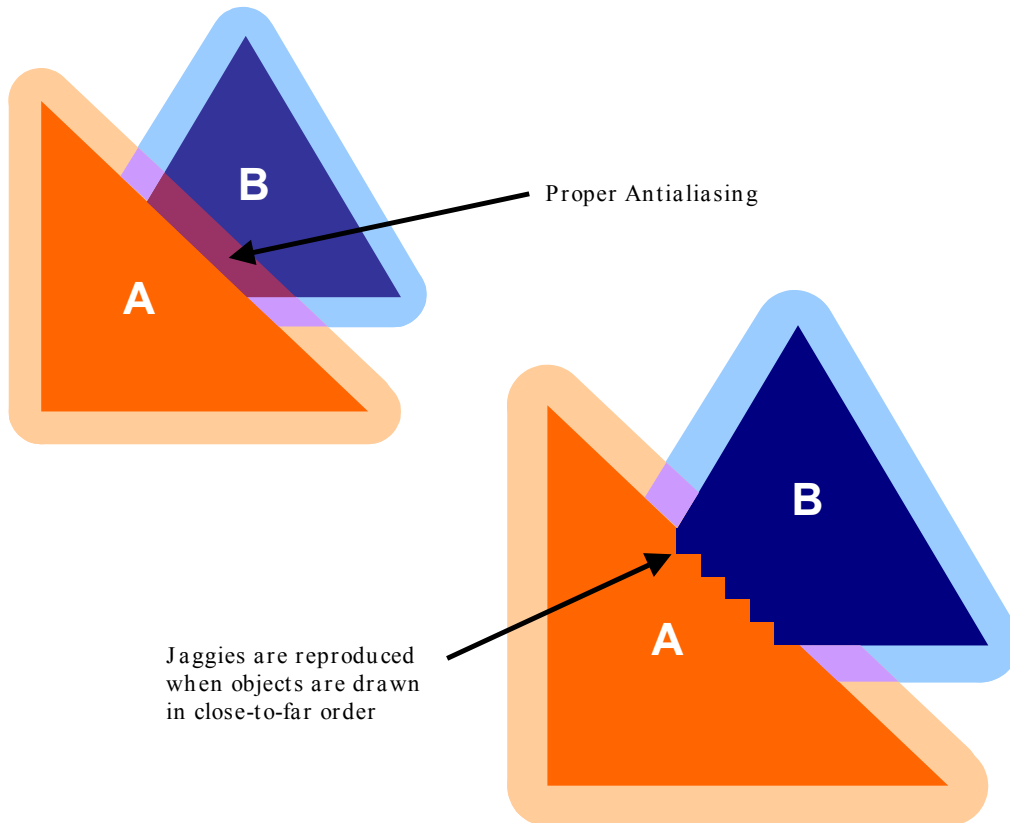


Figure 2-6 Effect on Drawing by Drawing Order

2.4.2. Width of Antialiasing Area

The width of the Antialiasing area is one pixel, which is taken from the drawing screen independently of the original primitive. Therefore, the Antialiasing area of a distant primitive or a primitive highly inclined toward the line of sight relatively becomes too large. Drawing such primitives may end in unnatural results.

You can avoid this problem by setting the AA1 function on or off according to the Z value and inclination of the primitive.

2.4.3. Extrapolation Value of Antialiasing Area

When drawing a Triangle, Triangle Strip, or Triangle Fan via AA1, the color values, texel coordinates, and Z values of the pixels in the Antialiasing area are determined by extrapolation, according to the gradient obtained from DDA, to draw the original primitive. As a result, these values may become larger or smaller than those within the original primitive, and look unnatural.

Extrapolating texel coordinates may cause an improper address in the texture buffer to be accessed. To avoid this, it is effective to specify the CLAMP mode as the texture wrap mode.

The Z (depth) test is performed to the Z value based on the extrapolated Z value. Therefore, when the Antialiasing area of a primitive inclined toward the line of sight overlaps with another primitive, their locations are not determined properly. The result may be unnatural, as if a distant primitive were located at a closer position.

Techniques of avoiding this problem are as follows:

- Sort the primitives in the direction of the Z axis and draw them in positional order (far to close)
- Set the AA1 function off when inclination of a primitive or gradient of the color value is large

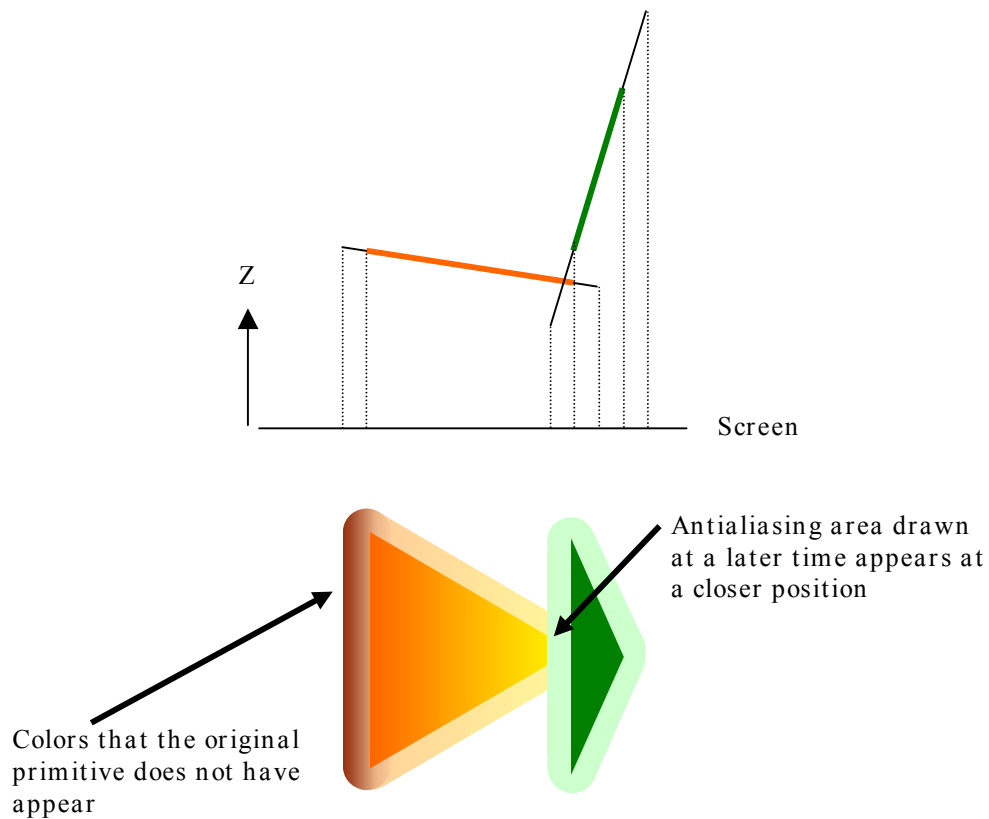


Figure 2-7 Possible Problem by Extrapolation

2.4.4. Thickness of AA1 Line

When drawing a Line or Line Strip via AA1, the color values, texel addresses, and Z values of the pixels in the Antialiasing area are used by simply enlarging the values on the Line in the expanded direction. The color of 2-pixel wide AA1 Line does not change.

Therefore, the thickness of the line may stand out unnaturally, when AA1 Line is drawn over a part with remarkable color changes.

There is a tendency that drawing by using a double buffer with a vertical resolution of 224 or 240 lines for standard interlaced scanning especially thickens horizontal lines. It makes an AA1 Line that forms a narrow angle with the X axis look much thicker. To display a higher quality image, perform the following steps:

1. Perform drawing in the frame buffer with a vertical resolution of 448 or 480 lines
2. When displaying the image, reduce it in a vertical direction using the bilinear filter or perform interlace read

3. Host Interface

3.1. Input FIFO

The input interface from the host to the GS has a 32-stage FIFO for buffering write instructions to the general-purpose registers.

Drawing commands transferred from the host during a time-consuming drawing process (such as clearing the entire screen with a Sprite) are accumulated in this FIFO. However, if accumulated commands reach a certain number, the FIFO becomes ALMOST FULL. The GS stops accepting drawing commands input from the host. If a space is created in the FIFO as drawing progresses, the GS starts accepting drawing commands again.

Write instructions to the privileged registers are not transferred through this FIFO. For this reason, SIGNAL can be cleared by writing to the CSR register if the state where drawing has stopped by redundancy in writing to the SIGNAL register.