Antialiasing Techniques for Improving Image Quality

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

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

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

Table of Contents

What is Aliasing?

Aliasing is a general term for noise that appears as shimmering and flickering on the screen. Since aliasing is a major factor in the loss of image quality, many techniques for eliminating it have been proposed for a long time, and these techniques are collectively referred to as antialiasing. This document introduces the antialiasing functions provided by the GS and several examples of programming techniques for supporting those functions.

The GS receives a sequence of drawing instructions from the EE and generates polygons of the corresponding shapes. Ultimately, this is sampled at the specified frame resolution (such as 640x480) and written to memory. However, this process generates signals that were not originally required. The generation of these unnecessary signals is what is referred to as aliasing. These signals are generated during sampling from the shifting of the frequency components that exceed half of the frame resolution to lower frequencies. The derivation of the term alias, which means assumed name or nickname, comes from the fact that a frequency component that should have been in another band has moved to a different location.

Since a frequency component that should not exist is added to the original signal, the added part visually appears as "screen shimmering," "flicker," or "edge jaggedness."

Since the aliasing component is physiologically bothersome compared to white noise (flat spectrum noise), it may present a problem when evaluating image quality.

Also, aliasing reacts sharply to the signal phase. When a shape on the screen changes with time, the aliasing signal fluctuates significantly more than the variation in the shape. Since the human eye is sensitive to a picture signal that varies with time, when a picture that includes aliasing is viewed as a moving image, the flicker becomes even more noticeable.

The major causes of aliasing are as follows.

- Texture pattern aliasing
- Polygon edge aliasing

First, texture pattern aliasing can be suppressed by performing bilinear mapping. Since bilinear mapping takes the weighted average of the four neighboring texels, this works as a low-pass filter (high band elimination filter), and the high band component that is the cause of the aliasing can be eliminated at this time.

However, bilinear interpolation is effective only when mapping in a direction that enlarges the texture pattern. When mapping is performed so that vertices degenerate such as for a distant object and the resulting texture is reduced, texture pattern aliasing will be a problem even if bilinear interpolation is performed. This can be resolved by using the MIPMAP texture function.

The second cause, polygon edge aliasing, results from a discontinuous step-shaped jump of the brightness values of the neighboring pixels that the edge is sandwiched between. The bigger this jump is, the larger the aliasing component will be. Therefore, among the polygon edges, processing will be more of a problem for the ones for which neighboring pixels are not derived from the same object, that is, for the polygon edges that constitute the edge of the object. This is because there is often a large change in the brightness values surrounding the edge.

GS Antialiasing Functions

The GS has a function called AA1 for dealing with aliasing for a polygon edge. AA is derived from Anti-Alias. This function adds (extrapolates) semi-transparent pixels in the direction of the normal line to the edge when drawing the edge. As a result, the brightness values from outside the edge are smoothly connected to the edge. AA1 can be used according to the following procedure.

1. Set the AA1 bit of the PRIM register to 1.
2. Specify the alpha-blending blend function.
3. For edge pixels, blend the alpha values according to the pixel coverage.

Although antialiasing that uses AA1 in this way often produces excellent results, the following restrictions also apply.

- It cannot be used together with alpha blending.
- The drawing order must be carefully considered.
- Since AA1 unconditionally widens the edge by one pixel, extrapolation is performed more than necessary in parts where the perspective is tight.

For the first two items, since the fundamental principle of the AA1 function is to "add semi-transparent pixels outside the edge," it cannot be used together with alpha blending, which uses the same resources, and the essential restrictions of alpha blending are directly inherited. The last problem is unique to AA1.

Due to the above restrictions, several antialiasing techniques are required to interpolate the AA1 function.

Antialiasing Functions Using AA1

Technique of controlling AA1 according to the inclination of the polygon

This technique ameliorates the defect of the AA1 function. It obtains the inner product of the polygon's normal line and the line of sight and uses this to determine the inclination of the polygon relative to the line of sight. The AA1 function is then selectively turned off for polygons having large inclinations. The harmful effect in which the edge appears to be thickened more than necessary, which is a defect of AA1, can be reduced by this technique.

Sample program:

```
ee/sample/graphics/anti
aa1-feather (R1 button)
```

Technique of redrawing edges by using semi-transparent lines

This technique uses antialiased lines to redraw portions corresponding to edges after drawing has been performed. Antialiased lines are drawn using the neighboring Z values, and it does not particularly matter whether the lines are drawn or not.

Similarly, after drawing has been performed, semi-transparent lines can be used to overwrite textures that you do not want to disappear (such as the center line of the ground). The AA1 function can also be used to together with this technique to combine polygons with textures that you do not want to disappear.

Also, overwriting with AA1 lines has a secondary effect of reducing texture aliasing in the background and making polygon subdivision gaps inconspicuous.

Sample program:

```
refmap-onepass-AAline
texmap-onepass-AAline
mountain
```

Restrictions:

When redrawing lines, drawing increases by the corresponding amount.

Antialiasing Techniques Using 2D Image Processing

There are also methods of performing antialiasing by executing normal 2D image processing for a frame buffer image that has completed drawing. The effectiveness of these easy-to-implement methods is limited. However, immediate effects can be expected because tests can be easily performed.

Some antialiasing techniques that are based on 2D image processing are introduced below.

Super Sampling Type

Super sampling is a general term for techniques that draw to a high resolution buffer and then down-sample it by passing the image through a low pass filter for display. Since the apparent sampling frequency increases due to super sampling, aliasing can be reduced by that amount.

Technique that draws using field mode (h=448) and then redraws by resampling for h=224

The image is drawn in the frame buffer by doubling the vertical resolution [H=448], and texture mapping bilinear interpolation is used for down-sampling to half the resolution.

Sample program:

```
refmap-448 (refmap-noAA does not eliminate aliasing)
```

Restriction:

The Z-buffer must be double the frame mode.

Technique that draws only the object in a work buffer and then redraws by resampling

This technique draws an object for which aliasing is a concern at a high resolution (magnified drawing), then uses texture mapping bilinear interpolation to reduce and remap it. An out-of-focus (fuzzy image) effect can also be applied as an additional function.

Restrictions:

Work VRAM is required.

Mutual interference between objects becomes more difficult to deal with.

Technique that draws the entire screen multiple times with sub-pixel offsets and then draws them using alpha addition

The entire screen is drawn multiple times while performing sub-pixel shifting (for example, by the offset amounts shown below).

(0.0,0.0), (0.5,0.0), (0.0,0.5), (0.5,0.5)

The output image is obtained by multiplying these by 0.25 and adding them together.

An out-of-focus (fuzzy image) effect or motion blur effect can also be applied as an additional function.

Sample program:

```
refmap-4times (refmap-noAA does not eliminate aliasing)
```

Restrictions:

An addition buffer is required.

All objects must be redrawn multiple times.

Technique that uses a 640x448 double buffer to read odd and even scan lines by using separate PCRTC circuits and performs alpha blending for output

This technique prepares a 640x448 double buffer for reading and blending the odd and even scan lines by using two PCRTC circuits. The display is interlaced by shifting the scan lines that are read according to the odd field and even field of the display.

An advantage of this technique is that display failures are difficult to notice even if the drawing time exceeds one field.

Sample program:

```
pcrtc-blend
```

Restrictions:

A large display buffer is required.

When drawing always ends in 1/60 second, GS memory can also be conserved by setting the display buffer to 640x224 and reducing each field when drawing.

2D Filter Type

This technique applies a suitable low-pass filter to the drawing image.

Technique that performs a half-pel shift of the buffer after drawing is complete and then resamples for drawing

This technique takes the drawing buffer as the texture source, shifts it by a half pixel, and then redraws it. This acts as a low-pass filter because an average of the four neighboring points is taken as a result of the texture mapping bilinear interpolation at this time.

- The filter effect can be increased and decreased by varying the pixel shift amount from 0 to a maximum of 0.5 (maximum effect at 0.5; no filter effect at 0.0).
- An out-of-focus effect can also be applied by recursively remapping multiple times.
- By using this together with a Z-test when remapping, the filter can be selectively turned on and off according to the Z-value.
- The filter can be selectively turned on or off for each object by remapping when the background is drawn and then drawing another object after that is done.

Restrictions:

If the same area is directly drawn back as a texture, recursive drawing may occur, and the image may be corrupted. Drawing has to be performed via a small intermediate buffer.

Also, in terms of picture quality, the image may fade more than necessary.

In particular, in frame mode for an interlaced display, the corruption in the vertical direction is striking. This is because in frame mode during interlace mode, the neighboring pixel value in the y-direction of the drawing buffer actually means the preceding pixel value.

Time Axis Filter Type

This technique eliminates aliasing by applying a low pass filter between the drawing buffer image and the previous field image (display buffer image).

Technique that performs a half pel shift of the display buffer, resamples, and adds the drawing buffer to this for mixing

In interlace mode, the display buffer pixel value that corresponds to the drawing buffer pixel does not indicate the same position during display. Therefore, the display buffer is shifted by a half pixel, bilinear interpolation is performed, the image is resampled, and this result is added to the drawing buffer value for mixing. The direction (+0.5 or -0.5) for shifting the display buffer by a half pixel differs for odd and even fields.

This is effective for eliminating the flicker that is characteristic of interlace mode. Also, the effectiveness of the filter can be controlled by changing the proportion for which the drawing buffer value is added (maximum effect at 0.5; no effect at 0.0).

In non-interlace mode, no display buffer pixel correction is needed.

Sample program:

```
aa1-feather (L1 and L2 buttons)
```

Restriction:

This technique is based on the assumption that the image pixel values vary continuously over a continuous time interval. Therefore, after images remain for images that move around intensely.

Depth Cue Type

This technique eliminates aliasing by reducing the differences in brightness values of distant objects.

Technique that sets the fog color to the average value for each object

This technique sets the average of the brightness values constituting an object as its fog color and switches the fog convergence value for each object. The aliasing within an object can be eliminated because the brightness values of the pixels constituting an object converge to the same color as the object goes into the distance.

Naturally, the mutual edge aliasing of objects may remain. However, even in that case, the result is no worse. The reason is that even if objects A and B are adjacent to each other, the difference between the average of the brightness values that are included in A and the average of the brightness values that are included in B will not exceed the maximum difference between the brightness values that are included in A and the brightness values that are included in B.

Object Level Antialiasing Techniques

Aliasing often occurs due to distant objects that have degenerated.

This is because the proportion of edges per unit drawing area increases because the texture is reduced when it is mapped as the vertex coordinates of the object degenerate, and edge aliasing becomes more obvious. The essential solutions for this will be mip mapping and mip modeling mechanisms and the examination of texture patterns and modeling that take into account antialiasing.

Problems associated with display methods also have model-based solutions. For example, in interlace mode, since neighboring pixel values in the drawing buffer vertical (Y) direction are actually displayed on every other line, directionality occurs in aliasing. This causes flickering to occur for the texture of the floor twice as easily as for the walls. As a result, texture pattern design must take into account such factors as the direction in which the pattern is affixed most often.

These problems necessitate investigations that return to the authoring stage or, in some cases, investigations that return to the basic theory of game display system design.

Sample Program Details

Processing is explained in more detail for the following antialiasing sample programs.

refmap-noAA, refmap-448, refmap-4times, refmap-onepass-AAline, texmap-onepass-AAline, and aa1-feather

refmap-noAA

This is a sample of a normal 640x448 interlace display with no antialiasing. The image is drawn according to the following processing flow.

sceGsResetGraph:	Sets up the GS.
sceGsSetDefDBuff:	Sets up the double buffer parameters.
sceGsSetDefTexEnv:	Sets up texture mapping parameters (sets up parameters in the TexEnv structure set by the user).
sceGsPutDrawEnv(&texenv.giftag):	Uses DMA to send the texture mapping parameters to the GS.
DMA via VU1 by My_dma_setup:	Performs DMA via VU1 for packets that include the texture, which were set in packet.dsm.
open pad processing	
while:	Statements from here down are repeated for each field (1/60 second).
sceGsSetHalfOffset:	Shifts xy-offset by 1/2 pixel in the V direction to match odd and even fields.
sceGsSwapDBuff:	Switches double buffer (DBP=0, DBP=70) display buffer and drawing buffer.
get pad status:	Gets pad information.
DMA via VU1 by My_dma_setup:	Performs DMA via VU1 for packets that include the texture, which were set in packet.dsm. (Although only the settings should actually be sent, everything is simply transferred by DMA here.)
SetVu1PacketMatrix:	Writes information such as the viewpoint or light source position that was calculated from the pad information to my_environment (data in packet.dsm which is written to VU1 MEM by next DMA transfer).
<Shading display>	
DMA via VU1 by My_dma_start:	Performs DMA transfer of matrix information in my_environment and polygon data to VU1 and starts up the VU1 light source calculation and perspective transformation program.
<Line display>	
DMA via VU1 by My_dma_line_start:	Performs DMA transfer of matrix information in my_environment and straight line data to VU1 and starts up the VU1 light source calculation and perspective transformation program (only GIFTAG differs for the data).

The processing flow shown above constitutes the basic refmap program. This conforms to the conventional reference mapping samples.

refmap-448

This program creates the drawing by using 640x448 PSMCT16 (R5,G5,B5,A1) format and reduces it to 640x224 PSMCT32 format for display.

Since 640x448 PSMCT16 and 640x224 PSMCT32 buffers are the same size, normal double buffers can be used. The Z-buffer takes an area that is doubled in size because the vertical resolution is doubled.

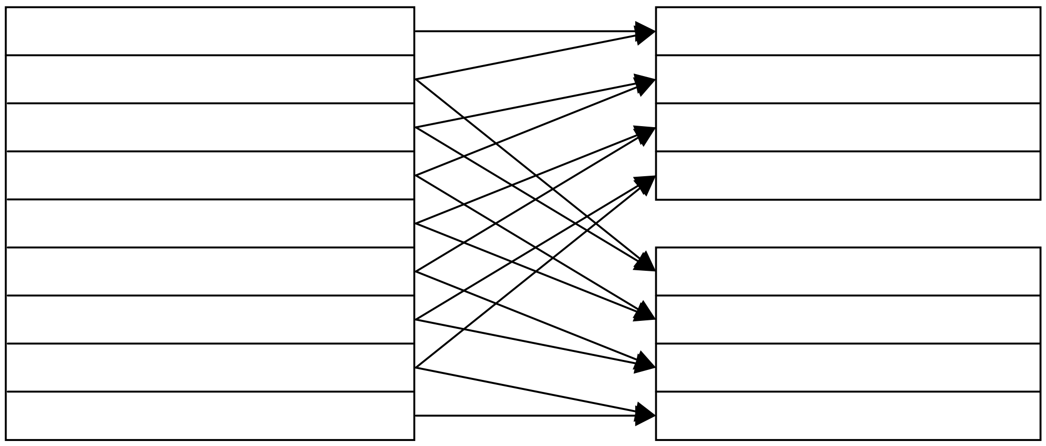
NTSC divides a screen with 640x448 resolution into two screens for the odd and even lines and draws each of these screens using 640x224 resolution. As a result, the Y-direction resolution is essentially cut in half. Since drawing is performed in a 640x448 buffer, this drawing has almost the same resolution in the vertical and horizontal directions.

When converting 640x448 resolution to 640x224 for output, the drawing can be displayed by using point sampling to extract only odd or even lines without specifically applying a filter. However, when the resolution changes, interlace-specific flicker can be seen. Therefore, a filter is applied to the 640x448 resolution to create a 640x224 output buffer.

When the select button has been pressed (filter_flg ==0), odd and even lines are blended in a 1:1 ratio. This is performed by using squeeze_frame or squeeze_frame1.

Y-shift for 1/2 pixel interlace is performed by shifting the texture address by 1 pixel when drawing.

Figure 1: Interlace Screen Synthesis With 1:1 Blending



When the start button has been pressed (filter_flg ==1), 3 lines are blended in a 1:2:1 ratio. This is performed by using squeeze_frame_50p, squeeze_frame1_50p, and the like.

Although the picture is dimmer compared to 1:1 blending, it has a more natural feeling.

sceGsResetGraph:	Sets up the GS.
sceGsSetDefDBuff:	Sets up the double buffer parameters.
use one buffer as 640x448x16	To use the drawing buffer with 640x448 PSMCT16 format, set xyoffset, scissor, dither, and the like.
sceGsSetDefTexEnv:	Sets up texture mapping parameters (sets up parameters in the TexEnv structure set by the user).

sceGsPutDrawEnv(&texenv.giftag):	Uses DMA to send the texture mapping parameters to the GS.
DMA via VU1 by My_dma_setup:	Performs DMA via VU1 for packets that include the texture, which were set in packet.dsm.
open pad processing	
while:	Statements from here down are repeated for each field (1/60 second).
sceGsSetHalfOffset: (None)	This cannot be used because the image is shifted in the V direction when the frame is copied. Actually, this can't be used because the scissor parameters are different.
squeeze_frame, etc:	Applies a filter when copying from the drawing buffer to the display buffer to match odd and even fields.
sceGsSwapDBuff:	Switches double buffer (DBP=0, DBP=70) display buffer and drawing buffer.
get pad status:	Gets pad information.
DMA via VU1 by My_dma_setup:	Performs DMA via VU1 for packets that include the texture, which were set in packet.dsm. (Although only the settings should actually be sent, everything is simply transferred by DMA here.)
SetVu1PacketMatrix:	Writes information such as the viewpoint or light source position that was calculated from the pad information to my_environment (data in packet.dsm which is written to VU1 MEM by next DMA transfer). Parameters are set so that the Y-direction size is doubled. (The sceSamp0ViewScreenMatrix argument has been changed.)
<Shading display>	
DMA via VU1 due to My_dma_start:	Performs DMA transfer of matrix information in my_environment and polygon data to VU1 and starts up the VU1 light source calculation and perspective transformation program.
<Line display>	
DMA via VU1 due to My_dma_line_start:	Performs DMA transfer of matrix information in my_environment and straight line data to VU1 and starts up the VU1 light source calculation and perspective transformation program (only GIFTAG differs for the data).

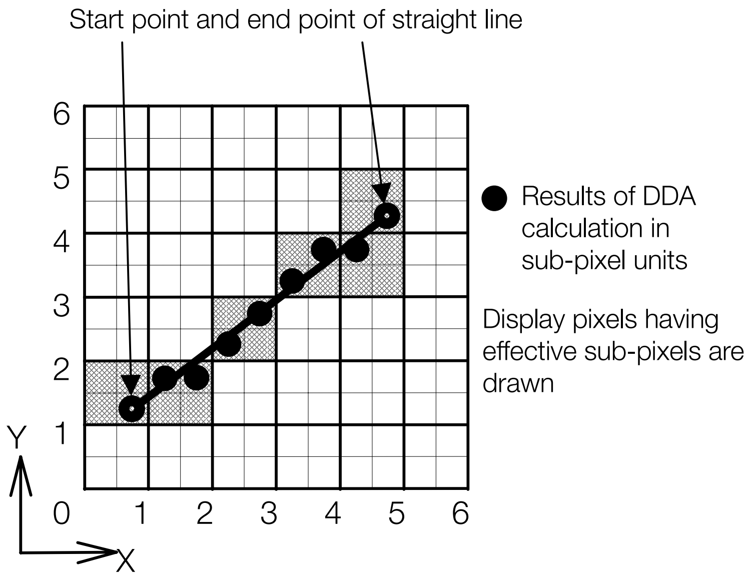
The processing flow shown above has been created by modifying the basic refmap program.

refmap-4times

This program increases the essential resolution and eliminates aliasing by performing drawing four times while shifting the position relative to the 640x224 PSMCT32 buffer in sub-pixel units and takes 1/4 the image each time for storage in the display buffer.

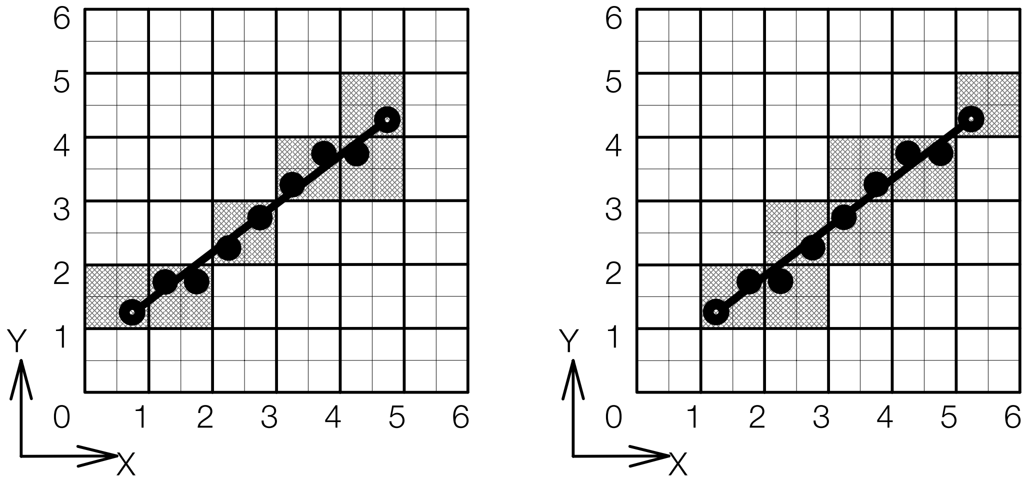
The theory of overwriting four times is explained below for straight line drawing.

Figure 2


Start point and end point of straight line
● Results of DDA calculation in sub-pixel units
Display pixels having effective sub-pixels are drawn
Y
X
0
1
2
3
4
5
6

Straight-Line Drawing with Sub-Pixel Accuracy

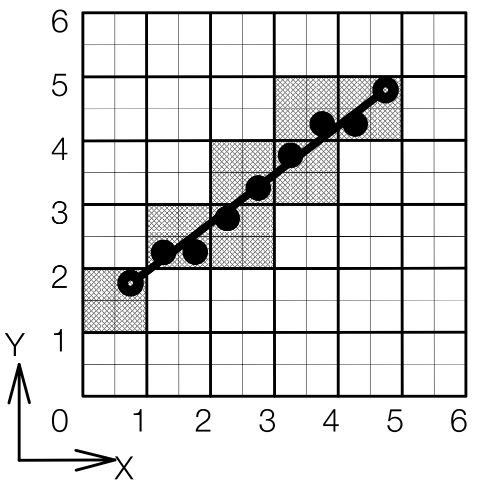
Figure 3



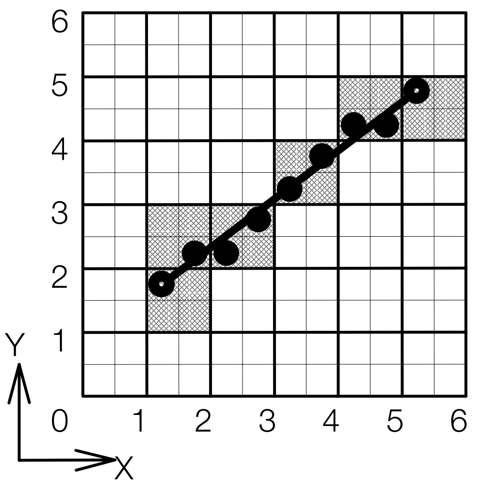
Display Pixels Shifted by dx=0.0, dy=0.0

Display Pixels Shifted by dx=0.5, dy=0.0

Figure 4

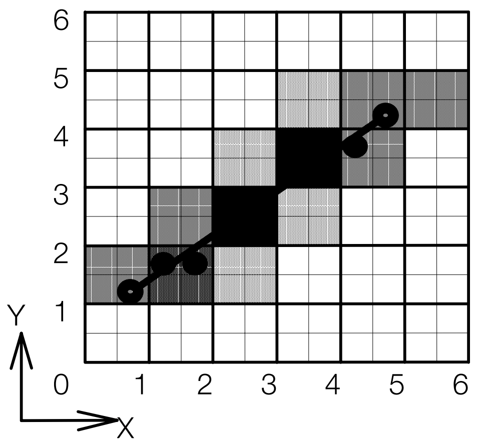


Display Pixels Shifted by dx=0.0, dy=0.5



Display Pixels Shifted by dx=0.5, dy=0.5

Figure 5



Antialiasing Line Produced from Four-Layered Shifted Images

Since the effect of this pixel shifting is even valid for the Z-buffer, the essential resolution of the entire screen increases.

The display mode (shading or line) of the object can be switched by using the L1 and L2 buttons. This enables you to see the difference in the effects of refmap-448 and refmap-4times.

This program uses a drawing buffer other than the normal double buffer. The parameters for this buffer are saved in the sb structure.

sceGsResetGraph:	Sets up the GS.
sceGsSetDefDBuff:	Sets up the double buffer parameters.

set drawing buffer environment	Set up the drawing buffer for drawing four times. To set the drawing parameters at one time, use context 2 to set the drawing parameters for the drawing buffer.
sceGsSetDefTexEnv:	Sets up texture mapping parameters (sets up parameters in the TexEnv structure set by the user). To draw in the double buffer area (even though this is only alpha blend copying performed four times), use context 1.
sceGsPutDrawEnv(&texenv.giftag):	Uses DMA to send the texture mapping parameters to the GS.
DMA via VU1 by My_dma_setup:	Performs DMA via VU1 for packets that include the texture, which were set in packet.dsm.
open pad Processing	
while:	Statements from here down are repeated for each field (1/60 second).
sceGsSetHalfOffset2:	Adds Y-direction offsets relative to the drawing buffer (context2) according to the odd or even lines.
sceGsSwapDBuff:	Switches double buffer (DBP=0, DBP=70) display buffer and drawing buffer.
get pad status:	Gets pad information.
DMA via VU1 by My_dma_setup:	Performs DMA via VU1 for packets that include the texture, which were set in packet.dsm. (Although only the settings should actually be sent, everything is simply transferred by DMA here.)
SetVu1PacketMatrix:	Writes information such as the viewpoint or light source position that was calculated from the pad information to my_environment (data in packet.dsm which is written to VU1 MEM by next DMA transfer). Parameters are set so that the Y-direction size is doubled. (The sceSamp0ViewScreenMatrix argument has been changed.)
sceGsPutDrawEnv(&sb.giftag):	Clears the drawing buffer.
set offset:	
sceGsPutDrawEnv(&lb1.giftag):	First, draws without pixel shifting.
<Shading display>	
DMA via VU1 due to My_dma_start:	Performs DMA transfer of matrix information in my_environment and polygon data to VU1 and starts up the VU1 light source calculation and perspective transformation program.
<Line display>	
DMA via VU1 due to My_dma_line_start:	Performs DMA transfer of matrix information in my_environment and straight line data to VU1 and starts up the VU1 light source calculation and perspective transformation program (only GIFTAG differs for the data).
add_frame:	Takes 1/4 the contents of the drawing buffer and adds it to the display buffer.
sceGsPutDrawEnv(&sb.giftag):	Clears the drawing buffer.
set offset:	sceGsPutDrawEnv(&lb1.giftag): Draws this time by shifting 1/4 pixel in the Y direction. (Since the display is interlaced, the Y-direction shift is 1/4 not 1/2.)

<The following is repeated until the images are overlaid 4 times>

The processing flow shown above has been created by modifying the basic refmap program.

refmap-onepass-AAline

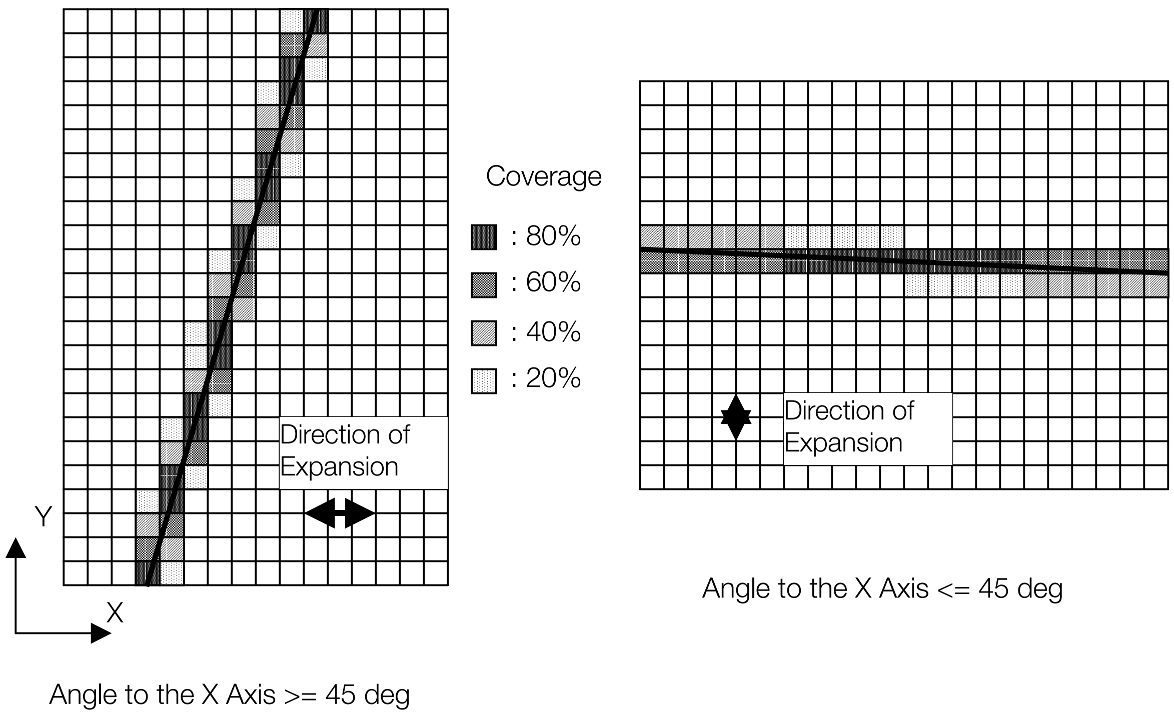
This program and texmap-onepass-AAline or mountain eliminates aliasing with AAline overwriting.

First, the theory of AAline overwriting is explained.

Line Antialiasing Function (AA1 Function) of the Graphics Synthesizer

The Graphics Synthesizer, which has a coordinate calculation function with sub-pixel precision, can draw straight lines for which antialiasing has been applied.

Figure 6



When antialiasing is applied to a LINE, the antialiasing effect is obtained by determining the direction for widening PIXELs according to the angle relative to the X-axis (horizontal direction), setting the coverage for a PIXEL of a LINE having width to the alpha value of that PIXEL, and using the following blending function for drawing. The coverage is determined so that the total of two PIXELS in the expansion direction is always 100%. (In the above figure, the four types of typical coverage are 20%, 40%, 60% and 80%. The coverage is actually calculated continuously with sub-pixel precision.)

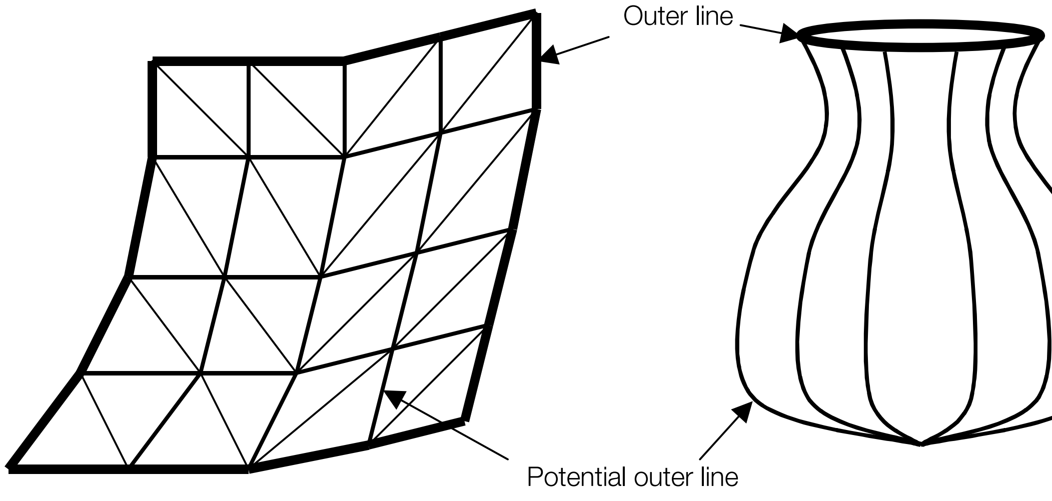
(Pixel value) = (Source color)x(Alpha) + (Destination color)x(1 - alpha)

Antialiasing Methods for Triangles and Triangle Strips

STEP1

Collect together the Triangles or Triangle Strips to be drawn and extract the silhouette lines or silhouette candidate lines of the shapes that are to be produced as Line or Line Strip data.

Figure 7

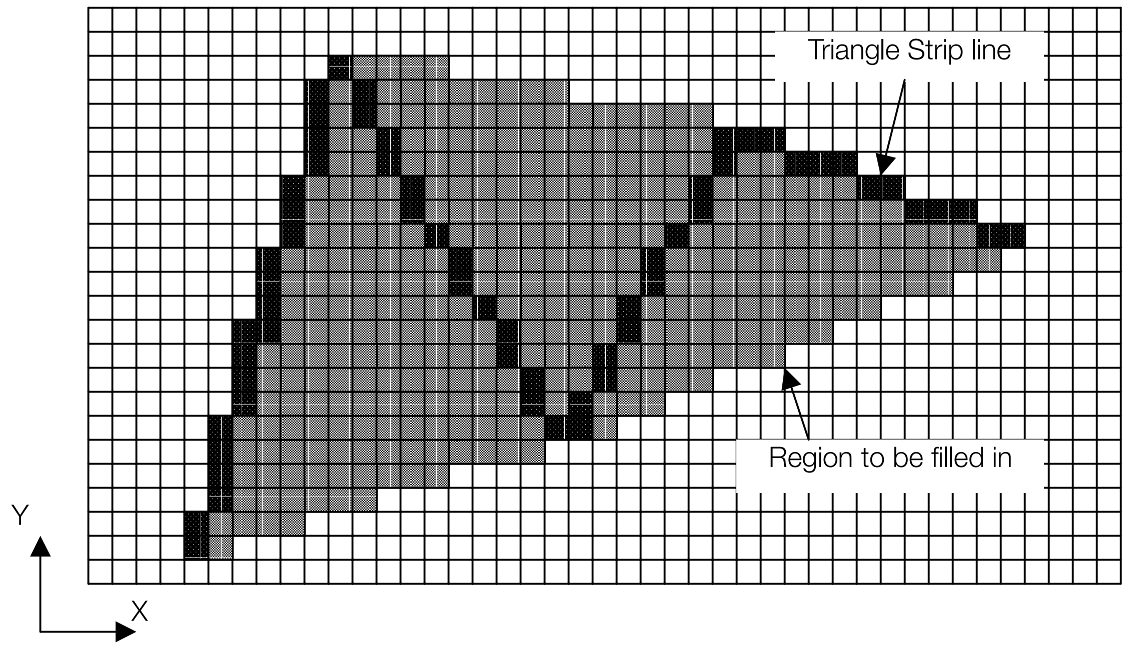


The above figure shows examples of silhouette lines and silhouette candidate lines of shapes. The vase-like shape on the right is covered by smaller Triangles. In three dimensions, a line that is the outline shape of a solid object regardless of the direction from which the object is viewed is called a silhouette line, and a line that is the outline shape according to the viewing direction is called a silhouette candidate line. A line having a low probability of being a silhouette line such as the diagonal of a rectangle is assumed not to be subject to drawing. This reduces the possibility of side effects occurring due to antialiasing drawing and also increases drawing speed.

STEP2

Draw Triangles and Triangle Strips with Z-buffer ON.

Figure 8

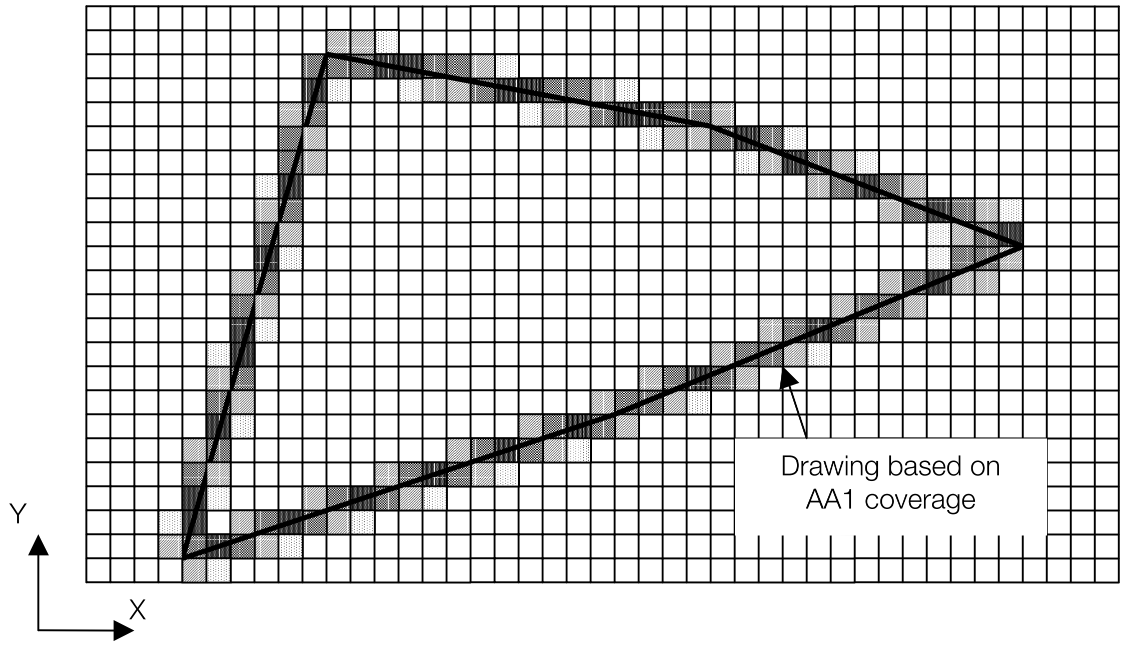


Polygons are painted out when aliasing occurs.

STEP3

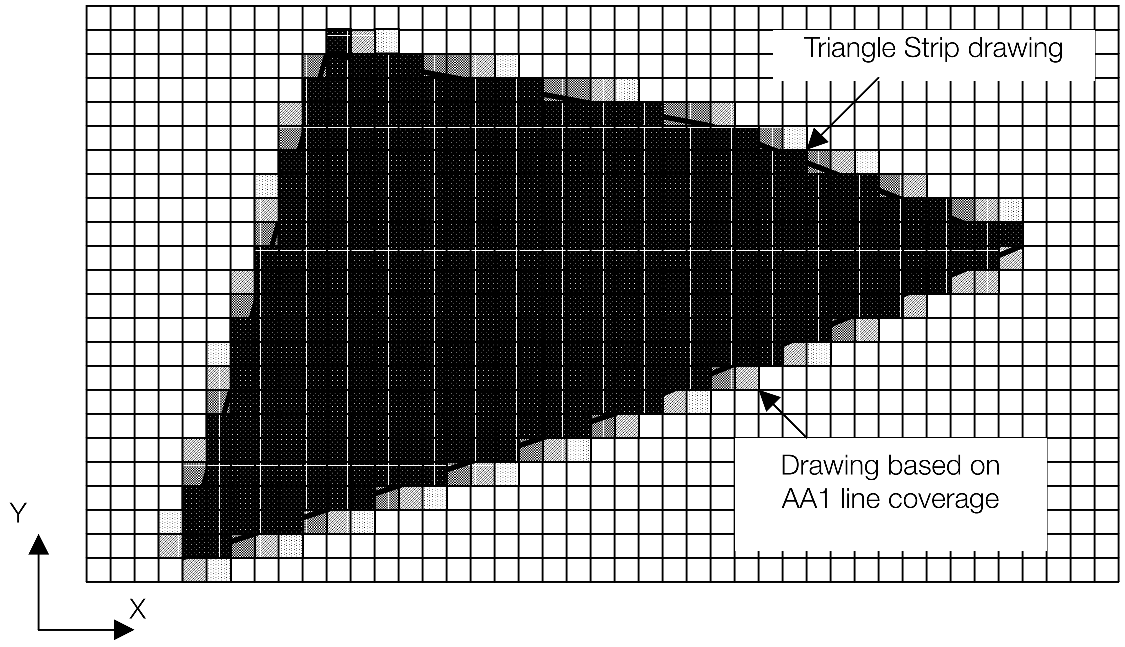
Draw silhouette lines and silhouette candidate lines if necessary by turning on the AA1 function and Z-buffer test. With the Graphics Synthesizer, when the AA1 function is turned on, only comparisons with the Z-buffer are performed, and writing is prohibited. The following figure shows PIXELs that are written when the AA1 function is turned on.

Figure 9



The above steps produce a figure for which antialiasing is applied to the edges as shown in the following figure.

Figure 10



The effects of refmap-onepass-AAline or texmap-onepass-AAline can be confirmed by toggling the L1 button to turn AA1 line on and off.

Also, with mountain, AA1 line is turned on by pressing L2 while holding down select and turned off by pressing R2 while holding down select.

Regardless of whether the 640x224 buffer is used, a high-quality image with little flicker can be obtained.

aa1-feather

Several antialiasing methods are implemented in aa1-feather.

These are explained below.

- Aliasing is eliminated by alpha blending the preceding field's frame (display frame) and adding it to the current drawing frame after the current drawing frame is written.

  L1 button: Alpha blended drawing with original brightness unchanged

  L2 button: Alpha blended drawing with brightness increased

  (If L1 and L2 are turned on at the same time, the image is skipped.)

  The program implements this by using copy_frame, copy_frame1. and the like. The display frame is drawn by alpha blending it in the drawing frame as texture. When speed must be increased, the sprites to be drawn are drawn 10 times at 64x224 resolution, not at 640x224 resolution. (The GS D-RAM page break frequency is decreased.) The drawing speed is approximately twice as fast.

- Processing is performed for eliminating AA1 side effects.

  When AA1 is applied in aa1-feather, processing with no side effects is always executed. Although no triangles or lines are displayed when aa1-feather is started up, the number of triangles or lines that are displayed is increased by continuously pressing the Circle button.

  a. When using AA1 to draw triangles

     The drawing object can be switched to triangles and lines by pressing the R2 button. Also, AA1 can be turned on and off by pressing the R1 button.

     If AA1 is used when drawing triangles, the triangles are corrupted on the screen (when the surface direction is perpendicular to the line of sight direction, AA1 should be turned off).

     This calculation is implemented by adding the normal line direction as data for a triangle at a time and transferring it to VU1, calculating the inner product of the line of sight vector and normal line in VU1, and turning off AA1 when that value is less than or equal to a threshold value. AA1 is turned on or off by sending a PRMODE command for each vertex.

     The calculations are summarized below.

     1. Calculate the vertex coordinates in the screen coordinate system for VF05.
     2. Calculate the polygon normal line in the screen coordinate system for VF06.
     3. Calculate the inner product VF05*VF06 and save it in VF29x.
     4. Substitute the distance from the viewpoint to the vertex by using VF05z and save it in VF08x.
     5. Calculate VF29x*8.0f - VF08x.
     6. In VF07, prepare a command that turns PRMODE AA1 ON or OFF according to the sign.
     7. Turn AA1 ON or OFF according to PRMODE each time one vertex is drawn.

     When the cosine of the angle between the line of sight vector and surface normal line is 1/8 or less, turn AA1 off.

b. When using AA1 to draw lines

When drawing AA1 lines, values cannot be written in the Z-buffer.

To prevent this, line drawing is divided into two steps. An AA1 line for which Z-buffer processing was performed can be drawn by first applying a mask to RGBA and drawing the line for which AA1 was turned off only in the Z-buffer and then turning AA1 on and drawing the AA1 line by only performing Z comparison.