Algorithms for the Detection and Elimination of Specular Aliasing

John Amanatides

Dept. of Computer Science
York University
North York, Ontario
Canada M3J 1P3

Abstract

This paper introduces an algorithm that, given the geometry and surface characteristics of an object (the Phong highlight model is assumed), detects when specular or highlight aliasing is expected and indicates the correct sampling rate to eliminate it. This is accomplished by noting the geometric properties of the surface, the direction and distances to the eye and light sources, and the specular shading parameters. Also, an auxiliary algorithm is presented that eliminates the specular aliasing without increasing the sampling rate. It accomplishes this by clamping the specular function parameters to values that will not introduce significant high frequency components.

Keywords

anti-aliasing, shading, highlights, image synthesis

Introduction

An area of research in computer graphics that continues to receive a lot of attention is anti-aliasing (Crow, 1977; Cook, 1986). Much work, however, has concentrated with the edges of the display primitives, noting, correctly, that the most noticeable aliasing artifacts would occur there. Unfortunately, this approach has ignored another form of aliasing, that introduced by the highlight or specular component of the shading function. For example, if we are rendering cylinders, the highlight down the center of the cylinder may be quite jagged if the surface is very shiny (Crow, 1981). These defects are also sometimes visible when bump mapping (Blinn, 1978). The standard approaches of anti-aliasing in computer graphics will not correctly handle specular aliasing. This paper will first review aliasing and then three earlier attempts to solve the specular aliasing problem. It then introduces an algorithm for detecting specular aliasing plus a new method of specular anti-aliasing, along with results produced using this new method.

Background

Aliasing is possible when the two dimensional image function $I(x, y)$, representing the image on the viewing screen, contains frequencies greater than those that can be faithfully reproduced on the display device. If we sample this function, the resulting pixel values will not be correct. Aliasing is manifested in the form of jagged edges, distortions of small objects, inconsistencies in areas of complicated detail and in animated sequences. To remove these errors, several techniques have been advanced (Crow, 1977). We can sample the image function at higher resolution. This expensive process can diminish the problem but will not eliminate it. The best approach is to filter $I(x, y)$ before we sample, and there are strong theoretical reasons for doing this (Pratt, 1978).
Idealy, if we perform the following convolution

$$I'(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} I(\alpha, \beta)\, H(x-\alpha)\, H(y-\beta)\, d\alpha\, d\beta$$

with the appropriate sampling filter H

$$H(u) = \frac{\sin(\omega_0 u)}{\pi u}$$

we can generate $I'(x, y)$, a function that is identical to $I(x, y)$ except that it contains no frequency components greater than ω_0 (ω_0 would be dependent on the output device resolution).

Unfortunately, this convolution is impractical; it is too expensive. Consequently, simplifications have been used in computer graphics to compute the convolution in reasonable time. The major simplifications are simpler sampling filters and capitalization of the coherence properties of the image function.

Let us assume that the object is modeled with polygons. The image function approximation begins with the observation that as we sample along a polygon the signal will change relatively slowly. It is only when we cross polygon boundaries that great

fluctuations in intensity will occur. Thus we can assume that within a pixel the intensity of each polygon is constant. This implies we have to calculate the shade of each polygon within a pixel only once, a great saving in computation. Consequently, the area and position that a polygon covers within a pixel along with just one shade computation for that polygon is enough to calculate that polygon's contribution to the intensity function at that pixel.

Over the last decade there have appeared many papers indicating solutions to the aliasing problem that are similar to the approach outlined above (Crow, 1977; Catmull, 1978; Crow, 1981; Crow, 1982; Carpenter, 1984; Duff, 1989; Tanaka, 1990) and, in general, they work very well. What most researchers have ignored, however, is that in regions where the surface curvature of an object is large, another form of aliasing becomes noticeable: specular aliasing. Consider Plate 1. It consists of two cylinders, one slightly tilted, each having a very shiny surface. A light source behind the viewer introduces a narrow highlight that is visible down the center of each cylinder. If we rotate the cylinder slightly, we see that the highlight breaks up, introducing aliasing artifacts. The high curvature causes the specular component of the shading function (the component most susceptible to orientation) to introduce high frequencies into $I(x, y)$, frequencies that cannot be represented by the current sampling rate. This aliasing cannot be removed by the traditional approaches to anti-aliasing because of the assumptions they make to simplify the low pass filtering. The problem does not occur at the border of the cylinder (where most algorithms expect it to be) but in the interior.

Previous Work

Solution 1

The first solution for performing specular anti-aliasing was outlined by Frank Crow (Crow, 1982). He performed this anti-aliasing by computing the specular component at a higher resolution in pixels where the surface normal changed significantly. The drawback to this approach was that the user had to manually define a "threshold curvature" which was then used to indicate when higher resolution highlights were required. As no information about the surface reflectance properties were included (the highlight is also a function of surface characteristics), this "threshold curvature" may not have been an accurate indicator of specular aliasing for a particular surface. Thus the user had to, at times, change the threshold and recompute the specular component if the aliasing was still noticeable (this manual approach would be very inconvenient in animations).

Solution 2

The second solution was reported by Lance Williams (Williams, 1983). It involves storing a spherical texture map representing the illumination from the scene. The specular shading computation is replaced by a look-up in the spherical texture map. Local surface curvature is used to compute how much of the texture map to integrate. Unfortunately this approach requires a great deal of storage for the texture map, cannot deal with varying surface shininess nor handle light sources that are at a finite distance.

Solution 3

A third, more limited solution, advocated by Saito, Shinya and Takahashi (Saito, 1989), for edges of planar surfaces, consists of developing special-purpose cylinder shaders and inserting thin cylinders over edges and drawing them as wire-frames.

Detecting Specular Aliasing

In this section we will derive a specular aliasing detection algorithm. The highlight function that we will use, $I = K_{spec}\cos^n(\alpha)$, is Phong's (Bui T. Phong, 1975). As the parameter n increases, the highlight becomes more concentrated and the surface appears more glossy. Let us look at the Fourier series of this function. Recall that every even periodic function with a period T can be represented by the cosine expansion (Hwei P. Hsu, 1970)

$$f(t) = \tfrac{1}{2}a_0 + \sum_{n=1}^{\infty} a_n \cos(n\omega_0 t), \qquad \omega_0 = \frac{2\pi}{T}$$

The cosine expansion of $\cos^n(\alpha)$ can be extracted from (Oberhettinger, 1973):

$$\cos^{2l}(\alpha) = 2^{-2l}(2l)! \sum_{n=0}^{l} \frac{\varepsilon_n \cos(2n\alpha)}{(l+n)!(l-n)!},$$

$$\varepsilon_0 = 1, \quad \varepsilon_n = 2, \quad n = 1, 2, 3, \cdots$$

$$\cos^{2l+1}(\alpha) = 2^{-2l}(2l+1)! \sum_{n=0}^{l} \frac{\cos((2n+1)\alpha)}{(l+1+n)!(l-n)!}$$

If we look at this Fourier transform of $\cos^n(\alpha)$ we see that the high frequency components increase as n does. In fact, the highest frequency is n radians. For a given n there is a sampling step size, $\Delta\alpha_n$ ($\Delta\alpha_n = \pi/n$), above which aliasing is inevitable. To detect specular

aliasing we must detect situations during rendering where we go above this sampling step size.

Consider the geometry of shading:

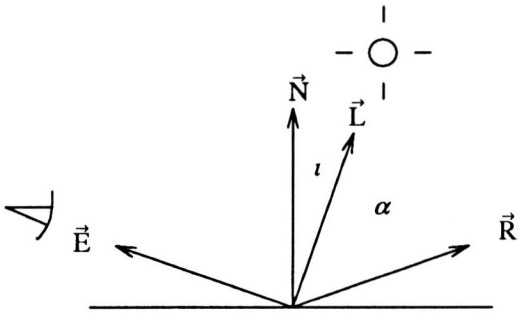

Figure 1

Here, $\vec{N}$, $\vec{L}$, $\vec{E}$ and $\vec{R}$ are unit vectors in the direction of the surface normal, light source, eye and reflected eye directions respectively. Changing any of $\vec{N}$, $\vec{L}$ or $\vec{E}$ will change α, the α of Phong's highlight function. If any of these change within a pixel when rendering, specular aliasing may occur (Williams, 1983). Luckily, unless the light source or eye are very close to the surface, their effect is minimal. The surface normal, however, has a more pronounced effect. It can change significantly within a pixel, thus noticeably affecting α. If the surface is shiny (a high value of n), the change in α ($\Delta\alpha$) within the pixel can exceed $\Delta\alpha_n$ (which is smaller for larger values of n) and aliasing results. Consequently, to detect specular aliasing, we must, within each pixel, determine how much the surface normal $\vec{N}$ changes and relate it to the maximum allowed by n. (Note: a change of $\vec{N}$ by Δ radians changes α by 2Δ radians).

The most straightforward way to compute how much the surface normal changes is to compute the normal at the corners of the pixel and find the pair which diverge the most (by computing the six dot products of the various pairs of normals and finding the minimum). The smallest of these six dot products, $dot_{\Delta\vec{N}}$, is compared to the smallest dot product allowed for the current value of n, $dot_{\Delta\alpha_n}$ (via a lookup on n into a pre-computed table). If it is greater than the value in the table, then no aliasing can occur. Otherwise, we have detected the occurrence of specular aliasing. We thus have derived a simple, analytic algorithm to detect specular aliasing on surfaces using the Phong highlight function. The only prerequisite is a good indicator of $\Delta\alpha$. Instead of using the normals at the corners of the pixel, another possibility would be to use the curvature of the surface and the size of intersection to compute $\Delta\alpha$.

The above approach of finding the how much the surface normal changes within a pixel has to be modified slightly when bump mapping is used. In this case, one can compute the change in normal when the bump map normal is generated and use this value instead.

The algorithm described above is conservative. Most of the energy in the power spectrum of $\cos^n(\alpha)$ is in the lower frequencies with very little near the high end. Consider the following figure:

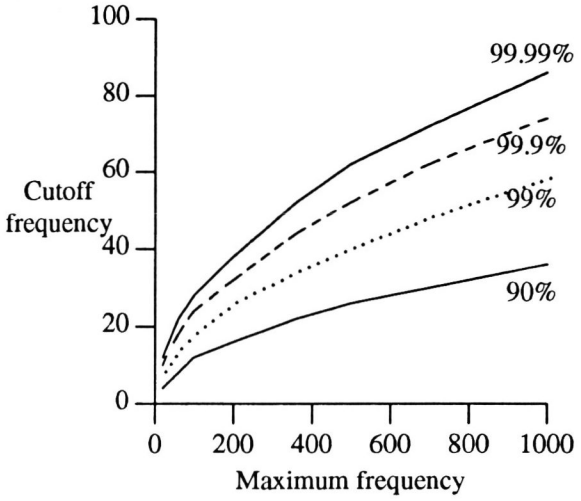

Figure 2

It is a frequency-frequency plot for several power levels. The abscissa indicates the maximum frequency in the power spectrum of $\cos^n(\alpha)$ (recall that the maximum frequency is n radians) while the ordinate represents the cutoff frequency below which a given percentage of the power of $\cos^n(\alpha)$ resides. For example, if n is 500, the maximum frequency in the power spectrum is 500 radians while 99.9 percent of the power is below 52 radians. Our table of minimum dot products could be computed more aggressively so that we don't have to perform specular anti-aliasing as frequently.

Once we have detected that specular aliasing is occurring we must be able to suggest a sampling rate that eliminates it. To do this we take the arc-cosine of the normal-pair dot product, $dot_{\Delta\vec{N}}$, (for efficiency we could perform a table lookup for the arc-cosine) to get $\Delta\alpha_{\Delta\vec{N}}$. We compare this with $\Delta\alpha_n$ and use the resultant ratio to indicate the oversampling rate.

Eliminating Specular Aliasing

We have just derived an algorithm for the detection of specular aliasing. Now we have to decide what

to do about it. Crow's standard solution for removing this aliasing can be used but it is unsatisfactory in that multiple shading computations per pixel per polygon are required; these are expensive. We will now derive an alternate solution that requires only one shading computation per pixel, a much more frugal approach.

The purpose of anti-aliasing is to remove the high frequencies in a signal that cannot be represented by the current sampling rate. If we do not want to change this sampling rate we must somehow change the signal so that the unrepresentable frequencies are absent. One way of accomplishing this low pass filter is described by Norton, Rockwood and Skolmoski in the work they did on texture mapping (Norton, 1982). Their original signal was constructed with a series of sine waves, each having a different frequency and phase angle. Their method of anti-aliasing was to clamp out the sine waves that were too high to be represented and only use the low frequency sine wave components. This approach inspired the specular anti-aliasing algorithm described below.

A different method to perform specular anti-aliasing involves clamping n to values that will not introduce aliasing. By replacing n by a smaller value, we guarantee that the new highlight function has no offending high frequencies. The new value of n, n', depends on the sampling rate $\Delta\alpha_{\hat{N}}$. What we are in fact doing is replacing the user specified highlight function with a duller one, one guaranteed not to alias. We only do this, however, in problem pixels so that in regions where no aliasing is occurring we use the original function.

The simple replacement of n' for n needs a little enhancement before it becomes the complete anti-aliasing algorithm. Replacing $\cos^{n'}(\alpha)$ for $\cos^{n}(\alpha)$ removes the high frequencies from the original signal but in the process it also boosts the lower frequencies, making the surface appear brighter than before. Some sort of normalization is in order. This normalization is encorporated in the algorithm if we multiply the new highlight function, $\cos^{n'}(\alpha)$, by the ratio of Maximum[n] to Maximum[n']. Maximum[] is an array that stores the value of the first (and largest) component in the signal $\cos^{n}(\alpha)$ for various values of n. By performing the above normalization we try to make sure that the overall level of the clamped signal is the same as the original signal. The effects of normalization is illustrated in Figure 6-3. It plots the ratio of the cosine components of the clamped signal to those in the original signal, $\cos^{160}(\alpha)$, for various values of n', starting at 20 and going to 140 in step sizes of 20. We see that in the low frequencies the original and clamped components are identical but as we go higher up in the frequency spectrum the cosine components in the clamped function quickly fall off.

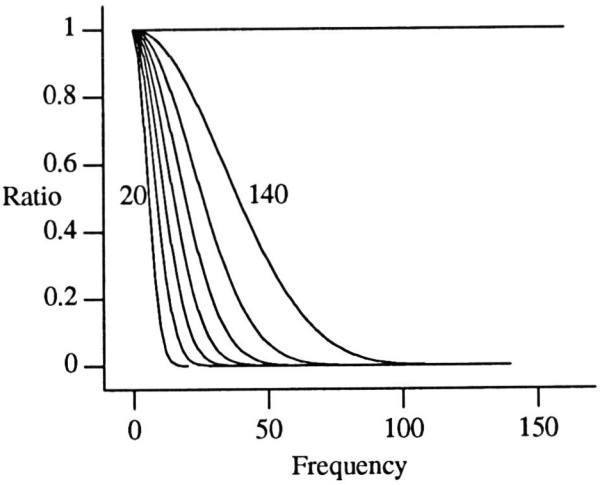

Figure 3

The resulting specular anti-aliasing algorithm is summarized in the code fragment below:

```
if(dot_ΔN̂ >= MinDot[n]) {
   /* no aliasing */
   n'= n;
   K_specular'= K_specular;
} else {
   Δα_N̂= 2*ArcCosine(dot_ΔN̂);
   sampleFrequency= π/Δα_N̂;
   n'= MaxnAllowed[sampleFrequency];
   K_specular'= K_specular*Maximum[n]/Maximum[n'];
}
```

$K_{specular}$ indicates the fraction of the light that the specular component contributes. The array Max$n$Allowed[] is required only if we are performing aggressive specular anti-aliasing. Otherwise, n' is assigned the value of sampleFrequency.

Results

The above algorithms for detecting and eliminating specular aliasing were implemented in a z-buffer rendering system. The z-buffer visible surface algorithm was chosen because it would guarantee that any anti-aliasing observed would have to come from the clamping algorithm. Adding the clamping algorithm to the z-buffer renderer was straightforward. The biggest implementation hurdle encountered was changing the tiler to have it compute $dot_{\Delta\hat{N}}$. The three

arrays, MinDot[], Maximum[] and Max$n$Allowed[], were pre-computed for efficiency. Plate 2 shows four cylinders, each with identical surface properties but different sizes and orientations computed to a resolution of 256 by 256 pixels. The large central cylinder and the one on the lower right exhibit severe specular aliasing. The cylinder on the upper left is also exhibiting specular aliasing even though none is visible at present. For if it is moved slightly, aliased highlights will appear on its surface. This is also true of the cylinder in the upper right. Plate 3 shows the same scene rendered using the highlight clamping algorithm. The highlight down the central and lower right cylinders are now smooth with no jaggies present. A highlight is faintly visible on the cylinder in the upper left. Now, even if the cylinder is moved, no highlight will flicker on and off. The cylinder on the upper right has no highlight visible as it is too faint. There is so much curvature in this cylinder that any visible highlight would cause aliasing due to the severe undersampling. Plate 4 shows the same scene rendered with more aggressive clamping (99.9 percent power). The highlights are brighter and not as spread out and aliasing is not noticeable. Plate 5 shows a scene with two cones with the one on the right being highlight anti-aliased. The cone shape is useful in that it illustrates a continuous transition from low curvature to high. As can be seen, the transition into the high curvature region is smooth when anti-aliased.

Conclusion

We have introduced a simple analytic algorithm that, given the change of α within the pixel, detects when specular aliasing is present and indicates a sampling rate to overcome it. We have also introduced a very fast and simple algorithm that removes specular aliasing without increasing the sampling rate.

I would like to thank Xerox PARC, and especially Frank Crow, for providing the facilities and support for much of this research project. Also, thanks to NSERC for their continuing support.

References

Blinn, 1978.
J.F. Blinn, "Simulation of Wrinkled Surfaces," *Computer Graphics,* 12(3), pp. 286-292 (August 1978).

Bui T. Phong, 1975.
Bui T. Phong, "Illumination for Computer Generated Pictures," *Comm. of the ACM,* 18(6), pp. 311-317 (June 1975).

Carpenter, 1984.
L. Carpenter, "The A-buffer, an Antialiased Hidden Surface Method," *Computer Graphics,* 18(3), pp. 103-108 (July 1984).

Catmull, 1978.
E. Catmull, "A Hidden-Surface Algorithm with Anti-Aliasing," *Computer Graphics,* 12(3), pp. 6-10 (August 1978).

Cook, 1986.
R.L. Cook, "Stochastic Sampling in Computer Graphics," *Trans. on Graphics,* 5(1), pp. 51-72 (January 1986).

Crow, 1977.
F.C. Crow, "The Aliasing Problem in Computer-Generated Shaded Images," *Comm. of the ACM,* 20(11), pp. 799-805 (November 1977).

Crow, 1981.
F.C. Crow, "A Comparison of Antialiasing Techniques," *IEEE Computer Graphics and Applications,* 1(1), pp. 40-48 (January 1981).

Crow, 1982.
F.C. Crow, "Computational Issues in Rendering Anti-Aliased Detail," *IEEE 1982 Spring COMPCON,* pp. 238-244 (1982).

Duff, 1989.
T. Duff, "Polygon scan conversion by exact convolution," *in Raster Imaging and Digital Typography, Edited by J. Andre, R. Hirsh, Cambridge Univ. Press; Proceedings of RIDT89 Intl. Conf., Lausanne, Switzerland,* pp. 154-168 (October 1989).

Hwei P. Hsu, 1970.
Hwei P. Hsu, "Fourier Analysis," Simon and Schuster, N.Y. (1970).

Norton, 1982.
A. Norton, A.P. Rockwood, and P.T. Skolmoski, "Clamping: A Method of Antialiasing Textured Surfaces by Bandwidth Limiting in Object Space," *Computer Graphics,* 16(3), pp. 1-8 (July 1982).

Oberhettinger, 1973.
F. Oberhettinger, "FOURIER EXPANSIONS: a collection of formulas," Academic Press, N.Y. (1973).

Pratt, 1978.
W.K. Pratt, "Digital Image Processing," Wiley-Interscience (1978).

Saito, 1989.
T. Saito, M. Shinya, and T. Takahashi, "Highlighting Rounded Edges," *CG International '89* (1989).

Tanaka, 1990.
T. Tanaka and T. Takahashi, "Cross Scanline Algorithm," *Eurographics '90,* pp. 63-74 (1990).

Williams, 1983.
L. Williams, "Pyramidal Parametrics," *Computer Graphics,* 17(3), pp. 1-11 (July 1983).

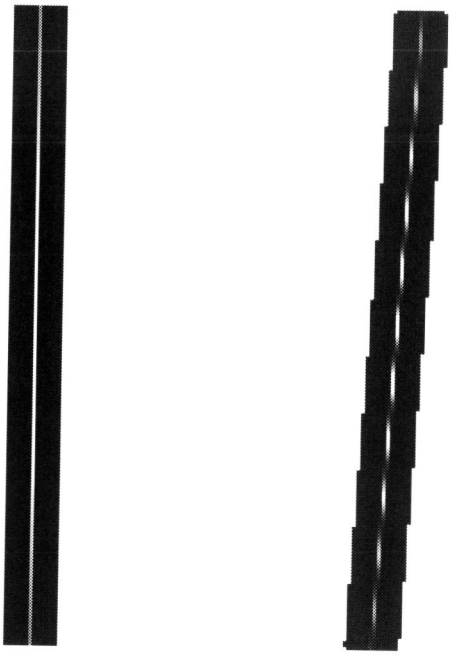

Plate 1

Plate 2

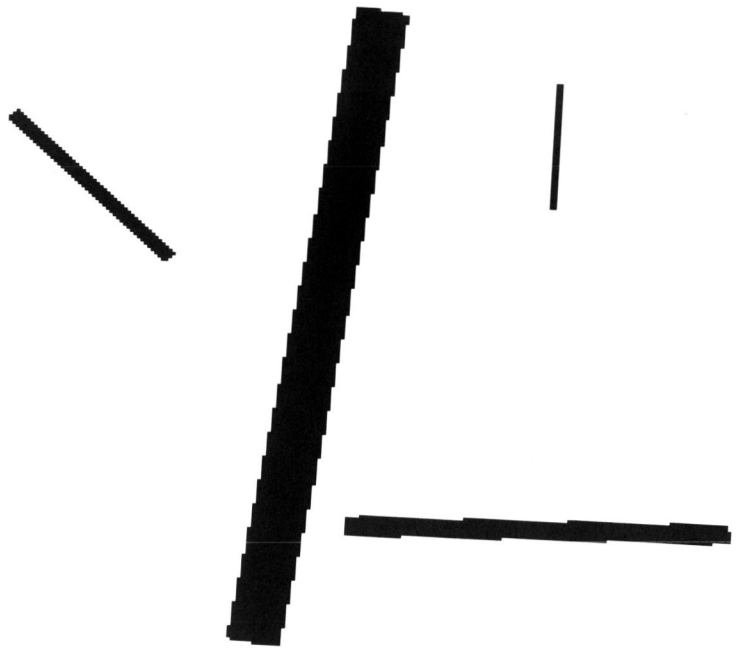

Plate 3

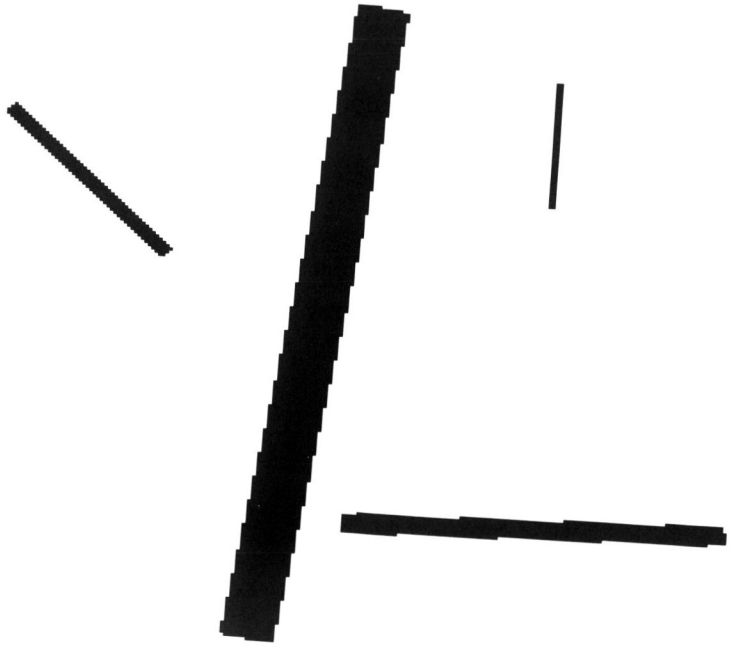

Plate 4

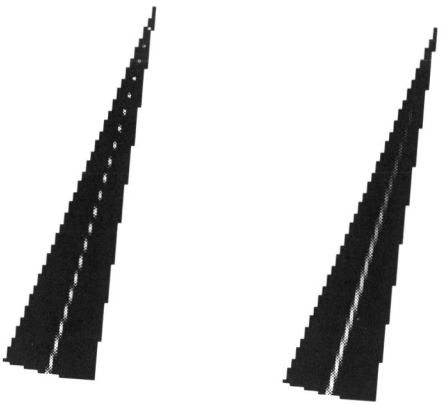

Plate 5