SMOOTHING DIGITIZED CONTOURS

JOHN D. HOBBY[1]

Abstract. We give a fast linear-time algorithm for finding a smooth polygonal approximation to a digitized contour such that the digitization of the polygonal contour matches the original input. The polygonal contour has the minimum possible number of inflections and obeys a localized best-fit property. Most of the vertices lie on a grid whose resolution is only twice that of the pixel grid, and the algorithm can be modified to force all vertices to obey this restriction.

Key Words. smoothing, digitized image, polygonal outline, polygonization

1. Introduction. In graphics and computer typesetting, black and white images are commonly represented as arrays of pixels, and it is convenient to manipulate such images via the contours that describe black-white boundaries. For instance, if pixels are thought of as unit squares that tile the plane, the contours that describe the digitized image of a letter "R" might appear as in Figure 1a. Of course, the smoother contours shown in Figure 1b are a much better rendition of the original design on which the figures are based. In fact, the digitized image was created by applying a digitization process to smooth contours similar to those in Figure 1b. We give a smoothing algorithm that computes Figure 1b from Figure 1a and guarantees that the digitization process would regenerate Figure 1a from Figure 1b. This inverse digitization property is particularly important because it enables the algorithm to eliminate digitization noise without obliterating subtle curves.

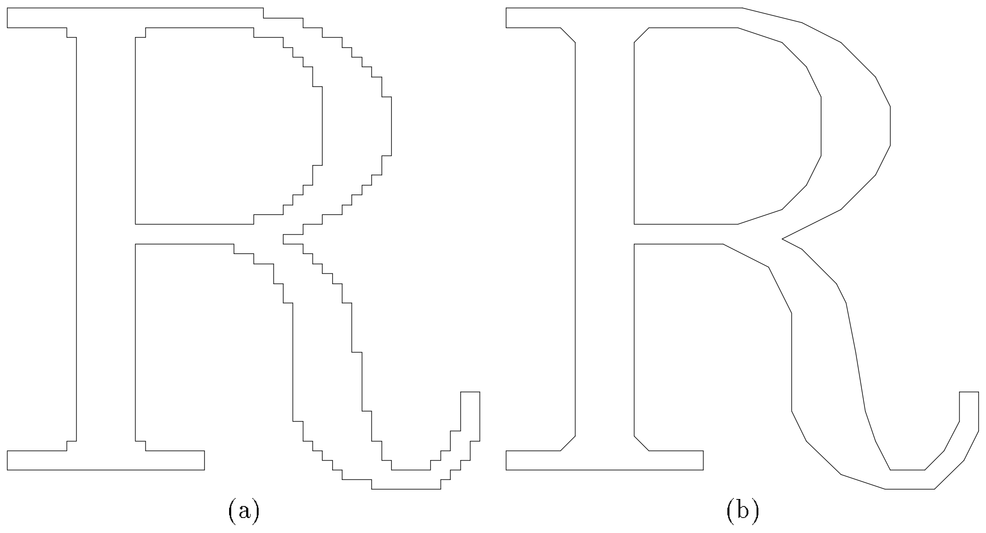

FIG. 1. *(a) Digitized contours for a letter "R" from Knuth's Computer Modern Roman typeface [4]; (b) corresponding smooth polygonal contours.*

The digitization of a smooth polygonal contour C_p is a well defined curve that can be computed by a fast linear algorithm similar to Bresenham's algorithm [1]. Applying the function

$$\rho(x,y) = \left(\lceil x - \tfrac{1}{2}\rceil, \lfloor y + \tfrac{1}{2}\rfloor\right)$$

[1] AT&T Bell Laboratories, 600 Mountain Ave., Murray Hill NJ 07974.

to each point on C_p yields a list of points

$$(m_0, n_0),\ (m_1, n_1),\ (m_2, n_2),\ \ldots,\ (m_k, n_k), \tag{1}$$

where $(m_k, n_k) = (m_0, n_0)$. Ordinarily, each difference $(m_i, n_i) - (m_{i-1}, n_{i-1})$ will be either $(0, \pm 1)$ or $(\pm 1, 0)$ and we may define the digitization $\mathcal{D}(C_p)$ to be the polygonal contour whose vertices are given by (1). The definition fails when a negatively sloped portion of C_p passes through ambiguous points of the form $(m+\frac{1}{2}, n+\frac{1}{2})$ for integers m and n, but we can get by with *non-ambiguous* polygonal contours that contain no such points. (A definition that does handle ambiguous points appears in [3].) Note that the definition also applies to non-closed curves in which case $(m_k, n_k) \neq (m_0, n_0)$.

The digitization $\mathcal{D}(C_p)$ is a polygonal contour made of integer-length vertical and horizontal edges as in Figure 1a. Any contour C_d with this property is called a *digital contour.* In order to have $\mathcal{D}(C_p) = C_d$, a contour C_p must pass sequentially through the squares

$$S(m_0, n_0),\ S(m_1, n_1),\ S(m_2, n_2),\ \ldots,\ S(m_k, n_k), \tag{2}$$

where

$$S(m, n) = \{\, (x, y) \mid m - \tfrac{1}{2} < x \le m + \tfrac{1}{2} \text{ and } n - \tfrac{1}{2} \le y < n + \tfrac{1}{2} \,\}$$

is the set of all (x, y) such that $\rho(x, y) = (m, n)$. Our smoothing process generates a polygonal contour C_p with this property. There are many ways to do this, but this paper gives a new approach that yields polygonal contours with many desirable properties.

The algorithm has important applications to image processing and computer typesetting. It does an excellent job of converting bitmap fonts into polygonal outlines, and it performs well in typesetting applications where the vertices of the polygonal outlines must be given to a limited precision. Other possible applications include scaling bitmap fonts and processing black and white images. The image processing application uses thresholding and edge detection to produce digital contours, while subsequent analysis requires smooth contours. (See Montanari [5] for a more detailed discussion.)

Previous work has concentrated on minimizing the total length of the smoothed contour subject to constraints on the deviation from the digitized input. In [8], Sklansky deals with constraints essentially identical to those implied by the inverse digitization property, and he outlines a fast linear algorithm for computing a polygonal contour that will be convex if any convex contour satisfies his constraints. In [5], Montanari gives a slower algorithm that applies to any digitized contour and deals with broader classes of constraints.

By minimizing total length, Sklansky and Montanari obtain polygonal contours that tend to maximize local deviations from the digitized input. A better approach is to reduce local deviations as much as possible without introducing any extraneous points of inflection. We therefore begin with two theorems that provide insight into the set of all contours that minimize the number of points of inflection without violating the inverse digitization property.

2. Polygonization and Single-Quadrant Paths. We now restrict our attention to portions of digital contours that are monotone in x and y. If σ_1 and σ_2 are two of the four direction vectors $(0, \pm 1)$ and $(\pm 1, 0)$ where $\sigma_1 \neq \pm\sigma_2$, a $\{\sigma_1, \sigma_2\}$ *digital* *path* is a polygonal line joining the integer grid points

$$(m_0, n_0),\ (m_1, n_1),\ (m_2, n_2),\ \ldots,\ (m_k, n_k),$$

in order, where each difference $(m_i, n_i) - (m_{i-1}, n_{i-1})$ is of the form $l_i\sigma_1$ or $l_i\sigma_2$ for some positive integer l_i.

In general, a path that is a $\{\sigma_1, \sigma_2\}$ digital path for some directions σ_1 and σ_2 is called a *single-quadrant digital path.* Since the smoothing process converts single-quadrant digital paths into more general *single-quadrant polygonal paths*, we define a $\{\sigma_1, \sigma_2\}$ *polygonal path* to be a polygonal line joining points

$$(x_0, y_0),\ (x_1, y_1),\ (x_2, y_2),\ \ldots,\ (x_k, y_k), \tag{3}$$

in order, where each difference $(x_i, y_i) - (x_{i-1}, y_{i-1})$ is $\alpha_{1i}\sigma_1 + \alpha_{2i}\sigma_2$ for some $\alpha_{1i}, \alpha_{2i} \geq 0$. In other words, $(x_i, y_i) - (x_{i-1}, y_{i-1})$ is a *nonnegative combination* of σ_1 and σ_2.

A $\{\sigma_1, \sigma_2\}$ digital path with vertices (m_i, n_i) for $0 \leq i \leq k$ is said to be *polygonizable* if for each positive $i < k$, at least one of the vectors $(m_i, n_i) - (m_{i-1}, n_{i-1})$ or $(m_{i+1}, n_{i+1}) - (m_i, n_i)$ has length one. The *polygonization* of such a path is the $\{\sigma_1, \sigma_2\}$ polygonal path obtained by connecting with straight lines the points $\frac{1}{2}(m_i + m_{i+1}, n_i + n_{i+1})$ for $0 \leq i < k$. In other words, the polygonization $\mathcal{P}(P)$ of a single-quadrant digital path P is obtained by connecting the midpoint of every edge in P as shown in Figure 2.

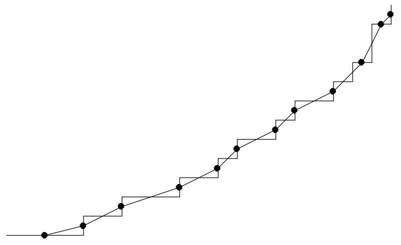

FIG. 2. *A polygonizable* $\{(1,0), (0,1)\}$ *digital path and its polygonization. The bold dots mark the points* $\frac{1}{2}(m_i + m_{i+1}, n_i + n_{i+1})$ *that are vertices of the polygonization.*

If a $\{\sigma_1, \sigma_2\}$ polygonal path P passes through points P_1 and P_2 in that order, $P_2 - P_1$ will always be a nonnegative combination of σ_1 and σ_2. Thus the *subpath of P from P_1 to P_2* contains all points z on P such that both $z - P_1$ and $P_2 - z$ are nonnegative combinations of σ_1 and σ_2. Another important operation on single-quadrant polygonal paths is a *simple extension* formed by lengthening the first and last edges; i.e., Q is a simple extension of P if P is a subpath of Q, whose first and last edges overlap nontrivially with those of Q.

The smoothing algorithm assumes that polygonal paths can be extended in order to achieve a desired digitization. Thus we say that a single-quadrant polygonal path P is *digitally extensible to* a single-quadrant digital path Q if Q can be expressed as the digitization of a subpath of a simple extension of P.

The two theorems below on digital extensibility are based on Lemmas 5.2.1 and 5.2.2 of [3], so we only justify them informally here. Theorem 2.2 depends on the concept of a *digital transformation*; i.e., an affine transformation that is 1 to 1 and onto on the set $\mathbf{Z}^2 + (\frac{1}{2}, \frac{1}{2})$ of points (x, y) where $x \equiv y \equiv \frac{1}{2}$ modulo 1. All such transformations are of the form $T(x, y) = (x - \frac{1}{2}, y - \frac{1}{2})A + (m + \frac{1}{2}, n + \frac{1}{2})$, where m and n are integers and A is a two-by-two integer matrix with determinant ± 1.

Some applications of digital transformations are discussed by Rothstein and Weiman in [7, 6]

THEOREM 2.1. *If P is a polygonizable single-quadrant digital path, then the polygonization $\mathcal{P}(P)$ is digitally extensible to P.*

THEOREM 2.2. *If P is a polygonizable single-quadrant digital path and there is a digital transformation T such that $T(\mathcal{P}(P))$ is a $\{(1,0),(0,1)\}$ digital path, then a non-ambiguous $\{(1,0),(0,1)\}$ polygonal path Q is digitally extensible to $T(\mathcal{P}(P))$ if and only if $T^{-1}(Q)$ is digitally extensible to P.*

Figure 3 illustrates the ideas behind Theorems 2.1 and 2.2. Figure 3a shows a $\{(1,0),(0,1)\}$ digital path P and the corresponding sequence of squares (2) through which a path must pass in order to have P as its digitization. The polygonization $\mathcal{P}(P)$ is not shown in the figure, but it could be formed by joining the bold dots with a polygonal line. Theorem 2.1 simply says that $\mathcal{P}(P)$ passes through the indicated squares when its first and last edges are suitably extended.

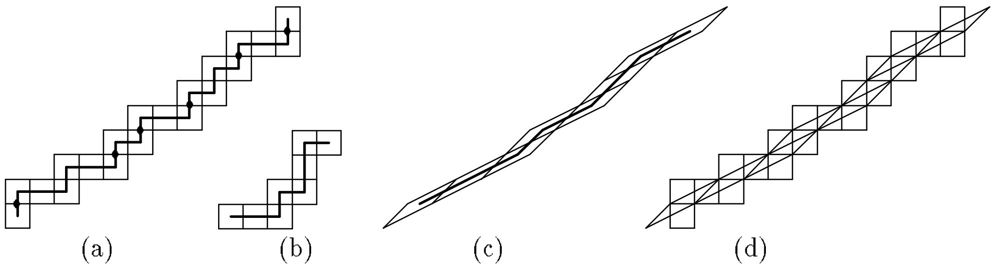

FIG. 3. *(a) A $\{(1,0),(0,1)\}$ digital path and squares where $\rho(x,y)$ is constant; (b) the polygonization transformed to yield a $\{(1,0),(0,1)\}$ digital path; (c) the same figure untransformed; (d) a superposition that illustrates Theorem 2.2.*

If P is the path shown in Figure 3a, Figure 3b shows the $\{(1,0),(0,1)\}$ digital path $T(\mathcal{P}(P))$, where

$$T(x,y) = (\tfrac{1}{2} + x - y,\ -x + 2y) \quad \text{and} \quad T^{-1}(x,y) = (2x + y - 1,\ x + y - \tfrac{1}{2}).$$

The squares in Figure 3b are where a path Q must pass in order to have $T(\mathcal{P}(P))$ as its digitization. Figure 3c is simply the result of applying T^{-1} to Figure 3b to obtain a sequence of parallelograms through which $T^{-1}(Q)$ must pass. Figure 3d illustrates Theorem 2.2 by superimposing the relevant portions of Figures 3a and 3c. The path $T^{-1}(Q)$ that passes through the parallelograms when its first and last edges are extended will also pass through the squares in Figure 3d when the first and last edges are extended sufficiently.

3. The Smoothing Algorithm. Theorems 2.1 and 2.2 suggest a simple recursive smoothing algorithm: Given a digital contour, break it into single-quadrant digital paths, take the polygonization of each path, and repeat for each polygonization that can be transformed into a single-quadrant digital path. If P is a single-quadrant digital path, Theorem 2.1 ensures that $\mathcal{P}(P)$ is digitally extensible to P. If there is a single-quadrant digital path of the form $T(\mathcal{P}(P))$, Theorem 2.2 ensures that we may recursively obtain a polygonal path Q that is digitally extensible to $T(\mathcal{P}(P))$, and $T^{-1}(Q)$ will be digitally extensible to P. Naturally, we extend the concept of polygonization so that we can compute $\mathcal{P}(\mathcal{P}(P))$ instead of $T^{-1}(\mathcal{P}(T(\mathcal{P}(P))))$. Thus, the recursive process just repeatedly polygonizes the polygonization.

To facilitate such repeated polygonization, we need a representation that captures the essential similarities between digital paths and digital transformations thereof. Any polygonal path with rational edge slopes can be represented as a root (x_0, y_0) and a list of edge nodes $e_1, e_2, \ldots, e_k$, where each edge node e_i contains a numeric length parameter $l(e_i)$ and a direction vector $d(e_i)$ of the form (a_i, b_i) for relatively prime integers a_i and b_i. The corresponding polygonal path has vertices $(x_i, y_i) = \sum_{j=1}^{i} l(e_j) \cdot d(e_j)$.

If $\sigma_1 \neq \pm\sigma_2$ are two vectors of the form $(\pm 1, 0)$ or $(0, \pm 1)$, a $\{\sigma_1, \sigma_2\}$ digital path representation has $(x_0, y_0) \in \mathbf{Z}^2$ and each $l(e_i) \in \mathbf{Z}$, where $\mathbf{Z}$ is the set of integers. In addition, the directions $d(e_1), d(e_2), \ldots, d(e_k)$ alternate σ_1, σ_2, σ_1, σ_2, etc. A digital transformation of such path has a very similar representation with σ_1 and σ_2 replaced by their images (a_1, b_1) and (a_2, b_2). The root will satisfy

$$(4) \qquad x_0 \equiv \tfrac{1}{2}(1 + a_1 + a_2) \pmod 1 \quad \text{and} \quad y_0 \equiv \tfrac{1}{2}(1 + b_1 + b_2) \pmod 1,$$

and the directions will alternate between (a_1, b_1) and (a_2, b_2), where $a_1 b_2 - a_2 b_1 = \pm 1$. If $a_1 a_2 + b_1 b_2 > 0$, the path is a *generalized digital path*, or more specifically, an $\{(a_1, a_2), (b_1, b_2)\}$ *digital path*. If no two adjacent $l(e_1)$ and $l(e_{i+1})$ are both greater than one, the path is *polygonizable* and the polygonization may be computed as shown in Figure 4.

```
function $polygonize(x_0, y_0, e_1, e_2, \ldots, e_k)$;
$i \leftarrow 1$; $n \leftarrow 0$;
$(\bar{x}_0, \bar{y}_0) \leftarrow (x_0, y_0) + \frac{1}{2} l(e_1) \cdot d(e_1)$;
while $i < k$
do { $j \leftarrow i + 1$;
      while $j < k$ and $l(e_{j-1}) = l(e_{j+1})$ do $j \leftarrow j + 1$;
      $n \leftarrow n + 1$;
      $d(\bar{e}_n) \leftarrow l(e_i) \cdot d(e_i) + l(e_{i+1}) \cdot d(e_{i+1})$;
      $l(\bar{e}_n) \leftarrow (j - i)/2$;
      $i \leftarrow j$; }
return $(\bar{x}_0, \bar{y}_0, \bar{e}_1, \bar{e}_2, \ldots, \bar{e}_n)$;
```

FIG. 4. *A function that computes the polygonization.*

Instead of giving up when *polygonize* is not directly applicable to its own output, we can try to break the offending path into subpaths that are polygonizable generalized digital paths. The *break* routine in Figure 5 shows how to find the required breakpoints, and the *polygonize_ext* routine polygonizes the resulting subpaths and copes with non-integral $l(e_1)$ and $l(e_k)$. These routines use a function $\tau(e_i, e_j) = a_i b_j - a_j b_i$, where $d(e_i) = (a_i, b_i)$ and $d(e_j) = (a_j, b_j)$ for edges e_i and e_j; the sign of $\tau(e_i, e_{i+1})$ determines whether the path turns left or right between e_i and e_{i+1}. Note that the argument to *break* is assumed to be part of a contour so that we may examine the edges e_0 and e_{k+1} before and after the path to be broken up.

Applying *break* to the polygonization in Figure 6a, successive values of $\tau(e_i, e_{i+1})$ are $2, -1, 1, 1, -1, 1, -1, 1, 1\ -1$, and we obtain the six subpaths labeled A through F in Figure 6b. Subpaths A, C and E are not polygonizable because they each contain only one edge; subpath B is a polygonizable $\{(2, 1), (3, 1)\}$ digital path; subpath D is a polygonizable $\{(2, 1), (1, 1)\}$ digital path; and subpath F becomes a polygonizable $\{(1, 2), (1, 1)\}$ digital path when *polygonize_ext* extends the last edge e_{11} so that $l(e_{11})$ is 1 as shown in Figure 6c instead of $\frac{1}{2}$ as shown in Figure 6b.

```
**function** $polygonize\_ext(x_0, y_0, e_1, e_2, \ldots, e_k)$;
$r \leftarrow l(e_1)$;
$l(e_1) \leftarrow \lceil (e_1) \rceil$;
$l(e_k) \leftarrow \lceil (e_k) \rceil$;
**if** $l(e_1) \neq r$ **then** $(x_0, y_0) \leftarrow (x_0, y_0) + \big(r - l(e_1)\big) \cdot d(e_1)$;
**return** $polygonize(\bar{x}_0, \bar{y}_0, \bar{e}_1, \bar{e}_2, \ldots, \bar{e}_n)$;

**procedure** $break(e_0, e_1, e_2, \ldots, e_k, e_{k+1})$;
**for** $i \leftarrow 1, 2, \ldots, k$
**do if** $\tau(e_{i-1}, e_i)$ and $\tau(e_i, e_{i+1})$ are both positive or both negative
    **then** break before and after $e_i$;
**for** $i \leftarrow 1, 2, \ldots, k-1$
**do if** $\big|\tau(e_i, e_{i+1})\big| > 1$ or $l(e_i) > 1$ and $l(e_{i+1}) > 1$
    **then** break between $e_i$ and $e_{i+1}$;
```

FIG. 5. *Routines for breaking a polygonal path into pieces that are digital transformations of single-quadrant polygonal paths.*

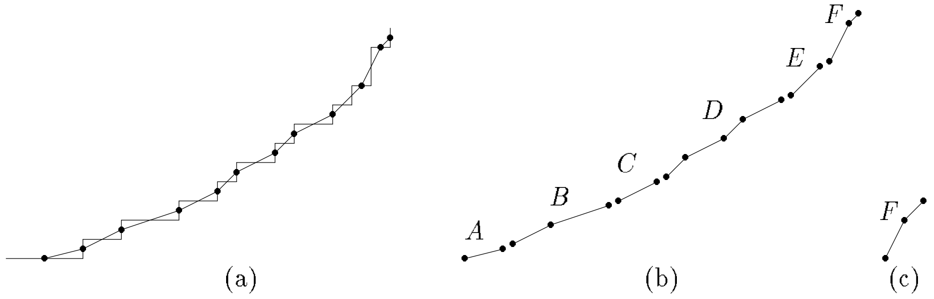

FIG. 6. *(a) A single-quadrant digital path and its polygonization with the vertices of the polygonization marked by dots; (b) subpaths produced by break; (c) a subpath produced by polygonize_ext.*

In general, the effect of the first loop in the *break* routine is to ensure that the subpaths generated will contain only inflection edges. The effect of the second loop is to ensure that $\tau(e_i, e_{i+1})$ alternates between 1 and -1. Since the input to *break* is either a digital contour or the polygonization of a polygonizable generalized digital path, it can be shown that the direction vectors come from a restricted set where $\tau(e_{i-1}, e_i) = -\tau(e_i, e_{i+1})$ implies $d(e_{i-1}) = d(e_{i+1})$. Lemma 3.2 from Section 3.1 then shows that the subpaths produced by *break* are legal input to *polygonize_ext*: We can construct digital transformations that map them into simple extensions of single-quadrant digital paths.

After repeatedly calling *break* and *polygonize_ext*, we obtain a list of polygonal paths with gaps where edges from *polygonize* need to be extended. The complete smoothing algorithm shown in Figure 7 uses a routine *join_paths* to eliminate the gaps by extending the first and last edges of each subpath until they intersect. For brevity, we use P, Q, and R in lieu of edge list representations for polygonal paths and lists of polygonal paths; we also introduce a routine $pop(P)$ that extracts the edges before the first breakpoint in P. When P is a complete contour consisting of sequences of edge nodes with breakpoints in between, $pop(P)$ removes one such sequence of edge nodes.

```
function smooth(P);
Make $Q'$ empty;
$Q \leftarrow break(P)$;
while $Q$ is not empty
  { $R \leftarrow pop(Q)$;
    if $R$ contains more than one edge
    then append $break(polygonize\_ext(R))$ to the beginning of $Q$
    else append $R$ to the end of $Q'$;
  }
return $join\_paths(Q')$;
```

FIG. 7. *The smoothing algorithm.*

Figure 8 illustrates the action of *smooth* on a simple convex shape. When we first reach the top of the **while** loop, the digital contour is broken into subpaths as shown by the dots at subpath boundaries in Figure 8a. We next reach the top of the loop after polyginizing the first subpath and breaking it up as shown in Figure 8b. The **if** test fails in the next two iterations, but the following iteration does call *polygonize*, obtaining a single edge of slope $\frac{3}{2}$ as shown in Figure 8c. The **while** loop terminates after 27 more iterations, obtaining Figure 8d.

The action of *join_paths* on Figure 8d is fairly simple: The list of polygonal paths can be thought of as a polygonal contour with gaps, and *join_paths* eliminates each gap by extending the surrounding edges until they intersect. For example, *join_paths*(Q) may be implemented as shown in Figure 9 if Q is a circular doubly linked list of edge and gap nodes where each gap node g_i contains a vector $\delta(g_i) = (x_i, y_i)$. (We use $\otimes$ for the operation $(x_1, y_1) \otimes (x_2, y_2) = x_1 y_2 - x_2 y_1$ so that $\tau(e_i, e_j) = d(e_i) \otimes d(e_j) = -d(e_j) \otimes d(e_i)$ for all i and j.)

The *join_paths* function given in Figure 9b contains a loop that eliminates edge nodes with nonpositive length parameters. Such "backward edges" are not very common, but they can be created when the loop "**for** each gap $g \in Q$ **do** *remove_gap*(g)" is applied to a Q that contains edges and gaps like those in Figure 10a and Table 1a.

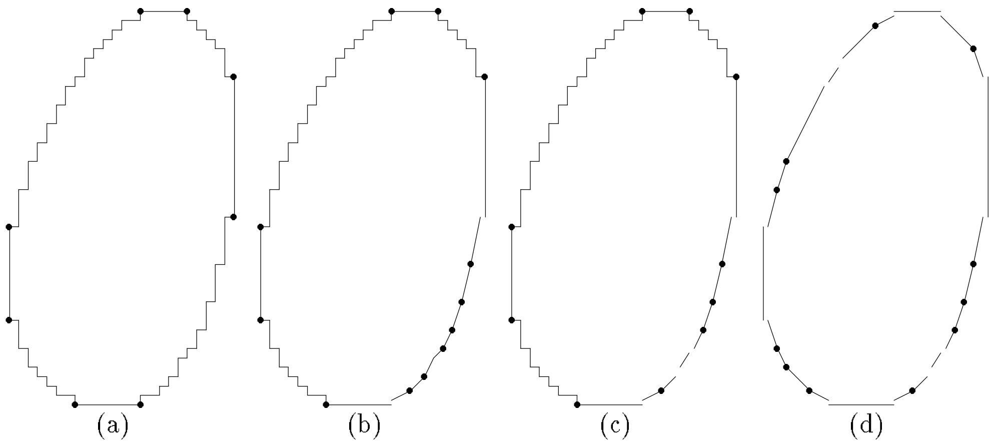

FIG. 8. *Snapshots of Q and Q' in smooth(P) after (a) zero iterations, (b) one iteration, (c) four iterations, (d) 31 iterations.*

```
procedure $remove\_gap(g)$;
Let $e_1$ and $e_2$ be the edges surrounding $g$;
$q \leftarrow \tau(e_1, e_2)$;
$l(e_1) \leftarrow l(e_1) - \big(d(e_2) \otimes \delta(g)\big) \;/\; q$;
$l(e_2) \leftarrow l(e_2) + \big(d(e_1) \otimes \delta(g)\big) \;/\; q$;
Remove $g$ from the linked list;
```

(a)

```
function $join\_paths(Q)$;
for each gap $g \in Q$ do $remove\_gap(g)$;
repeat
  $done \leftarrow true$;
  for each edge $e \in Q$ such that $l(e) \leq 0$
  do { $z \leftarrow l(e) \cdot d(e)$;
       Replace $e$ with a gap $g$ and set $\delta(g) \leftarrow z$;
       $remove\_gap(g)$;
       $done \leftarrow false$;}
until $done$;
return $Q$;
```

(b)

FIG. 9. *Routines for eliminating gaps by extending the surrounding edges.*

Parts b, c, and d of Figure 10 and Table 1 illustrate subsequent steps in the execution of *join_paths*(Q): The backward edge is replaced by a gap that is then removed.

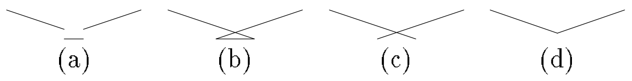

FIG. 10. *Polygonal paths (a) as given to join_paths, (b) after remove_gap(g) for each gap g, (c) after removing the backward edge, and (d) as computed by join_paths.*

node	l	d	δ
e_1	1	$(3,-1)$	
g_1			$(0,-\frac{1}{2})$
e_2	1	$(1,0)$	
g_2			$(0,\frac{1}{2})$
e_3	1	$(3,1)$	

(a)

node	l	d
e_1	$1\frac{1}{2}$	$(3,-1)$
e_2	-2	$(1,0)$
e_3	$1\frac{1}{2}$	$(3,1)$

(b)

node	l	d	δ
e_1	$1\frac{1}{2}$	$(3,-1)$	
g_3			$(-2,0)$
e_3	$1\frac{1}{2}$	$(3,1)$	

(c)

node	l	d
e_1	$1\frac{1}{6}$	$(3,-1)$
e_3	$1\frac{1}{6}$	$(3,1)$

(d)

TABLE 1
Edge and gap data corresponding to Figures 10a–d.

The motivation for the use of backward edges in *join_paths* depends on the concept of a *polygonal tracing* as described by Guibas, Ramshaw, and Stolfi [2]. Using the extension of digitization to polygonal tracings as described in [3], *join_paths* is intended to keep the digitization of Q invariant. A proof would be too tedious to be of much value, but extensive practical experience indicates that the smoothing algorithm given in this section does transform any digital contour P into a polygonal path whose digitization is P. Theorems 2.1 and 2.2 clearly show the validity of repeated polygonization, but it is harder to show that *join_paths* always yields the desired results.

3.1. The Localized Best-Fit Property. If the smoothing algorithm computes a polygonal contour Q from a digital contour P, the localized best fit property is that local changes to Q can only increase the maximum deviation

$$\max\bigl(\max_{z\in P}\operatorname{dist}(z,Q),\ \max_{z\in Q}\operatorname{dist}(z,P)\bigr)$$

between P and Q, where dist() refers to Euclidean distance. The local changes that we shall consider are shifts of a single edge parallel to itself as shown in Figure 11. In Figure 11a, the maximum deviation of the digital path shown in bold from the slope 1 edge of the polygonization is $1/\sqrt{8}$ pixels. Shifting the slope 1 edge as shown in Figure 11b or 11c doubles the maximum deviation.

A more precise statement of the best-fit property is "If $Q = \mathit{smooth}(P)$ has an edge node e with $l(e) \geq 1$, then a parallel shift of the edge represented by e cannot

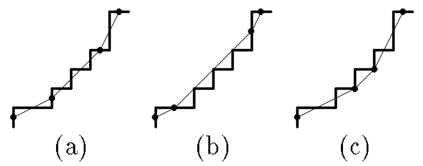

FIG. 11. *A digital path with its polygonization (a), and perturbations (b) and (c).*

decrease the maximum deviation between P and Q in the neighborhood of e." In other words, all sufficiently long edges of Q are optimally placed, where the definition of "sufficiently long" is liberal enough to apply to a large majority of the edges in Q. Theorem 3.3 below captures the key idea behind the best-fit property while avoiding the messy details of how the edges of Q fit together. Using the idea that for purposes of measuring maximum deviation, any sufficiently long rational-slope line segment behaves like an infinite line, we define a line ℓ of rational slope to be *optimally placed* if it is non-ambiguous and the maximum deviation between ℓ and its digitization $\mathcal{D}(\ell)$ is less than the maximum deviation between $\mathcal{D}(\ell)$ and any other line ℓ' parallel to ℓ.

LEMMA 3.1. *A line of the form $qx - py = c$ for relatively prime integers p and q is optimally placed if and only if $c \equiv \frac{1}{2}(1 + p + q)$ (modulo 1).*

Proof. Let ℓ be the line $qx - py = c$ and let $\mathcal{D}(\ell)$ be its digitization. The line ℓ is ambiguous if $qx - py = c$ has solutions where $x \equiv y \equiv \frac{1}{2}$ (modulo 1). In this case $c \equiv \frac{1}{2}p + \frac{1}{2}q$ and ℓ is not optimally placed.

The remaining case is where ℓ is non-ambiguous and $c \not\equiv \frac{1}{2}p + \frac{1}{2}q$. The maximum deviation between ℓ and $\mathcal{D}(\ell)$ is $\max_{(x,y)\in V} \alpha\,|qx - py - c|$, where $\alpha = 1/\sqrt{p^2 + q^2}$ and $V = \{\, \rho(x,y) \mid qx - py = c \,\}$ is the set of vertices of $\mathcal{D}(\ell)$. Since ℓ is non-ambiguous,

$$\begin{aligned}
V &= \{\, (x,y) \in \mathbf{Z}^2 \mid qx' - py' = c \text{ for some } (x',y') \text{ where } \rho(x',y') = (x,y) \,\} \\
&= \{\, (x,y) \in \mathbf{Z}^2 \mid qx' - py' = c \text{ for some } (x',y') \text{ where } x' \not\equiv \tfrac{1}{2} \not\equiv y' \text{ and } \rho(x',y') = (x,y) \,\} \\
&= \{\, (x,y) \in \mathbf{Z}^2 \mid qx' - py' = c \text{ for some } (x',y') \text{ where } |x' - x| < \tfrac{1}{2} \text{ and } |y' - y| < \tfrac{1}{2} \,\} \\
&= \{\, (x,y) \in \mathbf{Z}^2 \mid |qx - py - c| < \tfrac{1}{2}\,|p| + \tfrac{1}{2}\,|q| \,\},
\end{aligned}$$

where $\mathbf{Z}$ is the set of integers. Since $\{\, qx - py \mid (x,y) \in \mathbf{Z}^2 \,\} = \mathbf{Z}$, the deviation between ℓ and $\mathcal{D}(\ell)$ is

$$\max_{(x,y)\in V} \alpha\,|qx - py - c| = \max_{n\in N} \alpha\,|n - c|$$

where

$$\begin{aligned}
N &= \{\, qx - py \mid (x,y) \in V \,\} \\
&= \{\, n \in \mathbf{Z} \mid |n - c| < \tfrac{1}{2}\,|p| + \tfrac{1}{2}\,|q| \,\}.
\end{aligned}$$

If ℓ' is a line $qx - py = c'$, the above analysis shows that the maximum deviation $E(c')$ between ℓ' and $\mathcal{D}(\ell)$ is $\max_{n\in N} \alpha\,|n - c'|$. Clearly, ℓ is optimally placed if and only if the minimum of $E(c')$ occurs at $c' = c$. Since $c \not\equiv \frac{1}{2}p + \frac{1}{2}q$ (modulo 1), we have $c + r \notin \mathbf{Z}$ where $r = \frac{1}{2}\,|p| + \frac{1}{2}\,|q|$. Thus the largest and smallest elements of N are $\lfloor c + r \rfloor$ and $1 + \lfloor c - r \rfloor$, and the minimum of $E(c')$ occurs at

$$\frac{1 + \lfloor c - r \rfloor + \lfloor c + r \rfloor}{2}.$$

Setting $E(c') = c$ yields $c = \frac{1}{2}(1 + \lfloor c - r \rfloor + \lfloor c + r \rfloor) = \frac{1}{2}(1 + \lfloor c - r \rfloor + \lfloor c - r \rfloor + 2r) = \frac{1}{2}(1 + 2r) + \lfloor c - r \rfloor \equiv \frac{1}{2}(1 + 2r)$ (modulo 1). Thus $c \equiv \frac{1}{2}(1 + |p| + |q|) \equiv \frac{1}{2}(1 + p + q)$ as desired. □

Lemma 3.2. *Digital transformations map optimally placed lines into optimally placed lines.*

Proof. A digital transformation $T(x, y) = (x - \frac{1}{2}, y - \frac{1}{2})A + (m + \frac{1}{2}, n + \frac{1}{2})$ maps the line $qx - py = c$ into a line $q'x - p'y = c'$ where

$$c' = T(x, y)(q', -p')^T = (x, y)A(q', -p')^T - (\tfrac{1}{2}, \tfrac{1}{2})A(q', -p')^T + (m + \tfrac{1}{2}, n + \tfrac{1}{2})(q', -p')^T$$

when $qx - py = c$. Thus we may set $A(q', -p')^T = (q, -p)^T$ and $c' = c - (\frac{1}{2}, \frac{1}{2})A(q', -p')^T + (m + \frac{1}{2}, n + \frac{1}{2})(q', -p')^T$. When $c \equiv \frac{1}{2}(1 + p + q)$ (modulo 1), we have

$$\begin{aligned} c' &\equiv c - (\tfrac{1}{2}, \tfrac{1}{2})(q, -p)^T + (m + \tfrac{1}{2}, n + \tfrac{1}{2})(q', -p')^T \\ &= c - \tfrac{1}{2}q + \tfrac{1}{2}p + \tfrac{1}{2}q' - \tfrac{1}{2}p' + (m, n)(q', -p')^T \\ &\equiv \tfrac{1}{2}(1 + p + q) + \tfrac{1}{2}p + \tfrac{1}{2}q - \tfrac{1}{2}p' - \tfrac{1}{2}q' \\ &\equiv \tfrac{1}{2}(1 + p' + q'). \end{aligned}$$

□

Theorem 3.3. *If ℓ is the line determined by some edge in a polygonal contour computed by the smoothing algorithm, then ℓ is optimally placed.*

Proof. Let us say that an edge is *well placed* when the line determined by that edge is. We show by induction that all edges computed by the algorithm are well placed. Clearly, all edges the initial digital contour are well placed because we may apply Lemma 3.1 with $c \in \mathbf{Z}$ and $(p, q) \in \{(0, \pm 1), (\pm 1, 0)\}$.

All other edges are created by polygonizing to a digital transformation of a polygonizable single-quadrant digital path. It suffices to show that the line joining the midpoints of two adjacent edges of such a path is optimally placed. Then Lemma 3.2 allows us to apply the digital transformation that makes the edges in question go from $(-l_1, 0)$ to $(0, 0)$ and from $(0, 0)$ to $(0, l_2)$, where l_1 and l_2 are positive integers not both greater than one. By Lemma 3.1, the line $l_2 x - l_1 y = -\frac{1}{2}l_1 l_2$ joining the midpoints is optimally placed because

$$-\tfrac{1}{2}l_1 l_2 = \tfrac{1}{2}(1 + l_1 + l_2) - \tfrac{1}{2}(1 + l_1)(1 + l_2) \equiv \tfrac{1}{2}(1 + l_1 + l_2) \pmod 1.$$

Thus Lemma 3.2 shows that all new edges are well placed, and the theorem follows. □

3.2. Minimization of Inflections. An important property of smoothing algorithms is that they should eliminate extraneous points of inflection. Figure 1 exemplifies this: The digital contours in Figure 1a have 118 inflections while the smoothed versions in Figure 1b have only eight. In fact, no polygonal contours whose digitizations match Figure 1a can have fewer than eight inflections. The purpose of this section is to show why the results of *smooth* are always optimal in this way.

Since inflections in polygonal contours occur at edges, it is convenient to refer to edges as either *inflection edges* or *plain edges*. The main idea is that the *break* procedure only inserts gap nodes next to plain edges or at "sharp bends" where any contour with the correct digitization must have similar bends. The algorithm proceeds until edge and gap nodes alternate and each inflection edge is surrounded by either "sharp bends" or sequences of plain edges.

When counting the inflection edges in a polygonal path P with edges e_1, e_2, $\ldots$, e_k, we need to know the signs of $\tau_{\text{in}} = \tau(e_0, e_1)$ and $\tau_{\text{out}} = \tau(e_k, e_{k+1})$, where e_0 and e_{k+1} are the edges adjacent to P in the contour. Let $\mathcal{I}(\tau_{\text{in}}, P, \tau_{\text{out}})$ be the number of i where $1 \le i \le k$ and $\tau(e_{i-1}, e_i)$ and $\tau(e_i, e_{i+1})$ have opposite signs.

Let P be a path such that some simple extension P' of P is a single-quadrant digital path. Given the signs of τ_{in} and τ_{out}, we may apply *smooth* to yield a polygonal path $\mathcal{S}(\tau_{\text{in}}, P, \tau_{\text{out}})$. The computation is equivalent to evaluating *break*(*polygonize_ext*(P)) and then recursively applying *smooth* to each resulting subpath. We show by induction on the number of levels of recursion that

$$\mathcal{I}\big(\tau_{\text{in}},\, \mathcal{S}(\tau_{\text{in}}, P, \tau_{\text{out}}),\, \tau_{\text{out}}\big) \le \mathcal{I}(\tau_{\text{in}}, R, \tau_{\text{out}})$$

for any path R that is digitally extensible to P'.

Given a polygonizable $\{(1,0), (0,1)\}$ digital path P with polygonization $Q = \mathcal{P}(P)$, *break*(Q) produces subpaths Q_1, Q_2, $\ldots$, Q_n of Q, where each Q_i ends at the starting point of Q_{i+1}. For each Q_i there is a subpath P_i of P such that $Q_i = \mathcal{P}(P_i)$. The induction step depends on using the following lemma to apply Theorem 2.2 to each P_i and Q_i.

Lemma 3.4. *If P_1, P_2, $\ldots$, P_n are as defined above, then any polygonal path R that is digitally extensible to P is the union of disjoint subpaths R_1, R_2, $\ldots$, R_n such that each R_i is digitally extensible to P_i.*

Proof. Each interior vertex of $\mathcal{P}(P)$ lies at the midpoint of some edge e_j of P where $l(e_{j-1}) \ne l(e_{j+1})$. We cannot have $l(e_j) > 1$, because the polygonizability of P would force $l(e_{j-1}) = l(e_{j+1}) = 1$. The perpendicular bisector of e_j will be the boundary between the the squares $S(m_{j-1}, n_{j-1})$ and $S(m_j, n_j)$ from (2), where (m_{j-1}, n_{j-1}) and (m_j, n_j) are the endpoints of e_j. Any path whose digitization is P must cross this boundary exactly once.

Since R is digitally extensible to P, there is a subpath R' of a simple extension of R such that P is the digitization of R'. Breaking R' where it crosses the perpendicular bisectors of the e_j that contain endpoints of Q_1, Q_2, $\ldots$, Q_n, we obtain disjoint subpaths R'_1, R'_2, $\ldots$, R'_n such that each R'_i is digitally extensible to P_i. To find R_1, R_2, $\ldots$, R_n as required by the lemma, we modify R'_1, R'_2, $\ldots$, R'_n so that their union is R instead of R': Any initial subpath of R not in R' is appended to R'_1; any left-over final subpath of R is appended to R'_n; any R_i not fully contained in R has its first and/or last edges shortened as necessary. □

Each path Q_i produced by *break*(Q) is either a line segment or is extensible to a polygonizable generalized digital path. If I is the set of all i such that Q_i is nontrivial, then for each $i \in I$, there is a digital transformation T_i such that a simple extension of $T_i(Q_i)$ is a polygonizable $\{(1,0), (0,1)\}$ digital path. Lemma 3.4 produces an R_i that is digitally extensible to P_i, and Lemma 3.5 below allows us to assume without loss of generality that $T_i(R_i)$ is a $\{(1,0), (0,1)\}$ polygonal path for each $i \in I$. Thus Theorem 2.2 shows that for $i \in I$, $T_i(R_i)$ is digitally extensible to a simple extension of $T_i(Q_i)$.

Lemma 3.5. *For any polygonal path R with subpaths R_1, R_2, $\ldots$, R_n satisfying Lemma 3.4, there exists a path R' that satisfies the lemma and has the following properties: (1) There must be subpaths R'_1, R'_2, $\ldots$, R'_n of R such that the transformation T_i that maps Q_i into a $\{(1,0), (0,1)\}$ digital path maps R'_i into a $\{(1,0), (0,1)\}$ polygonal path. (2) All nonzero τ_{in} and τ_{out} satisfy $\mathcal{I}(\tau_{\text{in}}, R', \tau_{\text{out}}) \le \mathcal{I}(\tau_{\text{in}}, R, \tau_{\text{out}})$.*

Proof. A complete proof would be rather long and involved but the basic idea is fairly simple: For each i, R'_i is formed by applying a series of local modifications to R_i, where each modification preserves the digitization of R_i and does not increase the number of inflection edges. For example, if R_i passes through the squares in Figure 3d then R'_i will pass through the parallelograms in that figure. The process of constructing R'_i from i is similar to removing from R_i all portions not contained in the parallelograms and then replacing the missing subpaths with portions of the parallelogram boundaries. □

Since $T_i(Q_i)$ requires one less level of polygonization than P, the induction hypothesis shows that if $T_i(R_i)$ is a $\{(1,0),\,(0,1)\}$ digital path then

$$\mathcal{I}(\tau_i, T_i(Q_i), \tau'_i) \leq \mathcal{I}(\tau_i, T_i(R_i), \tau'_i) \tag{5}$$

for any nonzero τ_i and τ'_i. Since affine transformations preserve inflections, (5) still holds when $T_i(Q_i)$ and $T_i(R_i)$ are replaced by Q_i and R_i.

For each two integers $m < n$ belonging to I such that $i \notin I$ for $m < i < n$, let $\tau'_m = \tau_n = \tau(\bar{e}'_m, \bar{e}_n)$, where $\bar{e}'_m$ is the last edge in Q_m and $\bar{e}_n$ is the first edge in Q_n. A simple argument based on Lemma 3.5 allows us to assume that $\tau(\hat{e}'_m, \hat{e}_n)$ and $\tau(\bar{e}'_m, \bar{e}_n)$ have the same sign, where $\hat{e}'_m$ is the last edge in R_m and $\hat{e}_n$ is the first edge in R_n. Thus even if R has no inflection edges except those in some nontrivial R_i, each R_i will contain as many inflection edges as the corresponding Q_i. Hence the total number of inflection edges in R will be at least as great as the total for Q.

4. Refinements. Figures 4, 5, 7, and 9 give a complete implementation of the smoothing algorithm based on lists of edge and gap nodes, and we have already seen that the smoothing algorithm produces a polygonal path whose digitization matches its input while obeying a localized best-fit property and minimizing the number of inflections. These properties do not leave much flexibility, but we can still make some refinements that tend to simplify the smoothed contours.

4.1. Adjustments Prior to Polygonization. The first refinement affects the way *polygonize_ext* prepares a path for polygonization. After extending the first and last edges so that $l(e_1)$ and $l(e_k)$ are integers, it is sometimes desirable to make further adjustments as shown in Figure 12. In each part of the figure, bold dots denote points where gap nodes are inserted into the data structure to delimit subpaths suitable for polygonization. Figure 12a shows a $\{(1,0),\,(0,1)\}$ digital path where $l(e_1)$ is less than $l(e_3)$, $l(e_5)$, and $l(e_7)$. As shown in Figure 12b, e_1 may be replaced by a gap node because the smoothed version of e_2, e_3, …, e_7 will begin with an edge that can be extended so that its digitization matches e_1. Figures 12c and 12d show a $\{(1,0),\,(0,1)\}$ digital path where an extra gap may be inserted to force the e_1 edge to be treated as a separate subpath.

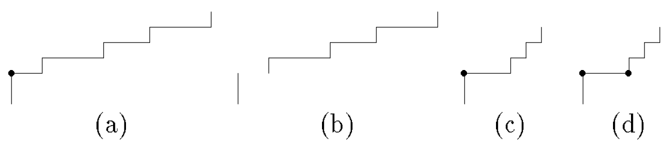

FIG. 12. *Digital paths before and after adjustments that help prepare for polygonization.*

Figures 13a and 13b show that the effect of the adjustment shown in Figures 12a and 12b is to eliminate an unnecessary edge of slope $\frac{1}{2}$ between the vertical edge and the edge of slope $\frac{2}{7}$. As Figures 13c and 13d demonstrate, the effect of the adjustment of Figures 12c and 12d is to replace an edge of slope $\frac{1}{3}$ with an edge of slope 0. The

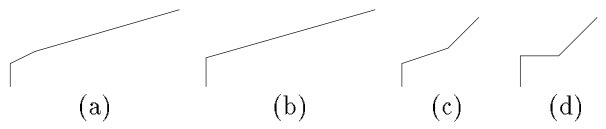

FIG. 13. *The results of applying smooth to Figures 12a–12d.*

slope 0 edge is an inflection edge, but this is safe here because an inflection is required in any case.

The routines in Figure 14 illustrate one way to decide when to apply the above adjustments. A call to *choice* with arguments $l(e_1)$, $l(e_2)$, …, $l(e_k)$ returns *omit* if e_1 should be removed as in Figure 12b; it returns *separate* if e_1 should be a separate subpath as in Figure 12d; and it returns *use* if no adjustment is desirable. A call to *choice* with arguments $l(e_k)$, $l(e_{k-1})$, …, $l(e_1)$ makes a similar decision for the last edge e_k of a polygonizable generalized digital path with edge list e_1, e_2, …, e_k.

```
function $tread\_change(l_1, l_2, \ldots, l_k)$;
for $i \leftarrow 2, 4, 6, \ldots$
do { if $l_i > 1$ then return $-1$;
     if $i = k$ then return $0$;
     if $l_{i+1} > l_{i-1}$ then return $1$;
     if $i + 1 = k$ then return $0$;
     if $l_{i+1} < l_{i-1}$ then return $-1$; }
                    (a)
```

```
function $choice(l_1, l_2, \ldots, l_k)$;
if $l_2 > 1$ or $k = 2$ then return use;
if $l_3 < l_1$ or $k = 3$
    then return (if $l_3 < l_1 - 1$ then separate else use)
else if $l_3 > l_1$ or $tread\_change(l_3, l_4, \ldots, l_k) > 0$
    then return omit
else return use;
                    (b)
```

FIG. 14. *Functions for deciding on adjustments prior to polygonization.*

4.2. Retaining Long Inflection Edges. Figure 15a shows one of the polygonizable single-quadrant digital subpaths produced by applying *break* to Figure 1a. Figure 15b shows that its polygonization has one inflection edge and the slope of that edge is -12. Treating the long edge separately by inserting gap nodes at the dots in Figure 15c yields Figure 15d after polygonizing and Figure 15e after using *join_paths*. Figures 15b and 15e both have the same number of inflections and they both satisfy the localized best-fit property, but Figure 15e may be preferred for aesthetic reasons.

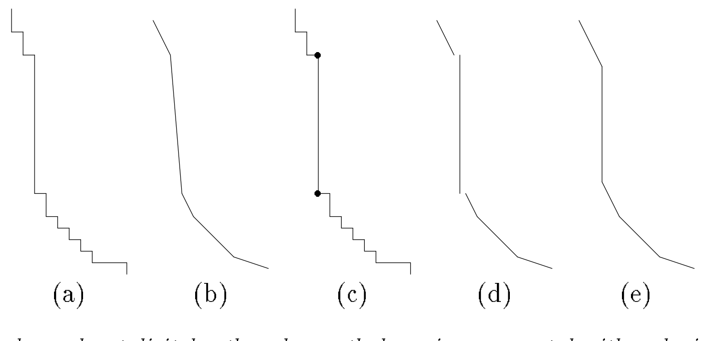

FIG. 15. *A single-quadrant digital path and smoothed versions computed with and without retaining the longest inflection edge.*

Given a polygonizable generalized digital path P with edge list e_1, e_2, …, e_k, an edge e_i is a *long inflection edge* if any path digitally extensible to P must have an inflection edge in the neighborhood of e_i. More precisely, $l(e_i)-1$ must be greater than both $l(e_{i-2})$ and $l(e_{i+2})$, and we must have $i-2>1$ and $i+2<k$. Aesthetic qualities govern the choice of which long inflection edges to retain, but a good compromise to to retain those where $l(e_i) > l(e_{i-2})+l(e_{i+2})$. This may be done by adding the following loop to the end of the *break* procedure in Figure 5:

```
for $i \leftarrow 4, 5, \ldots, k-3$
do if $l(e_i) > l(e_{i-2}) + l(e_{i+2})$ then break before and after $e_i$.
```

4.3. Redirecting Free Edges. When *smooth* is applied to the contour P shown in Figure 16a, the first thing it does is call *break*(P), and this immediately breaks the contour into eight trivial subpaths. This means that *smooth*(P) just returns the original contour P. For the bitmap font application, the contour shown in Figure 16b can be a much preferable result, and this is obtained by a simple refinement to the smoothing algorithm.

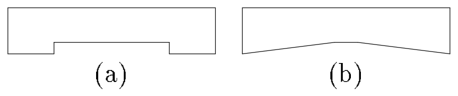

FIG. 16. *(a) A digital contour; (b) a smoothed version produced by the refinement discussed in Section 4.3.*

When *break*(P) makes the short vertical edges in Figure 16a into independent subpaths, they introduce some flexibility into the smoothing algorithm. Ordinarily, if Q is a digital path containing only one edge and Q' is a single-edge polygonal path that is digitally extensible to Q, the possibilities for the direction of Q' form a narrow range centered on the direction of Q, but the range becomes $180°$ when the Q edge is only one unit long. When the Q edge is also an inflection edge, we call it a *free edge*. We can recognize free edges by using the following block in place of the statement "append R to the end of Q'" in the *smooth* function:

```
{ Let $e$ be the edge in $R$;
  if $l(e) = 1$ and $e$ is an inflection edge then mark $e$ as a free edge;
  Append $R$ to the end of $Q'$ }.
```

Free edges need to be processed by a separate loop after all other edge directions have been determined. We therefore insert

```
for each edge $e \in Q'$ do if $adjustable(e)$ then $adjust(e)$
```

just before the **return** statement in *smooth*, where *adjustable* and *adjust* are the routines shown in Figure 17.

5. Conclusion. The basic smoothing algorithm is detailed in Figures 4, 5, 7, and 9 in Section 3. In spite of the complexity of the ideas behind the algorithm, the actual code is fairly simple. It is a fast linear-time algorithm that generates contours that have the localized best-fit property and minimize inflections subject to the constraint that the original contour must be the digitization of the smoothed contour. These properties do not uniquely determine the result of the smoothing algorithm, but the algorithm is intended to produce what might be regarded as the "best and simplest" polygonal approximation.

```
function $adjustable(e)$;
if $e$ is not a free edge then return $false$;
Let $e_-$ and $e_+$ be the edge nodes before and after $e$;
Let $g_-$ and $g_+$ be the gap nodes before and after $e$;
return $d(e_-) = d(e_+)$ and $|\tau(e_-, e)| = 1$ and $\delta(g_-) = \delta(g_+) = (0, 0)$;
procedure $adjust(e)$;
Let $g_-$ and $g_+$ be the gaps before and after $e$;
Let $e_-$ and $e_+$ be the edges before and after $e$;
Let $g_{-2}$ and $g_{+2}$ be the gaps before $e_-$ and after $e_+$;
Let $e_{-2}$ and $e_{+2}$ be the edges before $e_-$ and after $e_+$;
if $adjustable(e_{-2})$ then $l_- \leftarrow \frac{1}{2}l(e_-)$
else { $remove\_gap(g_{-2})$; $l_- \leftarrow l(e_-)$; }
if $adjustable(e_{+2})$ then $l_+ \leftarrow \frac{1}{2}l(e_+)$
else { $remove\_gap(g_{+2})$; $l_+ \leftarrow l(e_+)$; }
$\delta(g_-) \leftarrow \delta(g_-) - \min(l_-, l_+) \cdot d(e_-)$;
$\delta(g_+) \leftarrow \delta(g_+) - \min(l_-, l_+) \cdot d(e_-)$;
$d(e) \leftarrow d(e) + 2\min(l_-, l_+) \cdot d(e_-)$;
```

FIG. 17. *Routines for adjusting free edges.*

It is difficult to define exactly what is meant by "best and simplest," but the refinements discussed in Section 4 give a rough idea of the range of possibilities. The implementations make reasonable choices as to how and when each refinement should take effect, but the choices could be changed depending on the definition of "best and simplest."

Another possible change is related to the claim made in the abstract that "most of the vertices lie on a grid whose resolution is twice that of the pixel grid." For instance, Figure 1b contains 55 vertices, all but one of which lie in $(\frac{1}{2}\mathbf{Z})^2$, where $\frac{1}{2}\mathbf{Z}$ is the set of integer multiples of $\frac{1}{2}$. All the vertices do lie on such a grid if *join_paths* exits with $l(e) \in \frac{1}{2}\mathbf{Z}$ for all edges e, but Table 1 shows that this is not always true. The exceptions tend to occur where relatively long edges with nonsimple slopes intersect, and the exceptional vertices can usually be repositioned slightly without doing much damage to the smoothed contour. (It sometimes helps to apply a digital transformation before rounding the vertices to grid points.)

Another claim that we have not addressed so far is the linear running time of the smoothing algorithm. The intuition behind this is that the number of edges in the polygonization of a path P is at most a fixed fraction of the number of edges in P so that the total number of edges created during polygonization can be found by summing a geometric series. A simple inductive argument shows that the sum of $l(e_i)$ over all edges e_i created during polygonization is no more than the total number of inflections removed by the algorithm. For example, Figure 1a has 136 edges and 118 inflections, while Figure 1b has only eight inflections. Thus 110 inflections were removed and the 33 edges created by *polygonize* had $\sum_i l(e_i) = 48$. The linear bound on $\sum_i l(e_i)$ implies a linear bound on the number of edge nodes created, and this also bounds the number of gap nodes created.

The edge and gap implementation given here can be made reasonably efficient, but it should be noted that this model was chosen for ease of understanding rather than for maximum efficiency. In practice, most of the execution time is spent in the initial phases of the algorithm when there are a few single-quadrant digital subpaths, each containing a large number of edges. Rather than storing such subpaths as lists of edge nodes, we can save time and space by taking advantage of the fact that within a generalized digital path $e_1, e_2, \ldots, e_k$, the direction $d(e_i)$ depends only on whether the index i is odd or even. Thus all $l(e_i)$ values can be stored in a single array L, and a $\{d_0, d_1\}$ digital path with edges $e_1, e_2, \ldots, e_k$ becomes a 4-tuple (i_0, k, d_0, d_1) that refers to $L[i_0+1], L[i_0+2], \ldots, L[i_0+k]$, where $d(e_i) = d_{i \bmod 2}$ and $l(e_i) = L[i_0+i]$. This way, only the 4-tuples need to be stored in a doubly linked list and the gap displacements $\delta(g_i)$ can be merged with the 4-tuples. The only drawback is that it is not possible to represent an arbitrary sequence of edges without inserting undesired subpath boundaries. This means that the *polygonize* and *break* routines have to be merged together because the output of *polygonize* might not be a generalized digital path.

REFERENCES

[1] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4:25–30, 1965.

[2] L. Guibas, L. Ramshaw, and J. Stolfi. A kinetic framework for computational geometry. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, pages 100–111, 1983.

[3] John Douglas Hobby. *Digitized Brush Trajectories.* PhD thesis, Dept. of Computer Science, Stanford University, 1985.

[4] D. E. Knuth. *Computer Modern Typefaces.* Addison Wesley, Reading, Massachusetts, 1986. Volume E of *Computers and Typesetting.*

[5] U. Montanari. A note on minimal length polygonal approximation to a digitized contour. *Communications of the ACM*, 13(1):41–47, January 1970.

[6] J. Rothstein and C. Weiman. Pattern recognition by retina-like devices. Technical Report OSU-CISRS-TR-72-8, Ohio State University, Dept. of Computer and Information Sciences, 1972.

[7] J. Rothstein and C. Weiman. Parallel and sequential specification of a context sensitive language for straight lines on grids. *Computer Graphics and Image Processing*, 5:106–124, 1976.

[8] Jack Sklansky. Recognition of convex blobs. *Pattern Recognition*, 2(1):3–10, January 1970.