OpenGL - Anti Aliasing

CS475 - Computer Graphics

Sumair Ahmed
IIT Bombay

September 8, 2009

1 Introduction

Rasterization is inherently a discrete sampling process, since pixels (or raster elements) are discrete. The pixel size determines an upper limit to the amount of detail that can be displayed. This results in sampling artifacts in the rendered images which are called aliasing.

Aliasing is a common problem that arises whenever a continuous signal is reconstructed from a discrete set of samples.

2 Nyquist Rate

The **Sampling Rate or Sampling Frequency** defines the number of samples per second (or per other unit) taken from a continuous signal to make a discrete signal.

The **Nyquist-Shannon sampling theorem** states that perfect reconstruction of a signal is possible when the sampling frequency is greater than twice the maximum frequency of the signal being sampled. If lower sampling rates are used, the original signal's information may not be completely recoverable from the sampled signal.

Nyquist rate is the minimum sampling rate required to avoid aliasing, equal to twice the highest frequency contained within the signal.

$$f_N = 2B$$

where B is the highest frequency at which signal can have nonzero energy. To avoid aliasing, Sampling rate must exceed the Nyquist rate.

$$f_s > f_N$$

From Figure 1, You can see that a low frequency signal (sinusoid curve in example:a) is reconstructible. Both the curves(solid & Arrow) look similar. This is because the Sampling rate (1/t) is far greater than the signal frequency and follows the nyquist rate.

However in the case of high frequency sinusoid signal (example:b), the dotted curve is different from the actual signal. This is due to the low sampling rate (1/t) compared to the signal frequency. And so the curve cannot be reconstructible with this sampling rate. In general, any frequency component above $f_s/2$, is indistinguishable from a lower-frequency component . This phenomenon is known as **Aliasing**.

There are two methods that can be done to prevent or reduce aliasing:

- Increase the sampling rate, to above the twice of the frequencies that are aliasing.
- Introduce an anti-aliasing filter that restricts the bandwidth of the signal to satisfy the condition for proper sampling.

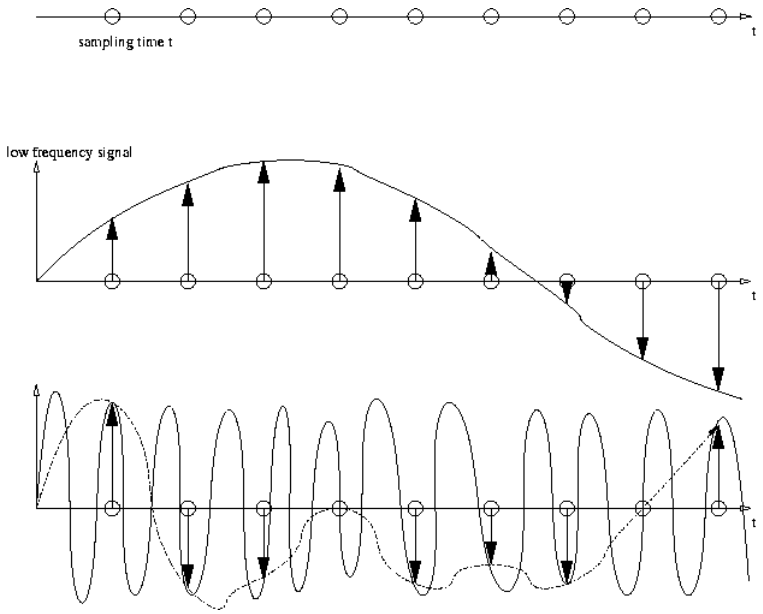

Figure 1: Nyquist Shannon Sampling Theorem: (a)low frequency signal (b)high frequency signal

3 Aliasing

The OpenGL renders a line/object as a series of pixels that lie on the pixel grid. Due to this approximation the Diagonal lines (as shown in the *Figure* 2) appear jagged. The borders of such an object remains square edged instead of smooth edges and such situation is called Aliasing.

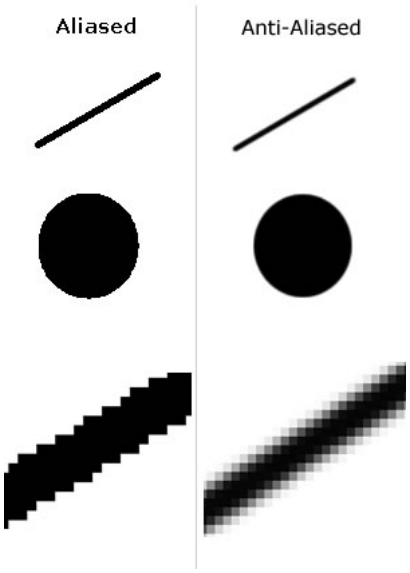

Figure 2: Benefits of Anti-Aliasing

OpenGL describes few Anti-Aliasing techniques to reduce the jaggedness and smooth the edges.

- Line and Point Anti-Aliasing
- Polygon Anti-Aliasing

- MultiSampling
- Anti-Aliasing with Textures
- Anti-Aliasing with Accumulation Buffer

3.1 Line, Polygon Anti-Aliasing

$$glEnable\,(GL\_POINT\_SMOOTH)\,;$$
$$glEnable\,(GL\_LINE\_SMOOTH)\,;$$
$$glEnable\,(GL\_POLYGON\_SMOOTH)\,;$$

The above function *glEnable* can enable many features of OpenGL; the feature you want to enable is provided as an input parameter. Enabling these features would anti-alias the points, lines and polygons respectively. However, line smoothing is by default enabled. With these enabled features, OpenGL computes an alpha value that represents the fraction of each pixel covered by the line or point as a function of the distance of the pixel center from the line center. It also gives an option to mention the amount of Quality desired. This can be done by calling *glHint*. Any of the following hint parameters can be given:

$$glHint(\ GL\_POLYGON\_SMOOTH\_HINT, GL\_NICEST\ );$$

- $GL\_FASTEST$: Most efficient option is chosen
- GL_NICEST: Best Quality option is chosen
- GL_DONT_CARE: No preference

Most of the polygons are represented as a set of triangles. With Polygon Smoothing, every triangle is anti-aliased against the next. This causes pixels on the edges of the polygon to be assigned fractional alpha values based on their pixel coverage. The Quality of the coverage value is controlled with $GL\_POLYGON\_SMOOTH\_HINT$. As these antialiased points are a form of transparent primitives, it requires blending to be enabled.

$$glEnable\,(GL\_BLEND)\,;$$
$$glBlendFunc\,(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA)\,;$$

Unfortunately the fact that these techniques use blending makes the rendering order-dependent. That is, the order in which the primitives are drawn affects the final result.

3.2 Using Textures

Antialiasing of points and lines can also be done using the filtering provided by texturing. Create an image of a circle where the alpha values are one in the center and go to zero as you move from the center out to an edge. This would then be used to blend the point with the pixel values already in the framebuffer.

For example, to draw antialiased points, create a texture image containing a filled circle with a smooth (antialiased) boundary. Then draw a textured polygon at the point making sure that the center of the texture is aligned with the point's coordinates using the texture environment $GL\_MODULATE$. This method has the advantage that any point shape may be accommodated simply by varying the texture image.

3.3 The Accumulation Buffer

The Accumulation buffer is the widely used technique for anti-aliasing. It can be used to antialias a scene without having to depth sort the primitives before rendering. A supersampling technique is used where the entire scene is offset by small, subpixel amounts in screen space, and accumulated. There are a number of effects that can be achieved if you can draw a scene more than once. You can do this by using the accumulation buffer.You can request an accumulation buffer with:

$$glutInitDisplayMode(...|GLUT\_ACCUM)$$

To perform Scene Antialiasing, first clear the accumulation buffer and enable it for reading and writing. GL_Accum reads each pixel from the buffer currently selected for reading and multiplies the R,G,B,and alpha values by $1/n$ and adds the result to the buffer. Scene gets drawn multiple times with slight perturbations(jittering), so that each pixel is the average of the images that intersect it. Finally, GL_Return copies the values into the color buffer for viewing.

$$glClearAccum(r, g, b, alpha)$$
$$glClear(GL\_ACCUM\_BUFFER\_BIT)$$
$$glAccum(GL\_ACCUM, 1/n)$$
$$glAccum(GL\_RETURN, 1.0)$$

3.4 MultiSampling

Multiple samples per pixel are taken in various ways and these are blended together to give the resulting pixel. Instead of using alpha for coverage, coverage masks are computed to help maintain sub-pixel coverage information for all pixels. Current implementations support four, eight, and sixteen samples per pixel. This method is available as an OpenGL extension.

This method supports Full Scene Anti-Aliasing(FSAA), since it prevents the aliasing on full-screen images as well. FSAA refers to blurring the edges of each polygon in the scene as it is rasterized in a single pass. With Multisampling, all subsamples of a pixel are updated in one pass. Though MultiSampling provides a modest performance, it uses a substantial amount of storage space (since sub-pixel samples of color, depth, and stencil need to be maintained for every pixel).

By setting the appropriate environment variable, you can enable full-scene antialiasing in any OpenGL application on these GPUs. Several antialiasing methods are available and you can select between them by setting the GL_FSAA_MODE environment variable appropriately. Note that increasing the number of samples taken during FSAA rendering may decrease performance.

Such OpenGL extensions allow to discover and enable FSAA from the application. It is possible to control FSAA with modern graphic cards. On Linux you can control this feature by setting an environment variable. FSAA is supported by most recent graphic cards.[7]
Settings for nVidia graphics card on windows is shown below.

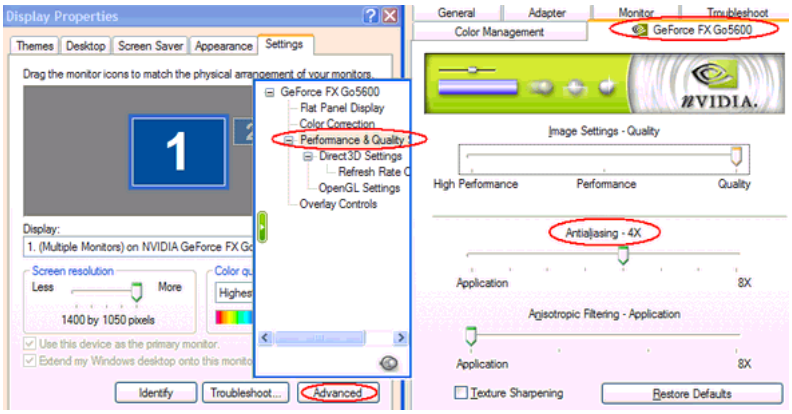

Figure 3: Anti-Aliasing settings on Windows [4]

4 References

1. MultiSampling - *http* : *//www.opengl*3.*org/wiki/Multisampling*
2. nVidia MultiSample - *http* : *//developer.nvidia.com/object/gdc_ogl_multisample.html*
3. OpenGL Programming Guide - *http* : *//glprogramming.com/red/chapter*06.*html*
4. OpenGL/Antialias - *http* : *//www.jsoftware.com/jwiki/OpenGL/Antialias*
5. OpenGL MultiSampling - *http* : *//nehe.gamedev.net/data/lessons/lesson.asp*?*lesson* = 46
6. Antialiasing - *http* : *//www.opengl.org/resources/code/samples/sig*99/*advanced*99 */notes/node*119.*html*
7. OpenGL Environment Variable Settings - *http* : *//us.download.nvidia.com/XFree*86 */Linux* − *x*86/185.18.29/*README/chapter* − 11.*html*