Lecture 5: Texture and Anti-aliasing

So far, we have considered objects to be made up of points and lines. If we project and draw these the result may be useful for some purposes, such as engineering drawing. However, for many applications the result would be a confusion of lines. To make any sense of the scene we need to draw it such that the faces of our polyhedral objects are opaque. This would eliminate all the lines that the eye cannot see. The simplest way to achieve this is called the painter's algorithm. What we do is sort all the polygons into depth order moving away from the viewpoint. We then draw them, furthest first, but instead of just drawing the edges, we draw a filled polygon for each. Thus, as the nearer polygons are drawn they replace the hidden parts of the scene. The same effect can be achieved by using what is called a Z-Buffer. This is simply an array, the same size as the screen. For each pixel a record is kept of the distance of the point in the scene that is visible at that pixel. Before the scene is drawn the Z-Buffer entries are all set to a value greater than the back clipping plane (ie further than the furthest object in the scene). When a polygon is to be drawn each pixel is checked. If the polygon is nearer than the Z-Buffer value, the pixel is set. If not the pixel remains unchanged.

The painter's algorithm and the Z-Buffer produce the same effect, but they have different implementation characteristics. The painter's algorithm requires the objects to be sorted, whereas the Z-Buffer does not. However the Z buffer needs checking for every pixel. In practice, the Z-Buffer is implementatble in hardware, and this means that it can achieve greater speeds than the painter's algorithm, and so is preferred in most graphics systems.

Texture

One simple and highly effective way of increasing the realism of any polygon based scene is to use a texture, rather than a plain colour, when filling in the polygons. Texturing is essentially a mapping between the texture space, and the polygon. In the simplest case, we are mapping one rectangular space into another as shown in Diagram 7.1.

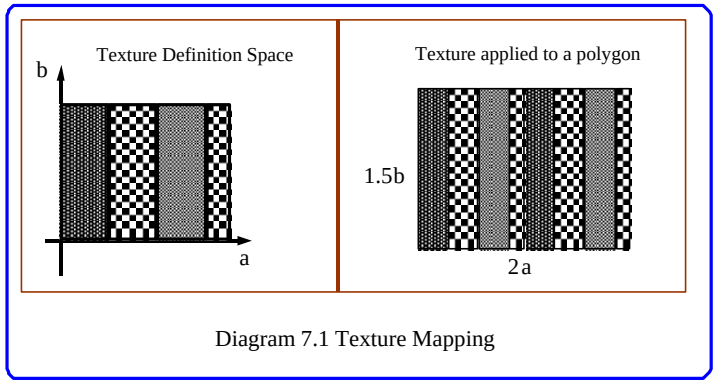

Diagram 7.1 Texture Mapping

In general however, we need to map the texture on a general quadrilateral, and this is done by bi-linear interpolation. This is simply illustrated using vectors. As shown in Diagram 7.2 any point in the quadrilateral can be expressed in terms of the edge vectors as:

$$\mathbf{p} = \alpha\mathbf{a} + \beta\mathbf{e}$$

where α and β are in the range [0..1]

$$\mathbf{e} = \mathbf{b} + \alpha(\mathbf{c} - \mathbf{b})$$

so $$\mathbf{p} = \alpha\mathbf{a} + \beta\mathbf{b} + \alpha\beta(\mathbf{c}-\mathbf{b})$$

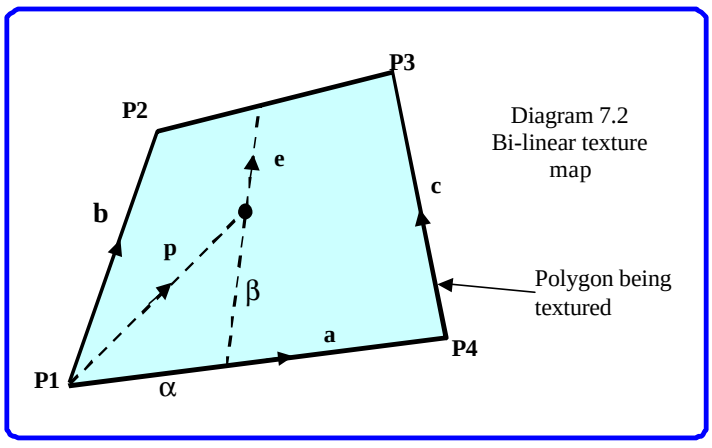

Diagram 7.2
Bi-linear texture map

Given a pixel position within the four projected corners of the quadrilateral, we can write down the edge vectors in device coordinates, and solve for α and β, the required roots being in the range [0..1]. Finally we look up the value of the pixel at the point $[\alpha,\beta]$ in the texture space, or possibly we can compute the value form a texture function $f(\alpha,\beta)$. It will be noted that the equation for texture mapping is quadratic in nature. The unfortunate consequence of this is that

straight line textures in the (α,β) space will become curved when mapped onto the image space. We note however, that for orthographic projection (which is affine), rectangles in three dimensional space will project into parallelograms in the image space. For this case, we have that **b**=**c** and the texture equation reduces to:

$$\mathbf{p} = \alpha\mathbf{a} + \beta\mathbf{b}$$

Which has a simple linear solution. However, it will be noted that in animated games it is rarely possible to use orthographic projection. This is because frequently the scene has distant parts which must appear smaller. However, as long as the change in depth of the polygon is small compared to its distance from the user the distortion of the texture will not be noticeable. The real problems occur with large polygons in the foreground.

One important use of texture mapping is to incorporate photographic material in our polygon rendering system. This has the advantage of adding an enormous amount of detail at a relatively low computational cost. It can be done in cases where the viewpoint does not change significantly relative to the polygon. For example, this is true of an airport, where the distant sky scrapers of a town, and the mountains behind can all be represented with one texture map, taken straight from a photograph.

Alias Effects

There is however a major problem with texture mapping which is referred to as alias effects. Basically an alias effect is caused by under sampling, and is found to cause difficulties not only in two dimensional graphics, but also in sound recording and signal processing. The problem with texture can be illustrated by a simple example. Suppose our texture is simply vertical stripes as shown in diagram 7.3. For simplicity, let us suppose that we are mapping this texture onto a polygon which has projected onto a rectangle in the image. Now consider moving away from that polygon being textured. The number of pixels across will also decrease. It will be seen that as the number of pixels decreases below a certain limit (six in our example, there are not enough samples to represent the texture, and the results are unpredictable. The effect is particularly noticeable in animated systems where a small movement of the viewer may change the number of pixels in a distant polygon from three to four pixels causing a distant shimmering appearance. Alias effects in 2D also occur when lines are drawn on raster terminals. The effect is the jagged staircase appearance of lines at certain angles, as seen in diagram 7.4. As the samples are increased (i.e. a higher resolution screen is used) so the effect is reduced.

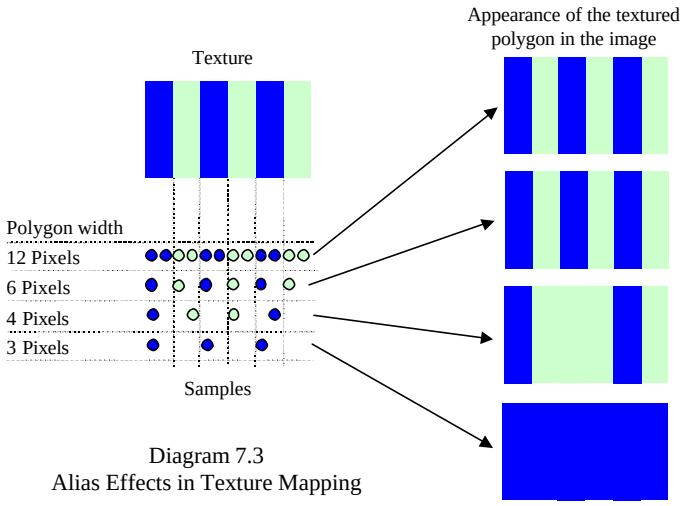

Diagram 7.3
Alias Effects in Texture Mapping

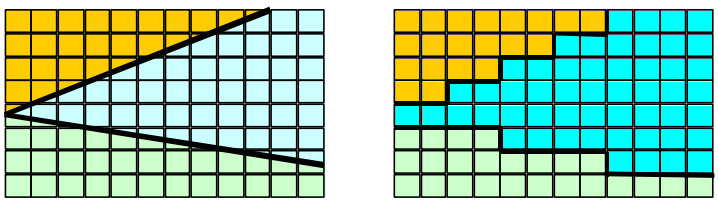

Diagram 7.4
Alias Effects at straight boundaries in raster images.

Anti-Aliasing

There are basically two ways to anti-alias a picture or a texture map. These are called super-sampling and low pass filtering.

The most effective is super-sampling in which we increase the

number of samples taken and average down the results. The super-sampling algorithm makes use of the property that alias effects reduce with resolution. If we are applying it to an image to reduce the effect of jaggies then the picture is computed at a higher resolution, say four times that of the display screen. Then each 4*4 block of pixels is averaged to get the actual pixel value to be set. Diagram 7.5 illustrates the approach for the boundary of a filled area. Although super-sampling will always work well when we are processing boundaries of fill areas, it does not prove satisfactory in cases where thin lines are drawn. The problem is illustrated by Diagram 7.6. Here it will be seen that only four of the super-samples in a block of sixteen corresponding to a real pixel will be set, thus the intensity of the whole line will be too low. Moreover, in the case shown, the alias effects will still be seen in the final image.

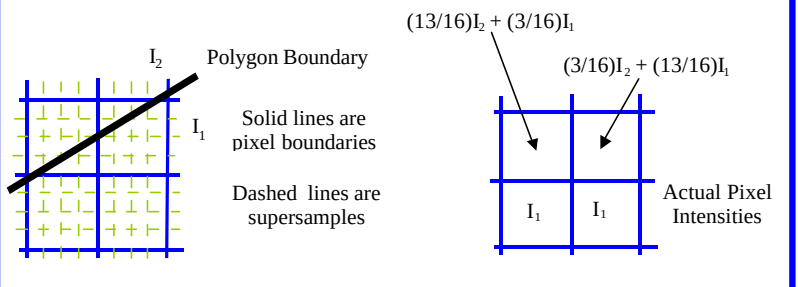

Diagram 7.5: Antialiasing by Supersampling

The disadvantage of super-sampling is that the computation of the higher resolution image will take longer. Much work has gone into ways of accelerating super sampling. For example, it is only necessary to super sample images where there are boundaries or a good deal of detail.

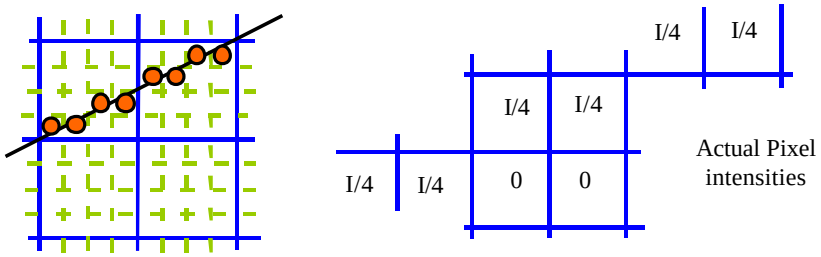

Diagram 7.6: Supersampling a thin line

The process is similar when applied to texture mapping. For each pixel in the texture map we take a set of samples (say 16) from the area of the pixel in the texture map and average them. Thus, the stripes in our example of diagram 7.3 will become just a blend of the two colours for distant low sampled polygons, and will not cause the disturbing shimmering appearance mentioned above.

Low pass filtering an image is in essence averaging it locally. The simplest way to achieve this might, for each pixel, to replace its value with the average of it and its eight neighbours. This however is not usually done since it can destroy too much of the information in the image. Instead a weighted average is taken, with the central pixel weighted the highest.

The most common way to do this is to use the following equation:

O(x,y):= I(x-1,y-1)*1/36 + I(x,y-1)*1/9 + I(x+1,y-1)*1/36 +
I(x-1,y) * 1/9 + I(x,y) * 4/9 + I(x+1,y) * 1/9 +
(x-1,y+1)*1/36 + I(x,y+1)*1/9 + I(x+1,y+1)*1/36

It is perhaps easier to see the process as placing a 3 by 3 mask over the pixel to be anti-aliased, and summing the product of the mask value and the pixel value.

1/36	1/9	1/36
1/9	4/9	1/9
1/36	1/9	1/36

The process is illustrated in diagram 7.7. The choice of filter values may look curious at first, however if we factor out 1/36 from the mask we get:

$$1/36 * \begin{matrix} 1 & 4 & 1 \\ 4 & 16 & 4 \\ 1 & 4 & 1 \end{matrix}$$

From which we see that high computational efficiency can be gained by carrying out most of the multiplies by shifting. With a little extra hardware the whole process can be reduced to little more than one divide per pixel. The mask is also a very close approximation to the well known Gaussian distribution function.

In essence all the low pass filtering does is blur the edges in the picture, while leaving the rest substantially the same. Although this may seem to destroy detail, it does have beneficial effects in enhancing visual appearance.

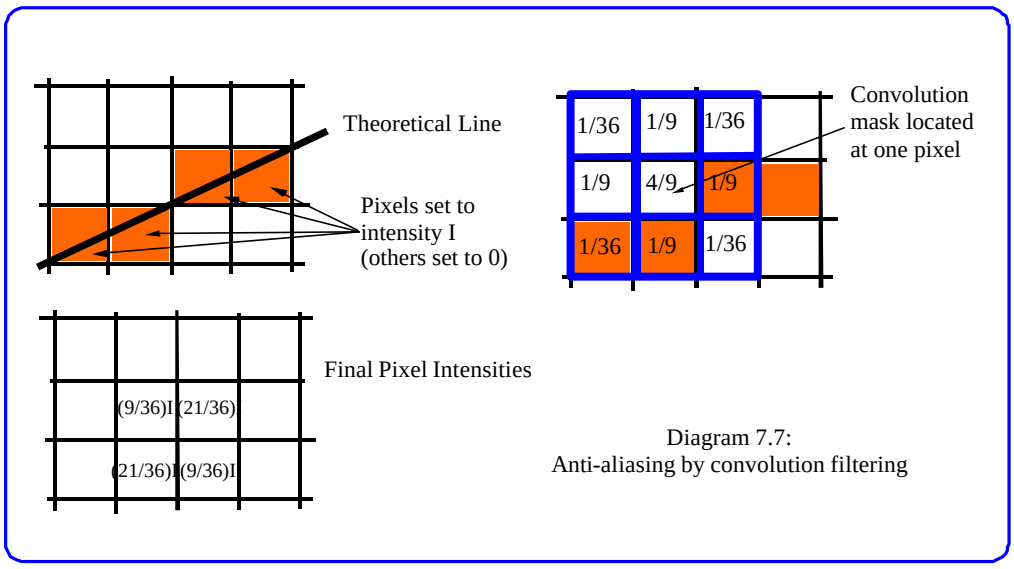

Diagram 7.7:
Anti-aliasing by convolution filtering