COMPUTER VISION, GRAPHICS, AND IMAGE PROCESSING 30, 331–344 (1985)

Vectorgraph Coding: Efficient Coding of Line Drawings

MICHAEL S. LANDY AND YOAV COHEN

Human Information Processing Laboratory, Psychology Department, New York University, New York 10003

Received September 24, 1984; revised January 4, 1985

A method for the efficient coding of line drawings is discussed. The intent is to provide an extremely low bandwidth representation for images, preserving "intelligible" image content but not necessarily image quality. This has led to an image transformation involving edge enhancement, detection, line thinning, and polygonal splining which is termed the *polygonal transformation*. The graph-like image which results from this transformation is coded quite efficiently by using *vectorgraph coding*, which codes a series of vectors in a manner similar to that used in many computer graphics systems. This technique has been applied to a body of images of American Sign Language and the results are encouraging. © 1985 Academic Press, Inc.

1. INTRODUCTION

This paper describes a particular approach developed for extremely efficient coding of images. The work is motivated by a project concerning the viability of a low-bandwidth visual communication system for American Sign Language (ASL, a visual–gestural language used with and among the deaf and hearing impaired). The object of this research was to determine the feasibility of such a system subject to the bandwidth capacity of the current switched telephone network. Thus, the success of this project would result in a system whereby speakers of ASL could effectively communicate visually over the telephone [1–4]. Because of the stringent coding constraints (considering rates as low as 5 to 10 Kbaud), this project has involved transformations of the original grey-scale images subject to the constraint that the resulting images remain "intelligible" to speakers of ASL.

The image transformation and coding technique to be described here results in the lowest bandwidth of any technique that we have developed. In this procedure, the images are treated one frame at a time; no compensation for motion between frames has been incorporated at this time. Edge enhancement and detection are applied to each frame, and the resulting images are thinned and chained. This yields a line drawing which is then approximated using polygonal splining. Finally, the approximated picture is coded as a series of vectors, in a manner similar to that used in many computer graphics systems.

The techniques used here are by no means new. Thinning is very common in image processing and computer vision, and representative algorithms may be found in [5–11]. Splining has also become common in recent years both as a means of image and object representation for pattern recognition and for image compression. These techniques include use of cubics [12–14], conics [15–16], polygonal splining [17–22], and hybrids between these techniques [23]. A number of schemes have been used for line drawing representation. In [7], cubic splines are used for compression and representation. The chain codes of Freeman [24] have been used for both recognition [5, 20, 25] and data compression [7, 26–28]. Other techniques include the

0734-189X/85 $3.00

use of Fourier descriptors based on the chain code [29, 30] and hierarchical coding [31]. Finally, polygonal splines such as those used in this paper have been suggested as tools for both pattern recognition [5] and data compression [26, 32].

The intent of this paper is to outline a particular use of polygonal splining techniques subject to the constraints of intelligibility and extremely low bandwidth. The paper describes the edge enhancement, detection, and thinning techniques used. The polygonal splining procedure is outlined, and results in a *polygonal transformation* of the original image. The coding technique, *vectorgraph coding*, is described in detail. Finally, results of this method on a body of ASL images are discussed.

2. THE POLYGONAL IMAGE TRANSFORMATION

The transformation which begins with an original grey-scale image and produces a polygonal line-drawing representation of that image will now be described. The procedure begins by applying an edge enhancement operator, performing edge detection and thinning, and finally applying a polygonal spline approximation to the resulting curves. The goal is to produce a drawing consisting of a (hopefully small) number of straight line segments or vectors which can then be coded using the vectorgraph technique described in the subsequent section.

Edge Enhancement

The edge enhancement operator used is a convolution mask based on the edge detection technique described by Marr and Hildreth [33]. Their work describes the use of the Laplacian of a Gaussian operator applied to an image (the so-called $\nabla^2 G$ filter) as the ideal edge enhancement tool. This operator can be effectively approximated by a difference of two Gaussians, with an excitatory center and an inhibitory surround (with a larger variance), resulting in a "Mexican hat"-shaped impulse response or convolution mask. The use of the difference of Gaussians (or *dog*) formulation allows for a more efficient implementation of the convolution since the 2-dimensional Gaussian is separable in x and y, allowing the use of four convolutions with 1-dimensional convolution profiles instead of one convolution with the resulting 2-dimensional profile. (For more efficient filtering schemes, see Burt [34].) In our work, the resolution of the dog filter yielding the most intelligible image had a central Gaussian of variance 0.6 pixels and a surround with variance 0.96 pixels (based upon the low-resolution images used in this work—see Sect. 4 below), and a mask size of 7×7.

Edge Detection

The edge enhanced image is next subjected to an edge detection operation. A number of detection schemes have been examined, including the zero crossing detection suggested by Marr and Hildreth [33]. With the ASL images used in this work, the most effective technique, however, is to apply a threshold to the filtered image, rendering the strongest *negative* peaks as black pixels on a primarily white background (see Fig. 6B). With this procedure, edge pixels are placed in a position displaced slightly towards the dark side of each edge. In the case of ASL images, this results in lines around the face, eyes, nose, lips, and fingers, and the image is thereby more intelligible [4; also see 35–36].

Thinning

The next stages of the polygonal transformation process thin the lines in the binary edge-detected image, and chain and categorize the remaining pixels in preparation for the polygonal splining which follows. The thinning and categorizing algorithms are based upon those of Sakai *et al.* [5]. It turns out to be much easier to chain the pixels which remain after thinning if the thinning process removes as many pixels as possible, leaving one-pixel wide lines and curves. In order to do this, the thinning algorithm described by Sakai *et al.* [5] was extended substantially.

The thinning process proceeds in two stages of pixel deletion. The first stage is repeated until no more pixels can be deleted, and then the second stage is applied in a similar iterative fashion. The two deletion stages are most simply described in terms of three characteristics of a given edge pixel P. The first, $N(P)$, is the number of 8-neighbors of the pixel. The second, $T(P)$, is the number of black-to-white and white-to-black *transitions* as the 8-neighbors are examined cyclically. This number characterizes the number of paths leading to this edge pixel. Finally, $G(P)$ is the number of *gaps* in the 8-neighbor set of this pixel. If the 3×3 square of pixels centered on the given pixel is considered in isolation, $G(P)$ is the number of 8-connected components that would remain if the central given pixel were deleted. Alternatively, it is the number of unconnected gaps that would be found proceeding around the deleted pixel's neighbors cyclically. Examples of these concepts are illustrated in Fig. 1.

Given these definitions, the two thinning stages are the following:

(1) Delete all pixels P, where

$$N(P) \in \{3,4,5\} \qquad \text{and} \qquad T(P) = 2.$$

Repeat until no more pixels may be deleted.

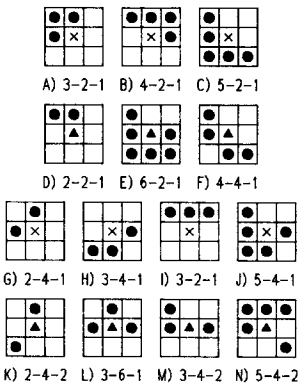

FIG. 1. The thinning process. Thinning determines whether to delete a pixel based on the set of 8-neighbors. The central pixel in each square will be deleted in the first thinning pass for the pixels represented in A–C, but left alone for those in D–F. The second pass deletes the central pixels in G–J, but not those in K–N. The three numbers listed for each central pixel are N, the number of 8-neighbors, T, the number of transitions, and G, the number of gaps.

(2) Delete all pixels P, where

(a) $N(P) = 2$ and $T(P) = 4$,

or

$N(P) = 3$ and $T(P) \in \{2, 4\}$.

or

$N(P) \in \{4, 5, 6\}$.

and

(b) $G(P) = 1$.

Repeat until no more pixels may be deleted.

The first stage does the primary thinning work. The second stage ensures that diagonal lines and curves are at most one pixel wide and junctions have as many pixels removed as possible without changing the 8-connectivity (except for the case of the T junction shown in Fig. 1L, which is preserved for a better spline). A number of representative cases of pixels deleted or left alone by each stage is given in Fig. 1.

Pixel Categorization

The pixels remaining after thinning are next categorized as described by Sakai *et al.* [5]. Pixels are designated as *isolated* (or I), *endpoints* (E), *multi-way junctions* (M or MM, see below), or *uninteresting* (U, for the internal pixels of line segments). The categorization, illustrated in Fig. 2, is initially the following:

if $N(P) = 0$ then I
if $N(P) \leq 2$ and $T(P) = 2$ then E
if $N(P) = 2$ and $T(P) = 4$ then U
if $T(P) \geq 6$ then M
if $N(P) > 2$ and $T(P) = 4$ then MM.

This categorization is exhaustive, since a point with $N(P) > 2$ and $T(P) = 2$ would have been deleted in the thinning process.

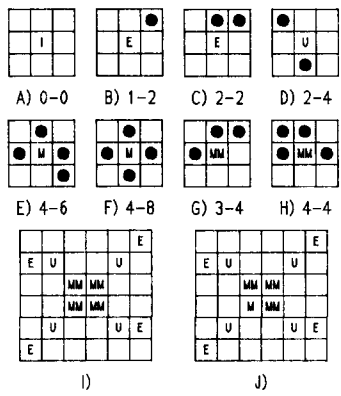

FIG. 2. Pixel categorization. The first pass of categorization classifies pixels as isolated (I), multi-way junction (M), potential multi-way junction (MM), endpoint (E), or uninteresting (U) based on the 8-neighbors. Representative examples are given in A–H. The two numbers given are N, the number of 8-neighbors and T, the number of transitions. In the second pass of categorization, in clusters of MM points (I), a single point is designated as an M point and acts as the attractant for incoming paths (J).

Before we go any farther, a few definitions concerning the chaining and splining process are in order. The output of the chaining process which follows the thinning is a graph structure where the nodes represent single thinned edge pixels, and arcs represent 8-neighbor relations. A subset of the nodes are identified as *knots* for the splining process. Knots are the endpoints and branch points which are used to anchor the splining process. The splining acts separately on each *arc* of the chained image, which is the sequence of neighboring pixels on a path which leads from one knot to another.

In order to minimize the code which will result from the entire procedure, it is necessary to minimize the number of knots and arcs. Thus, in connected clusters of multi-way branch points, it will be useful to designate a single pixel as the *attractant* of all incoming arcs to that cluster. These multi-way knots will be the M points resulting from the categorization process. After categorization and chaining, arcs will consists of a series of chained points beginning and ending with an E or M point, with a sequence of U and MM points in between. The categorization process serves to define the E and M knots, differentiating between the M and MM branch points. Chaining will link the other points in linear arcs to these knots.

The final stage of categorization chooses the knots in clusters of branch points. When there is a large cluster of multi-way junctions, a single point will be designated as the attractant of incoming paths, if possible, rather than having a large group of small segments between each M and MM in the cluster. M points categorized in the first categorization pass (such as in Figs. 2E and F) clearly need to be designated as knots. In the final categorization stage, for any cluster of 8-connected MM points remaining where there are no M points in the group, the MM point nearest the center of gravity of that group (with respect to a city-block metric) is changed into an M point, and becomes the knot for all paths which approach that group (Figs. 2I and J).

Chaining

The thinned and categorized image is next chained, determining the sequence of pixels joining each E or M knot (by definition, the I knots are isolated and require no further processing). The process first joins adjacent edge pixels starting from Es and stopping at Es, Ms, and MMs, with a priority given to Ms over MMs. Next, MMs are further chained to nearest Ms, and Ms are linked to neighboring Ms. Finally, any closed cycle of U points is broken at one point, which is changed to an M, and chained as one arc.

The thinning and categorization process is illustrated in Figs. 3A through D. This example includes a cluster of MM points, where a single element of the cluster was changed to an M and became the knot for all incoming arcs to the cluster. This occurs very rarely in the images because the thinning algorithm usually can delete enough pixels so that one element of the cluster is designated as an M on the first pass. Also note that this example includes a closed contour of U points which is broken during the splining process at an arbitrary point (the lower left point), which then becomes the M knot for the rest of the process.

Splining

At this point, the image is ready to be splined. The image representation now consists of a set of knots (Is, Es, and Ms), connected to each other by arcs of

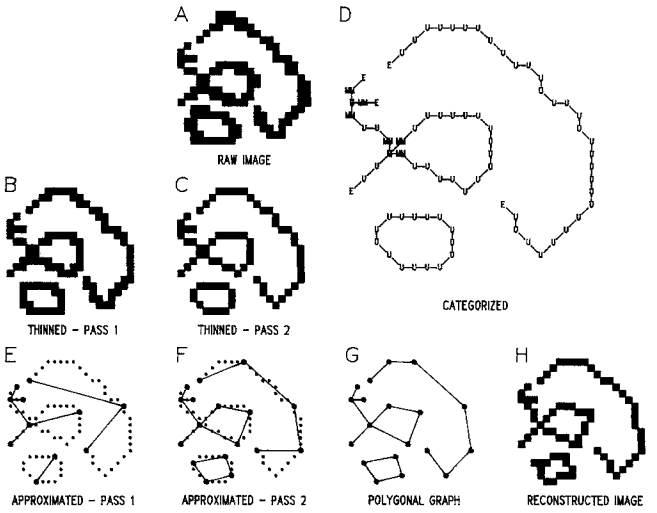

FIG. 3. The polygonal transformation. In the polygonal transformation an edge-detected image (A) is thinned in two passes (B and C) and the pixels remaining are categorized and chained (D). Next, the M and E points (including a new M chosen from the closed contour of Us in the lower left) act as knots for a polygonal splining process. This process chooses as new splining points those points which are furthest from the current approximation until all points are within criterion. After two intermediate passes of this splining (E, F) the final polygonal graph representation is achieved (G). In the same resolution of pixel representation, this graph is reconstructed as (H).

8-connected pixels. The splining process will approximate any given connected pair of knots and their intervening arc with one or more straight line segments.

The method used here is derived from the method described by Ramer [17]. In this procedure, a single arc will be approximated by choosing interior pixels of the arc as *cut points* (or C points), and approximating the arc by a series of line segments joining one end of the arc through each intervening cut point to the other end of the arc. The recursive procedure to choose cut points described by Ramer [17] is the following:

To choose cut points between points A and B,

(1) Find the point C on the arc joining A and B which is the greatest distance, *dmax*, from the straight line segment $\overline{AB}$.

(2) If *dmax* < *criterion* return, else add point C as a cutpoint and repeat the procedure recursively on arcs AC and CB.

Ramer [17] discusses procedures for making the algorithm more efficient in terms of the manner of computing all of the distances from points to $\overline{AB}$, but since the correct spline requires the distance from each point to line *segment* $\overline{AB}$ as opposed to the distance from each point to the line which passes through A and B, the efficient method described by Ramer fails on arcs which contain points whose projection onto AB fall beyond line segment $\overline{AB}$. Therefore, the actual Euclidean distance to segment $\overline{AB}$ is used here. The entire process is illustrated in Fig. 3. It results in the polygonal approximation given in Fig. 3G, which reconstructs on the original raster as in Fig. 3H, although one might at this point use a vector-based imaging device or higher resolution anti-aliased raster technology for the reconstructed image.

3. VECTORGRAPH CODING

The final result of the polygonal image transformation is a new graph structure, the *polygonal graph*, where nodes are knots and cut points, and arcs represent the straight line segments which connect the nodes. The compression scheme for this graph structure will initially represent the graph as a series of commands such as used in most computer graphics systems to control a plotter or vector graphics terminal. The representation consists of commands describing visible and invisible vectors to be drawn, tracing line by line through the graph.

Graph Traversal

The method for choosing a series of vectors to draw the graph begins by considering each maximally connected component of the graph. A given component is traversed beginning with an E point, if possible, and otherwise with an M point. An "initial point command" (I command) describes the invisible vector to this point. The arcs of this component are then traversed from this point, one by one, resulting in "continuation commands" (C commands), which draw the visible vectors corresponding to each arc. The traversal is only allowed to draw a given arc once, and so the process marks each arc as it is traversed, and does not consider marked arcs again. When no further unmarked arcs emanate from the current point, an invisible vector is drawn back to a previously visited M node which still has unmarked arcs to traverse by issuing a "back to command" (B command). This point becomes the current node, and the process continues until all arcs of this component have been traversed.

The next component is chosen based on the smallest possible invisible vector from the current point to a new component (for coding reasons to be described shortly). This yields the I command for the next component. The component traversal and choice of subsequent component are repeated until the entire graph has been traversed. This process is illustrated in Fig. 4.

Binary Coding

It remains to describe how the symbolic description of the traversal is encoded in binary form. In order to code the graph traversal commands efficiently, it is necessary to take into account the characteristics of the set of images to be coded. The important characteristics include the resolution of the images, the variety and frequency of occurrence of different length vectors in the splined image, and the number and position of different maximally connected components of the graph. The polygonal images derived from the ASL originals used in our research (see Fig. 6C), consist of 96×64 pixels and average about 30–40 arcs in 15–20 components. In this section a method will be described which has been found to be highly efficient for these images.

Each command is encoded as a command type (I, C, B, or E), followed by the arguments for that command. Given the frequency of the four command types as occurs in the ASL images, the command code chosen is $\{C = 1, I = 01, B = 001, E = 000\}$, which is the Huffman code [37] for these four words.

The most complicated coding is used for the C command, since it is the most frequent, and optimization here is effective. Nominally, the only arguments to a C

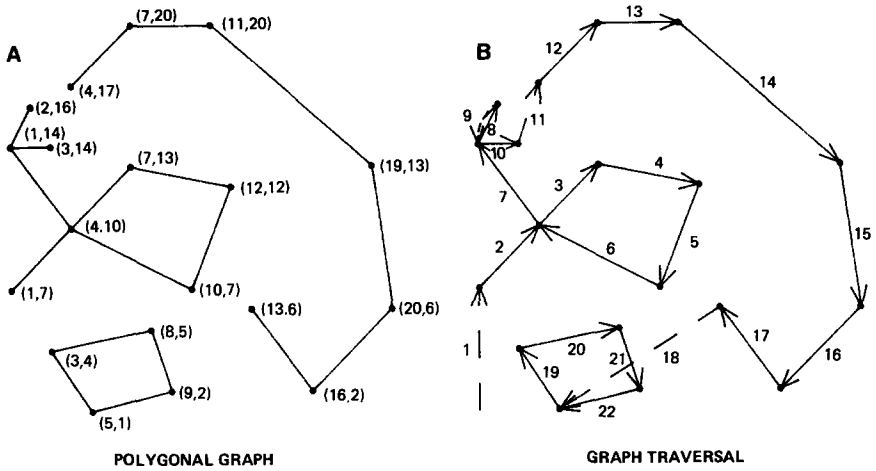

FIG. 4. Traversal of the polygonal graph. The polygonal graph from the previous example (A) is traversed by a series of visible (solid line) and invisible (dotted line) vectors in a component by component fashion (B).

command are the x and y coordinates of the next point to which this vector will be drawn. Given a 96 × 64 image, the address of an arbitrary point requires up to 13 bits if simply coded as the binary representation for x (6 bits) and the binary representation for y (7 bits). The vectors that result from the polygonal transformation are usually fairly short; one can take advantage of this by coding the address as a quantity relative to the previous point, rather than as an absolute coordinate. The coding method used here is 3-tiered, allowing for short vectors, medium-length vectors, and absolute coordinates. The address is then given as an addressing mode (medium-length vector = 0, short vector = 10, absolute = 11), followed by an address (medium-length vector = 8 bits, short vector = 4 bits, absolute address = 13 bits).

The 4 bits of short vector are used to describe a 4 × 4 square around the previous point. In general, this square covers vectors where

$$-1 \leq \Delta x \leq +2 \qquad \text{and} \qquad -1 \leq \Delta y \leq +2,$$

but if the previous point is near the border of the frame, the square is offset so as to lie entirely within the frame in order to maximize the likelihood of being able to use the short vector mode (Fig. 5).

The 8 bits of a medium-length vector are used in a similar manner, describing a set of points around the previous point which moves when the point is near a border. Because a vector will never occur from the previous point to itself, the actual shape of the points reachable by a short vector is one point bigger than a 2 × 2 square. Also, since a short vector will always be chosen in lieu of a medium-length vector if possible, the points reachable by a medium-length vector are actually contained in a 16 × 17 rectangle. The points reachable by the two vector modes are illustrated in Fig. 5.

The I points also take the address of the next point as an argument. Again, allowance is made for relative and absolute modes for these points, but no provision

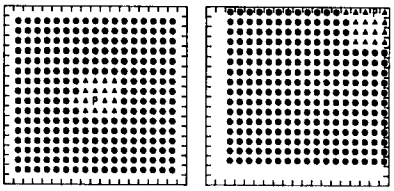

FIG. 5. Vector reach. The points reachable from pixel P by a short vector are designated by triangles, and the points reachable by medium-length vectors are shown as circles. Vectors to all other points would require absolute addressing. In (A) the unconstrained case is shown, where P is near the center of the grid. When the pixel is nearer to the boundaries of the grid, the reachable vectors are displaced so as to maximize the use of vectors and short vectors, as shown in (B).

is made for the short vector mode since it would occur too infrequently. Thus, there is one bit for mode (relative or absolute), and either 8 bits for relative mode (in this case describing a 16 × 16 area), or 13 bits for absolute mode.

The argument for a *B* command gives the number of a previously-described point to return to, relative to the beginning of this component (the most recent *I* command). The code for this argument takes advantage of the possible range of the argument, i.e., the number of points that might be returned to, which is the number already described other than the last point drawn. The coding scheme is given in Table 1 and the net result of this binary coding on the example is illustrated in Table 2.

One final elaboration of this image transformation and coding scheme has been examined. The idea was that once the polygonal graph was achieved, an intelligible image might result from a lower resolution version of the polygonal graph (which was derived originally from a higher resolution image). Thus, the vectorgraph scheme is applied to the polygonal graph with one half of the resolution in each dimension (so that the address arguments are now on a 48 × 32 grid). The coding scheme proceeds unchanged, except that absolute coordinates now take 11 bits, and all vectors have twice as long a reach, so that vector modes are used more often.

4. EVALUATION

The image transformations and coding procedures described above were applied to a library of ASL image sequences. These images of a deaf person signing were digitized to a resolution of 512 × 512 at 30 frames per second. They were then

TABLE 1
Coding Scheme for "Back-to" Commands

Number of potential candidates	Code
1	Not coded (0 bits)
2	Binary code (1 bit)
3, 4	Binary code (2 bits)
5, 6, 7, 8	Binary code (3 bits)
> 8	3 Bit shift code [38]

TABLE 2
Vectorgraph Coding Example

	Command	Number of command bits	Argument	Point number	Mode	Number of mode bits	Number of address bits
1	I	2	(1,7)	0	Vector	1	8
2	C	1	(4,10)	1	Medium vector	1	8
3	C	1	(7,13)	2	Medium vector	1	8
4	C	1	(12,12)	3	Medium vector	1	8
5	C	1	(10,7)	4	Medium vector	1	8
6	C	1	(4,10)	1	Medium vector	1	8
7	C	1	(1,14)	5	Medium vector	1	8
8	C	1	(2,16)	6	Short vector	2	4
9	B	3	5	5	3-Bit binary	NA	3
10	C	1	(3,14)	7	Short vector	2	4
11	I	2	(4,17)	0	Vector	1	8
12	C	1	(7,20)	1	Medium vector	1	8
13	C	1	(11,20)	2	Medium vector	1	8
14	C	1	(19,13)	3	Medium vector	1	8
15	C	1	(20,6)	4	Medium vector	1	8
16	C	1	(16,2)	5	Medium vector	1	8
17	C	1	(13,6)	6	Medium vector	1	8
18	I	2	(5,1)	0	Absolute	1	13
19	C	1	(3,4)	1	Medium vector	1	8
20	C	1	(8,5)	2	Medium vector	1	8
21	C	1	(9,2)	3	Medium vector	1	8
22	C	1	(5,1)	0	Medium vector	1	8
23	E	3	NA	NA	NA	NA	NA

Note. This sequence of commands results from the vectorgraph coding of the example in Figs. 3 and 4.

cropped, reduced by a factor of four in both dimensions by pixel averaging, and the temporal resolution was reduced in half by deleting every other frame. This resulted in sequences of 30 to 45 frames (2–3 s at 15 frames/s), and a resolution of 96 × 64 pixels (Fig. 6A). The images contained the hands, arms, face, and upper body of the signer. There were 80 such sequences, each containing a single ASL sign. The images were digitized with a Grinnell GMR 27-30 image processor and processed using the HIPS system for image processing [39–40].

As described above, each frame was convolved with a difference of Gaussians filter (center variance—0.6 pixels, surround variance—0.96 pixels), and thresholded so that 5% of the negative peaks were preserved as edge elements (Fig. 6B). These images were thinned and categorized. Also, in order to lessen the number of vectors to code, a noise cleaning operation was applied which deleted components which contained only one or two pixels (which includes all *I* points). The cleaned images were chained and polygonally splined with a criterion distance of 1.5 pixels. Finally, all images were vectorgraph coded at both full and half resolution (Figs. 6C and D).

The resulting polygonally transformed images (80 full and 80 half resolution) were vectorgraph encoded. For comparison, the same images were also encoded using

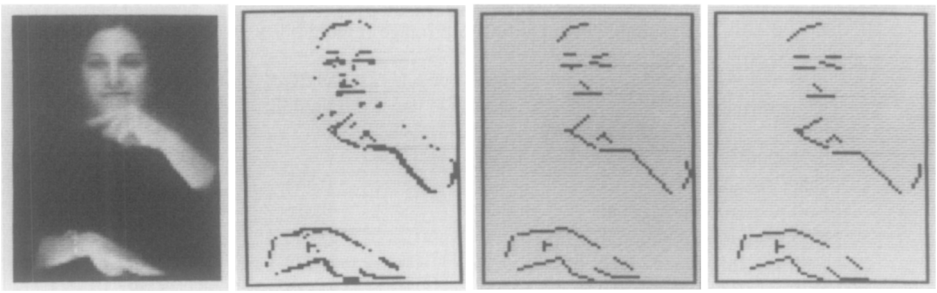

FIG. 6. ASL images: (A) A 96 × 64 image from an image sequence of a sign in ASL. (B) The same image after a convolution with a difference of Gaussians filter and a thresholding which designates the 5% of the most negative-going peaks as edge pixels. (C) The reconstruction of the full resolution polygonal transformation. (D) The reconstruction of the half resolution polygonal transformation.

run-length encoding and a highly efficient hierarchical code, binquad coding [31]. The binquad method was run on images sampled at 30 frames per second rather than 15, so the information rates for that method are slightly underestimated since this is a method which takes advantage of inter-frame correlations. The results are given in Table 3. The vectorgraph coding method achieves a data rate significantly lower than the other two methods. Note that the binquad method is an interframe method which requires 16 frames of look-ahead in order to achieve the rate given.

The important question remaining is to determine whether the impoverished images produced by the polygonal transformation are still intelligible. In previous research [4], we have shown these images to deaf subjects in order to gauge their intelligibility. In Table 4, a sampling of the results from this work are given. The image transformations listed are precisely those shown in Fig. 6. For the "original" subsampled images, the information rate was based on a nominal rate of 3 bits per pixel. The edge detected images were coded using the binquad method, and the polygonally transformed images were vectorgraph coded. Note that the polygonal images were also run at a number of lower frame rates.

The relative intelligibilities listed in Table 4 were derived as follows: 14 deaf persons viewed the sets of transformed images, where each image consisted of a

TABLE 3
Coding Comparison

Image	Coding method	Bits per pixel	Baud rate
Full Resolution	Run-length	0.294	27.1K
	Binquad	0.126	11.6K
	Vectorgraph	0.097	8.9K
Half Resolution	Run-length	0.295	27.2K
	Binquad	0.126	11.6K
	Vectorgraph	0.084	7.8K

TABLE 4
Low Bandwidth Coding and Image Intelligibility

Transformation	Frames per second	Relative intelligibility	Baud rate
"Original"	15	1	276.5K
Edge detected	15	0.865	14.3K
Full resolution polygonal	15	0.816	8.9K
Full resolution polygonal	10	0.695	6.0K
Half resolution polygonal	15	0.738	7.8K
Half resolution polygonal	10	0.617	5.2K
Half resolution polygonal	7.5	0.560	3.9K

single isolated sign in ASL. They were requested to write down an English gloss (or translation) for each sign. For each transformation, the percent correct was computed averaged across all subjects and all single signs. This percent correct was then normalized by the percent correct for the originals in order to gauge the content of the image independent of the difficulty of the task. This assumed that performance with the "original" stimuli was as good as might be expected from any image including the video originals, which did appear to be the case.

As is visible in the table, there is certainly a decrease in intelligibility with information rate, and this is to be expected. On the other hand, the performance rate at the worst condition (the last listed) is surprisingly high given the seriously impoverished images that these data represent. Given no context clues whatsoever, the subjects recognized more than half of the signs in this condition, both relative to the originals and, in fact, in absolute terms, since the percent correct figure for this last condition was 51.3%. It appears that a potentially effective communication scheme might result from the techniques described here for rates as low as 3900 Baud!

5. CONCLUSIONS

This paper has described a method for approximating and coding edge detected images. The approximation method involves thinning and polygonal splining. The coding method results in a series of vector drawing commands which traverse the splined image. The images are fairly intelligible, as was measured for images of American Sign Language and tested with speakers of ASL, and the coding is efficient and extremely low in bandwidth requirements.

A number of useful directions might be followed at this point. In the direction of more compact coding, the obvious next step is to investigate interframe methods. We have considered a method which attempts to match the components of one polygonal graph with those in the subsequent frame. In this scheme, any component in the previous frame may be matched to a component in the subsequent frame using a combination of scaling, rotation, and translation. The hybrid coding that results begins with a bit map which designates the components which matched. Next, matched components are described as either stationary or moving, and moving components are further described by the three parameters for scaling, rotation, and translation. Finally, new components are given in the manner described in this

paper. A further savings might result from subsampling the moving components further in time, and interpolating their motion path. We estimate that a further savings of 15–25% may result from these schemes, but it would be computationally expensive.

On the other hand, the images which result from the polygonal transformation are quite impoverished and already at an extremely low information rate. Thus, the method is now at a point where the important goal may no longer be to reduce the baud rate, but to improve the image quality as much as possible while maintaining the low rates. One possible scheme would be to use a smoother splining technique such as cubic or conic splining in the reconstruction process. This would lessen the extremely jagged quality of the polygonally transformed images, and would be more pleasing to the eye, especially if the reconstruction took place at a higher resolution. In any case, as it stands now this work has resulted in intelligible very low bandwidth images.

ACKNOWLEDGMENTS

The work on image processing of American Sign Language was supported by National Science Foundation, Science and Technology to Aid the Handicapped, Grant PFR-80171189. Special thanks to George Sperling, who conceived the American Sign Language project and provided the support and encouragement that made this image processing research a reality. We also wish to acknowledge the assistance of Thomas Riedl, August van der Beek, and Robert Picardi, and the helpful comments of Robert Hummel.

REFERENCES

1. G. Sperling, Bandwidth requirements for video transmission of American Sign Language and finger spelling, *Science (Washington, D.C.)* **210**, 1980, 797–799.
2. G. Sperling, Video transmission of American Sign Language and finger spelling: Present and projected bandwidth requirements, *IEEE Trans. Commun.* **COM-29**, 1981, 1993–2002.
3. G. Sperling, M. Pavel, Y. Cohen, M. Landy, and B. Schwartz, Image processing in perception and cognition, in *Physical and Biological Processing of Images. Rank Prize Funds Int. Symp. Roy. Soc. London* (O. J. Braddick and A. C. Sleigh, Eds.), pp. 359–378, Springer-Verlag, Berlin, 1983.
4. G. Sperling, M. Pavel, M. S. Landy, and Y. Cohen, Encoding of ASL imagery at extremely low information rates, in *Mathematical Studies in Perception and Cognition* 85–2, New York University, 1985.
5. T. Sakai, M. Nagao, and H. Matsushima, Extraction of invariant picture sub-structures by computer, *Comput. Graphics Image Process.* **1**, 1972, 81–96.
6. C. Arcelli, Pattern thinning by contour tracing, *Comput. Graphics Image Process.* **17**, 1981, 130–144.
7. P. Suetens, P. Dierckx, R. Piessens, and A. Oosterlinck, A semiautomatic digitization method and the use of spline functions in processing line drawings, *Comput. Graphics Image Process.* **15**, 1981, 390–400.
8. T. Pavlidis, *Algorithms for Graphics and Image Processing*, Computer Sci., Rockville, Md., 1982.
9. T. Pavlidis, A thinning algorithm for discrete binary images, *Comput. Graphics Image Process.* **13**, 1980, 142–157.
10. T. Pavlidis, An asynchronous thinning algorithm, *Comput. Graphics Image Process.* **20**, 1982, 133–157.
11. I. S. N. Murthy and K. J. Udupa, A search algorithm for skeletonization of thick patterns, *Comput. Graphics Image Process.* **3**, 1974, 247–259.
12. K. Harada and E. Nakamae, An isotropic four-point interpolation based on cubic splines, *Comput. Graphics Image Process.* **20**, 1982, 283–287.
13. F. Yamaguchi, A new curve fitting method using a CRT computer display, *Comput. Graphics Image Process.* **7**, 1978, 425–437.

14. J. E. Midgley, Isotropic four-point interpolation, *Comput. Graphics Image Process.* **11**, 1979, 192–196.
15. F. L. Bookstein, Fitting conic sections to scattered data, *Comput. Graphics Image Process.* **9**, 1979, 56–71.
16. P. D. Sampson, Fitting conic sections to "very scattered" data: An iterative refinement of the Bookstein algorithm. *Comput. Graphics Image Process.* **18**, 1982, 97–108.
17. U. Ramer, An iterative procedure for the polygonal approximation of plane curves, *Comput. Graphics Image Process.* **1**, 1972, 244–256.
18. C. M. Williams, An efficient algorithm for the piecewise linear approximation of planar curves. *Comput. Graphics Image Process.* **8**, 1978, 286–293.
19. U. Montanari, A note on minimal length polygonal approximation to a digitized contour, *Commun. ACM* **13**, 1970, 4–47.
20. R. Nevatia and K. R. Babu, Linear feature extraction and description, *Comput. Graphics Image Process.* **13**, 1980, 257–269.
21. Y. Kurozumi and W. A. Davis, Polygonal approximation by the minimax method, *Comput. Graphics Image Process.* **19**, 1982, 248–264.
22. T. Pavlidis and S. L. Horowitz, Segmentation of plane curves, *IEEE Trans. Comput.* **C-23**, 1974, 860–870.
23. A. Albano, Representation of digitized contours in terms of conic arcs and straight-line segments, *Comput. Graphics Image Process.* **3**, 1974, 23–33.
24. H. Freeman, Computer processing of line-drawing images, *Comput. Surv.* **6**, 1974, 57–97.
25. I. Chakravarty, A single-pass, chain generating algorithm for region boundaries, *Comput. Graphics Image Process.* **15**, 1981, 182–193.
26. T. Pavlidis, Techniques for optimal compaction of pictures and maps, *Comput. Graphics Image Process.* **3**, 1974, 215–224.
27. D. Proffitt and D. Rosen, Metrication errors and coding efficiency of chain-encoding schemes for the representation of lines and edges. *Comput. Graphics Image Process.* **10**, 1979, 318–332.
28. T. H. Morrin, Chain-link compression of arbitrary black–white images, *Comput. Graphics Image Process.* **5**, 1976, 172–189.
29. F. P. Kuhl and C. R. Giardina, Elliptic Fourier features of a closed contour, *Comput. Graphics Image Process.* **18**, 1982, 236–258.
30. H. C. Lee and K. S. Fu, Using the FFT to determine digital straight line chain codes, *Comput. Graphics Image Process.* **18**, 1982, 359–368.
31. Y. Cohen, M. S. Landy, and M. Pavel, Hierarchical coding of binary images, *IEEE Trans. Pattern Anal. Mach. Intell.*, **PAM I–7**, 1985, 284–298.
32. K. Ramachandran, Coding method for vector representation of engineering drawings, *Proc. IEEE* **68**, 1980, 813–817.
33. D. Marr and E. Hildreth, Theory of edge detection, *Proc. Roy. Soc. London Ser. B* **207**, 1980, 187–217.
34. P. J. Burt, Fast filter transforms for image processing, *Comput. Graphics Image Process.* **16**, 1981, 20–51.
35. D. E. Pearson and J. A. Robinson, Visual communication at very low data rates, *IEEE Proc.* **73**, 1985.
36. P. Letelier, M. Nadler, and J.-F. Abramatic, The Telesign project, *IEEE Proc.* **73**, 1985.
37. D. A. Huffman, A method for the construction of minimum redundancy codes, *Proc. Inst. Radio Eng.* **40**, 1952, 1098–1101.
38. R. C. Gonzalez and P. Wintz, *Digital Image Processing*, Addison–Wesley, Reading, Mass., 1977.
39. M. S. Landy, Y. Cohen, and G. Sperling, HIPS: A UNIX-based image processing system, *Comput. Vis. Graphics Image Process.* **25**, 1984, 331–347.
40. M. S. Landy, Y. Cohen, and G. Sperling, HIPS: Image processing under UNIX, Software and Applications, *Behav. Res. Methods Instrum. and Comput.* **16**, 1984, 199–216.