5.3 the split at k_2 is the last split and the final approximating curve $P' = [p_1, k_1, k_2, p_n]$ contains only three segments.

One interesting question that arizes concerning this algorithm is its time complexity in the worst case. It is not difficult to create examples of polygonal curves for which this a straight forward implementation of this algorithm runs in $O(n^2)$ time. However, with clever techniques from computational geometry one can reduce this complexity to $O(n \log n)$ time [HS92].

6. References

[AII90] Asano, A., Itoh, K. and Ichioka, Y., "The nearest neighbor median filter: Some deterministic properties and implementations," *Pattern Recognition,* vol. 23, No. 10, 1990, pp. 1059-1066.

[BB82] Ballard, D. H. and Brown, C. M., *Computer Vision,* Prentice-Hall, Inc., 1982.

[Br84] Brownrigg, D. R. K., "The weighted median filter," *Communications of the ACM,* vol. 27, 1984, p. 807.

[Cl65] Clemens, J. K., "Optical character recognition for reading machine applications," Ph.D. thesis, Dept. Elec. Eng., M.I.T., August 1965.

[DH73] Duda, R. O. and Hart, P. E., *Pattern Classification and Scene Analysis*, John Wiley & Sons, 1973.

[DP73] Douglas, D. H. and Peucker, T. K., "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *The Canadian Cartographer*, vol. 10, No. 2, December 1973, pp. 112-122.

[HS92] Hershberger, J. and Snoeyink, J., "Speeding up the Douglas-Peucker line-simplification algorithm," Tech. Rept. 92-7, Department of Computer Science, University of British Columbia, Vancouver, April 1992.

[KC36] Kasnr, E. and Comenetz, G., "Groups of multi-point transformations with applications to polygons," *Scripta Mathematica*, vol. 4, January 1936, pp. 37-49.

[Mc87] McMaster, R. B., "Automated line generalization," *Cartographica*, vol. 24, No. 2, 1987, pp. 74-111.

[Pa82] Pavlidis, T., *Algorithms for Graphics and Image Processing*, Computer Science Press, 1982.

[Ra72] Ramer, U., "An iterative procedure for the polygonal approximation of plane curves," *Computer Graphics and Image Processing*, vol. 1, 1972, pp. 244-256.

[Se82] Serra, J., *Image Analysis and Mathematical Morphology*, Academic Press, 1982.

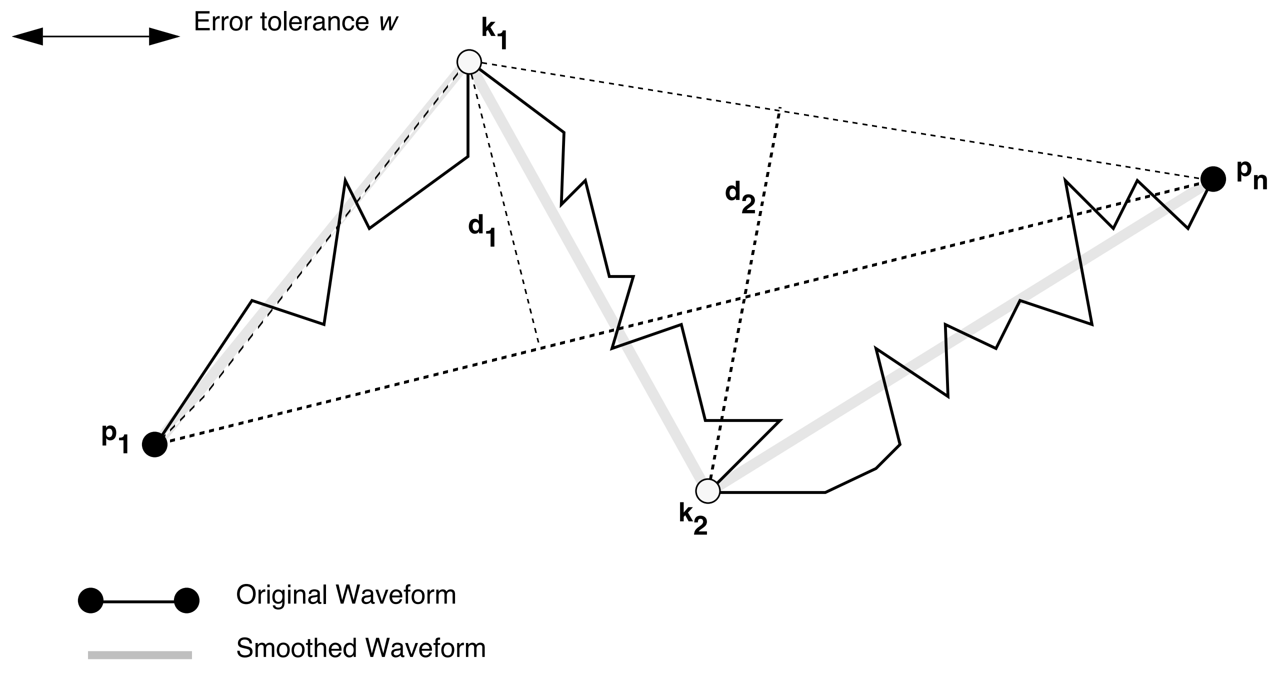


Fig. 5.3 Illustrating the Ramer-Douglas-Peucker algorithm.

mation problem in itself does not have smoothing as an explicit goal. However, most algorithms end up smoothing as a by-product and in some cases smoothing is actually explicitly incorporated [BB82], [Pa82].

5.3.1 Iterative end-points fit

We will describe in this section an elegant and simple algorithm favored by image processors [DH73] and cartographers [DP73], [Mc87]. The algorithm is called *iterative end-points fit* by Duda and Hart [DH73] but was independently discovered by Ramer [Ra72] in an image processing context and Douglas and Peucker [DP73] in a geographic information systems context and so it is often referred to in the literature as Ramer's algorithm or the Douglas-Peucker algorithm. We will refer to it as the Ramer-Douglas-Peucker algorithm.

Consider the polygonal curve $P = [p_1, p_2,..., p_n]$ and refer to Fig. 5.3. An error tolerance w is pre-specified which determines the amount of deviation we are willing to allow between the original curve and the approximation. The algorithm proceeds in a greedy fashion. First it tries to use only one segment $[p_1, p_n]$. A line $L(p_1, p_n)$ is drawn through p_1 and p_n. The vertex of P whose perpendicular distance to $L(p_1, p_n)$ is greatest is determined. Let this vertex be k_1 and the corresponding distance be d_1. If this distance is less than the error tolerance w we are finished. If not then we try approximating p with a curve of two segments by splitting the segment that failed the tolerance test at the furthest vertex, in this case k_1. We therefore obtain a candidate approximation $P' = [p_1, k_1, p_n]$. We test $[p_1, k_1]$ against $[p_1, p_2,..., k_1]$ for the error tolerance and discover it passes. We therefore abandon this portion of the curve and test $[k_1, p_n]$ against $[k_1,..., p_n]$ for the error tolerance of k_2. We discover that d_2 is greater than w and we split $[k_1,..., p_n]$ into two curves at k_2. We continue recursively in this way until each piece satisfies the error tolerance. In our example in Fig.

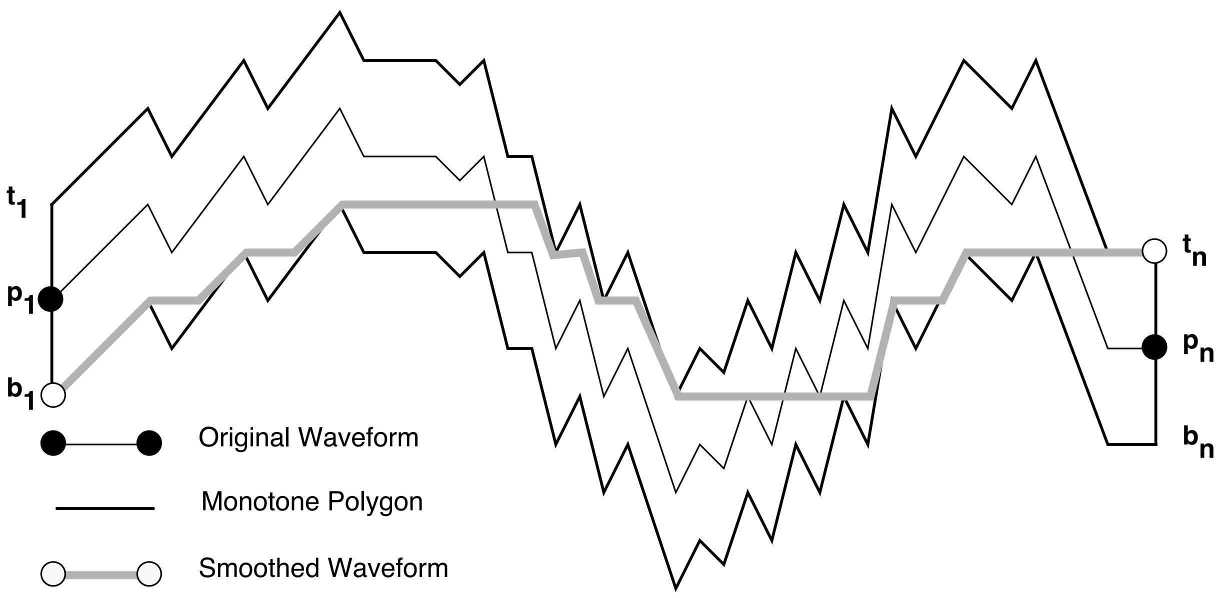


Fig. 5.2   Hysteresis smoothing produces the “water-fall” path.

negative y direction. Finally connect the line segments $[b_1, t_1]$ and $[b_n, t_n]$ to T and B to form a monotonic polygon $TB = [t_1, t_2,..., t_n, b_n, b_{n\text{-}1},..., b_1]$. We may now produce the smoothed waveform as follows. Arbitrarily choose either t_1 or b_1 as a starting point, say b_1. Now imagine rotating the polygon by 90 degrees so that b_1 is on top and TB is monotonic in the y direction. Also imagine b_1 provides a continuous supply of water and trace the path of the resulting waterfall in the “cave” TB. The path of the waterfall is precisely the resulting hysteresis-smoothed waveform. It is not difficult to design an O(n) time algorithm for computing the *waterfall* path of a monotone polygon.

Hysteresis smoothing is an efficient algorithm for smoothing monotone polygons and it is both conceptually simple and easy to program. However it has its limitations. For arbitrarily shaped polygons non-trivial modifications would have to be made. More seriously, if data reduction is very important hysteresis smoothing may not yield satisfactory results. Note that whenever the smoothed curve proceeds horizontally from a point x on the upper (lower) curve to a point y on the lower (upper) curve, it does so in one single line segment. Therefore all vertices of the original curve that lie between x and y are discarded. Although this holds for all horizontal segments in the final smoothed curve there are many other portions of this curve which are identical copies of portions of the upper and lower curves and therefore may contain many vertices. Clearly hysteresis smoothing does not embody data reduction as its primary goal. In the next section we consider the problem of data reduction as a primary goal.

5.3   Polygonal Approximation

The polygonal approximation problem is concerned with receiving as input a simple polygon or a polygonal chain P such as that obtained from a waveform consisting of n vertices, where n is usually very large, and obtaining a new polygon or polygonal chain P’ that has m vertices such that m is much smaller than n and yet P’ is a “good” approximation of P. The polygonal approxi-

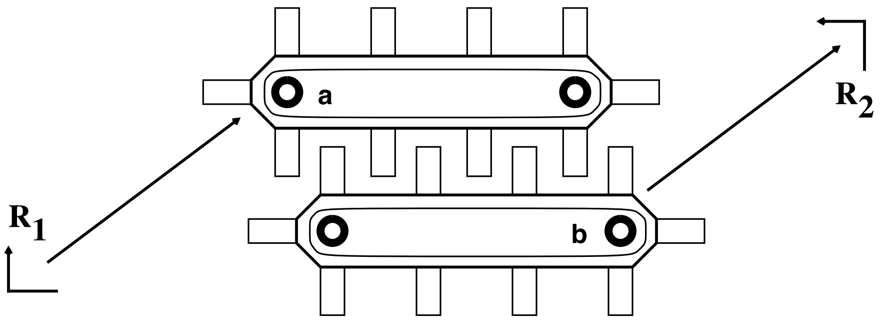


Fig. 5.1 A mechanical-gear implementation of hysteresis smoothing.

tions we consider algorithms that smooth and perform data reduction.

5.2 Hysteresis Smoothing

In this section we describe an elegant, widely applicable and practical method for smoothing polygonal waveforms that also performs data reduction as a by-product and was first introduced into the field of pattern recognition by J. K. Clemens at M.I.T. in 1965 [Cl65] in the context of designing a reading machine for the blind that could handle different machine typed fonts. It is called *hysteresis smoothing*.

Before explaining the algorithm for polygonal curves it is instructive to consider how hysteresis smoothing is done mechanically with gears. Assume that we want to transfer rotational motion from one location in a machine to another via connecting rods. We have a rod *a* that receives "signals" in the form of successive clockwise and counter-clockwise rotations and we would like to transfer these motions to another rod *b*. However, we want to ensure that only significant rotations are transferred. In other words small perturbations in rotations due to vibrations in the machine and other "noise" should not be transferred to rod *b*, they should be "filtered-out" or smoothed away. How do we connect the two rods to accomplish this task? The answer is simple and illustrated in Fig. 5.1. R_1 is a rod connected to sprocket *a* of the upper sprocket-wheel-chain system. R_2 is a rod connected to sprocket *b* of the lower sprocket-wheel-chain system. Both chains have "teeth" sticking outwards with a suitable gap between each pair of consecutive teeth. This gap is the smoothing tolerance parameter. If R_1 rotates by a suitably small amount the teeth of the upper sprocket-wheel-chain system do not engage the teeth of the lower sprocket-wheel-chain system and R_2 does not respond. On the other hand, if R_1 is rotated, say in a clockwise direction, by a sufficiently large amount the teeth will engage and a resulting counter-clockwise rotation of R_2 will result.

Consider now how we can interpret hysteresis smoothing of a polygonal curve. Accordingly let $P = [p_1, p_2,..., p_n]$ denote the polygonal chain monotonic in the x direction and refer to Fig. 5.2. First pick a threshold value v, analogous to the gap between the teeth in the sprocket-wheel-chain system described above, which will determine the amount of smoothing done. Next let $T = [t_1, t_2,..., t_n]$ denote the polygonal chain monotonic in the x direction which is an exact copy of P but translated by $v/2$ in the positive y direction. Similarly, let $B = [b_1, b_2,..., b_n]$ denote the polygonal chain monotonic in the x direction which is an exact copy of P but translated by $v/2$ in the

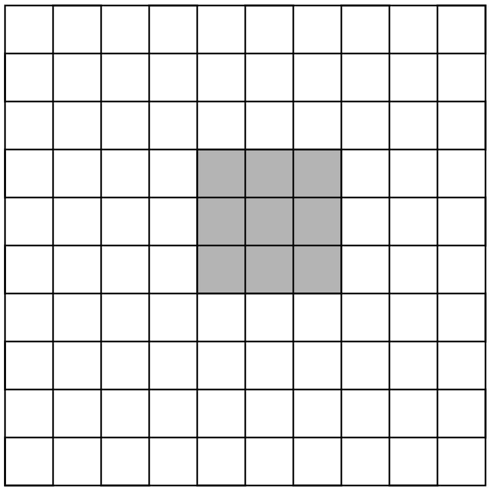


Fig. 3.1 *Median filtered* image of pattern in Fig. 2.1 (a).

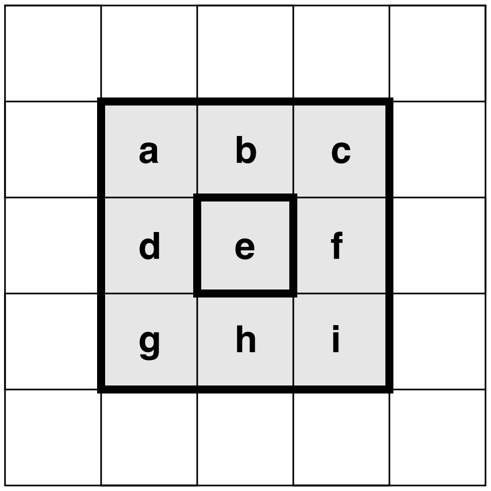


Fig. 4.1 A 3x3 window operator used in *logical* smoothing.

ation by the following Boolean expression:

$$e' = (\sim e) \otimes (a \otimes b \otimes c \otimes d \otimes f \otimes g \otimes h \otimes i) \oplus$$
$$e \otimes (a \oplus b \oplus c \oplus d \oplus f \oplus g \oplus h \oplus i)$$

where ~, $\otimes$ and $\oplus$ denote the Boolean operators *not*, *and* as well as *or*, respectively.

5. Polygonal Smoothing and Approximation

5.1 Midpoint Smoothing

In Chapter 1 we saw a surprisingly simple algorithm for smoothing polygons: the *mid-point* algorithm. The mid-point algorithm creates a *new* polygon by successively joining the mid points of the edges of the *old* polygon in the order in which they appear. This operation when performed a sufficient number of times yields a progressively smoother polygon. If the polygon has *n* vertices and the procedure is run *k* times the algorithm clearly takes O(*nk*) time and is trivial to program. In situations where this algorithm is appropriate *k* may be a small constant and hence the algorithm would be quite efficient in practice. However, in many applications where the “noisy” portions of the polygons that must be cut off are significantly larger than the average edge length of the polygon the mid-point algorithm may not be satisfactory. Furthermore, in many applications it is desirable not only to perform smoothing but also *data reduction*. In other words the smoothed polygonal curve should have fewer vertices than the original curve. Since the mid-point algorithm yields smoothed curves that contain exactly the same number of vertices as the original curves, it is completely useless for this purpose. We close discussion on this algorithm by mentioning that the mid-point algorithm has implications other than practical. Indeed, mathematicians have been interested in studying some surprising properties of these smoothed polygons [KC36]. In the following sec-

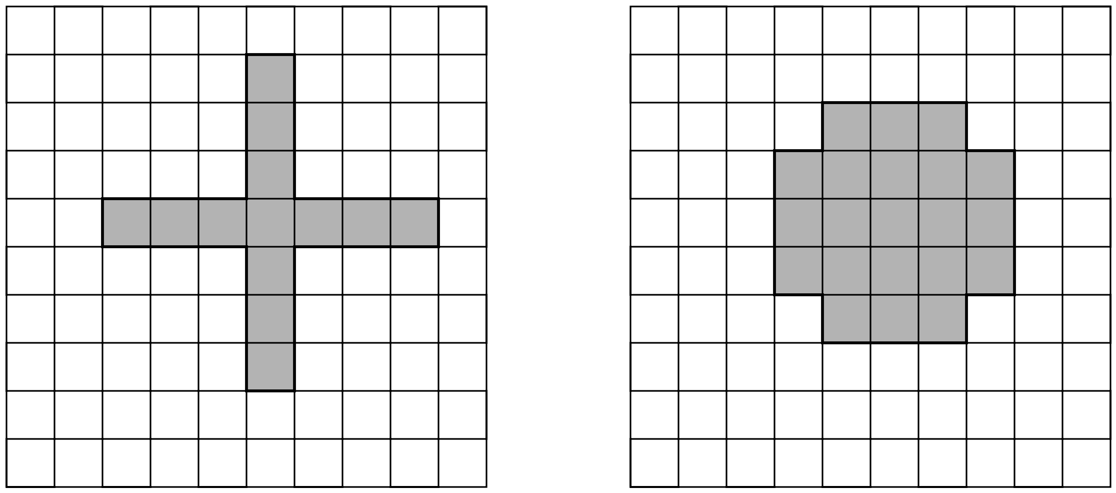
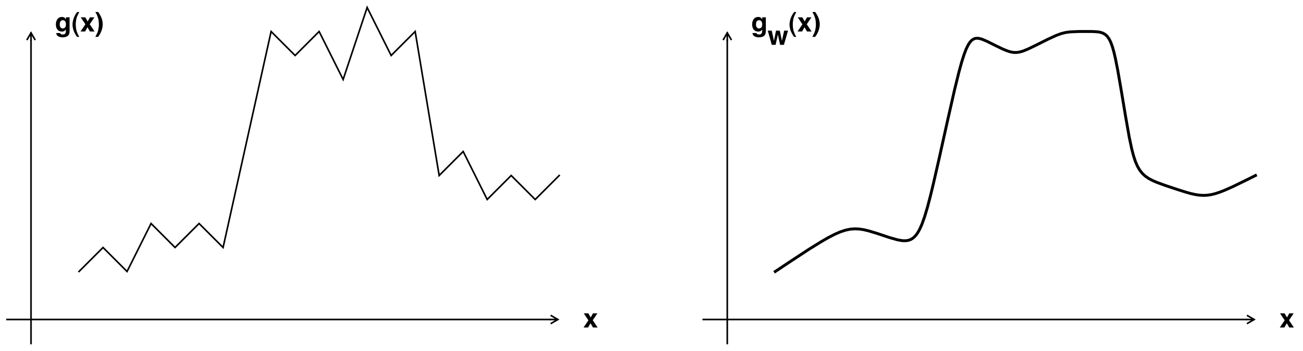

(a) Original pattern (b) Smoothed pattern

Fig. 2.1 Two dimensional averaging.

of averaging, instead of computing the arithmetic average of the pixel values in a neighborhood, we compute the *median* value [Se82]. This will automatically give a value among those available in the input picture. This is called *median filtering*. If you apply median filtering to the image of Fig. 2.1 (a) you will obtain the pattern in Fig. 3.1 since a pixel will become a one (or zero) if at least 5 of the 9 pixels involved have a value of one (or zero). Many variations of this idea, for example *weighted* median filters [Br84], exist for both binary and grey-level pictures. For a comparative study of the latest techniques the reader is referred to [AII90].

4. Logical Smoothing

In logical smoothing the pixels appearing within the averaging window are treated as *Boolean variables* and the value of the smoothed picture function at a point is defined by a Boolean function of those variables. Therefore with this technique we can be very specific as to the description of unwanted (or wanted for that matter) patterns for removal. For example, in *salt-and-pepper* noise we want to flip the value from *one* to *zero* and vice-versa of isolated pixels, i.e., pixels with a value of *one* completely surrounded by pixels of value *zero* and vice-versa. If we choose a 3x3 window operator to effect this smoother then the following algorithm will do the job. Let the pixels be labelled according to the convention illustrated in Fig. 4.1. Then, for each pixel *e* in the original image change its value from *zero* to *one* if all its 8-neighbors are *one* and change its value from *one* to *zero* if all its 8-neighbors are *zero*. Let *e'* denote the smoothed pixel. We can express this oper-



(a) Original waveform (b) Smoothed waveform

Fig. 1.1 One dimensional *regularization*.

it in a manner analogous to the definition of one dimensional regularization

Let $g(x,y)$ be a single-valued two dimensional function. Typically $g(x,y)$ is a light intensity function across the field of vision. Let $g_W(x,y)$ denote the regularized version of $g(x,y)$ given by:

$$g_W(x,y) = (A_W)^{-1} \int\int g(u,v)\, du\, dv$$

where integration is carried out over $w(x,y)$, a window of area A_W centered at (x,y). In other words at each point (x,y) we are substituting the intensity value $g(x,y)$ by the value $g_W(x,y)$ which is an average over a small neighborhood around (x,y).

In the case of a digital image the value of a pixel is usually substituted by the average value of the values of the pixel itself and its eight neighbors. Larger neighborhoods can obviously be used and neither do the neighborhoods have to have a square shape. However, it has been observed frequently in practice that a 3x3 neighborhood is sufficient to give good results in many situations. For example, consider the digital binary pattern in Fig. 2.1 (a) where the black and white pixels have values *one* and *zero*, respectively. First each pixel in the new smoothed pattern is computed as the sum of its own value and the values of its 8-neighbors. Dividing each pixel value by 9 would give the result. However, absolute values are not important at intermediate stages here since we have to quantize all the resulting values into *zeros* and *ones* again to obtain our output binary picture. As a quantization rule we use the following: if the grey level value of the pixel is 3 or greater make it a *one*, otherwise make it a zero. Applying this rule to the pattern of Fig. 2.1 (a) yields the pattern in Fig. 2.1 (b). Note that the ends of the cross have been cut off somewhat and the concave corners have been partially filled in. For additional discussion see [DH73].

3. Median Filtering

Often in image processing, when we have as an input a picture with a given set of integers representing the grey level intensities of the pixels (such as the 0,1 image of Fig. 2.1 (a), we compute some function that gives output pixel values that fall above, below or in between the input grey level values necessitating the quantization illustrated in the previous section. We can often do away with this quantization stage by appropriately modifying the function we compute. In the case

CHAPTER 3

SMOOTHING

Godfried Toussaint

ABSTRACT

This chapter introduces the basic ideas behind smoothing images. We begin with *regularization* of one dimensional functions. For two dimensional images we consider four widely used methods: spatial averaging, median filtering, logical smoothing and hysteresis smoothing. Finally, we consider methods for smoothing and approximating the boundaries of shapes represented as simple polygons.

1. Regularization

Let $g(x)$ be a single-valued one dimensional function. It could be some function of the human voice during speech or the elevation of crops above the ground as a function of distance travelled by a truck driving by with scanning equipment and so on. We are interested in *smoothing* $g(x)$, i.e., removing the noise or kinks from it. If the original $g(x)$ looks something like the curve in Fig. 1.1 (a) we would like to obtain something like that illustrated in Fig. 1.1 (b). Let us call the smoothed curve $g_w(x)$. Mathematically we can define $g_w(x)$ as follows:

$$g_w(x) = 1/w \int g(u)du$$

where integration is carried out from $u = x - w/2$ to $u = x + w/2$. This is known as regularization. There are many interesting properties of regularized curves. For example:

(1) if $g(x)$ is *discontinuous* then $g_w(x)$ is *continuous*.

(2) if $g(x)$ is *continuous* then $g'_w(x)$ is *continuous*, where $g'_w(x)$ denotes the first derivative of $g(x)$.

In the above equation, w is a smoothing parameter that must be "tuned" so as to give "good" results for the application at hand. If w is too large the entire function will be turned to a flat line, the average value of the entire data! If w is too small no change will result.

2. Spatial Smoothing

The following experiment readily performed by anyone illustrates clearly the idea behind spatial smoothing. From a piece of paper cut out a pattern such as a letter. Now make all manner of "random" incisions along the boundary of the pattern to make the boundary quite jagged. Next, place the pattern on an overhead projector and observe the screen as you progressively blur the image by adjusting the focus control. The jagged edges will disappear and the original smooth pattern will appear instead. This process of blurring an image is smoothing. Mathematically we can define