SOME METHODS OF ENCODING SIMPLE VISUAL IMAGES FOR USE WITH A SPARSE DISTRIBUTED MEMORY, WITH APPLICATIONS TO CHARACTER RECOGNITION

Louis A. Jaeckel

July 1989

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS Technical Report 89.29

NASA Cooperative Agreement Number NCC 2-408 and NCC 2-387

Research Institute for Advanced Computer Science
An Institute of the Universities Space Research Association

SOME METHODS OF ENCODING SIMPLE VISUAL IMAGES FOR USE WITH A SPARSE DISTRIBUTED MEMORY, WITH APPLICATIONS TO CHARACTER RECOGNITION

Louis A. Jaeckel

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS Technical Report 89.29
July 1989

Abstract. To study the problems of encoding visual images for use with a Sparse Distributed Memory (SDM), I consider a specific class of images: those that consist of several pieces, each of which is a line segment or an arc of a circle. This class includes line drawings of characters such as letters of the alphabet. I give a method of representing a segment or an arc by five numbers in a continuous way; that is, similar arcs have similar representations. I also give methods for encoding these numbers as bit strings in an approximately continuous way. The set of possible segments and arcs may be viewed as a five-dimensional manifold M, whose structure is like a Mobious strip. An image, considered to be an unordered set of segments and arcs, is therefore represented by a set of points in M, one for each piece. I then discuss the problem of constructing a preprocessor to find the segments and arcs in these images, although a preprocessor has not been developed. I also describe a possible extension of the representation. A later report will describe some implementations of an SDM based on these encoding methods.

Work reported herein was supported in part by Cooperative Agreements NCC 2-408 and NCC 2-387 between the National Aeronautics and Space Administration (NASA) and the Universities Space Research Association (USRA).

INTENTIONALLY BLANK

SOME METHODS OF ENCODING SIMPLE VISUAL IMAGES FOR USE WITH A SPARSE DISTRIBUTED MEMORY, WITH APPLICATIONS TO CHARACTER RECOGNITION

1. INTRODUCTION

A Sparse Distributed Memory (SDM), as described by Kanerva (1988), is a new design for a computer memory system that can respond to stimuli that are only approximately like those with which it has been trained. Thus it can be applied to tasks such as visual pattern recognition. It is assumed in this report that the reader is familiar with the concept of an SDM.

This report describes some methods of representing and encoding a class of visual images for use as input to an SDM system. A later report will describe some ways to implement an SDM system based on these encoding methods, and will give the results of a small-scale simulation.

The task that I have set for the SDM system is to recognize visual images of simple two-dimensional objects such as letters of the alphabet. I will work with a specific class of possible images: I assume that each image is made of several *pieces*, each of which is a line segment or an arc of a circle. This class is broad enough to include a variety of shapes for line drawings of the familiar characters. I will assume that a visual image is sent first to a preprocessor, which finds the segments and arcs

in the image and encodes a description of each segment and arc in a suitable way. This encoded information representing the pieces found in the image can be used in various ways to generate a description or a representation of the entire image. When applied to an SDM, these representations may be used either as an address to the memory or as the data to be stored in the memory.

Since we want to consider other classes of images and other possible ways of representing them, we have not constructed a preprocessor to find the segments and arcs in an image, as envisioned in this report. However, I believe that such a preprocessor could be built, and in Section 11 I will discuss some of the issues involved in designing a preprocessor. It is assumed that the preprocessor would not have any information built into it concerning the particular set of objects that the SDM will be trained to recognize; that information would be stored in the SDM. The task of the preprocessor would simply be to find the segments and arcs in a fairly mechanical way. However, if we use such a preprocessor, we would be implicitly assuming not only that these features are present in the images, but also that they are important and useful for the task to be performed.

In order to make the discussion more focussed, I will usually assume that the set of objects to be recognized is a set of alphabetic characters, and I will use them as examples, with the understanding that the ideas herein would apply more generally to other sets of simple objects that can be represented by segments and arcs. Thus I will use the term *character* to

refer to any of the objects that the SDM might be trained to recognize.

The SDM system will attempt to recognize and distinguish between a given set of characters, based on the encoded information it receives from the preprocessor. First the system is "trained" on a set of characters by writing representations of one or several images of each character to the memory, using an encoded description of each image of a character as an address to the memory. In other words, the SDM learns the characters "by example". Then, when it is presented with an image which it must try to recognize — that is, identify it or classify it as an instance of one of the characters stored in the memory — the system reads from the memory using an encoded description of the image as the read address. If the image presented is similar to one of the stored characters, the SDM should be able to recognize it. The ability to recognize, or respond to, a stimulus that is only approximately like the stimuli with which the memory was trained is one of the fundamental properties of the SDM.

The question of how to represent the images is crucial. If the SDM is to accomplish its task, the method of encoding must be such that similar images are given similar encodings, and dissimilar images are given encodings that are far apart. Since the nature of the problem will determine which images should be considered as similar and which as dissimilar, the choice of a representation depends on the task to be performed.

The purpose of this work is twofold. The first goal is to explore the problems of representing and encoding the elements of

a visual image so that the encoded information may be used as input to an SDM, or to some other form of associative memory. As stated above, I have chosen to work with a relatively well-defined class of images: those that contain a small number of line segments and circular arcs, which would be identified in the image by a preprocessor. By studying this class of images, we can gain some insight into the problems of representing visual images more generally.

The second goal of this work is to provide a more or less "real-life" example of sensory data encoded for use as input to an SDM. The data that can be generated by the methods below can be used to explore and experiment with various aspects of the performance of an SDM system, and to compare the performance of alternative SDM designs. In a later report I will give some ways of designing and implementing an SDM system based on the encoding methods described in this report, and I will describe some small-scale simulation experiments. This work will help us to understand how the SDM concept may be adapted to fit particular applications.

Since this work is exploratory in nature, I will often discuss various alternative ways to accomplish a particular task, and at times I will indicate which method is used in the present version of the system. However, my choosing a certain design option or parameter value does not mean that I think it is the best one. Sometimes I choose a method because it is simple or because I can evaluate it mathematically, and sometimes my choices are arbitrary. The reason for these design choices is so

that we can have something concrete to experiment with.

I begin in Section 2 by defining the class of images under consideration and stating my assumptions more specifically. I then outline some design considerations in Section 3.

Since we have not developed a preprocessor for finding the segments and arcs in an image, I enter images into the system by drawing them on graph paper, finding the segments and arcs by hand, and entering their approximate coordinates through the keyboard. The procedure is described in Section 4.

In Section 5 I give the method of representing segments and arcs. Line segments are treated as special cases of circular arcs in a continuous way; that is, if a segment is similar to an arc, its representation is close to that for the arc. A segment or arc is described by five parameters: two for its relative location, one for its relative size, and two which jointly represent its orientation and shape, as defined below. The set of possible segments and arcs thus forms a five-dimensional manifold M whose structure is determined by the nature of the similarities between the segments and arcs. We will see in Section 6 that in the two dimensions corresponding to orientation and shape, the manifold is topologically like a Möbius strip. Since I consider an image to be an unordered set of pieces, an image is represented by an unordered set of points in M, one point for each piece.

In Section 7 I consider some measures of distance in M, and in Section 8 I apply these measures to the question of comparing two images, each represented by a set of points in M.

In Section 9 I give some ways of converting the numbers representing a segment or arc to bit strings. This is done in a way that approximately preserves the relative distances between the numbers. For the linear parameters this is straightforward. The two numbers that together represent orientation and shape constitute an unordered pair of angles, or points on a circle; I give a method for converting this pair to a bit string that preserves the topological structure and avoids some of the problems inherent in using numbers to represent these quantities.

I then discuss, in Section 10, the idea of representing an image as a bit string composed of blocks of bits, one block for each piece of the image. To do this we would have to impose an ordering on the pieces. I will argue, however, that there is no natural, continuous way to do this, and therefore I will treat the image as an *unordered* set of pieces. The SDM implementations to be described in a later report are based on this premise.

In Section 11 I return to the issue of how a preprocessor for finding the arcs in an image might be constructed. I point out some of the difficulties and suggest some possible algorithms.

Finally, in Section 12, I describe a possible extension of the method of representing an image to include *critical points* such as intersections of pieces, corners, and endpoints. It would be useful to include such features in the representation, because they provide explicit information on how the individual pieces are related to one another.

2. ASSUMPTIONS ABOUT THE IMAGE RECOGNITION PROBLEM

I will now try to formulate the problem more specifically. The goal is to construct a system to recognize a set of simple two-dimensional objects. First we must decide on a class of possible images that the system can accept as input. The class of images must be such that the objects to be recognized can be depicted by images in the class. Each image in the class I will use is made of several *pieces*, say between two and eight, each of which is a line segment or a circular arc. A preprocessor might be used to find these segments and arcs in the images. (The images must be such that the segments and arcs in them can be identified.) I will assume that each arc is less than or equal to 180^0 of a circle. (Larger arcs that are less than 360^0 could be included, but for now I will not go above 180^0.) Since a line segment will be treated below as a special case of an arc, I will sometimes use the term *arc* to include line segments. In the representation I will use, each arc in an image will be described by five numerical parameters, which give its location relative to the other pieces, its relative size, its shape, and its orientation. In Section 5 these terms will be defined, and the way in which the parameters are computed will be described.

Next, I assume that there is a set of simple objects, which I will call *characters*, that the memory will be taught to recognize. For each character, there are many possible images that should be considered as instances of that character; in fact, there may not be a unique "correct" version of the character. In other words, each character corresponds to a

subclass of the class of images. (Many of the possible images will not lie in any of these subclasses.) The outer limits of these subclasses may not be clearly defined. However, we must have some idea of what these subclasses are, so that we can evaluate the memory's performance. Moreover, the structure of these subclasses should bear some relationship to our notion of whether two images are "similar", since we expect that two similar images will usually (except for borderline cases) belong to the same subclass.

I will use the term *character* in two ways. In the preceding paragraph I used it to refer to the totality of all of the possible images of a character. I will also use the term to refer to a particular image of a character, that is, to refer to a member of a subclass of possible images. This double usage seems to be common, and does not seem to cause any confusion.

To make the discussion more focussed, I will assume from now on that the characters to be recognized are line drawings of some of the letters of the alphabet, although the ideas and methods below would apply more generally to other sets of simple objects that can be represented by segments and arcs. Because of our experience with alphabetic characters, we have some intuitive notions of when such images are similar. The encoding methods described in this report attempt to embody some aspects of our intuition about these images.

Note, however, that I am not beginning with a specific set of characters to be recognized. I want to construct a system that will be able to learn any set of simple characters that can

be drawn with a small number of segments and arcs, *without specifying in advance what the set of characters will be.* I assume that there will not be too many characters in the set, and that they will not be too similar to one another. The SDM will be trained on these characters by writing examples of them to the memory; that is, the training set will include one or several images of each character, and a representation of each image (or some response to give for that character) will be written to the memory, using an encoded description of the image as an address to the memory. After the memory is trained, we will use it to recognize characters as follows: We present it with a new image — that is, we read from the memory using an encoded description of the image as a read address — and the memory is supposed to respond by classifying the image as one of the characters on which it was trained.

The reasons for considering segments and arcs to be the basic elements in the visual images are as follows: First of all, I do not want to use pixels as the features or elements making up the representation of the image. This is because a small change in the image of a character can make a big difference in which pixels are *black* (part of the figure) or *white* (part of the background). For example, if we consider an "A" made of three line segments, then a tall and thin "A" and a slightly shorter and wider "A" will have very few black pixels in common. The same is true if we compare an "A" to one that is slightly rotated or leaning to one side. In other words, if we represent the image as a vector of pixel values, the

representation is not continuous, in the sense defined below. On the other hand, the line segments making up these "A's" are very close to the corresponding segments of the other, similar, "A's", in terms of their locations relative to the other pieces, their relative lengths, and their orientations. Since a small change in the image, of the kinds described above, corresponds to only a small change in the descriptions of the three segments comprising the "A", it would be better to have a representation of a character that is based on descriptions of the pieces. To provide useful input to an SDM, an encoding method must represent similar images similarly.

We could choose line segments to be the fundamental elements of an image, or line segments and circular arcs, or some richer set of image elements. Of course, with a richer set of visual elements, a system could do more interesting and flexible recognition tasks, but it would also be more complex. I decided to use segments and arcs for this study because line segments alone are not sufficient for the usual forms of the familiar characters, and circular arcs, the next step up in complexity, allow us to draw good approximations of a great variety of shapes, including the familiar characters.

Since the relative thickness of the arcs is not very useful for recognizing the familiar characters, I will assume that the arcs in the images are much longer than they are wide, so that they resemble the "strokes" in hand-printed characters. For this reason I will not include the thickness of an arc in the representation. Kahan et al. (1987) have developed a system for

recognizing characters of various fonts and sizes. They observe (at p. 275) that we think of characters as made of strokes, and they therefore use a "thinning" process to reduce the image to a kind of skeleton.

When we choose a representation based on certain image elements, we are building into the system some assumptions as to which image features are important or useful. Thus, by working with the class of images defined above, I am implicitly assuming that the character recognition task depends mainly on the structure of the relatively large-scale components of the image, of which there are only a small number. The fine structure or detail in the image is mostly irrelevant to this task, and should therefore be filtered out by the preprocessor. In other recognition problems, other aspects of the image might be more important.

Whatever set of image elements is chosen, a preprocessor would have to be constructed to find those elements in an image and encode them in some suitable way. Finding these image elements could be difficult, depending on the nature of the images to be presented to the system. Some of these problems are discussed in Section 11. The encoded information would then be used as the input to an SDM, or to some other associative memory system. The preprocessor would be programmed to find the image elements in a fairly mechanical way, without having any built-in information concerning the particular set of characters that will be written to the SDM as the training set. As stated above, the role of the SDM in the system is to learn the characters in the

training set "by example", and to recognize a given character if it is similar to one of the characters in the training set.

I assume that the system is given images of characters one at a time, in isolation, so that there is no information on the context in which the character is situated. For example, an elliptical shape might be an "O", an "o", or a numeral "0", depending on the context. Such information, if it were available, could help a system in a variety of ways; for example, if a character appears in a line of text, its position and size relative to the other characters gives us useful information. The semantic content of the text can also be used; for example, Kahan et al. (1987), p. 283, have used a spelling checker with their system to correct errors. However, to simplify the problem here, I will assume that the context of the characters is not given.

Some English letters, such as a lower case "g", have two or more distinct forms, and the system would naturally see them as different characters. If we want to consider these forms as different forms of the same character, we would have to train the system to give the same response to these different forms.

There are some inherent limitations in reducing images of characters to segments and arcs. Some variations in the form of a character, which may appear small to our eyes, will make a big difference in how the character appears to the system. For example, a letter may or may not have serifs; an "A" may be made of three line segments or it may have a horizontal line across the top; an "O" may look like a circle, like an ellipse, or like

a rectangle with rounded corners. If a character contains curves, for example a "C" or an "S", how to break it up into circular arcs will often be unclear. A small change in the shape of these letters could cause a big difference in how the preprocessor breaks up the curves into arcs, and could even change the number of pieces comprising the character. These problems can be dealt with to some extent by including several instances of each character in the training set, to cover the various possibilities. At this stage I will just assume that the images used can be broken up into arcs fairly unambiguously; this restriction on the set of possible images is the price we must pay for using a relatively simple set of image elements.

3. SOME DESIGN CONSIDERATIONS

There are a few design considerations that I will try to follow.

Location and scale invariance: The system is designed to be location and scale invariant in the following sense: It is assumed that the preprocessor will find the character within the visual field, find a central point for it, and compute a scale factor, in effect drawing a square around the character. The parameters for the pieces of the character will be computed relative to the central point and scale factor of the character as a whole. Thus, since the character is automatically centered and scaled, the system is location and scale invariant. However, the system is not designed to be rotation invariant; a sideways or upside-down "A" is different from a right-side-up "A", and the

system is not intended to see them as the same character.

Continuity: I will use this term loosely, to mean that a small variation in an image will cause only a small change in the representation or encoding of the image, without trying to give it a precise definition. Note that this is not the mathematical definition of continuity; the concept above can apply to discrete objects such as bit strings. Kahan et al. (1987), p. 276, call this property a "shape-similarity smoothness" property. Since the SDM is intended to recognize and deal with objects that are approximate but not exact, we must have a notion of similarity or distance between possible objects, whether it is defined explicitly or is imposed on us implicitly by the nature of the system. When representing or encoding visual images (or any other data) for use as input to an SDM, we want the representation to be as continuous as possible, in the sense that objects we consider similar should have similar representations, and the reverse for dissimilar objects. The problem with representing an image directly by pixels, discussed above, is that the representation is not continuous in this sense. Using segments and arcs will give us a large degree of continuity, but, as mentioned earlier, the method does have limitations; some images that we would call similar would not be seen as similar by this system because they would be broken up into arcs in very different ways.

Representing an unordered set: A problem that will come up in a number of ways is how to represent or describe an unordered set in a unique and continuous way. To describe a set containing

several elements, we would ordinarily give one element first and then another, and so on, with the understanding that the set is an unordered set. But if we do this, each possible ordering of the elements of the set constitutes a different representation of the same set. For example, a line segment can be described by specifying its two endpoints, as an unordered pair of points in the plane. If we represent this set by giving one point first and then the other, then, unless we have a rule like the one in the next paragraph, there are two possible representations. Without such a rule, in order to compare two such pairs of points to see whether they represent the same segment, we would have to try all of the possible permutations. In this case there are only two possibilities to try, but with larger unordered sets there would be many permutations, making the problem more difficult.

To make the representation unique, we could have a rule such as this: Give the endpoint of the segment with larger Y coordinate first, and if both endpoints have the same Y coordinate give the endpoint with smaller X coordinate first. (In other words, scan the image from the top down, and within horizontal lines from left to right.) But if we do this, we lose continuity. If a horizontal line segment is rotated counterclockwise slightly, its endpoints suddenly switch positions in the representation, and two very similar segments will have very different representations; that is, we have a discontinuity. We can get around this problem for line segments by representing them in a different way, as explained in Section

5. We will encounter a similar problem later, when we need to represent an unordered pair of points on a circle in a way that is unique and continuous.

A more serious problem involving ordering occurs if we try to list the pieces of a character in some order. A general property of most classes of visual images is that there is no natural one-dimensional ordering for the elements in the image. A simple "A", for example, consists of three line segments, but there seems to be no way to give the pieces a natural ordering without creating discontinuities in the representation. For this reason, I will consider the set of arcs comprising a character to be an unordered set. We will return to this issue later in Section 10.

4. ENTERING AN IMAGE OF A CHARACTER INTO THE SYSTEM

Since we have not developed a preprocessor, it might be helpful at this point to explain how an image of a character is entered into the present version of the system. The system now consists of a program that performs a rough simulation of one possible implementation of an SDM for this problem; it will be described in a later report. First the character is drawn on graph paper, and its pieces are identified by hand, each piece being a segment or an arc of not more than 180^{o}. The maximum number of pieces is arbitrarily set at eight. Since the program will center and scale the character itself, the location and scale on the graph paper do not matter; any convenient pair of orthogonal coordinate axes will do, as long as the character is

not rotated. Each piece, whether a segment or an arc, is specified by finding the X and Y coordinates, on the graph paper, of its two *endpoints* and its *midpoint* (the point on the arc equidistant from the endpoints). Since these coordinates will be treated in a continuous way, they do not have to be exact. Figure 1 gives an example of a "P". Note that since the curved part of the "P" is wider than a semicircle, that part is broken up into an arc and two short horizontal line segments. Hence the "P" consists of four pieces. The figure shows the coordinates of the endpoints and midpoints, and also gives the encoded values for the pieces.

When this information is to be entered through the keyboard, the program requests, first, a name or identifier for the character, then the number of pieces, and then, for each piece, the X and Y coordinates of one endpoint, then the midpoint, and then the other endpoint. It does not matter in what order the pieces are entered. Also, for each piece, either endpoint may be entered first. (This description of the image may also be stored on a disk file, from which it may later be read into the computer.)

The program then centers and scales the character as a whole, and then converts the input information into five parameters for each piece of the character. The centering and scaling are done as follows: First the program finds the minimum and maximum X coordinates of all of the entered endpoints and midpoints of all of the pieces, and computes the difference. Then it does the same with the Y coordinates. The larger of

these two differences is used as an overall scale factor. A central point for the character is defined by averaging the minimum and maximum X coordinates above, and by doing the same with the Y coordinates. This information is then used to transform the image so that the entered endpoints and midpoints all lie in the unit square. The effect is essentially like drawing a square around the character. (Since for simplicity the square is drawn around the entered endpoints and midpoints, it is possible for part of an arc to lie slightly outside of the square. But that should not matter, since all images are treated in the same way.) The information is then encoded as described below.

An alternative method of centering and scaling a character would be to compute the average and the standard deviation of the X coordinates of all of the endpoints and midpoints of the pieces, and then do the same with the Y coordinates. These quantities could be used to determine a central point and a scale factor for the character. This method would be less dependent on the locations of the most extreme points in the image.

If there were a preprocessor, it would be given a visual image, from which it would find the pieces of the character, center and scale the character, and then express each piece in terms of five parameters, as explained in the next section.

5. REPRESENTING LINE SEGMENTS AND ARCS OF CIRCLES

Each piece of a character is a line segment or a circular arc. I will represent each piece by five parameters, two for the

coordinates of a center point, one for the size, and two parameters that together will represent orientation and shape. (As mentioned earlier, the thickness of the arc is not included in the representation.) The representation is designed to treat a line segment as a special case of an arc of a circle in a continuous way; that is, if a segment is bent slightly into an arc, the arc will be encoded in a way that is close to the encoding for the segment. Since a nearly straight arc might look like a line segment, or vice versa, these similarly shaped pieces should be encoded in a way that preserves their closeness to one another.

Consider first the simpler problem of encoding only line segments. A segment may be described by four numbers, for example the X and Y coordinates of its endpoints. But, as we saw above, if we use this representation, we have the problem of which endpoint is which; either the representation is not unique, or it is not continuous. Another way to represent a segment by four numbers (since there must be that many) is to give the X and Y coordinates of its midpoint, the length of the segment, and a number for its orientation. This representation is unique; there is no problem of which endpoint is which. For the orientation of the segment, we could use the angle from the positive X-axis to the direction of the segment; this angle would be between 0^{o} and 180^{o}. We would not want to use the slope of the segment as a measure of orientation, because it is inhomogeneous: It does not change uniformly as we rotate the segment, and for a vertical segment it is undefined. If we express the angle as a number, we

have a discontinuity at 0^{o} and 180^{o}. Except for this problem, the above representation of a line segment is continuous. We will return later to the issue of representing angles in a continuous way. (Kahan et al., 1987, p. 276, give a somewhat different representation for a segment, which is also unique and continuous.) I will not use this four-number representation because I want to consider a segment to be a special case of an arc.

To represent an arc (including line segments as a special case), we need five numbers. The first two will be the X and Y coordinates of a *center point* for the arc, to locate the arc relative to the character as a whole. This point is different from the midpoint, defined earlier, which is a point on the arc. A simple way to define a center point for an arc is to average the three sets of entered coordinates for the piece (for the two endpoints and the midpoint), after they have been centered and scaled based on the coordinates of all of the pieces. The present program computes the average giving the midpoint double weight, so that the center point is nearer to the midpoint than it would be if each entered point were given equal weight. It is not clear how best to define a center point. It could be argued that the three entered points should be given equal weight, or that the endpoints should receive greater weight than the midpoint. We could compute the true center of gravity of the arc, but it would not be worth the trouble. The real issue in creating an encoding scheme is how we want the encoding to be affected by a change in the arc; that is, which arcs are

relatively similar to each other and which are not. Note that since the present program centers and scales the image so that the endpoints and midpoints (and therefore the center points) of the pieces lie in the unit square, the X and Y coordinates of the center points will lie in the interval [0,1].

Next I define a number to represent the *size* of the arc. This quantity is distinct from the shape of the arc, as defined below. The size is a measure of the overall extent of the arc, relative to the character as a whole. We could compute the arc length, but this is more complicated than necessary. A simple quantity to use is the distance d between the two scaled endpoints. The program computes this distance and transforms it somewhat, so that if d is large, a small change in d is not as important as the same change would be if d is small. Since all of the endpoints lie in a unit square, d must be between 0 and $\sqrt{2}$. I replace d with $d - 0.2071 \times d^2$. Since for d in the above interval, this function is an increasing function with decreasing slope, the transformation reduces the effect of a small change in d when d is large, somewhat like taking the logarithm. The transformation results in a number in the interval [0,1]. This measure of size gives us a number that is intended to represent similar arcs similarly. The function above was chosen somewhat arbitrarily; many other functions would have a similar effect.

To represent the orientation and shape of an arc, I will need a way to represent directions in the plane. A *ray* is a line segment with a direction, like a vector. Consider the set of all

rays emanating from a point P. If we draw a circle about P, the direction of a ray can be represented by the point on the circle where the ray (extended if necessary) intersects it. In other words, the set of possible directions is topologically like a circle. To have a number to represent the direction of a ray emanating from P, I draw a ray from P whose direction is the same as the positive X axis, and I measure the angle, in degrees, counterclockwise from that ray to the given ray. This gives us a number between 0^{0} and 360^{0}. Using a number to represent a direction introduces a discontinuity at 360^{0}, but if we think of a direction as equivalent to a point on a circle, we see that there is no real discontinuity here. In Section 9 I will represent directions in a way that does not involve a discontinuity of this kind. Since I will express directions in terms of angles, I will usually refer to them simply as angles, but they should really be thought of as points on a circle.

To measure the (unsigned) difference, or angular distance, between two directions, or angles, I will treat them as two points on a circle, and I will always measure the distance by the smaller of the two parts of the circle joining the two points, so that the distance is at most 180^{0}. To compute this distance numerically, I take the absolute value of the difference between the two angles, and if the result is greater than 180^{0}, I subtract it from 360^{0}.

I need to define two numbers to represent the orientation and shape of an arc in a unique and continuous way, with line segments included in the representation. By *shape* I mean the

number of degrees of a circle comprising the arc (or zero for a line segment). Note that shape is not the same as curvature; two arcs could have the same shape but be of different size, in which case they would have different curvature. The orientation of an arc (excluding line segments for a moment) may be thought of as the position of the arc on the circle of which it is a part. A way to represent the orientation would be to give the direction from the midpoint of the arc to the center of the circle. But this representation cannot be extended to line segments in a continuous way; if we deform a nearly straight arc into a segment, and then into an arc curving the opposite way, the direction to the center makes a sudden jump of 180^{o}. We will see in the next section that the set of all allowable segments and arcs is topologically like a Möbius strip, which is a non-orientable surface. For this reason, we cannot assign to an arc a number for shape and a number for orientation in a way that is continuous for the entire set of segments and arcs.

Instead of giving a number for shape and another number for orientation, I will define two numbers which together will represent the orientation and shape of an arc or a segment in a unique and continuous way. Given an arc or a segment, I draw a ray from the midpoint of the arc (which is a point on the arc) to each endpoint, and I find the angle that represents the direction of each ray. These two angles, as an *unordered* pair, jointly represent the orientation and shape of the arc, uniquely and continuously. Note that neither endpoint is treated as being "first" or "second". The representation is unique in the sense

that (1) only one unordered pair of angles can represent an arc's orientation and shape, and (2) given the two angles for an arc, we can recover its orientation and shape. It is continuous because if we bend or rotate the arc slightly, there is only a small change in the two angles (except for the discontinuity at 360^{o}, due to representing an angle as a number). This is true even if we deform an arc continuously, without rotating it, from an arc curving one way to a segment to an arc curving the other way. Some examples will be given in the next section.

Figure 1 gives the five parameter values for each of the pieces of a "P". Note that the two angles for each piece are to be considered an unordered pair. The endpoints of each piece were entered into the computer in an arbitrary order, and the corresponding angles are listed in the same order.

If the arc is a line segment, as are three of the pieces in Figure 1, the two angles are 180^{o} apart. If the arc curves 180^{o}, the maximum allowable, the two angles would be 90^{o} apart (using the measure of the difference between angles given above). For example, if an arc goes from "12 o'clock" through "3 o'clock" (the midpoint) to "6 o'clock", the two angles would be 135^{o} and 225^{o}. The curved piece in Figure 1 comes very close to this. For lesser arcs the two angles are more than 90^{o} apart. In general, if an arc comprises θ degrees of a circle ($\theta = 0$ for a segment), the difference between the two angles (measured as above) is $180 - \theta/2$ degrees. This difference represents the shape of the arc. If we think of the orientation of an arc (excluding line segments) as the direction to the center of the

circle on which it lies, this direction would be represented either by the average of the two angles, or by the average plus or minus 180^{0}. For a segment, this average would be one of the directions perpendicular to it.

We can recover an arc from the values of the five parameters, as follows: Suppose the midpoint of the arc is at the origin. Draw two unit vectors from the origin, using the directions indicated by the two given angles. The endpoints of these vectors, together with the origin, give us three points, from which we can determine a circle or a line. The arc or segment connecting these three points has the same shape and orientation as the arc to be recovered. The size and location of the arc can then be determined from the other three parameters.

We now have five numbers that represent a piece of a character. The set of all possible arcs, that is, possible pieces of a character, may be viewed as a five-dimensional manifold M, perhaps embedded in some higher-dimensional space. (To be precise, the interior of M is a manifold.) An image of a character, then, is represented as an unordered set of several points in M, one for each of its pieces. We will see in the next section that for the two parameters representing orientation and shape, the topological structure of this manifold resembles a Möbius strip. The other three parameters, representing location and size, are each like points on a line segment, since they are all limited to [0,1] by the scaling I applied to the character. If we consider the Cartesian product of these three line segments and a Möbius strip, that is, the set of all combinations of

values for the five parameters, we see that **M** is a proper subset of this Cartesian product; that is, not all of the points in the product can represent possible pieces of a character. For example, a point in the product might correspond to a large arc centered near a corner of the square drawn around the character; such an arc might have an endpoint outside of the square, which is not possible with the method of centering and scaling described above. Thus the five-dimensional manifold **M** is a subset of the Cartesian product of three line segments and a Möbius strip.

6. A MÖBIUS STRIP

Consider the pair of angles representing orientation and shape, each represented by numbers between 0^{o} and 360^{o}. For example, for the vertical line segment " | ", the angles are {90,270} — as an unordered pair; for the arc " (", they are roughly {80,280}; and for the arc ") ", they are roughly {100,260}. The set of all allowable unordered pairs of angles is equivalent to the set of all unordered pairs of points on a circle such that the two points are at least 90^{o} apart (measured as explained above). Suppose that the arcs above all have the same location and size parameters, so that we can ignore those parameters for now. If the " | " is rotated 180^{o}, it comes back to itself. If the " (" is rotated 180^{o}, it becomes a ") ". The effect of this rotation on the pair of angles for the " (" is to take the 80^{o} angle to 260^{o} and the 280^{o} angle to 100^{o}; that is, the unordered pair of angles for " (" becomes

the unordered pair of angles for ") ". On the other hand, we can move directly from " (" to " | " to ") " by gradually bending the arc without rotating it. This operation entails making small, continuous changes in the two angles, changing 80^{o} to 100^{o} and 280^{o} to 260^{o}. Similarly, rotating any arc other than a segment 180^{o} changes it to its opposite, a result that can also be obtained by bending the arc without rotating it, while rotating any segment 180^{o} brings it back to itself.

It follows that this set of pairs of points on a circle is topologically like a Möbius strip; that is, each of these pairs corresponds to one point on the strip. Figure 2 shows the Möbius strip, cut along the vertical line AB and laid flat. Some of the arcs and their corresponding pairs of angles are shown on the strip. The pairs representing line segments form a circle running along the middle of the strip like an equator. A 180^{o} rotation of a line segment corresponds to travelling once around the middle of the strip, back to the starting point. Moving away from the "equator" toward the edge of the strip in either direction corresponds to bending the segment into an arc. For example, if we start with " | " and move perpendicularly to the equator in one direction, we come to " ("; if we move the other way, we come to ") ".

The outer edge of the strip corresponds to the 180^{o} arcs. The strip has one continuous edge; if we follow the edge until we return to our starting point, we go around the strip twice, corresponding to a rotation of 360^{o}. Going only once around the strip, following the edge, brings us to an arc with the same

shape but with the opposite orientation — a 180^0 rotation. The same is true of any arc that is not a line segment; for example, starting at " (" and going around the strip once, parallel to the equator, we come to ") ". If the line segments form the equator of the strip, then the shape of an arc (represented by the difference between the two angles) corresponds roughly to latitude, or distance from the equator, and the orientation of an arc (the average of the two angles, or the average plus or minus 180^0) corresponds roughly to longitude. In other words, moving on the strip parallel to the equator represents a rotation, or change in orientation, of an arc, without changing its shape, and corresponds to increasing or decreasing both of the angles representing the arc by the same amount. Moving perpendicularly to the equator represents changing the shape of the arc without changing its orientation, and corresponds to increasing one of the angles and decreasing the other by the same amount. On the other hand, if we change one of the angles representing an arc but hold the other angle fixed, the point on the strip representing the arc moves in a diagonal direction. These ideas are illustrated in Figure 2.

If we wanted, we could reparameterize the strip by replacing the pair of angles with a number for shape (latitude) and a number for orientation (longitude). I will not do that because the twist in the strip would introduce a discontinuity somewhere in the representation, although algorithms could be devised to work around the discontinuity.

7. MEASURES OF DISTANCE IN M

For any given set of visual image elements, the nature of the similarities and dissimilarities between them imposes a topological structure of some sort on the set of elements, such as the Möbius strip above. For the set M, it will be useful to have a measure of distance between any two points, so that we can do some computations. Choosing a metric is an attempt to quantify our beliefs about the relative similarity or dissimilarity of different arcs, although it may be that a metric cannot fully capture the concept of similarity of arcs. There are many different distance measures that could be used, such as L1 (taxicab) distance, L2 (Euclidean) distance, or the uniform metric. The difference between these measures is in the relative importance they attach to the following two situations:

- Changing one parameter by ϵ, with the others held fixed;
- Changing all five parameters by ϵ.

All three of the metrics above treat the first case the same, while for the second case, the L1 distance would be large, the L2 distance not so large, and the uniform distance even smaller. Which one we should use depends on what we consider to be more important, a larger change in one parameter or a smaller change in several.

These measures of distance make sense when two points are near each other, since any small portion of the manifold M is like Euclidean space. For two points far from each other, however, it may not be clear how to measure distance, since M curves around on itself. That is, for any two points, there are

two paths connecting them, depending on which way we go around the Möbius strip. I will define the distance between the two points to be the length of the shorter path. But in any case, if two arcs are very different from each other, it does not matter how different they are, as long as they are called "far apart". When dealing with image features, it may in general be more useful to think in terms of similarity rather than distance.

With any of these distance measures, we can define a weighted distance, with different weights given to each of the five parameters to reflect the relative importance of a change in one of them, with the others held fixed, compared to a change in one of the others. For example, we can ask how a change in the location of the center point of an arc compares to a change in size, or to a change in shape or orientation, in terms of how much the image of the character is affected by such a change. Note that if we replaced the two angles, which together represent the shape and orientation of an arc, by a number for shape (latitude on the Möbius strip) and another number for orientation (longitude), it would be easy to adjust the relative importance of changing the shape of an arc compared to rotating it, by changing the weights assigned to those parameters. However, the same thing can be done indirectly, by redefining the distance between two pairs of angles.

The present system uses L2 distance, with relative weights for the parameters that seem reasonable. One reason for using L2 distance is that it is relatively insensitive to a reparameterization of M (which would be like a rotation of a

coordinate system, at least locally), whereas other metrics are more dependent on how M is parameterized. For example, suppose we replaced the pair of angles by a parameter for orientation and a parameter for shape, as mentioned above; this would correspond to defining a new set of local coordinate axes at each point on the Möbius strip, at an angle of 45^{o} from the local axes corresponding to the pair of angles. Assuming the new parameters are scaled properly, this change would have no effect on the L2 distance between nearby points, whereas if a different metric were used, the change in axes would cause a change in the distance between points.

The method of finding the distance between two unordered pairs of angles (or pairs of points on a circle) is as follows: Since these are unordered pairs, there are two possible ways to match the members of one pair with the members of the other pair. For each way of matching the angles, the program finds the difference between each angle in the first pair and the angle in the second pair with which it is matched, using the measure of the difference between angles given above, and it computes the sum of the squares of the differences between the matched angles. It then chooses the matching that gives the smaller sum of squares. For example, to find the distance between {90,270} and {80,280}, the 90 should be matched with the 80, and the 270 with the 280. (The other way of matching the angles would correspond to going around the Möbius strip the long way.) To find the L2 distance between two points in M, the sum of the squares of the differences between the matched angles is combined

with the sum of the squares of the differences for the other three parameters.

8. COMPARING TWO IMAGES

If we have a measure of distance between points in M, we can try to define an overall measure of distance (or similarity) between two images, each represented as a finite, unordered set of points in M. For two similar images, we might try to match the pieces in one image with the pieces in the other, and then measure the distance between the images by combining the distances between these corresponding pieces. Since we do not know in advance which piece of one image corresponds to each piece of another, possibly similar, image, we would need a procedure for matching the pieces in one image with the pieces in the other. However, if the images are not similar, there may be no piece-by-piece correspondence at all.

One way to compare two images is to compute the distance between each piece of the first and each piece of the second, and arrange the results in a rectangular matrix. Then, if the two images are similar to each other, and have the same number n of pieces, the $n \times n$ matrix will contain n relatively small entries, arranged so that exactly one small entry lies in each row and in each column. The positions of these small entries indicate how the pieces of the two images correspond to one another. Even if the images have different numbers of pieces, there may be some correspondences between their pieces. For example, if we compare the "P" in Figure 1 with an "R" consisting

of five pieces, four of which are similar to the pieces of the "P", we have the following 4×5 matrix of distances:

.2	5.5	3.6	5.1	3.1
5.4	.3	5.1	2.6	4.9
3.5	5.1	.2	5.1	3.5
4.9	2.5	5.1	.3	3.5

In each row there is a small entry in a different column, showing that each piece of the "P" is similar to a piece of the "R". The fifth column, with no small entries, shows that the diagonal line in the "R" is not like any of the pieces of the "P". (Note that the pieces of either image could have been entered in a different order, in which case the entries in the matrix would be the same, but the rows or the columns of the matrix would be rearranged.)

There are various things that could be done with this matrix of distances. If the two images have the same number of pieces, so that we have an n×n matrix, we could try to match pieces in the best way; that is, we could try to find n entries in the matrix such that each row and each column are included once, and such that the sum (or some other function) of these entries is minimized. This minimum value could then be thought of as a measure of the distance between the two images. When we compare two images with different numbers of pieces, it is not clear whether we should try to match pieces at all. But since we might need to try to recognize a character that is missing a piece, or has an extra piece, due to "noise", we may want to try to match pieces, and define an overall distance function that includes a

term for unmatched pieces.

There may be other ways to define a measure of the distance or similarity between two images, which do not involve trying to match the individual pieces. We will see in a later report that when we choose a particular design for an SDM, the design will impose an implicit measure of similarity on the class of images, based on the size of the *access overlap*, that is, the number of memory locations activated by both of two images when used as addresses to the memory.

9. CONVERTING THE PARAMETER VALUES TO BIT STRINGS

Since we usually think of the SDM as operating on bit strings (binary vectors), I will now give a method for converting the numbers and angles representing an arc to bit strings. This will necessarily cause some rounding off, because I will represent a continuous quantity by a bit string chosen from a small set of available strings. But since we are working with approximate information anyway, this will not be a major problem. We will see that there are advantages to converting the pair of angles to a bit string. In order to maintain continuity, the conversion must preserve relative distances, at least approximately. That is, numbers near each other must be converted to bit strings near each other in Hamming distance, and numbers far apart to bit strings far apart. In the latter case, relative distances do not matter much, as long as the distance is large.

First I will represent numbers lying within a limited range,

say the interval [0,1]. Consider the following sequence of nine six-bit strings:

1 1 0 0 0 0

1 1 1 0 0 0

0 1 1 0 0 0

0 1 1 1 0 0

0 0 1 1 0 0

0 0 1 1 1 0

0 0 0 1 1 0

0 0 0 1 1 1

0 0 0 0 1 1

Note that adjacent strings have different numbers of 1's. If they all had the same number of 1's, we would not have as many six-bit strings to work with. Varying the number of 1's permits more efficient use of the bits, but at the cost of some added complexity in recovering the bits when reading from the memory.

These strings are defined so that the Hamming distance between each string and the next is 1, the distance between each and the second from next is 2, and for strings farther apart in the sequence, the Hamming distance is greater than 2. For very distant strings, however, the Hamming distance is not a monotonic function of the distance between them in the sequence, but it is always at least 4.

I will use each of these strings to represent the points in a different subinterval of [0,1]. The subintervals do not have to be of equal width. For example, in the present system the first string represents any number between 0 and 0.15, the second

any number between 0.15 and 0.25, and so on, continuing with intervals of width 0.1, except for the last; the ninth bit string represents numbers between 0.85 and 1. In Figure 1, the first three parameters for each piece are represented by these six-bit strings. If we used longer bit strings, we would of course have higher resolution.

There is a kind of duality here in how we describe the correspondence between bit strings and intervals: For each bit string we can give the set of numbers it is used to represent, as I did above, or, for each of the six bit positions in the string, we can give the set of numbers for which that bit is *on* (set to 1). For example, if we use these nine strings for the nine subintervals of [0,1] defined above, we have the following rule for when each of the six bits is on: If x is the number to be converted to a bit string, then the first bit is on whenever $0 \leq x \leq 0.25$, the second bit is on whenever $0 \leq x \leq 0.45$, the third when $0.15 < x < 0.65$, and so on. There is a biological analogy here to some sensory neural systems: If we think of the six bits as neurons, whose collective function is to report, say, the angle of a knee joint or the frequency of a sound, each neuron is activated by any stimulus within some interval, and these intervals overlap, like the intervals defined by the inequalities above. Some examples are given by Albus (1981), p. 39-40 and p. 58.

Now I will represent a point on a circle as a bit string. I will use a set of 12-bit strings to represent the points on different parts of the circle, just as I used strings for

subintervals above. Imagine 12 bit positions arranged counterclockwise around a circle, 30^{0} apart, with the first bit 15^{0} above "3 o'clock", and the last bit 15^{0} below "3 o'clock". (The bits are only conceptually in a circle, not physically.) Each of the bit strings I will use contains two or three 1's in adjacent positions — that is, adjacent around the circle, so that Bit #12 is adjacent to Bit #1. There are 24 such bit strings:

1 0 0 0 0 0 0 0 0 0 0 1

1 1 0 0 0 0 0 0 0 0 0 1

1 1 0 0 0 0 0 0 0 0 0 0

1 1 1 0 0 0 0 0 0 0 0 0

0 1 1 0 0 0 0 0 0 0 0 0

0 1 1 1 0 0 0 0 0 0 0 0

and so on, up to

0 0 0 0 0 0 0 0 0 1 1 1

0 0 0 0 0 0 0 0 0 0 1 1

1 0 0 0 0 0 0 0 0 0 1 1

If we write the bits in one of these strings around the circle, beginning at 15^{0} above "3 o'clock", we see that the block of 1's in the first string is centered on "3 o'clock", the block of 1's in the second string is centered on the point on the circle 15^{0} above "3 o'clock", and so on, moving counterclockwise around the circle.

Now divide the circle into 24 equal 15^{0} arcs (not to be confused with the arcs that are pieces of characters), beginning with the 15^{0} arc centered at "3 o'clock". The first bit string

above will represent any point on this arc, and so on around the circle, with each arc having the same center as the 1's in the bit string that represents it. Thus any point on the circle is represented by one of these 24 bit strings. Another way to express it is this: To find the bit string that represents a given point on the circle, set to 1 those bits that are within 37.5^{o} of the point.

As before, the Hamming distance between each bit string and the next in the sequence above is 1, and the distance between the first string and the last is also 1, because they are adjacent to each other in the sense that they represent adjacent arcs of the circle. Note that we no longer have to worry about a discontinuity at 360^{o}. We also have the same kind of duality as before; that is, we can give a rule for when each bit is on. For example, the first bit is on whenever the point to be represented is within 52.5^{o} above "3 o'clock", or within 22.5^{o} below "3 o'clock".

Now I can represent an unordered pair of points on the circle as a bit string in a natural way: I just take the logical OR of the bit strings for each point; the resulting bit string represents the unordered pair of points. For example, the " | ", whose angles are {90,270}, is encoded as

0 0 1 1 0 0 0 0 1 1 0 0

(This string is used for Piece #1 in Figure 1.) The " (", whose angles are {80,280}, is encoded as

0 1 1 1 0 0 0 0 1 1 1 0

This representation is used in the present system. Since I am

using only pairs of points on the circle that are at least 90^{o} apart, there will be no overlap between the 1's in the strings for the two points. Therefore, we can recover the two points, at least approximately, from the combined bit string. Note that no preference is given to either of the two points; that is, neither of them is marked as "first" or "second". So we have a unique and approximately continuous representation of an unordered pair of points on the circle. Figure 1 gives the strings representing the pairs of angles for the pieces of the "P".

If two pairs of points are close to each other in the sense discussed earlier, then the Hamming distance between the bit strings for the pairs will be small. For example, the distance between the two strings above is 2. We no longer have to worry about which point to match with which; that is taken care of automatically, as is the discontinuity at 360^{o}. Note that the Hamming distance between two such bit strings representing arcs does not correspond exactly to the L2 distance between the two points on the Möbius strip representing the same arcs, but it does agree with the L2 distance in a qualitative way; actually, the Hamming distance here is more like L1 distance on the strip.

10. A WAY TO REPRESENT AN IMAGE AS A BIT STRING

Using the methods above, we can represent each piece of an image by a 30-bit string: six bits each for the X and Y coordinates of the center, six bits for the size, and 12 bits for the pair of angles representing orientation and shape. If there were a natural way to order the pieces of a character, we could

represent an image of the character as a long bit string consisting of one 30-bit block for each piece. The length of the string would depend on the number of pieces. Since I will arbitrarily limit the images to a maximum of eight pieces, the strings would be no longer than 240 bits. A similar representation will be used in one of the SDM implementations to be described in a later report. However, that representation will be used as data to be written to the memory, rather than as an address.

Note that we could reconstruct the image, at least approximately, from such a bit string, since we can recover the approximate values of the five parameters for each piece. Also, if we wanted to increase the resolution, we could use the methods above with longer bit strings.

But it seems to be impossible to order the pieces in an image in a continuous way. That is, for any ordering scheme there will be images for which a very small change in the image would change the ordering of the pieces. If images are encoded as long bit strings as described above, a different ordering would result in a very different bit string. Then, if the memory system is attempting to recognize the character in an image, using the bit string as a read address, whether it succeeds will depend on which ordering of the pieces was used when an example of the character was stored in the memory. This would be true of any representation that depends on putting the pieces in some order. For example, suppose we assign an ordering to the pieces of an image, something like this: Scan the image from top to

bottom, with a succession of horizontal lines or strips. Within each strip, scan from left to right. When you first encounter a point on a piece, call that piece "number 1"; call the next piece found "number 2", and so on. If two or more pieces are encountered at the same point, for example the two legs of an "A", we would have a rule for breaking ties, perhaps based on the direction from which the piece approaches the point. The problem with any scheme of this kind is that a small change in the image can change the ordering. If we apply the scheme above to an image of an "A" in which one leg extends slightly above the top of the other leg, the pieces will be put in a different order than they would be if the other leg were slightly higher.

A possible way to get around this problem is to use more than one ordering of the pieces when we write an image of a character to the memory. For example, when we store an image in the memory, we could examine it for alternative orderings of the pieces that could easily occur if the image is perturbed somewhat, and then write to the memory using each of the resulting bit strings as an address. (We would not want to write all $n!$ possible orderings of the pieces to the memory.) Then, when we read from the memory in order to try to recognize a character in an image, we hope that the ordering of the pieces in the image matches one of the stored orderings. This seems to be an inefficient use of space in the memory. As an alternative to this idea, or in addition to it, when we read from the memory, we could try several plausible orderings of the pieces in the image being read. But this is also inefficient in that it increases

the time that it takes to read from the memory. Schemes such as these are not impossible, but they are awkward and inefficient.

Therefore, in the SDM implementations to be described in a later report, when I use an encoding of an image as a read or a write address, I will use representations that treat the pieces of the image as an *unordered* set. An important aspect of those implementations is that they are designed so that an unordered set of elements can be used as an address to the memory.

11. SOME PREPROCESSOR ISSUES

In this section I present some thoughts about designing a preprocessor for finding the segments and arcs in the images under consideration. Since we intend to consider other schemes for encoding relatively simple visual images for use as input to an SDM, we are not planning to build a preprocessor for the particular method of representation described in this report. I will, however, discuss some of the issues involved in building such a preprocessor, and outline some possible algorithms, so that we can see that it is indeed possible to build one.

There is a large body of literature on edge and line detection in visual images; see for example Ballard and Brown (1982). There has also been much research on finding and analyzing curves and contours in an image; see for example Parent and Zucker (1985). For an overall review of the field see Olshausen (1988). So if we want to build a preprocessor to embody an encoding scheme of this kind, there is much previous work that we can draw upon.

As stated earlier, I assume that the preprocessor is constructed to find certain image elements, without any built-in information on the particular set of characters that the memory system will have to learn. Of course, any encoding scheme is based on some assumptions, both about the nature of the images to be presented to the system, and also about what features in the images are likely to be useful for the task to be performed. Any preprocessor would necessarily embody some such set of assumptions, either explicitly or implicitly.

Assume for now that the images have the following properties: The images consist of a rectangular array of binary pixels, say, *black* for the figure and *white* for the background. The images contain a small number of segments and arcs, somewhat discretized by the grid of pixels. The thickness of the segments and arcs is fairly uniform; they are at least a few pixels wide, and they are much longer than they are wide, so that each piece is more or less clearly defined. Also, the images are relatively free of noise, and it is fairly clear (at least to our eyes) how to decompose each image into segments and arcs. Moreover, the resolution is fine enough so that the shapes of the segments and arcs are not distorted too much by the discreteness of the pattern of pixels. For example, if the pixels form a 32 by 32 grid, the resolution should be adequate for this class of images. A finer grid would be even better.

There are a variety of algorithms that could be used. An inefficient but conceptually simple procedure would be to use templates, one for each possible line and circle that crosses the

visual field. Each template, comprising a subset of the pixels, would be compared to the image; if a large number of black pixels are found in a template, those pixels are examined to see whether they constitute a segment or arc within the template. Since these templates would be a certain number of pixels wide, only a finite number of templates would be required, but the number of templates needed would be so large that this method would be very inefficient. Moreover, the method does not generalize well: If we want to consider images made of a larger family of image elements, the number of templates required would be hopelessly large.

One type of algorithm is a search procedure, something like this: First, we scan the image in horizontal rows, beginning at the top, until a black pixel, or a cluster of them, is found. This gives us a starting point. (Many other ways of finding a first point could be devised.) Draw a small circle about that point, with a radius of several pixels, and locate clusters of black pixels on that circle. Each such cluster represents a possible point on an arc that may go through the starting point. Then draw small circles using each of these new points as centers, and locate clusters of black pixels on these circles. Now, if we are lucky, we will have three points on an arc, or sets of three points on each of a few arcs. Since three points (clusters of black pixels) on an arc give us a rough idea of the curvature of the arc, we can estimate the direction in which the arc will continue, and search for black pixels in that direction. So we choose a possible arc to follow, and trace along it,

refining our estimate of the curvature as we find more points. If we do not find more points on the arc, either we have passed an endpoint, or we were not on an arc at all; the three points we found might have been on different arcs, or some of them might have been noise. If we are following an arc, we must search in both directions from the first three points found until we find both of its endpoints. This search can be done in jumps several pixels long, rather than by crawling one pixel at a time. Then, when we find the endpoints, we should check the entire arc to confirm that we have really been following one arc rather than hopping from one arc to another, and also to estimate the parameters of the arc more accurately.

After we find each arc, we search for other arcs. As starting points for these searches, we could use apparent points of intersection with the arcs already found; these would probably appear as large clusters of black pixels near or connected to an arc already found. If there are no such points, we could scan the image for a new starting point. As each arc is found, its pixels could be marked as accounted for. We do not want to remove these pixels from the image, however, because some pixels will lie on more than one arc, and removing them would alter the remaining arcs. We continue to search in the image until every black pixel is accounted for, either as part of an arc or as noise.

A variation of the above method would be to scan across the image from many directions to find some of the outer boundaries of the character. The outermost pieces could then be found and

stripped away, exposing the pieces in the interior. This process would continue until all of the pieces are found.

Another possible type of algorithm is based on finding edge elements and line elements, putting them together to form parts of segments and arcs, then finding intersections and endpoints, and finally integrating this information to identify the pieces. Since an edge element or a line element is a local feature of an image in the sense that it depends only on the pixels in a small part of the image, a parallel processing device using standard edge-detection methods could test for the existence, and maybe the orientation, of such elements, simultaneously at many points in the image. Nearby edge elements with the proper orientations would then be grouped together to form tentative partial arcs, which would "grow" as more elements are added to them. It is not clear how best to do this. We would need some sort of efficient search through the set of edge and line elements found. This could probably be done by a partly parallel computation. One possible method is the "Hough transform", described in Ballard and Brown (1982). Points where arcs intersect would appear to the first part of the process as large or irregular black clusters; their locations could be marked for later interpretation as the arcs begin to be identified. After all of the pieces are found, a final pass could be made to confirm the results and to estimate the parameters of each piece more accurately.

Once the pieces are found, the preprocessor or some other part of the system would center and scale them, and then compute

the five parameter values for each piece.

There are some images that will be difficult or ambiguous for any preprocessor. Because of the limited resolution, a segment or an arc appears as a long, thin set of black pixels with some thickness and with inexact boundaries. An example of the kind of problem that can occur is the following: If two line segments have a common endpoint and are such that the angle between them is close to 180^{0}, it will be hard to distinguish them from one long arc with a slight curvature, especially if we allow for some noise in the image. That is, the preprocessor might consider them as two short pieces, or as one long, slightly curved piece. A similar problem would be caused by two arcs of similar curvature that have a common endpoint at which they are tangent (or nearly so), as could occur in a "C"; it would be hard to tell just where one arc ends and the next begins. Problems would also be caused by two pieces that are partially overlapping for part of their length, so that it is hard to separate them, or by several pieces that intersect one another near a point, but whose intersections are not at exactly the same point. Such combinations would be difficult for any preprocessor.

More sophisticated preprocessors could be designed to cope with various complications. For example, if the images are noisy, a preprocessor could filter out some forms of noise. It could also remove other details in the images that are not relevant to the recognition problem. If the thickness of the pieces can vary, the preprocessor could be made to handle such pieces. Some systems have "thinning" algorithms for this

purpose; see Kahan et al. (1987). If the pixels can represent shades of gray, the preprocessor could adjust for overall brightness and contrast, and could be made to find pieces whose boundaries were either sharp or fuzzy. In order to give the specifications for an image preprocessor, it is not sufficient merely to list the image elements that the preprocessor is to identify; we must also describe the possible variability in the appearance of these elements, the nature of the background or context in which they might occur, and the possible kinds of noise that might be present in the images.

Finally, if an image is ambiguous, the preprocessor could give several possible decompositions of the image, or it could receive some form of feedback from the other parts of the system, in order to help it decide among various possible interpretations of the image.

12. ENCODING CRITICAL POINTS IN AN IMAGE

When we look at characters, we attach great importance to image features such as vertices, angles, intersections, isolated endpoints of pieces, and changes in the curvature of a smooth curve. Since such features are formed by the relative positions of two or more pieces of the character, they provide explicit information on how the individual pieces are related to one another. Thus, a possible extension of the representation described in this report would be to include these more complex features in the encoding of an image, so that they could form part of the read or write address when accessing the memory

system. If the system were given this information on how the pieces are related, rather than being given only information describing the pieces individually, it should be better able to recognize characters. Information on these kinds of features is of course contained implicitly in the parameters describing the pieces, in the sense that the information can be computed from those parameters, but the memory system is not able to make direct use of such implicit information.

I will consider a *critical point* in an image of a character to be a point where two or more pieces intersect, including a point on a smooth curve where one arc ends and another begins. I will also include terminations (that is, isolated endpoints of pieces) and dots in this category. More specifically, I will define a critical point to be either a dot, or a point with one or more segments or arcs going through it or radiating out from it in various directions, but not a point lying on only one segment or arc, unless it is an endpoint. For example, an "A" made of three line segments has five critical points. This definition is consistent with the above decomposition of an image into segments and arcs; thus, the representation of an image as a set of pieces can be enhanced by including descriptions of the critical points.

A critical point is a local feature in an image, in the sense that its existence can be detected by examining a small neighborhood about a point in the image. The part of the image seen in a small circular window centered at a critical point would appear as a line segment or a kind of "star". Most of the

critical points described in the definition above can be distinguished in this way from an ordinary point lying on only one piece. (Some critical points may not be identifiable until we begin to find the pieces in the image.) Thus, it should be possible to build a preprocessor to find both the pieces and the critical points in an image. In fact, identifying the critical points would help in accurately determining the parameters of the pieces.

I have not yet done any experiments with representations of these critical points. But since this seems to be a natural next step, I will describe a way to represent such points that is similar to the representation of segments and arcs given above.

A critical point can be described by giving the position of the point and the direction at which each segment or arc (if any) radiates away from it. Note that since I want to identify a critical point from what can be seen in a small neighborhood of it, an arc passing through the point will appear locally as two rays emanating from the point in approximately opposite directions. Hence, I will include both of these directions in the representation, as if they are parts of different arcs. The information can be encoded as a bit string as follows: First, the X and Y coordinates of the position of the point can be encoded as was done earlier for the coordinates of the center point of an arc. To encode the directions of radiation from the point, imagine a string of, say, 24 bits, arranged conceptually in a circle, like the circle of 12 bits in Section 9. I then define a sequence of 48 24-bit strings, like the sequence defined

earlier: Each string contains a block of two or three 1's, so that the Hamming distance between adjacent strings (and between the first and last strings) is 1. Each of these strings will represent 7.5^{0} of the circle of possible directions of radiation. Then, for each arc radiating from the critical point, I choose the bit string representing the arc's direction of radiation, and I take the logical OR of those strings. In other words, I set to 1 the two or three bits on the circle nearest to the direction of radiation of each arc, using a rule similar to that used for the circle of bits in Section 9. If the critical point is a dot, all of these bits will be 0.

This representation is independent of the order in which the radiating arcs may have been listed; that is, they are treated as an unordered set. Therefore the representation is unique, in that there is only one way to encode the critical point. It is also approximately continuous, in the sense that if an arc is moved slightly, or is added or deleted, there is only a small change in the bit string. Moreover, the length of the bit string is constant, no matter how many radiating arcs there are. This method is limited in resolution, however, depending on the length of the bit strings used. If there are several arcs radiating in similar directions, the bit string will have a blur of 1's, and the number of arcs and their individual directions will not be recoverable from the encoded information. (The same is true of our own visual systems, if we are allowed only a brief glance at the point.) But even in this case, two similar critical points will have similar encodings, in the sense that the Hamming

distance between the strings will be small. If higher resolution is desired, longer bit strings could be used.

If we use 24 bits to represent the set of directions of radiation from a critical point, we can define the set C of all possible representations of critical points to be a subset of the Cartesian product of a unit square (for the position of the point) and a 24-dimensional binary vector space (for the set of directions). To measure the similarity between two critical points, we can define a distance function for the set C; for example, we could use a weighted sum of the L2 distance between the positions of the two points, plus the Hamming distance between the bit strings representing the directions of radiation. The critical points found in an image would be represented by an unordered set of points in C. Thus, in this enhanced representation, an image would be represented by a set of points in the manifold M, together with a set of points in C.

I would like to thank the members of the SDM group for their many helpful comments during the course of this work.

REFERENCES

Albus, J. S. (1981). *Brains, Behavior, and Robotics.* Byte Books. Peterborough, N. H.

Ballard, D. H. and C. M. Brown (1982). *Computer Vision.* Prentice-Hall. Englewood Cliffs, N. J.

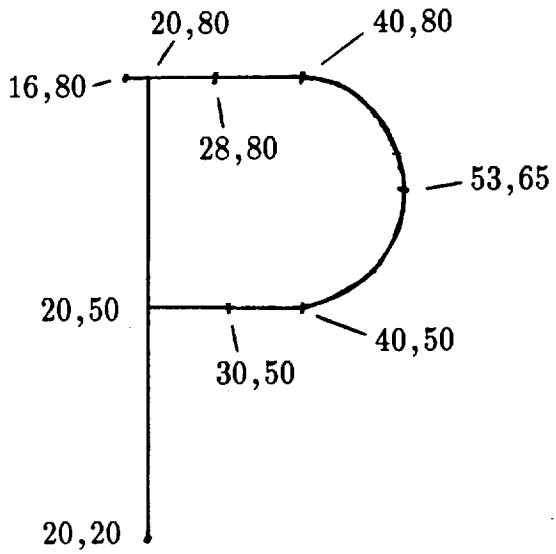
Kahan, S., T. Pavlidis, and H. S. Baird (1987). On the Recognition of Printed Characters of Any Font and Size. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-9,* 274-288.

Kanerva, P. (1988). *Sparse Distributed Memory.* MIT Press. Cambridge, Mass.

Olshausen, B. (1988). A Survey of Visual Preprocessing and Shape Representation Techniques. RIACS Technical Report 88.35.

Parent, P. and S. W. Zucker (1985). Trace Inference, Curvature Consistency, and Curve Detection. Computer Vision and Robotics Laboratory Technical Report CIM-86-3. McGill University.

PAGE ___ INTENTIONALLY BLANK

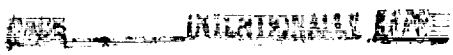


Piece	Description	Endpoint	Midpoint	Endpoint
1	Vertical segment	20,20	20,50	20,80
2	Upper hor. segment	16,80	28,80	40,80
3	Curve at right	40,80	53,65	40,50
4	Lower hor. segment	40,50	30,50	20,50

PARAMETER VALUES AND BIT STRINGS

Piece	X	Y	Size	Angle	Angle
1	.26	.50	.79	270	90
	011000	001100	000111	001100001100	
2	.39	1.00	.37	180	0
	011100	000011	011100	100001100001	
3	.70	.75	.45	131	229
	000110	000111	011100	000111111000	
4	.43	.50	.31	0	180
	011100	001100	011000	100001100001	

Figure 1: This character was drawn on graph paper and broken up into four pieces. The pieces were entered into the computer in an arbitrary order, by typing in the coordinates of the endpoints and the midpoint of each piece. The program then computed the five parameter values and the corresponding bit strings for each piece.



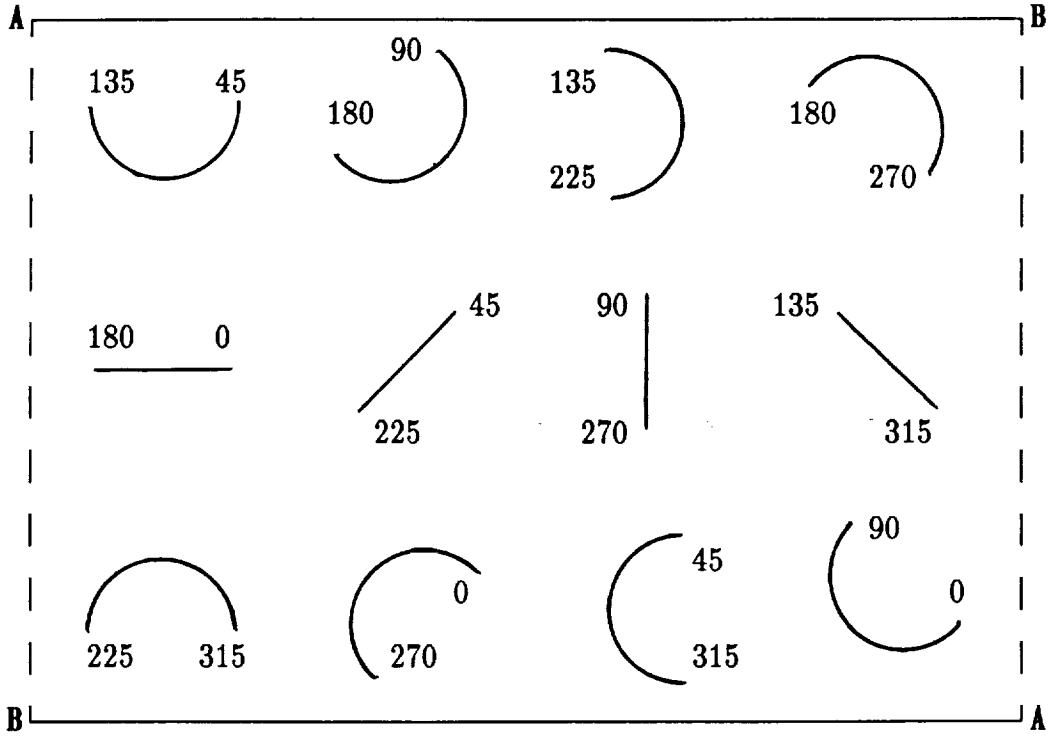


Figure 2: The Möbius strip of Section 6, cut along the vertical line **AB** and laid flat. Each point on the strip represents a segment or an arc. Some of the segments and arcs, with their corresponding pairs of angles, are shown above. The unordered pair of angles for an arc is found by drawing a ray from the midpoint of the arc to each endpoint, as described in Section 5.

The points representing line segments lie along the "equator" of the strip. Moving horizontally on the strip corresponds to rotating an arc, while moving vertically corresponds to bending an arc without rotating it. If you begin at the upper left, at **A**, and move along the top of the strip to **B**, the next step after the arc at the upper right is the arc at the lower left; continuing along the bottom of the strip from left to right, you then go from the lower right, at **A**, back to the upper left.