

REPRESENTING SHAPES FOR VISUAL RECOGNITION

by

Donald David Hoffman
B.A. University of California, Los Angeles
(1978)

SUBMITTED TO THE DEPARTMENT OF PSYCHOLOGY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
OF THE DEGREE OF DOCTOR OF PHILOSOPHY IN
COMPUTATIONAL PSYCHOLOGY
AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1983

© Massachusetts Institute of Technology 1983

Signature of Author _____

Department of Psychology, May 11, 1983

Certified By _____

Professor Whitman Richards, Thesis Supervisor

Accepted By _____

Professor Richard Held, Chairman, Department of Psychology

MASSACHUSETTS INSTITUTE
OF TECHNOLOGY

JUN 03 1983

Archives
LIBRARIES

REPRESENTING SHAPES FOR VISUAL RECOGNITION

by

Donald David Hoffman

Submitted to the Department of Psychology
on May 11, 1983 in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy
in Computational Psychology

ABSTRACT

With but a glance we can recognize a seemingly unlimited variety of objects. Our proficiency at recognition, however, belies the complexity of the task and the multiplicity of information sources we exploit in elegant coordination. This thesis explores how one source of information, the shapes of objects, can provide an initial clue to the identity of objects. We ask, What properties of the bounding contours and surfaces of objects should be represented, whether by people or vision machines, to facilitate visual recognition? How can these properties be computed from images? What principles and natural constraints underly our choice of properties?

One principle is key. Our descriptions of a shape should not change with every small shift in our viewing position. That is, our descriptions should decouple the shape of an object from its position and orientation with respect to the viewer. This principle is used, along with computational arguments and psychophysical observations, to motivate a scheme for carving curves and surfaces into parts and then describing those parts. For smooth plane curves the scheme involves extrema and inflections of signed curvature. For smooth surfaces the scheme involves extrema and inflections of the principal curvatures along lines of curvature. In both cases minima of curvature define part boundaries; maxima and inflections anchor the internal part descriptions. The scheme is then extended to include curves with cusps and surfaces with discontinuities of the tangent plane.

Thesis Supervisor: Whitman Richards, Professor of Psychophysics

Biography

I received a B.A. summa cum laude in Quantitative Psychology from the University of California at Los Angeles, June 1978. My junior and senior years at UCLA were supported by bachelor fellowships from the Hughes Aircraft Company. In July 1976 I joined Hughes Aircraft as a student engineer, becoming a Member of the Technical Staff in June 1978. Among my projects at Hughes were the Digital Avionics Information System (DAIS), for which I wrote the majority of the display software, and the Advanced Medium Range Air to Air Missile (AMRAAM) real-time three-dimensional color display system, for which I was project engineer. I began graduate study at the Massachusetts Institute of Technology in 1979, working jointly in the Department of Psychology and the Artificial Intelligence Laboratory. My graduate study was supported by Hughes Aircraft Company fellowships.

Publications

"The interpretation of biological motion", *Biological Cybernetics*, **42**, 3, 197-204, 1982. (with B.E. Flinchbaugh). Also available as MIT AI Memo 608.

"Inferring local surface orientation from motion fields", *Journal of the Optical Society of America*, **72**, 7, 888-892, 1982. Also available as MIT AI Memo 592.

"Equation counting and the interpretation of sensory data", *Perception*, 1983 (in press). (with W.A. Richards and J.M. Rubin). Also available as MIT AI Memo 622.

"Interpreting time-varying images: The planarity assumption", *IEEE Workshop on Computer Vision*, 92-94, 1982.

"Representing smooth plane curves for recognition: Implications for figure-ground reversal", *Proceedings of the International Conference of the American Association for Artificial Intelligence*, 5-8, August 1982. (with W.A. Richards).

"The precision attack enhanced (PAE) remote terminal simulator", *Hughes Aircraft Co. Technical Document*, 1978.

Acknowledgements

I thank the members of my committee, Chris Brown, Shimon Ullman, Berthold Horn, and Whitman Richards for their comments, criticisms, encouragement, and careful reading of earlier drafts of this thesis. I particularly thank Whitman Richards for being an ideal advisor and friend, always willing to make time to talk when I needed it, unselfish with his ideas, creative in his thoughts, treating me and his other students with respect. It was largely my interactions with him that made my graduate work at MIT enjoyable, challenging, and interesting.

I thank my parents David and Loretta Hoffman for their encouragement, their genuine interest in my research, and especially for their prayers. I thank my wife Rebecca for her patience, love, and support, and my daughter Melissa for many playful days and sleepless nights.

Several faculty members, staff, and students critiqued earlier versions of this thesis. Of particular help were the comments of Molly Potter, Steven Pinker, Alan Yuille, and Joe Shipman.

Finally, I thank John Rubin, Sandy Pentland, Andy Witkin, Aaron Bobick, and Joseph Scheuhammer, erstwhile fellow graduate students who shared ideas, interests, and many experiences.

Contents

	<i>Page</i>
Title	0
Abstract	1
Biography	2
Acknowledgements	3
Contents	4
1. Introduction	6
1.1 Three observations	6
1.2 The problem	7
1.3 Recognition	7
1.4 Representations	10
1.5 Representations for recognition	13
1.6 Previous 2-D representations	15
1.7 Generalized cylinders	18
1.8 Philosophy of the approach	22
2. Representing plane curves	25
2.1 Overview	25
2.2 Notation and terminology	25
2.3 Goals	26
2.4 Decoupling rotation and position	27
2.5 Scale independent parts	27
2.6 Contour codons	29
2.7 A perceptual note	34
2.8 Positive minima	35
2.9 Quantitative description of parts	36
2.10 Curves with cusps	37
2.11 Codon description extended for cusps	39
2.12 Codon hierarchies	42
2.13 Summary	43
3. Representing surfaces	45
3.1 Surfaces of revolution	45
3.2 Notation and terminology	46
3.3 Goals	47
3.4 Decoupling rotation and position	47
3.5 Scale independent parts	48
3.6 Segmentation of developable surfaces	49
3.7 Segmentation of surfaces of revolution	52
3.8 Segmentation of the torus	56
3.9 Segmentation of the general ellipsoid	58
3.10 Flattened surfaces of revolution	63
3.11 Surfaces with discontinuities	67
3.12 Elbows	68
3.13 Summary	71

4. Discovering natural scales	72
4.1 Differential quantities on fractal curves	73
4.2 Natural scale	74
4.3 Natural scale and effective dimension	87
4.4 Summary	90
5. Relation to human perception	91
5.1 Predictions	91
5.2 Experiments	92
5.3 Demonstrations	97
5.4 Summary	106
Epilogue	108
References	111
Appendix 1. Lisp code	114

1. Introduction

1.1. Three observations

We easily interpret squiggles or grey dots on a sheet of paper as objects in various spatial relationships. This ability is demonstrated in figure 1-1. From this figure we can make three straightforward but informative observations about our recognition of visual objects. First, we recognize most of the objects in the figure although we have no prior idea what those objects might be when we first inspect the figure. Second, we can recognize several of the objects even though the figure only informs us of their shapes. Finally, we do not immediately recognize those objects for which the figure provides an insufficient amount of shape.¹

What do these observations tell us about our visual recognition of objects? To understand the import of the first observation we must realize that recognition, reduced to its essentials, requires matching a description of what is seen against a store of descriptions already in memory. This matching could proceed in one of two basic ways – brute force or intelligent guessing. A brute force matcher simply dredges up the descriptions in memory one by one and compares them with the visual description until it finds a match. On average the brute force matcher must examine half of the items in its memory, a formidable task for memories of the magnitude we presumably possess. An intelligent guesser only searches through certain items in memory, being informed in its selection either by contextual knowledge, such as the time and place, or by properties of the visual description of the object, or by both. Assuming that we are intelligent guessers, the first observation indicates that though ordinarily we may direct our forays through memory using both context and properties of the visual description of the object, we can if necessary drop our reliance on context and use only properties of the description. We can do this, according to the second observation, even when the visual description of the object is restricted to statements about its geometry. As indicated by the third observation, however, the geometrical descriptions must exceed some minimum level of richness or we will fail to recognize the object.

These descriptions of shape, their construction from images, their necessary properties, their likely properties, and their role in initiating the first searches through memory, these are the subjects of this investigation. We restrict our attention to these topics, not because context is unimportant, not because the more general problem of recognition is less interesting, but because the more elementary topics are important building blocks for a

¹Quantification of shape is discussed in chapters two and four.

comprehensive theory of visual recognition and because they form a well circumscribed set of issues with the promise of a greater degree of tractability than the general problem.

1.2. The problem

What is the precise problem to be solved? I propose the following:

- How can early representations of shape be transformed into representations suitable to initiate the recognition process?

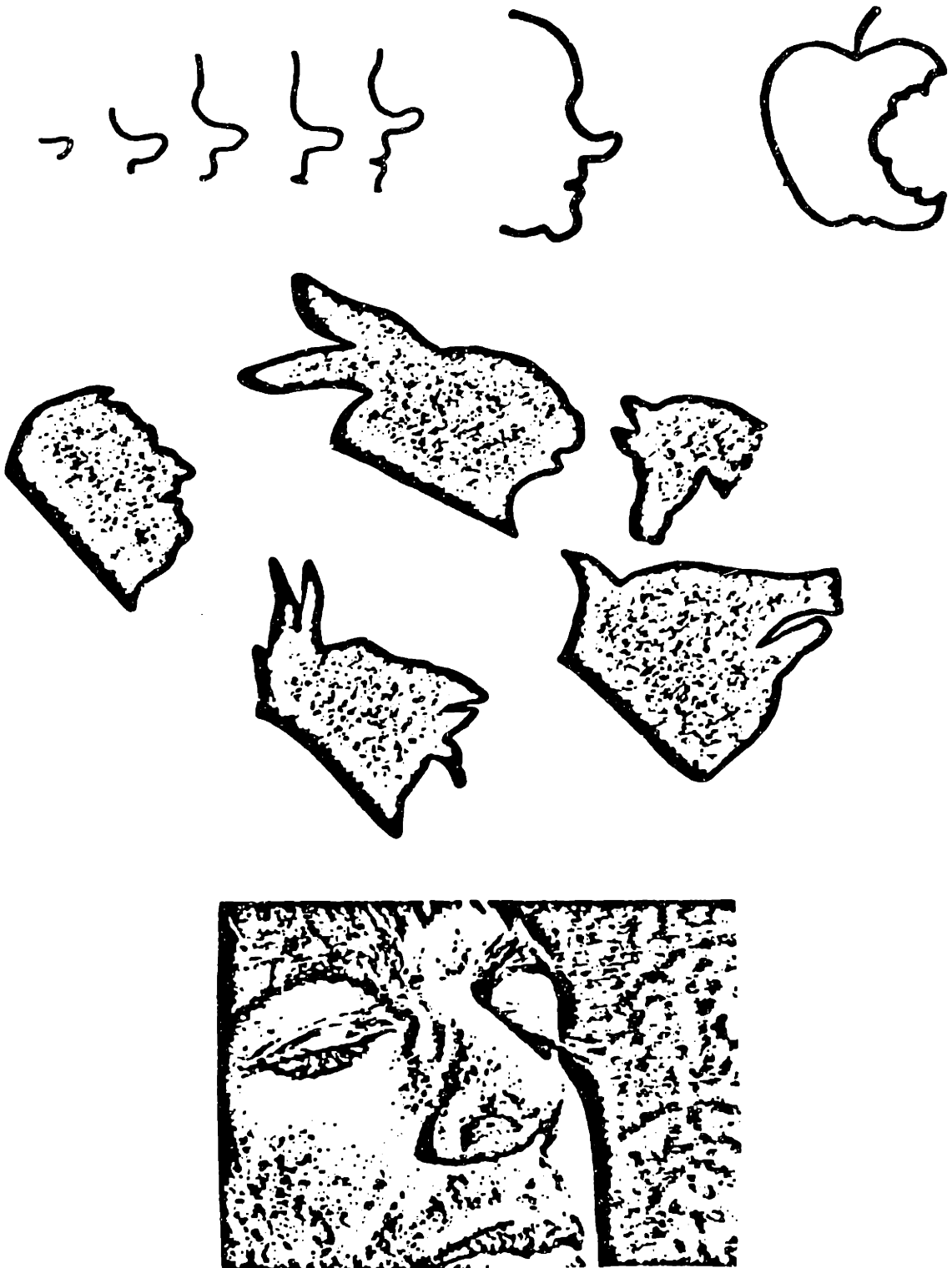
Much is packed into that question. To define the problem more clearly, in this chapter we will look briefly at the meaning of the terms *representation* and *shape*. We will then explore what is meant by the phrases *early visual representations of shape* and *representations appropriate for recognition*.

Once the problem question is defined we will spend the latter half of the chapter examining the relevant literature, using this literature as a means to further clarify the problem. The chapter closes with a statement of the philosophy of the approach. Chapter 2 develops a representation for recognition of plane curves. Chapter 3 extends this representation to surfaces. Chapter 4 introduces the notion of "natural scale", a necessary notion if the theories of chapters 2 and 3 are to be applied to the images of real shapes, not just to shapes admitting a concise analytic specification. Chapter 4 then describes an implementation of the representation for curves developed in chapter 2. Chapter 5 explores the relation between these representations and the results of some experiments on human visual recognition. The appendix is a listing of the lisp code described in chapter 4.

1.3. Recognition

In the simplest case, recognition involves matching a description with an item in memory. For example, the description "has four wheels and flies" might be matched with the item "airplane" in memory, which has a similar, though probably more detailed description. Of course things can get arbitrarily more complicated. The first attempted matches may not succeed, but may initiate efforts to augment the original description in specific promising details, while in parallel different parts of memory are being searched. Such extra effort might turn up a different match for "has four wheels and flies", such as "trash truck".

Figure 1. Some squiggles and grey-level images



Looked at from a broader perspective, recognition is a paradigm example of a form of nondemonstrative inference – induction. An inductive inference is an argument whose conclusion is supported by its premises but is not logically guaranteed to be true by the premises. For visual recognition the role of premises is played by the descriptions of shape, color, motion and texture provided by earlier vision, together with any relevant background knowledge or contextual information possessed by the observer. The conclusion, via nondemonstrative inference, is the identity of the visual object.²

When making an inductive inference one is well advised to assess all the relevant knowledge in one's possession before making a conclusion. There is evidence that human observers do in fact consider background knowledge and contextual constraints in deciding the identity of objects in their field of view. For instance, an ellipse in one context might be taken to be an eye (e.g., in the context of a schematic face), while in another context it might be taken as a foreshortened circle. The unhappy conclusion is that visual recognition is not normally performed in a modular fashion (see Fodor 1982 on modularity). That is, anything known by the observer could potentially be used to help decide what is being observed. This nonmodularity does not promote research tractability.

To circumvent this, the strategy taken here is to investigate a subproblem of the visual recognition problem, a subproblem for which there is evidence of functional modularity. This evidence was discussed before and illustrated in part by figure 1-1. In brief summary, although one cannot reasonably predict the contents of the figure prior to seeing it, and although for several objects only geometrical information is presented, the objects contained in the figure are, with few exceptions, readily recognized.

It appears, then, that we can recognize some objects using shape information alone. We can do so even without prior expectations regarding object identity and, consequently, no prior contextual clues as to what parts of memory should be searched first or what background knowledge is applicable.³ This raises the question, How can the recognition process get started without prior expectations or contextual cues? How does one decide what parts of memory to search first and what background knowledge to exploit?

It seems clear, in this case, that the only possible answer is that the decision to explore a certain part of memory, or to invoke specific background knowledge, must be based on

² Realizing that visual recognition is a species of induction, we can gather insights on our problem from the relevant philosophical literature. Goodman (1955), for instance, notes that the predicates used to state one's premises can dramatically alter the conclusions drawn. In fact, given a set of data, one can choose the predicates used to describe that data such that conflicting inductive conclusions are supported. Further, for any conclusion one chooses, there exists a manner of describing the data which licenses the desired conclusion. Thus, in the case of the visual recognition of shapes, one's choice of primitives and of organization for the representations is critical and must be well motivated. Much of the discussion of chapters 2 and 4 is devoted to this issue.

³ Again, the functional modularity claim here is that we can use shape in isolation, not that in general we do use shape in isolation.

computations involving the object's shape alone. Somehow there must be an analysis of the object's shape, performed independently of general background knowledge⁴ (i.e. performed "bottom up"), which provides a sufficiently useful first key or index into memory to get the recognition process started. This suggests the utility of a set of bottom up rules for the initial analysis and description of curves and surfaces. The majority of this investigation is devoted to discovering what these rules might be. These rules provide an important answer to our original question, How can early visual representations of shape be transformed into representations suitable to initiate the recognition process?

1.4. Representations

A representation is a formal system which models some domain. A familiar example is the common road map. A road map is a representation which models, among other things, the layout of streets, railroads and important buildings of a region. To use a representation effectively as a model one must know what are its primitive symbols and what entities of the domain they represent. On a road map the primitive symbols and their meanings are typically indicated in a key. The key might tell, for instance, which symbols stand for airports, roads and schools. In essence, the key states what information about the domain is made explicit in the representation. Other information may be contained implicitly in the representation, such as, in the case of the road map, how many government buildings lie between the airport and the high school along the shortest path between them. Recovering implicit information from a representation generally requires more work than recovering what it makes explicit. Thus it is important to design a representation to make explicit the information that is needed to accomplish the task at hand.

1.4.1. Representations of shape

The word *shape* is used here to refer to the geometry of an object's surface in three dimensions, or to the geometry of a region's bounding contour in two dimensions. A *shape representation*, then, is a formal system which makes explicit some aspects of the geometry of the bounding surfaces or contours of objects.

Unfortunately, it appears that the aspects of shape most easily and directly made explicit by early visual processes are not the aspects best suited for the task of recognition. This statement will be defended shortly. The consequence, however, is that somehow the early shape representations must be transformed so that the appropriate, and hitherto implicit, aspects of shape are made explicit. This leads us back to our original question:

⁴More precisely, the only background knowledge available is knowledge of geometry and topology.

How can early visual representations of shape be transformed into representations suitable to initiate the recognition process?

1.4.2. Early visual representations of shape

The earliest visual representation available to a system is generally a two-dimensional image which gives the light intensity⁵ (and perhaps chromaticity) at each point in the image. However the starting point for the work presented in chapters 2–4 is not the raw image. Instead it is assumed that *bitmap* representations are available for plane curves and that *depth map* representations are available for surfaces in three dimensions. In this section these early visual shape representations are described briefly.

Two points about these early representations should be noted: 1) they represent the visible curves or surfaces completely, and 2) they represent curves and surfaces in a manner which depends critically on the position of the viewer.

1.4.3. Early curve representations: Bitmaps

A bitmap representation is a two-dimensional array (or grid), $B(x, y)$, which contains a 1 in locations where a curve segment is present, and a 0 otherwise. A bitmap representation of the curve $y = x$, for example, would contain 1's along one diagonal of the array and 0's elsewhere.

Considerable work on the detection of edges in images makes it reasonable to expect that a vision system can produce something equivalent to a bitmap representation of curves (see Marr & Hildreth 1980; Hildreth 1980; Zucker, Hummel & Rosenfeld 1977; Horn 1973; Falk 1972). However there are several problems yet to be solved before this endeavor can be considered successful, particularly problems regarding the detrimental effects of noise and variations in illumination on the connectivity and shape of the curves computed by current approaches (Richards, 1983).

1.4.4. Early surface representations: Depth maps

A depth map representation is a two-dimensional array which contains in each array entry the (relative) distance between the viewer and the nearest patch of surface in the corresponding visual direction.⁶ The terms *2½D-sketch* (Marr 1982) and *intrinsic image*

⁵More precisely, the image irradiance is given.

⁶Of course, contiguous elements of the array needn't be located next to each other in whatever physical system actually implements the array. This non-isomorphism between array element contiguity and physical location of the storage of elements is common in the storage of arrays on computers.

(Tenenbaum & Barrow 1978) are also commonly used to refer to some variant on the notion of depth maps. Considerable work on the problems of inferring three-dimensional shape from shading (Pentland 1982; Bruss 1981; Ikeuchi & Horn 1981; Horn 1975), texture (Witkin 1981; Stevens 1981; Ikeuchi 1980), motion (Hoffman 1982; Hoffman & Flinchbaugh 1982; Prazdny 1980; Longuet-Higgins & Prazdny 1980; Ullman 1979; Koenderink & Van Doorn 1976) and stereo (Grimson 1981; Marr & Poggio 1979) makes it reasonable to expect that a vision system can produce something equivalent to a depth map representation of surfaces.

1.4.5. Properties of early shape representations

To a large extent the nature of these early shape representations is dictated by what can be computed from images with relative ease. They are not tailored in any way to the requirements of recognition, nor should they be since, presumably, they are also to be employed in tasks other than recognition, such as manipulation.

One consequence of this is that a shape description couched in one of these representations depends essentially on the relative positions of the observer and the observed. This *viewer dependence* of the descriptions can be illustrated by considering analytic descriptions of the ellipsoid $ax^2 + by^2 + cz^2 = 1$. If the observer's line of sight lies along the z axis, the description is of the form $x(u, v) = (u, v, \sqrt{(1 - au^2 - bv^2)/c})$, where u and v are the observer's coordinates. If the line of sight moves to the x axis, the description changes to $x(u, v) = (u, v, \sqrt{(1 - bv^2 - cu^2)/a})$. Clearly any change in viewing geometry changes the resulting description. It is not difficult to convince oneself that the same is true of the bitmap and depth map shape representations. As will be seen, this is an undesirable property if one's task is recognition.

There is an important sense in which these early representations are complete representations of the visible contours and surfaces. The sense is that, within the limits of resolution imposed by the imaging system, these representations in principle contain sufficient information to specify uniquely the geometry of the visible portions of surfaces and contours. For example, from a depth map one can compute the coefficients of the first and second fundamental forms (Spivak 1972 volume 2) at each point, which, by the fundamental theorem of surfaces (Do Carmo 1976), specify the surface uniquely up to rotation and translation. Consequently, implicit in these early representations is all possible information regarding the geometry of the visible curves and surfaces, even though only certain aspects of shape are made explicit in these representations, and even though the aspects made explicit are not directly suitable for recognition. Thus our goal is to determine which implicit aspects of shape need to be made explicit for the task of visual recognition, and how this transformation is to be accomplished.

1.5. Representations for recognition

We have hinted that the early representations of shape already described are inappropriate for the task of visual recognition, their inappropriateness due not to a dearth of information, but to the form of the representations. The wrong information is made explicit. The information needed for recognition is buried in an implicit form.

What information should be made explicit, and in what form, to expedite the recognition process? How are we to evaluate a representation for recognition? Perhaps the most thorough analysis of this problem is given by Marr and Nishihara (1978) who propose three criteria for judging the usefulness of a representation for shape recognition. These criteria, briefly, are: 1) *computability*, the feasibility of computing the representation from images,⁷ 2) *scope and uniqueness*, the range of shapes that can be described using the representation and how close to canonical are the descriptions for each shape, and 3) *stability and sensitivity*, the degree to which the representation captures the similarities between shapes within its scope while simultaneously noting important subtle differences.⁸ They note that these criteria affect three aspects of a representation's design: 1) The *coordinate system* chosen (whether or not it is dependent on the viewing geometry), 2) the *primitives*, the basic units of shape in the representation, and 3) the *organization* imposed on the shape descriptions by the representation. The next two sections examine these criteria and design considerations in more detail.

1.5.1. The criteria in more detail

The computability criterion is the obvious requirement that a proposed representation for recognition should, in principle, be possible to construct from the available image data. This notion can be extended to the criterion that, other factors being equal, a representation is preferable if it requires less computation for its construction.

The scope/uniqueness and stability/sensitivity criteria can be understood by a simple example. Suppose we propose a representation which assigns the description "99" to every shape. Such a representation has the broadest possible scope; since every shape imaginable is, by stipulation, given the description "99" no shape is outside the scope of the representation. This representation also satisfies the uniqueness criterion. Regardless of viewing geometry, or any other factor, each shape is assigned the same description every

⁷Marr and Nishihara call this criteria *accessibility*.

⁸The stability/sensitivity issue has long been a subject of philosophical discussion under the names "unity and diversity" and "form and freedom".

time it is seen. Since there is no possibility for multiple descriptions to arise for a given shape, the uniqueness criterion is satisfied.⁹

The patent absurdity of this representation is due to its severe violation of the stability/sensitivity criterion. The description "99" in no way reflects the stable properties of shapes while simultaneously capturing the subtle, yet salient, nuances which differentiate them. Note the difference here between stability and uniqueness. A representation is unique for a *single* shape if the same description is computed time and again when the shape is seen. It is stable for a *class of shapes* if the description makes explicit those aspects of shape which are common to the class.

Evaluating one of the early visual representations of shape against these criteria is also a useful exercise, not only to understand the criteria but to understand why the early representations are inappropriate for recognition. Consider depth maps. The depth map representation of surfaces has the broadest possible scope since every surface can be assigned a depth map. However the uniqueness criterion is badly violated. Any change in viewing geometry will, in general, change the depth map description assigned to a surface. Whereas the uniqueness criterion states that ideally only one description should ever be produced for a shape, in principle the number of depth map descriptions of a surface is unlimited (assuming unlimited resolution). This viewer dependence of the early visual shape representations will be dealt with in more detail in chapters 2 and 4.

Depth maps also violate the stability/sensitivity criterion. Nothing in depth map descriptions makes explicit the commonalities among members of a class of shapes while simultaneously noting the variations. Depth maps of various goblets, for example, do not make explicit the fact that they all have three basic parts: a base, a stem, and a bowl. Nor do they make explicit how the shape of one stem varies from that of another.

Finally, depth maps are computable from images as evidenced by work on inferring three-dimensional shape from shading, texture, motion and stereo. Thus at least the weaker form of the computability criterion is satisfied.

1.5.2. The design considerations in more detail

Coordinate System. An important design consideration when trying to satisfy the uniqueness criterion is the type of coordinate system used in the representation. As noted by Marr and Nishihara, there are two basic types of coordinate system. A representation whose descriptions depend on the viewing geometry is said to employ a *viewer dependent*

⁹ Remember that multiple descriptions are undesirable for recognition because as the number of potential descriptions of a shape increase the memory needed to store them and the computations needed to find the right match also increase.

coordinate system. Since multiple descriptions can arise using a viewer dependent coordinate system, representations intended for recognition should instead be designed using *viewer independent* coordinate systems. Viewer independent coordinate systems can be established by exploiting the geometry of the object of interest. Using the natural axes of shapes is a popular method for doing this.

Primitives. The primitives of a representation are its basic units of shape. In a depth map the primitive is the distance, from the observer's current viewpoint, to each point on an object. Among representations intended for recognition two basic types of primitives have been proposed: region based and boundary based. A region based primitive makes explicit properties of the volume (in three dimensions) or area (in two dimensions) occupied by a shape. The most common region based primitive is the axis, both in two dimensions and three dimensions. A boundary based primitive makes explicit properties of the surface (in three dimensions) or bounding curve (in two dimensions) of a shape. The representations proposed in chapters 2 and 3 use boundary based primitives.

Organization. Finally, the structure or organization of a representation can be used to encode various properties of shapes such as the spatial relationships of the primitive shape units. Particularly useful is a hierarchical organization of the primitives which assigns descriptions of more prominent shape features to more prominent positions in the hierarchy and descriptions of finer details to lower positions (Marr and Nishihara 1978; Hollerbach 1975). Such an organization facilitates the simultaneous satisfaction of the stability and sensitivity criteria.

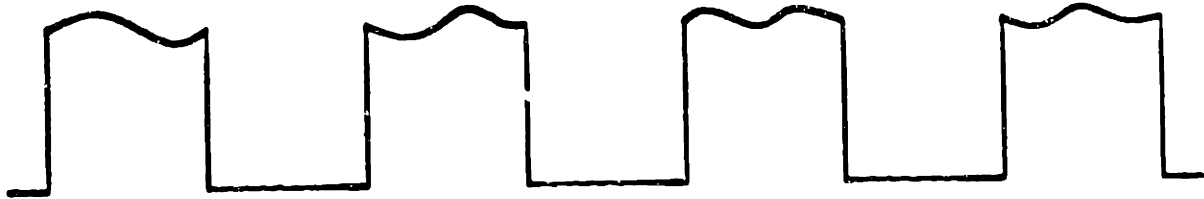
1.6. Previous 2-D representations

In this section we examine some of the previous literature on representing curves with the intent to clarify the issues already discussed.

1.6.1. Fourier descriptors

The method of Fourier descriptors represents a plane curve by the coefficients of its Fourier transform. This representation can be constructed with relative ease once the curve has been isolated in an image and therefore satisfies the computability criterion. Strictly speaking, its scope is limited to periodic curves, but it has been shown to represent arbitrary curves quite well. If the coefficients of the Fourier series are properly parametrized,

Figure 2. Stability & sensitivity is not low versus high frequency.



the Fourier description can satisfy the uniqueness criterion by being invariant under two-dimensional rotations, translations, and uniform scaling.

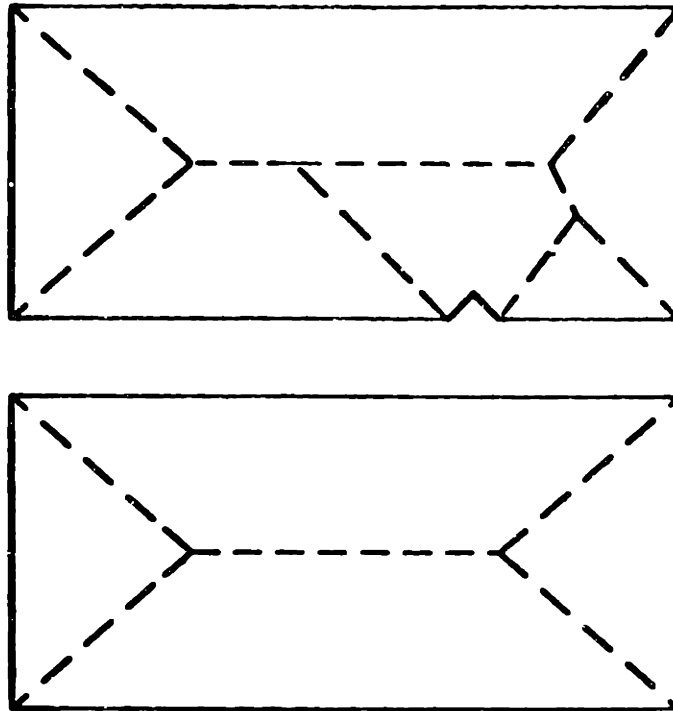
It might also seem that Fourier descriptors satisfy the stability/sensitivity criterion since low frequency Fourier components are decoupled from the higher frequency ones. However the decoupling needed to separate the stable aspects of a shape from its finer aspects is in the *spatial* domain, not the frequency domain.¹⁰ For example, in figure 1-2, a natural decomposition of the curve is into a square wave, each cycle of which has its own unique minor pattern of squiggles superimposed. The square wave is the basic form (to be captured by a representation satisfying the stability criterion), the minor squiggles constitute the freedom (to be captured by a representation satisfying the sensitivity criterion). But note that this decomposition is not one of low versus high frequencies. In fact the square wave contains higher frequencies than the minor squiggles. Low pass filtering the curve would not necessarily eliminate the minor squiggles, and would simultaneously eliminate the high frequencies needed for the square wave.

1.6.2. Symmetric axis and smoothed local symmetry representations

Blum (1973) introduced a representation of two-dimensional shapes that exploits natural axes of symmetry. These axes can be found by taking the union of maximal disks that touch at least two points of the bounding contour of the shape. Equivalently, if one set fire to the entire boundary of the shape and noted where the fires first met in its interior, the resulting contours would be Blum's axes of symmetry.

¹⁰In addition, some of the predicates of shape description which are convenient in the spatial domain, such as "cusp", are represented by an infinite series in the Fourier domain.

Figure 3. Stability/sensitivity violation.



The symmetric axis representation is easily derived from images, satisfying the computability criterion. Its scope is all two-dimensional shapes with closed bounding contours. With a modicum of effort the axes could be represented in a manner that is invariant under two-dimensional rotations, translations and uniform scaling, thus satisfying the uniqueness criterion (in two dimensions, not three). However, as noted by Agin (1972), the symmetric axis representation can violate the stability/sensitivity criterion severely. Figure 1-3, taken from Agin, shows how a minor change in a bounding contour can dramatically change the entire representation. Efforts are underway to repair this deficiency using a different means of computing axes of symmetry, called the method of smoothed local symmetries (Brady 1982a, Brady 1982b).

1.6.3. Contour representations using curvature

As will be explained in more detail in chapter 2, representations of bounding contours using curvature decouple the rotation and translation of a contour from its shape. More work must be done, however, to decouple the scale and thus satisfy the uniqueness criterion. Constructed properly, a curvature representation can have all smooth contours within its scope, and can satisfy the stability/sensitivity criterion (chapter 2). The computability of such a representation from images is an open question, but first steps toward an implementation are discussed in chapter 2.

Figure 4. Attneave's cat



Curvature has been used by several investigators to segment a curve into parts.¹¹ Attneave (1954) has demonstrated that extrema (minima and maxima) of curvature are perceptually salient to human observers. In one clever figure, he redrew a line drawing of a cat by connecting points of high curvature with straight lines. The result, shown in figure 1-4, is quite recognizable. Duda and Hart (1974) suggest as a result that extrema of curvature might be used for curve segmentation. This is also proposed by Brady (1982a). Hollerbach (1975), in his analysis of Greek vases, segments their bounding contours into parts at zero-crossings of curvature. For computational reasons, and to account for the fact that a curve seems to have different parts when it undergoes a figure-ground reversal, in chapter 2 we propose that minima of curvature, not extrema or zeroes, be used for curve partitioning. For curves with C^1 discontinuities this rule is extended to partitioning at concave, but not convex, cusps.

1.7. Generalized cylinders

Since Binford's (1971) introduction of a class of shapes called generalized cylinders, much work has been devoted to developing them as an axis-based means of representing three-dimensional shapes for recognition (Brooks 1981; Marr & Nishihara 1978; Nishihara 1977; Marr 1977; Vatan 1976; Nevatia 1974; Agin 1972). A generalized cylinder is more

¹¹Partitioning a curve into parts is an important step toward designing a representation which satisfies the stability/sensitivity criterion. More on this in chapter 2.

general than a cylinder in that it may have a curved axis, its cross section need not be circular and the cross section may be scaled as a function of position along the axis.

A representation whose primitives are generalized cylinders has a scope restricted to those shapes naturally decomposed into (a hierarchy of) generalized cylinders, such as the trunks and limbs of animals. Faces, for instance, are not naturally decomposed into generalized cylinders and are thus outside the scope of these representations.¹²

An advantage of generalized cylinders is that representations employing them can satisfy the uniqueness and stability/sensitivity criteria. For example, Marr and Nishihara propose that the cylinders be organized in a hierarchy, with cylinders capturing larger portions of a shape positioned higher in the hierarchy. The position and orientation of a cylinder is specified relative to the one above it in the hierarchy. This effectively decouples the overall disposition of a shape from the shape itself, allowing uniqueness of description. The hierarchical organization decouples the larger and more stable aspects of a shape from the finer details, satisfying the stability/sensitivity criterion.

Unlike the previous representations discussed, the most serious issue with generalized cylinders is computability. There is as yet no satisfactory way to derive generalized cylinder descriptions from depth maps or images despite much effort on the problem. Several approaches to the problem proposed in the literature are examined by Nishihara (1977) and found wanting. Rather than review these again, we will examine the approach proposed by Nishihara.

Nishihara (1977) proposed a "ridge operator" for finding the axes of generalized cylinders. The ridge operator looks at local regions of a depth map and, as the name suggests, determines for each region if there is a ridge passing through. A ridge is a local extrema (or a contour of local extrema) in the depth map. The local ridge assertions are linked into chains to form the axes.

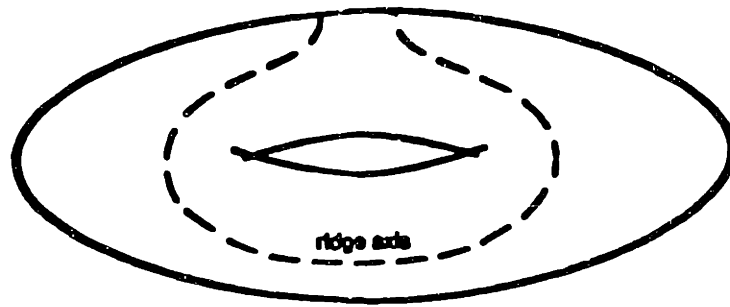
In general a ridge operator axis is not uniquely associated with a locus of points on the surface nor with the volume defined by the surface. Thus the axes generated by the ridge operator violate the uniqueness criterion.¹³

As an example of this violation of the uniqueness criterion, consider figure 1-5 which illustrates the differences in axes picked out by the ridge operator in two different views of a torus (a generalized cylinder with a planar curved spine and a constant sweeping rule, resembling a doughnut). When viewed from above, the axis defined by the ridge operator is a closed planar contour. When viewed obliquely, the axis becomes open and non-planar.

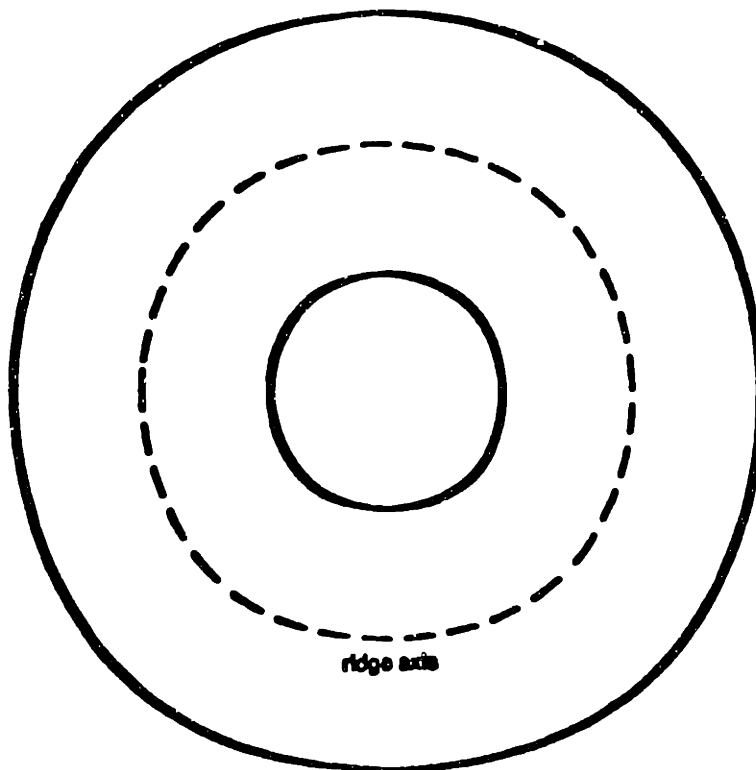
¹²The observation that faces are easily recognized, but not easily represented by generalized cylinders, was the original impetus for developing the representations presented in chapters 2 and 3.

¹³Or, conversely, the scope of the ridge operator is limited.

Figure 5. Ridge operator axes on a torus

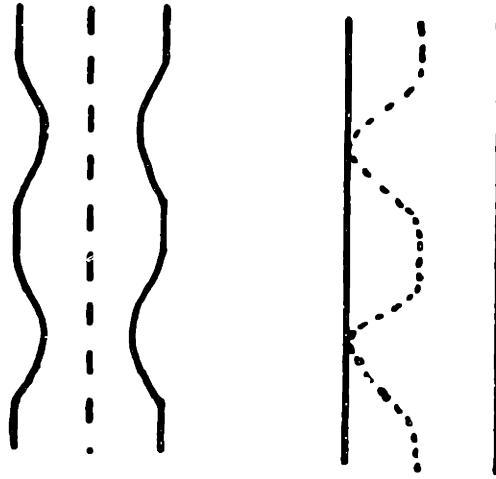


SIDE VIEW



TOP VIEW

Figure 6. Ridge operator axes on a volume with a straight spine



The difficulty in the torus example is due to the torus' curved spine. However, one can easily construct counterexamples to the ridge operator using volumes with straight spines. Consider the sinusoidally elliptic cylinder (figure 1-6) given by:

$$\mathbf{x} = \mathbf{x}(\theta, \phi) = (\theta, \cos \phi, \cos \theta \sin \phi)$$

where $-\infty \leq \theta \leq \infty$, and $0 \leq \phi < 2\pi$. A change in θ is associated with translation along the axis of this surface. At $\theta = 0$ the cross section of the surface is a circle of radius i . As θ increases (or decreases) the cross section is in general an ellipse whose eccentricity varies with $\cos \theta$.

The locus of points picked out by the ridge operator on the sinusoidally elliptic cylinder is a sinusoid whose amplitude depends on how the sec is rotated about its spine with respect to the observer. That the locus of points is in general a sinusoid rather than, say, a straight line, is not a problem. One may in fact want the locus of points computed at a fine enough scale to be curved in a manner that reflects important properties of the surface. What is a problem is the fact that the amplitude of the sinusoid is a function of the viewer's position with respect to the surface. This violates the uniqueness criterion.

To summarize, representations employing generalized cylinders can in principle provide uniqueness, stability, sensitivity and large scope. Attempts to compute generalized cylinders from depth maps or images have had limited success.

1.8. Philosophy of the approach

Our discussion of the problem of visual recognition thus far has intentionally been ambiguous about whether we are considering machine or human recognition. Actually the theory to be presented here is intended to be a (partial) theory of both machine and human visual recognition.

This has proved distressing to some psychologists. Some have argued that since the theory is an idealized account of visual recognition, ignoring much of the nonmodularity and little understood feedback systems known to exist in the human visual system, it is therefore irrelevant to human vision. While the observation is accurate, the conclusion is quite surprising, certainly counter to the conclusion drawn under similar circumstances in the other natural sciences. Newtonian physics, and relativistic physics for that matter, are also idealizations of true physical law. Few would conclude from this that the theories of Newtonian and relativistic physics are irrelevant to a true understanding of physical law. A more appropriate conclusion is that it is the nature, perhaps also a limitation, of scientific inquiry to study a subject by constructing successively better idealizations.

How, then, can the psychological validity of the theory of visual recognition be tested, if it is to be taken seriously as a psychological theory? Don't the psychological data underdetermine the theory? If that is the case, isn't it impossible to ascertain the psychological validity of the theory?

Again, the observation is true, but the conclusion is surprising. It is true that the psychological data underdetermine a theory of visual recognition. But this is true of the data in all fields of natural science. That is what makes theory construction interesting. Chomsky (1980, 11-12) gives a cogent description of what constitutes supporting evidence for a psychological theory:

"I am interested, then, in pursuing some aspects of the study of mind, in particular, such aspects as lend themselves to inquiry through the construction of abstract explanatory theories that may involve substantial idealization and will be justified, if at all, by success in providing insight and explanations. From this point of view, substantial coverage of data is not a particularly significant result; it can be attained in many ways, and the result is not very informative as to the correctness of the principles employed. It will be more significant if we show that certain fairly far-reaching principles interact to provide an explanation for crucial facts—the crucial nature of these facts deriving from their relation to proposed explanatory theories. It is a mistake to argue, as many do, that by adopting this point of view one is disregarding the data. Data that remain unexplained by some coherent theory will continue to be described in whatever descriptive scheme one chooses, but will simply not be considered very important for the moment."

This approach, of showing "that certain fairly far-reaching principles interact to provide an explanation for crucial facts" is adopted here. Chapters 2 and 4 develop some principles in detail. These principles are expected to pertain to human and machine vision. Chapter 5

relates them to relevant observations about human visual recognition in an attempt to justify them as a psychological theory.

We can make significant progress toward elucidating useful principles by noting that vision is *goal oriented*. Human observers are presented with two time varying images and must attempt to infer from them useful properties of the world. These inferences of useful properties of the world are, for the most part, inductive inferences because the mapping from the world onto the images is many to one. That is, for any set of images there is an infinite set of possible configurations of the world that are equally consistent with the images. To decide among them the observer must bring to bear knowledge regarding the world he happens to inhabit.¹⁴

For example, if one wants to infer the three-dimensional structure of an object from time-varying images of its features (i.e., from motion), one is faced with a fundamental ambiguity. Regardless of the quantity of image data available, there are an infinite number of structures which are consistent with the data. If, however, one brings to bear knowledge about objects in our visual world, particularly the knowledge that objects are often rigid, it is possible to determine whether the image data are consistent with a rigid interpretation, given that certain sufficiency conditions are met by the data (Hoffman 1982, Hoffman & Flinchbaugh 1982, Longuet-Higgins & Prazdny 1980, Ullman 1979). Experiments have shown that if there is a unique rigid interpretation the human visual system takes it (Ullman 1979).

In short, three categories of knowledge are particularly germane for visual non-demonstrative inferences:

- the image data
- relevant knowledge about the structure of the visual world
- knowledge of lawful relationships between the world and the images

An important aspect of understanding vision, then, is to discover what are useful goals, what properties of the world are desirable to be inferred from images, what knowledge about the world is necessary to make the inferences, and under what conditions the image data is sufficient to grant credence to the inferences. A theory of vision which addresses these questions is called by Marr and Poggio (1977) a *computational theory*. A computational theory is to be distinguished from the particular *algorithm* which is used to implement the theory and the *hardware* (or *wetware*) in which the algorithm is embodied (Dennett 1978, Marr & Poggio 1977). It is at the level of the computational theory that we hope to "...

¹⁴The more *general* the knowledge the better. Other things being equal, knowledge of geometry, mathematics and physical law is preferable in early vision to more specific knowledge, such as that I am in an office and will probably be seeing office furniture. This is because the general knowledge is applicable more often than the specific and lends itself to the design of "informationally encapsulated" vision modules (see Fodor 1982, on informational encapsulation).

show that certain far-reaching principles interact to provide an explanation for crucial facts ...” and to answer our problem question, How can early visual representations of shape be transformed into representations suitable to initiate the recognition process?

2. Representing plane curves

2.1. Overview

In this chapter the problem of representing plane curves for recognition is addressed. Smooth curves are discussed first followed by curves with a finite number of discontinuities of the tangent. The treatment in this chapter assumes the curves are given at least piecewise by analytic functions so that the tools of differential geometry can be applied. In chapter 4, where an implementation of the ideas in this chapter is presented, the problem of extending notions of differential calculus to nowhere differentiable functions is addressed so that bitmap representations of curves, not just analytic representations, can be transformed into representations for recognition.

2.2. Notation and terminology

A plane curve can be specified by the vector function:

$$\alpha(t) = (x(t), y(t)), \quad t \in \mathbb{R}$$

where t is an arbitrary parameter. If the parameter is arc length, s , along the curve then $\gamma(s)$ is used rather than $\alpha(t)$ to specify the curve. We restrict our attention to curves $\alpha: [a, b] \mapsto \mathbb{R}^2$ which are C^1 for all $t \in [a, b]$ and which are *immersions*, i.e. which satisfy $\alpha'(t) = d\alpha/dt \neq 0$ for all $t \in [a, b]$.

Derivatives with respect to a parameter other than arc length are indicated by primes (e.g. $\alpha'(t)$ is the first derivative of $\alpha(t)$). Derivatives with respect to arc length are indicated by dots (e.g. $\ddot{\gamma}(s)$ is the second derivative with respect to arc length of $\gamma(s)$). The tangent of the curve at $\gamma(s)$ is $\dot{\gamma}(s)$.

The *orientation* (or *direction*) of a curve is not the same as its *rotation*. A curve admits only two orientations, the two directions along which the curve may be traversed. A curve admits a continuum of rotations in the interval $[0, 2\pi]$, obtained by spinning the curve in the plane.

The magnitude of the curvature at $\gamma(s)$ is given by $|\ddot{\gamma}(s)|$. For space curves in general the curvature is not a signed quantity. However for a plane curve it is possible to give the curvature a sign. This can be done by assigning a unit vector, called the *principal normal unit vector*, to each point on the curve such that the principal normal is always orthogonal to the tangent vector $\dot{\gamma}(s)$ and always points to the right side of the curve when the curve is traversed in the chosen orientation. The sign of curvature at a point positive if the angle between the principal normal and $\ddot{\gamma}(s)$ is zero. The sign is negative if this angle is 180 degrees. An important consequence of this definition is that the curvature at each point along a curve flips sign when the orientation (direction of traversal) of the curve is reversed.

2.3. Goals

Our goal in this chapter is to transform vector function representations of plane curves into representations which are suitable for the task of object recognition. Criteria were discussed in the first chapter by which the suitability of a representation for recognition may be judged. The basic ideas are simple. First, a representation for recognition should articulate shapes into a natural hierarchy of parts and spatial relations, decoupling the basic form of a class of shapes from the minor variations exhibited by members of the class. That is, the representation should capture the form and freedom displayed by objects in the world. This is the stability/sensitivity criterion. Second, the representation should be able to describe the desired range of shapes. In our case the range should be as broad as possible since we are interested in computing shape descriptions bottom up and using them as a first index into a memory of shapes. Third, changes in viewing geometry should not lead to multiple descriptions of the same object. Ideally only one shape description should be produced from all viewing positions. Realistically one should hope to keep the number of descriptions to a minimum. The problem of course is that multiple descriptions require more memory for storage and more processing for matching during recognition. Note that the intent is not to ignore information about position and rotation of the shape. (For instance, the rotation of a shape is important information when trying to recognize the shape, as evidenced by the fact that a square and diamond look different or that a map of the United States turned on its side looks like a face). Rather the intent is to decouple the rotation, position and overall scale of an object from its shape so that these factors can be dealt with independently in the recognition process. Finally, the representation should be computable from images or it is useless. This will be the burden of chapter three.

2.4. Decoupling rotation and position

A description of a plane curve using the representation $\gamma(s) = (x(s), y(s))$ depends critically on the position, rotation and overall scale of the curve. Consequently, any rules for partitioning a plane curve into parts based on this representation must also be dependent on viewing geometry. The parts defined by such rules would have to be different, in general, when the viewing position changed. A prerequisite to a viewpoint independent articulation of a curve into parts, then, is a representation which decouples the disposition of the curve in space from its shape.

A major step toward accomplishing this is provided by the fundamental existence and uniqueness theorem for space curves (Do Carmo 1976, Lipschutz 1969).

Theorem: Let $\kappa(s)$ and $\tau(s)$ be arbitrary continuous functions on $a \leq s \leq b$. Then there exists, except for position and rotation in space, one and only one space curve γ for which $\kappa(s)$ is the curvature, $\tau(s)$ is the torsion and s is a natural parameter along γ .

Since we are dealing with plane curves here, $\tau(s)$ is always zero. Thus we have that for plane curves a representation based on the curvature $\kappa(s)$ as a function of arc length yields descriptions which are unique up to position and rotation in the plane. This provides a means to effectively decouple the position and rotation of the curve from its shape. We could represent a curve, for example, by the four-tuple $(\theta, x, y, \kappa(s))$ where θ is the rotation of the curve, x and y its translation and $\kappa(s)$ its curvature parametrized by arc length. We have decoupled, not eliminated, the disposition of the curve from the shape of the curve. The disposition of the curve is still available to be used in the recognition process.

2.5. Scale independent parts

Curvature is independent of rotation and translation, but not independent of scale. For example, two circles of differing radii have differing curvatures (since curvature is the inverse of the radius) even though they have the same shape. Therefore a description in terms of curvature alone does not provide complete viewer independence. Nor does it yet partition a curve into parts and spatial relations.

A solution to both problems can be found in the inflections and extrema of curvature. An inflection of curvature on a curve $\gamma(s)$ is a point where the curvature $\kappa(s)$ changes sign. An extremum of curvature is a point where $d\kappa(s)/ds = 0$. The property of being an inflection point or a local extremum of curvature is independent of scale. If some combination of inflections and extrema or both is used to carve a curve into parts then

Figure 1. Minima of curvature (slashes). Arrows indicate curve orientation.



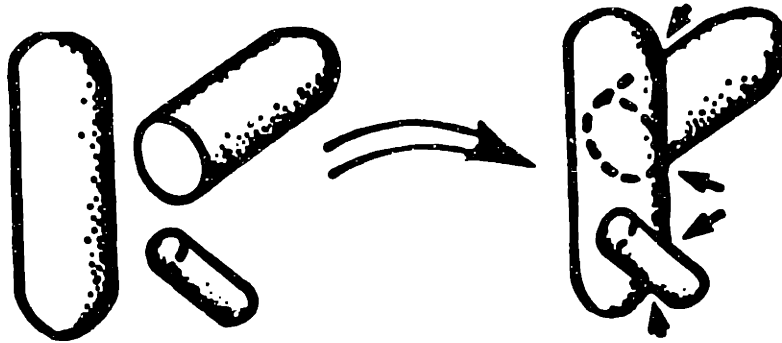
the part definitions will be completely viewer independent. The question is, which points should be used? Attneave (1954), Duda and Hart (1973) and Brady (1982) all suggest that maxima and minima points should be used to define part boundaries. Hollerbach (1975) used inflections.

An important technical point should be clarified here. An extremum of curvature can be either a point of local maximum or minimum curvature depending upon which of the two possible orientations (directions of traversal) of the curve is selected. A change in orientation of a plane curve flips the sign of curvature everywhere along the curve so that a point of maximum curvature becomes a point of minimum curvature and vice-versa.

A change in orientation of a plane curve can be uniquely associated with a reversal in which side of the curve is considered "figure" and which side is considered "ground". The convention adopted here is that the figure of a curve is to the left and its ground to the right as the curve is traversed in the chosen orientation. Thus knowing which side of a curve is figure determines the choice of orientation on the curve, or, conversely, choosing an orientation determines which side is figure by convention. Minima are then typically associated with concavities of the figure and maxima with convexities (see figure 2-1). It is possible however for minima to have positive curvature, as in the case of convex closed curves, or for maxima to have negative curvature, as when the orientation of the convex closed curve is reversed.

Maxima, minima and inflections of curvature are all candidate points for partitioning a curve into units in a viewer independent manner. To choose among them we should require

Figure 2. Joining parts yields concave cusps in the resulting silhouette.



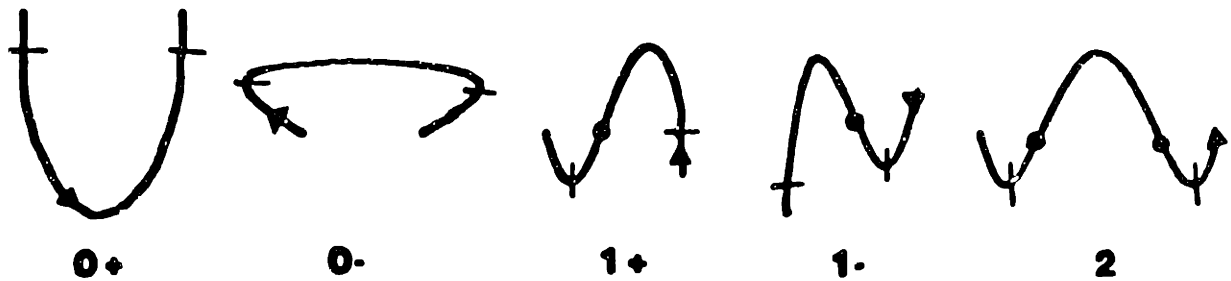
that the units chosen reflect natural parts of shapes (Marr 1977, Marr 1982). Fortunately, when 3-D parts are joined to create complex objects concave cusps will generally be created in the imaged silhouette (figure 2.2). If one assumes the parts are joined at random, then by general position the probability that this is the case is one. By smoothing the silhouette we have that the parts meet each other at minima of curvature. This suggests that only minima of curvature, not maxima or inflections, be used to segment a curve into parts since *segmentation of the image at minima of curvature immediately encodes in a straightforward manner an important property of the natural world which is not captured by maxima or inflections.*

2.6. Qualitative description of parts: Contour codons

The arguments of the previous section lead to the conclusion that minima of curvature should be used to define parts along plane curves. The first minima encountered in traversing a part in the chosen orientation is called the *tail* of the part. The second (and last) minima encountered is called the *head*. This does not imply that maxima and inflections have no role to play in the representation. Maxima and inflections will be used to provide successively more detailed viewer independent descriptions of the individual parts.

The first level of description of the individual parts is a qualitative one. Five qualitatively

Figure 3. Contour codons. Slashes are minima, dots are inflections.



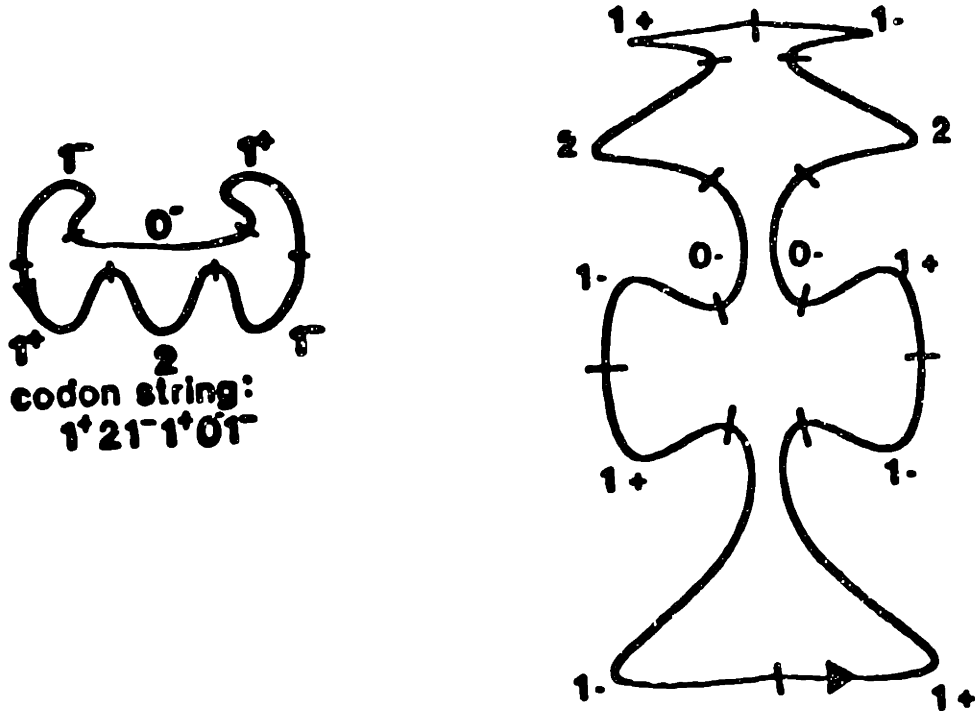
different types of parts can be identified based on the sign of curvature of the point of maximum curvature on the part and the signs of the two minima which define the beginning and end of the part. These five types of parts, called "contour codons", are defined in table 2-1 and illustrated in figure 2-3. The first column, labelled "codon", gives the name of the codon. The second column, labelled "tail", gives the sign of curvature of the tail of the codon. The third column gives the sign of curvature of the head of the codon. The final column gives the sign of curvature of the point of maximum curvature of the codon. These five codon types exhaust the qualitatively different possibilities (assuming that when $\kappa(s) = 0$, $d\kappa(s)/ds \neq 0$).

The names of the codons given in table 2-1 are based upon the number of inflections the codon has and the sign of the tail of the codon. A type 0+ codon, for instance, has no inflections of curvature and has a positive tail. A type 2 codon has two inflections. 2 is actually an abbreviation for 2-, since there is no 2+ codon possible.

Contour Codon Definitions			
Codon	Tail	Head	Maximum
0-	-	-	-
0+	+	+	+
1-	-	+	+
1+	+	-	+
2	-	-	+

Table 2-1

Figure 4. Codons strings for some curves. Arrows for orientation.



A top level description of a curve using the codon notation is a string whose elements are in the set $0+, 0-, 1+, 1-, 2$. The sequence of the string indicates the sequence of the codons on the curve. Some example curves with their top level codon descriptions are given in figure 2-4.

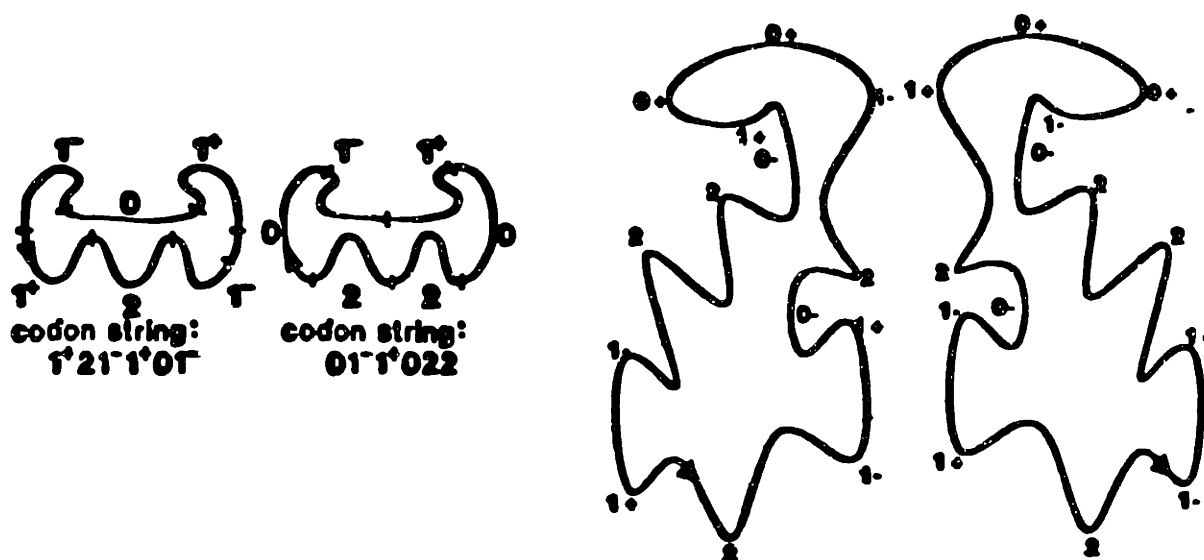
Not all codons strings of length two are allowable. A codon string of length two is allowable only if the head of the first codon of the string has the same sign of curvature as the tail of the second codon. Table 2-2 shows for each conceivable codon double whether or not that double is allowable. The column gives the first codon of the double, the row is the second codon of the double. A + indicates that the pair is allowable, a - that it is not. One can see from the table that of the 25 possible codon doubles only 13 are allowable.

Legal Codon Doubles					
Codon	0-	0+	1-	1+	2
0-	+	-	+	-	+
0+	-	+	-	+	-
1-	-	+	-	+	-
1+	+	-	+	-	+
2	+	-	+	-	+

Table 2-2

The fact that not all conceivable codon combinations are allowable suggests that the top level codon description of curves may be amenable to available error correction techniques. Consider, for example, the codon string $\dots c_{j-1}c_jc_{j+1}\dots$ if all codons combinations were

Figure 5. Orientation and mirror reversals



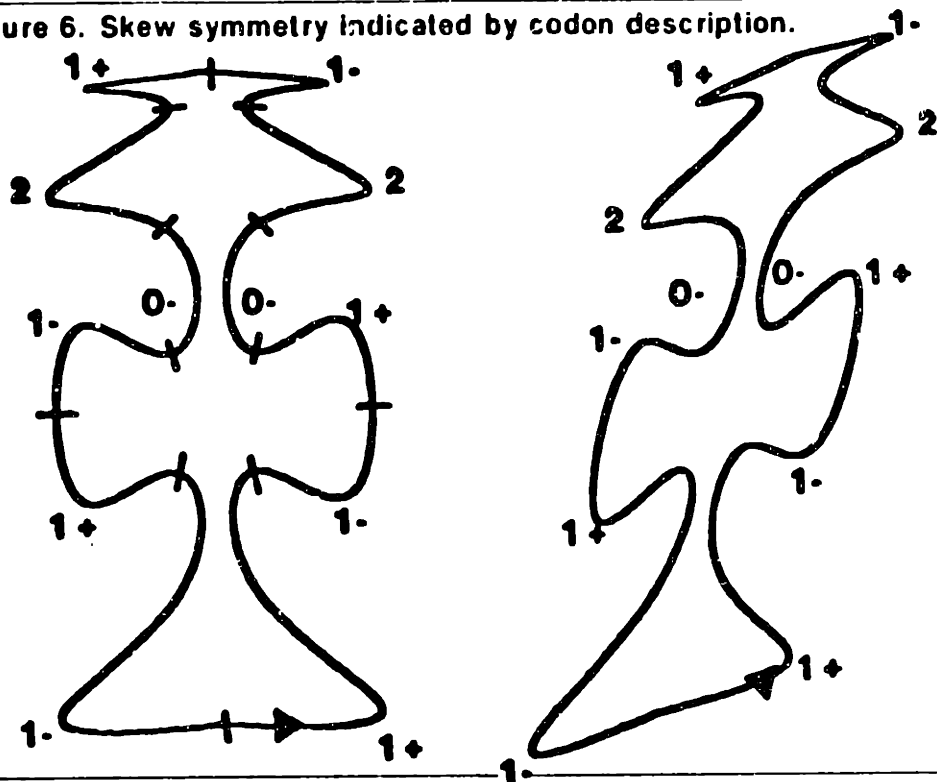
allowable then c_j could take any one of the five codon values. Thus the value of c_j would not be constrained by its context. However, as shown in table 2-3, the value of a codon is highly constrained by the values of its two surrounding neighbors. The row specifies the codon to the left of the middle codon of the triple. The column specifies the codon to the right. The entry for a given row and column shows all the allowable codons for that particular context. From the table it can be seen that 64 per cent of the possible contexts constrain the allowable set of codons to a unique value. On average the allowable set size is 1.36.

Legal Codon Triples					
Codon	0 -	0 +	1 -	1 +	2
0 -	0-2	1 -	0-2	1 -	0-2
0 +	1 +	0 +	1 +	0 +	1 +
1 -	1 +	0 +	1 +	0 +	1 +
1 +	0-2	1 -	0-2	1 -	0-2
2	0-2	1 -	0-2	1 -	0-2

Table 2-3

The top level codon representation of a curve decouples aspects of the shape of the curve from its disposition in space and its overall scale. Consequently the codon string description is invariant under rotations, translations and uniform scalings of the contour. However the codon description is not invariant when the orientation (direction of traversal) of the curve is reversed or when the curve is subjected to a mirror reversal (see figure 2-5). The question naturally arises, Are there simple rules that define how the codon description

Figure 6. Skew symmetry indicated by codon description.



of a contour is transformed when the contour undergoes a mirror reversal or a change of orientation?

In the case of a mirror reversal the rule is quite simple. The mirror transform of a codon string is obtained by reversing the direction in which the string is read (right to left rather than left to right) and reversing the sign attached to each type 1 codon. This rule can be used to find symmetries within a single contour. If, for example, one half of a codon string is found to be the mirror transform of the remainder of the string, a necessary condition for the curve to be symmetric has been found (see figure 2-6). Note that this applies to skew symmetry as long as zeroes of curvature are not made to appear or disappear by the skew.

When the orientation of a curve is reversed the codon string transformation rule, called the lock-key transform, is unique but apparently not simple. It is perhaps most easily specified as a map from pairs of concatenated codons to codon singletons. The codon doubles which map to each codon singleton are:

$$\begin{aligned} \{(0^+ 0^+)(0^+ 1^+)(1^- 0^+)(1^- 1^+)\} &\mapsto 0^- \\ \{(0^- 0^-)\} &\mapsto 0^+ \\ \{(0^- 0^-)(0^- 1^-)\} &\mapsto 1^- \\ \{(1^+ 0^-)(2 0^-)\} &\mapsto 1^+ \\ \{(1^+ 1^-)(1^+ 2)(2 1^-)(2 2)\} &\mapsto 2 \end{aligned}$$

2.7. A perceptual note

Before developing the codon description in more detail, it is interesting to note a parallel between the human perception of contours and the codon representation of contours. In figure 2-7(adapted from Attneave 1971) are two semicircular regions each having a wavy boundary where typically a straight boundary is drawn. The question is, What is the relation between the wavy bounding contours of the two regions?

Most first time viewers deny that there is any obvious relation between the two wavy contours. In fact the two contours are identical! The contours fit together like two pieces of a jigsaw puzzle to form a complete circular region. Why does the same curve look so different?

From an information processing perspective, the same curve could look different if for some reason the curve is given different descriptions by the human visual system. Apparently the only salient contextual difference between the two curves is that the figure-ground relations, induced by which side of each curve is darkened, are reversed between them. Thus it appears that *the human visual system describes the same curve quite differently when the curve undergoes a figure-ground reversal.*

As mentioned earlier, a reversal in figure and ground of a curve also reverses the orientation (direction of traversal) of the curve. This induces a change in the sign of curvature along the entire curve. In particular, maxima and minima of curvature swap places on the curve. Since the codon parts are defined by the minima of curvature, the part definitions and descriptions change when figure and ground reverse (see figure 2-5). The parts actually move around on the curve. The codon strings transform according to the lock-key rule of the previous section: Consequently *the codon system describes the same curve quite differently when the curve undergoes a figure-ground reversal*, an interesting parallel with human vision.

This result would not obtain if the part boundaries were defined using only inflections of curvature, or if both maxima and minima were used as suggested by some previous investigators. If boundaries are defined by minima, maxima, minima and inflections, or maxima and inflections then the part definitions will change under an orientation reversal. Among these four possibilities, however, minima are used since segmentation of a curve at minima of curvature encodes in a straightforward manner an important property of the natural world which is not captured by the other options, namely natural part boundaries.

Figure 7. Two wavy curves



2.8. Positive minima

The justification for using minima of curvature as part boundaries, which is illustrated in figure 2-2, is that randomly placed parts meet at minima of curvature. However one can see that figure 2-2 only shows that parts will meet at minima with *negative* curvature, not positive curvature. But the codon definitions presented earlier use positive minima as boundaries in addition to the negative minima. Has a mistake been committed?

Not really (or the subject probably would not be raised!). Whatever descriptive scheme one chooses, positive minima will figure prominently, along with inflections and maxima, since they allow scale invariant descriptions. One could define part boundaries only at negative minima and throw positive minima into the descriptions of these larger parts along with the maxima and inflections. Alternatively one could opt for using all minima as part boundaries, leading to smaller parts and simpler descriptions for the parts. As long as one is clear about what is going on either approach is acceptable. The second alternative is used here because of the convenient codon notation that arises from using all minima as part boundaries, and because the description of each part becomes simpler and more uniform.

There is another, more subjective, reason. Consider the possible partitions of an ellipse presented in figure 2-8. Of all the partitions shown, the partition at the (positive) minima seems the most natural.

Underlying this issue is the notion of the "strength" of a part boundary. Roughly, the more negative the curvature is at a minimum the stronger the part boundary. Minima of positive curvature have the least strength. Figure 2-9 shows a progression of part boundaries going from strongest to weakest. Note that cusps, having infinite curvature, can give the strongest part boundaries.¹

¹Cusps are discussed in more detail later in this chapter.

· **Figure 8. Partitions of an ellipse**

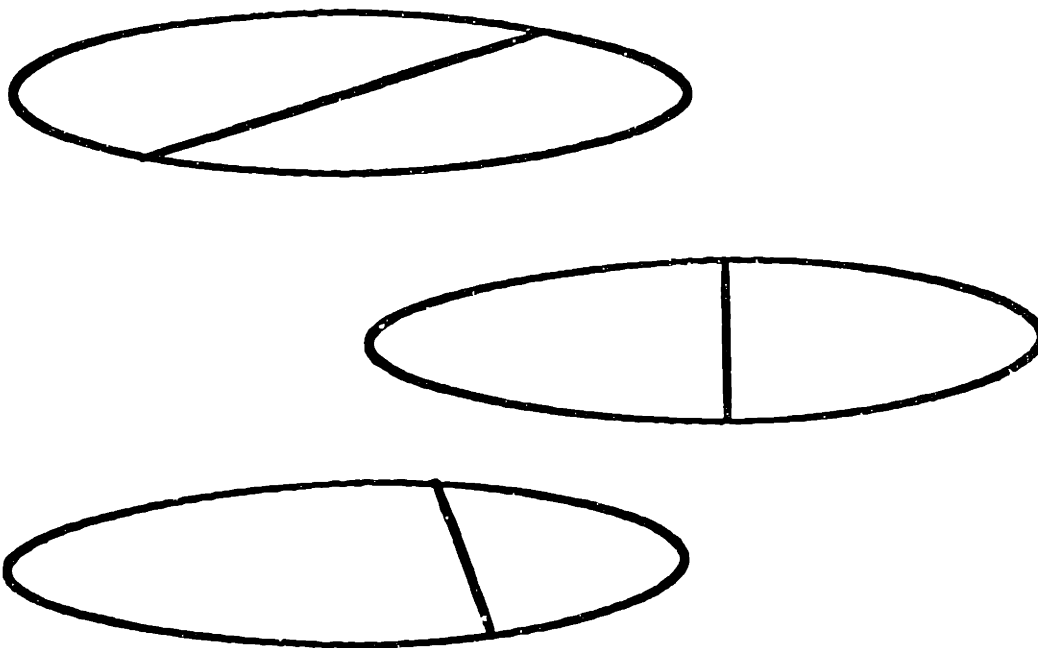


Figure 9. Part boundaries of decreasing strength



2.9. Quantitative description of parts

The codon string is but the top level description of a curve. Infinitely many different curves can have the same codon string. What is needed is more quantitative description attached to each element of the string. Yet this quantitative information must itself be represented in a form which decouples shape from disposition and scale. This is possible by exploiting extrema and inflections of curvature. What follows is a list of the most obvious quantitative measures.

Since all elements of a codon string need not represent equal arc length curve parts, an important quantitative measure is the *relative arc length* of each codon. The relative arc length can be computed by dividing the arc length of each codon by either the arc length of the longest codon of the curve or by the arc length of the longest codon within a certain neighborhood whose size is specified in terms of number of codons. This gives a measure between zero and one inclusive which does not depend on disposition or overall scale.

Although the number of inflections and maxima contained in a codon can be inferred from the codon type, their positions within the codon cannot. If their positions are described as a percentage of the total length of the codon segment then they are appropriately invariant.

The curvature at the extrema of curvature in each codon can be described either relative to the extremum with maximum curvature over the entire curve or relative to an extremum with maximum curvature over some local neighborhood.

As illustrated in figure 2-10, two codon segments can have identically placed maxima and inflections, identical curvatures at the maxima and minima, and yet appear quite different. The difference is the behavior of the curvature between the extrema and inflections. This behavior can be summarized in an appropriately invariant manner by the integral of curvature between each of these points. There is a quite simple method for determining this integral since:

$$\int_a^b \kappa(s) ds = \theta(b) - \theta(a),$$

where $\theta(s)$ is the angle of the tangent at $\gamma(s)$ given by $\theta(s) = \tan^{-1}(y'(s)/x'(s))$. A representation which notes the integral of curvature between these points will give different descriptions for the two curves in figure 2-10.

2.10. Curves with cusps

2.10.1. Terminology

The codon representation of smooth plane curves can be extended to plane curves with a finite number of cusps. A cusp is a point on a curve where the first derivative is not continuous (see figure 2-11a). The following terminology is useful for discussing cusps (figure 2-11b). Let $\gamma(s)$ be a plane curve parametrized by arc length, and let $\gamma(s_c)$ be a cusp point. At $\gamma(s_c)$ there are two tangents. The "first tangent" is defined as

Figure 10. Curves with differing integrals of curvature



$$t_1 = \lim_{\Delta s \rightarrow 0} \frac{\gamma(s_c - \Delta s) - \gamma(s_c)}{-\Delta s}$$

where $\Delta s \rightarrow 0$ through positive values. The "second tangent" is defined as

$$t_2 = \lim_{\Delta s \rightarrow 0} \frac{\gamma(s_c + \Delta s) - \gamma(s_c)}{\Delta s}$$

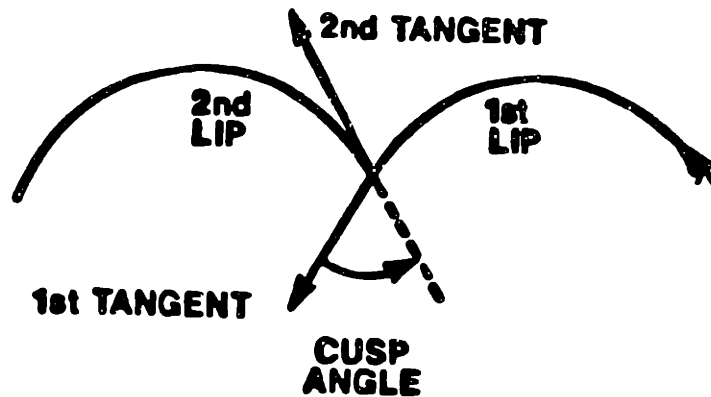
where again $\Delta s \rightarrow 0$ through positive values. Intuitively the first tangent is associated with the side of the cusp first traversed in an orientation and the second tangent with the other side.

The "first lip" of the cusp is the image under γ of the open interval $(s_c - \epsilon, s_c)$ where ϵ is sufficiently small that there are no extrema or inflections of curvature and no cusps in the lip. Similarly the "second lip" is the image under γ of the open interval $(s_c, s_c + \epsilon)$ where ϵ is sufficiently small.

If t_1 and t_2 are the first and second tangent respectively, then the cusp angle, θ_c , is

$$\theta_c = \cos^{-1}(t_1 \cdot -t_2)$$

Figure 11. Cusp terminology



2.10.2. Segmentation

For purposes of partitioning a curve into codon segments, cusps are divided into two classes. If an ϵ step along the first tangent at the cusp takes one to the left of γ , i.e. into the figure side of the curve for the chosen orientation, then the cusp is called *concave*. If the ϵ step takes one to the right of γ , i.e. to the ground side of γ , then the cusp is *convex*. The argument presented in section 2-5 (and figure 2-2) provides motivation for partitioning curves at concave cusps, but not at convex cusps. Consequently our extended segmentation rule is to partition a curve into codon segments at minima of curvature and at concave cusps (figure 2-12).

Let minima of curvature and concave cusps be referred to collectively as "generalized minima". Similarly let maxima of curvature and convex cusps be referred to collectively as "generalized maxima". Then the segmentation rule can be characterized succinctly as partitioning at generalized minima.

With the inclusion of cusps it is clear that two codons, a and b , may be joined smoothly or at a cusp. Smooth joins are indicated by the operation $a \circ b$. Joins at cusps are indicated by $a * b$. Smooth joins are subject to the restrictions of table 2-1 whereas cusp joins are not.

2.11. Codon description extended for cusps

For purposes of description cusps are divided into two types. Cusps whose first and

Figure 12. Segmentation at generalized minima

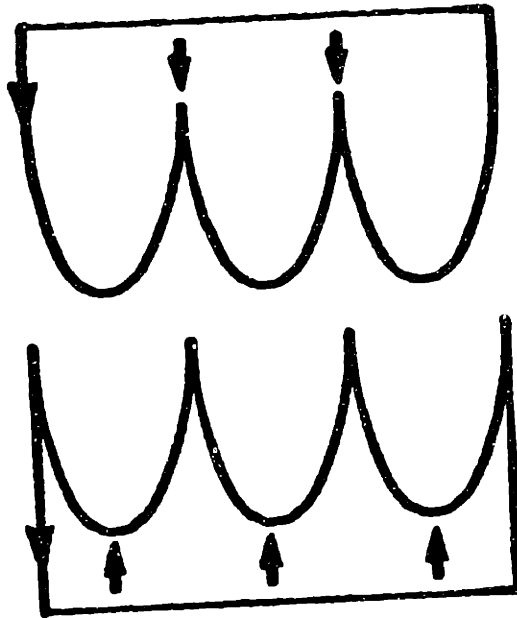
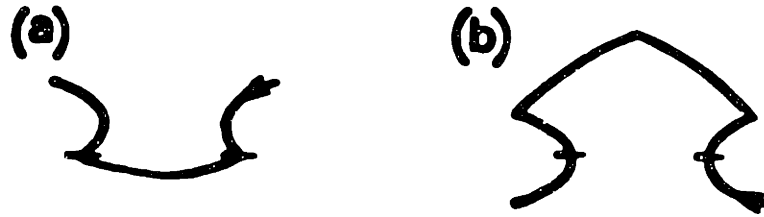


Figure 13. Cusp types crossed with concave/convex

	CONCAVE	CONVEX
SYMMETRIC +		
SYMMETRIC -		
ASYMMETRIC +		
ASYMMETRIC -		

second lips have the same sign of curvature are called *symmetric*. Those whose lips have opposite signs are called *asymmetric*. Each type is further described by noting the sign of curvature of the first lip, leading to four categories illustrated in figure 2-13. A convenient shorthand for the cusp types is $S+$, $S-$, $A+$ and $A-$. Figure 2-13 also illustrates that each of the four types of cusps can be either concave or convex.

Figure 14. Number of generalized maxima varies



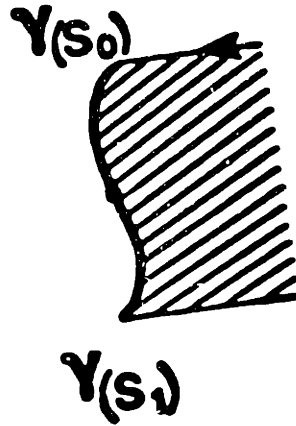
When cusps are allowed, the number of generalized maxima in one codon segment can range from zero to a large number (figure 2-14 shows a case where there are none and a case where there are three). The number of generalized maxima in a codon segment becomes, then, an important part of the codon description. A detailed description would include the type of each generalized maximum and the type of each cusp.

Finally, let generalized minima, generalized maxima and inflections of curvature be referred to collectively as "distinguished points". Then describing the normalized positions of maxima and inflections in smooth codon segments can be generalized to describing the normalized positions of distinguished points for curves with cusps. Likewise, the integral of curvature between extrema and inflections may be generalized to the integral of curvature between distinguished points.

2.11.1. Inflections without minima

Simply for thoroughness it should be noted that it is possible to have arbitrarily many inflections within a codon segment when cusps are allowed. The reason is that there need not be a point of minimum curvature between two generalized maxima even though there is an inflection between them. For this to happen at least one of the two generalized maxima must be a convex cusp. An example is given in figure 2-15 for the case where both generalized maxima are cusps. Let $\gamma(s_0)$ be the first cusp traversed and $\gamma(s_1)$ the second. Let the arc length from $\gamma(s_0)$ to $\gamma(s_1)$ be l . Then if the curvature between $\gamma(s_0)$ and $\gamma(s_1)$ is given by $\kappa(s) = c(s - l/2)$, $0 < s < l$ and c constant, then there is no point of minimum curvature but there is an inflection at $s = l/2$. There is no minimum (or maximum)

Figure 15. Inflections without minima



of curvature in this example because there is no minimum (or maximum) s . For any $s \in (0, l)$ it is the case that $s/2 < s < (s + l)/2$, giving elements of $(0, l)$ strictly less or greater than s .

2.12. Codon hierarchies

The treatment of codons thus far has intentionally ignored, for sake of simplicity, the fact that *the codon description of a curve depends upon the resolution with which the curve is examined*. At one degree of resolution a curve may appear to contain only a handful of codons. Closer inspection reveals, however, that nested within each of these bigger codons is a handful of smaller codons. Again, nested within each of these smaller codons is yet another handful of even smaller codons, and so on ad infinitum.

Clearly, then, one level of codon string description in isolation is necessarily incomplete. Each codon of the codon string at one level must have a pointer to another subordinate codon string which describes the codons nested within it. Each member of each of these subordinate codon strings has a pointer to its own subordinate codon string, etc. The result is that a curve is described by a hierarchy of codon strings rather than by a single string.

A hierarchy of codon string descriptions is a useful means to satisfy the stability/sensitivity criterion, at least in part. The hierarchical descriptions decouple some grosser aspects of the shape of a curve from some of the finer aspects.

Figure 2-16 gives an idea how a hierarchy of codon strings could be organized. It also shows how the more quantitative descriptive information could be attached to the qualitative codon descriptions. Of course the number of levels in the hierarchy is limited by the resolution of the human visual system. At some point the hierarchy must be terminated and finer scale effects summarized perhaps as texture predicates attached to the smallest scale codons that are represented in the hierarchy.

2.13. Summary

I propose the following rule for dividing smooth plane curves into parts: *Divide a curve into parts at minima of signed curvature.* For curves with cusps this rule extends to dividing a curve into parts at concave cusps in addition to minima of curvature.

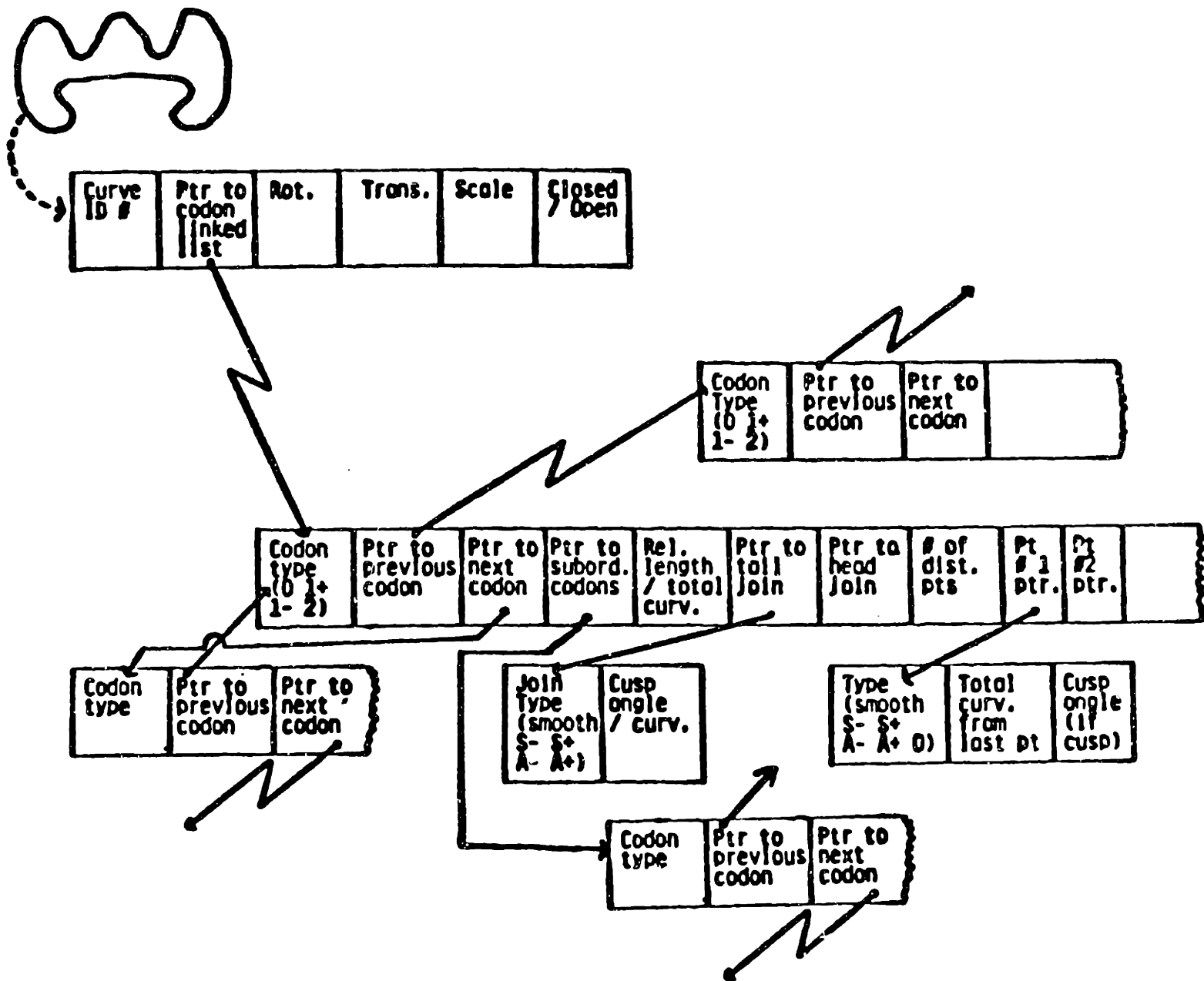
I argue that this rule is useful because of the following: 1) the parts defined by this rule do not vary with similarity transforms in the plane, 2) when two objects are intersected concave cusps or minima of curvature (for smoothed curves) will generally be created in the imaged silhouette (figure 2-2) and 3) the parts defined by this rule look natural. Evidence for the psychological role of this rule is presented in chapter 5.

Conveniently, an exhaustive qualitative classification of parts can be given. I call these the five codon types.² In addition metrical properties can be assigned to codon parts in a manner that is invariant under similarity transforms in the plane.

Much is left to do to understand our ability to recognize objects from their silhouettes. Of particular interest is the question of how a codon description of a curve is used as a first index into a memory for shapes. The duality between bounding contours and the areas they bound also needs to be developed. It is likely that both contour and area based predicates figure in our judgements of similarity and identity of objects.

²This nomenclature was suggested by W. Richards.

Figure 16. A possible codon representation scheme



3. Representing surfaces

The previous two chapters develop a representation of plane curves for recognition. In this chapter the project is to develop a similar representation for surfaces in three dimensions. For a large class of surfaces it turns out that it is a simple matter to extend the notions applied to plane curves. This is somewhat surprising since surfaces are much more complicated in their local geometry than plane curves. For example, rather than having a single curvature at each point, as is the case with curves, there are two principal curvatures associated with orthogonal directions at each point on the surface.

Smooth surfaces are discussed first followed by surfaces with contours of discontinuity of the tangent plane. I propose that for purposes of segmentation it is advantageous to analyze the principal curvatures independently rather than to use a measure such as the Gaussian curvature (where the principal curvatures are multiplied together). In particular, the segmentation rule argued for in this chapter is the following: *Divide a surface into parts at minima, along the lines of greatest curvature, of the greatest principal curvature. Or instead divide a surface into parts at minima, along the lines of least curvature, of the least principal curvature.*

3.1. Surfaces of revolution

Before engaging in a detailed mathematical analysis of surfaces, it would be helpful to develop our intuitions on a simple class of surfaces, surfaces of revolution. A surface of revolution can be obtained by spinning a plane curve about an axis which lies in the plane and does not touch the curve (figure 3-3). For image planes parallel to the axis of rotation, the projected silhouette of a surface of revolution is simply the original plane curve used to generate the surface.

From chapter 2 we already have a rule for dividing the plane curve into parts. Whatever rule we have for surfaces should divide the surface of revolution into parts in a manner consistent with the rule we already have for dividing its silhouette into parts. To satisfy this constraint the partitioning contours on a surface of revolution must be circles on the surface which pass through the minima of curvature of the generating plane curve (see figure 3-4 for examples). The rule stated earlier for partitioning surfaces does in fact define exactly these circles as the dividing contours. To see this, however, will require a brief tour through the differential geometry of surfaces.

3.2. Notation and terminology

Tensor notation, which allows concise expression of surface concepts, is adopted in this chapter (see Dodson and Poston 1979, Lipschutz 1969). A vector in \mathbb{R}^3 is $\mathbf{x} = (x^1, x^2, x^3)$. A point in the parameter plane is (u^1, u^2) . A surface patch is $\mathbf{x} = \mathbf{x}(u^1, u^2) = (x^1(u^1, u^2), x^2(u^1, u^2), x^3(u^1, u^2))$. Partial derivatives are denoted by subscripts:

$$\mathbf{x}_1 = \frac{\partial \mathbf{x}}{\partial u^1}, \quad \mathbf{x}_2 = \frac{\partial \mathbf{x}}{\partial u^2}, \quad \mathbf{x}_{12} = \frac{\partial^2 \mathbf{x}}{\partial u^1 \partial u^2}, \quad \text{etc.}$$

A tangent vector is $d\mathbf{x} = \mathbf{x}_1 du^1 + \mathbf{x}_2 du^2 = \mathbf{x}_i du^i$ where the Einstein summation convention is used. The first fundamental form is

$$I = d\mathbf{x} \cdot d\mathbf{x} = \mathbf{x}_i \cdot \mathbf{x}_j du^i du^j = g_{ij} du^i du^j$$

where the g_{ij} are the first fundamental coefficients and $i, j = 1, 2$.

The differential of the normal vector is the vector $d\mathbf{N} = \mathbf{N}_i du^i$ and the second fundamental form is

$$II = d^2\mathbf{x} \cdot \mathbf{N} = \mathbf{x}_{ij} \cdot \mathbf{N} du^i du^j = b_{ij} du^i du^j$$

where the b_{ij} are the second fundamental coefficients and $i, j = 1, 2$.

A plane passing through a surface S orthogonal to the tangent plane of S at some point P and in a direction $du^i:du^j$ with respect to the tangent plane intersects the surface in a curve whose curvature at P is the *normal curvature* of S at P in the direction $du^i:du^j$. The normal curvature in a direction $du^i:du^j$ is $k_n = II/I$. The two perpendicular directions for which the values of k_n take on maximum and minimum values are called the *principal directions*, and the corresponding curvatures, k_1 and k_2 , are called the *principal curvatures*. The *Gaussian curvature* at P is $K = k_1 k_2$. The *mean curvature* is $H = (k_1 + k_2)/2$. A *line of curvature* is a curve on a surface whose tangent at each point is along a principal direction. A point P is *planar* if $k_1 = k_2 = 0$, *parabolic* if $K = 0$ and either $k_1 \neq 0$ or $k_2 \neq 0$, *elliptic* if $K > 0$, and *hyperbolic* if $K < 0$.

Just as a plane curve can have one of two orientations, so an orientable surface can have one of two orientations of its field of surface normals.¹ On a sphere, for instance, the surface normals can either point inward to its center or outward like a porcupine. Reversing the orientation of a surface reverses the sign of the principal curvatures on the surface just

¹Avoiding technicalities, a surface is orientable if one can consistently assign a field of surface normals over the entire surface. A Moebius strip is an example of a surface where this is not possible.

as reversing the orientation of a plane curve reverses the sign of curvature all along the curve.

3.3. Goals

As mentioned in the first chapter a surface patch representation $x(u^1, u^2)$ for a surface S is inappropriate for recognition because it is a viewer dependent representation, violating both the uniqueness criterion and the stability/sensitivity criterion. Our goal in this chapter is to transform a surface patch representation into a representation which is suitable to initiate the recognition process. This transformation will proceed in the same three stages as for plane curves. First the rotation and translation of the surface will be decoupled (not discarded) from the shape. Then scale independent parts will be defined. Finally, the parts will be given viewpoint independent descriptions.

Simpler surfaces will be examined first followed by more complicated ones. Simplest are the *developable surfaces*, those containing only parabolic and planar points. Then *surfaces of revolution* will be studied. Finally, more general smooth surfaces and surfaces with contours of discontinuity of the tangent plane will be studied.

3.4. Decoupling rotation and position

For plane curves it is possible to decouple the shape of a curve from its disposition in the plane by using the curvature as a function of arc length. The analogous move for surfaces is to use the two principal curvatures. The principal curvatures at a point do not depend upon the viewing geometry and are thus an effective means of decoupling the shape of a surface from its rotation and translation in space. That the principal curvatures at a point do not depend on the viewing geometry can be proved as follows. Each principal curvature is a normal curvature. The normal curvature in some direction is given by the ratio of the first and second fundamental forms in that direction, i.e. $k_n = II/I$. Since both the first and second fundamental forms are invariant under a parameter transformation (Lipschutz 1969, p.172, 175) the normal curvature is also invariant. Hence the principal curvatures do not depend upon the viewing geometry.

However the matter is not as simple for surfaces as for curves. One can show that $\kappa(s)$ specifies a plane curve uniquely by the fundamental theorem of curves. But it is not true

that by specifying k_1 and k_2 as a function of the two surface parameters one has specified the surface uniquely up to disposition in space. What is missing at each point are the principal directions. Consequently our analysis will deal with the principal curvatures and with lines of curvature. It will be possible in this manner to define and describe parts in a viewer independent way.

3.5. Scale independent parts

The principal curvatures, like the curvature of plane curves, are not independent of the overall scaling of the shape. For example, two spheres of differing radii have differing principal curvatures² even though they have the same shape.

The way out of this problem for plane curves is to base the segmentation rules and part descriptions upon the inflections and extrema of curvature. For surfaces the analogous way out is to base the segmentation rules and part descriptions upon *inflections and extrema of the normal curvature of lines of curvature*.³ Note that the claim is that the rules depend upon extrema and inflections of the *normal* curvature, not merely the curvature, of the lines of curvature. The two curvatures are equal only when the lines of curvature are also geodesics, i.e. curves for which the geodesic curvature, κ_g , is everywhere zero.⁴ Note also that only one of the two normal curvatures is associated with a given line of curvature. For our purpose of segmentation into parts, *only the extrema and inflections of the normal curvature associated with that line of curvature are of interest in the analysis along that line of curvature*.

A given extremum point of normal curvature along a line of curvature can be either a maximum or minimum depending upon the orientation of the field of surface normals on the surface. From a practical perspective, a surface is simply a boundary between the object on the inside and the ground on the outside. A convention is needed here, in a manner similar to the case of plane curves, to associate the two possible orientations of the field of surface normals with the figure-ground relations. For plane curves the figure side of the curve is to the left and the ground side to the right as the curve is traversed in an orientation. For surfaces a convenient convention is to say that the surface normals point to

²Actually every normal curvature on a sphere is a principal curvature since all normal curvatures are identical.

³Recall that a line of curvature is a curve on a surface whose tangent at each point is along a principal direction.

⁴The total curvature of a surface curve at a point can, in general, have a component along the surface normal at that point and a component in the tangent plane at the point. The component along the normal is the normal curvature. The tangent plane component is the geodesic curvature.

the figure side of the surface. If a sphere is considered the bounding surface of a solid ball, for instance, then the surface normals are taken to be inward pointing by convention. If the sphere is a bubble in some surrounding medium then the surface normals are taken to be outward pointing. Thus if the figure-ground relations are known then the surface orientation is known. If the surface orientation is known then the figure-ground relations are known by convention. Minima of normal curvature of a line of curvature are then generally associated with concavities of the figure and maxima with convexities. It is possible however for minima to have positive normal curvature, as in the case of an ellipsoid with inward pointing surface normals. It is also possible for maxima to have negative normal curvature, as in the case of an ellipsoid with outward pointing normals.

Maxima, minima and inflections of normal curvature along lines of curvature are all candidate points for partitioning a surface into units in a viewer independent manner. To choose among them we require that the units chosen reflect natural parts of shapes, as in the case of plane curves. When 3-D parts are joined to create complex objects, the contour of the join will generally be concave (figure 2-2). This suggests that only minima of normal curvature, not maxima or inflections, be used to segment a surface into parts since, again, segmentation of a surface at minima of normal curvature along lines of curvature immediately encodes in a straightforward manner an important property of the natural world which is not captured by maxima or inflections. This leads to the following segmentation rule: *Divide a surface into parts at minima, along the lines of greatest curvature, of the greatest principal curvature. Or divide a surface into parts at minima, along the lines of least curvature, of the least principal curvature.*

This segmentation rule is applied to several classes of surfaces in the next sections.

3.6. Segmentation of developable surfaces

Developable surfaces are a special case of ruled surfaces. A ruled surface is a surface generated by a one parameter family of lines (Do Carmo 1976). A one parameter family of lines $\{\underline{a}(u^1), \underline{w}(u^1)\}$ is a correspondence that assigns to $u^1 \in (a, b) \subset \mathbb{R}$ a point $\underline{a}(u^1) \in \mathbb{R}^3$ and a vector $\underline{w}(u^1) \in \mathbb{R}^3$, $\underline{w}(u^1) \neq 0$, such that both $\underline{a}(u^1)$ and $\underline{w}(u^1)$ depend differentiably on u^1 . For each $u^1 \in (a, b)$, the line $L(u^1)$ which passes through $\underline{a}(u^1)$ and is parallel to $\underline{w}(u^1)$ is called the line of the family at u^1 .

Given a one parameter family of lines $\{\underline{a}(u^1), \underline{w}(u^1)\}$, the associated ruled surface is given by the parametrization

$$\mathbf{x}(u^1, u^2) = \underline{\alpha}(u^1) + u^2 \underline{w}(u^1), \quad u^1 \in (a, b) \subseteq \mathbb{R}, \quad u^2 \in \mathbb{R}.$$

The curve $\underline{\alpha}(u^1)$ is called a *directrix*, and the lines are called the *rulings* of the surface \mathbf{x} . In what follows we assume, without loss of generality, that u^1 is arc length along $\underline{\alpha}$ and that $|\underline{w}(u^1)| = 1$. A ruled surface is said to be *developable* if the scalar triple product $(\underline{w}, \underline{w}_1, \underline{\alpha}_1) = 0$ everywhere on the surface. Intuitively this means that \underline{w} , \underline{w}_1 and $\underline{\alpha}_1$ lie in a plane.

In the next two subsections we determine the segmentation contours for two nonexhaustive cases of developable surfaces: the cylinder and the cone.

3.6.1. Cylinders

A cylinder is a developable surface where $\underline{\alpha}$ is contained in a plane and $\underline{w}(u^1)$ is parallel to a fixed direction in \mathbb{R}^3 , that is $\underline{w}_1 = 0$. For a cylinder $\mathbf{x}_1 = \underline{\alpha}_1 + u^2 \underline{w}_1 = \underline{\alpha}_1$ (since $\underline{w}_1 = 0$) and $\mathbf{x}_2 = \underline{w}$. The first fundamental coefficients are then

$$g_{ij} = \mathbf{x}_i \cdot \mathbf{x}_j = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

The surface normal is

$$\mathbf{N} = \frac{\mathbf{x}_1 \times \mathbf{x}_2}{|\mathbf{x}_1 \times \mathbf{x}_2|} = \frac{\underline{\alpha}_1 \times \underline{w}}{|\underline{\alpha}_1 \times \underline{w}|} = \underline{\alpha}_1 \times \underline{w}$$

The second fundamental coefficients are

$$b_{ij} = \mathbf{x}_{ij} \cdot \mathbf{N} = \begin{pmatrix} (\underline{\alpha}_{11}, \underline{\alpha}_1, \underline{w}) & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} |\underline{\alpha}_{11}| & 0 \\ 0 & 0 \end{pmatrix}$$

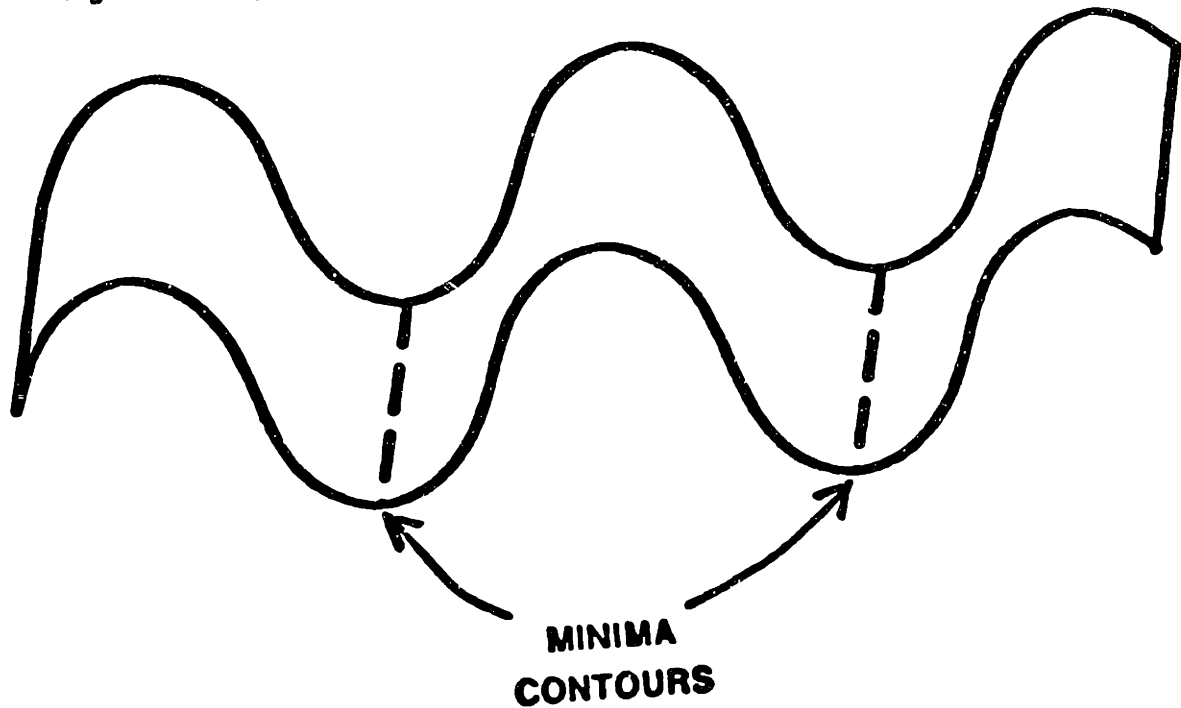
Since $g_{12} = b_{12} = 0$ the principal curvatures on a cylinder are

$$\begin{aligned} k_1 &= b_{11}/g_{11} = |\underline{\alpha}_{11}| \\ k_2 &= b_{22}/g_{22} = 0 \end{aligned}$$

The expression for k_1 is the magnitude of the second derivative of α with respect to arc length (with sign determined by the orientation of the field of surface normals) which is simply the curvature along the directrix $\underline{\alpha}$. The directrix and its translations are, in fact, one set of lines of curvature and the rulings the other set.⁵ As expected, the curvature along the

⁵The u^1 and u^2 parameter curves on a patch without umbilical points are lines of curvature if and only if at every point on the patch $g_{12} = b_{12} = 0$. When the parameter curves are lines of curvature the expressions for the principal curvatures are much simplified, as illustrated in the last displayed equation. (See Lipschutz 1969, 186).

Figure 1. A cylinder with segmentation and figure ground reversal



rulings, k_2 , is zero. Consequently no segmentation contours arise from the rulings (since there are no minima of the normal curvature, k_2). Only the minima of k_1 along the directrix and its translations are used for segmentation.

Figure 3-1 shows a cylinder and its segmentation contours (dotted lines) for one of the orientations of the field of surface normals. The segmentation contours break the cylinder into parts that seem natural to human observers. If one examines the figure long enough one can experience a figure ground reversal similar to that for plane curves. When this happens the bumps of the surface become dips and vice-versa. Notice that when figure and ground reverse the natural segmentation lines shift away from the indicated dotted lines to the contours that were previously maxima of k_1 . This occurs because the figure ground reversal is associated with a reversal in the surface orientation and, hence, in the sign of k_1 everywhere on the surface. Contours of maxima of k_1 and contours of minima of k_1 swap places and the new segmentations along the new minima become apparent.

Segmentation rules which use Gaussian curvature, rather than analyzing the principal curvatures independently, fail on this example and on cones. They fail because the Gaussian curvature is everywhere zero, making impossible any segmentation based only upon the Gaussian curvature. Yet human observers readily and consistently segment into parts surfaces whose Gaussian curvature is everywhere zero.

3.6.2. Cones

For analysis of ruled surfaces we may take the *line of striction* to be the directrix without loss of generality (Do Carmo 1976, p. 190). The line of striction is a parametrized curve $\underline{\alpha}(u^1)$ such that $\underline{\alpha}_1 \cdot \underline{w}_1 = 0$, $u^1 \in (a, b)$ and α lies on the trace of x . In the case of a cone $\underline{\alpha}_1 = 0$ and the line of striction is simply the vertex of the cone. With this choice of directrix for cones $x_1 = u^2 \underline{w}_1$ and $x_2 = \underline{w}$. The metric tensor is

$$g_{ij} = \mathbf{x}_i \cdot \mathbf{x}_j = \begin{pmatrix} (u^2)^2 (\underline{w}_1 \cdot \underline{w}_1) & 0 \\ 0 & 1 \end{pmatrix}$$

The surface normal is

$$\mathbf{N} = \frac{\mathbf{x}_1 \times \mathbf{x}_2}{|\mathbf{x}_1 \times \mathbf{x}_2|} = \frac{u^2 (\underline{w}_1 \times \underline{w})}{|u^2| |\underline{w}_1 \times \underline{w}|} = \frac{\underline{w}_1 \times \underline{w}}{|\underline{w}_1|}$$

The second fundamental coefficients are

$$b_{ij} = \mathbf{x}_{ij} \cdot \mathbf{N} = \begin{pmatrix} u^2 (\underline{w}_1, \underline{w}, \underline{w}_{11}) / |\underline{w}_1| & 0 \\ 0 & 0 \end{pmatrix}$$

Since $g_{12} = b_{12} = 0$, the principal curvatures on a cone are

$$k_1 = b_{11}/g_{11} = \frac{u^2 (\underline{w}_1, \underline{w}, \underline{w}_{11})}{(u^2)^2 (\underline{w}_1 \cdot \underline{w}_1) |\underline{w}_1|} = \frac{(\underline{w}_1, \underline{w}, \underline{w}_{11})}{u^2 |\underline{w}_1|^3}$$

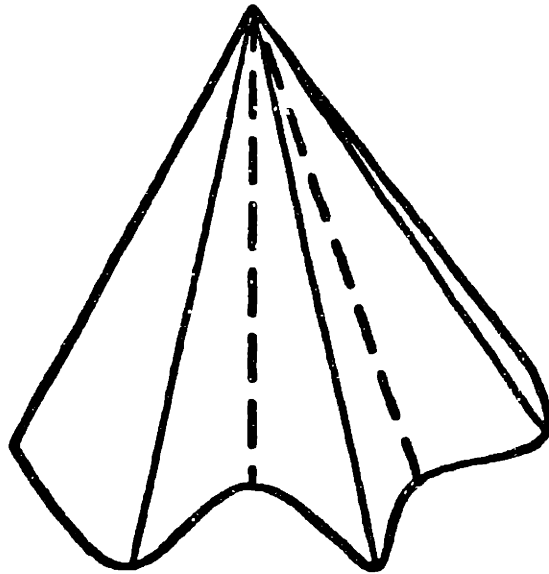
$$k_2 = b_{22}/g_{22} = 0$$

The u^1 and u^2 parameter curves are lines of curvature. As expected, the principal curvature along the u^2 parameter curve, k_2 , is everywhere zero. The expression for k_1 along the u^1 parameter curves (where u^2 is constant) does not depend on u^2 . Thus the contours of minima of k_1 are straight lines which pass through the vertex of the cone. An example cone is shown in figure 3-2 with segmentation contours indicated by dotted lines. The resulting parts seem the natural ones.

3.7. Segmentation of surfaces of revolution

A surface of revolution is a set $S \subset \mathbb{R}^3$ obtained by rotating a regular plane curve $\underline{\alpha}$ about an axis in the plane which does not meet the curve. Let the $x^1 x^3$ plane be the plane of $\underline{\alpha}$ and the x^2 axis the rotation axis. Let

Figure 2. A cone with dotted segmentation contours



$$\underline{\alpha}(u^1) = (x(u^1), z(u^1)), \quad a < u^1 < b, \quad z(u^1) > 0$$

Let u^2 be the rotation angle about the x^3 axis. Then we obtain a map

$$x(u^1, u^2) = (x(u^1) \cos(u^2), x(u^1) \sin(u^2), z(u^1))$$

from the open set $U = \{(u^1, u^2) \in \mathbb{R}^2; 0 < u^2 < 2\pi, a < u^1 < b\}$ into S (figure 3-3). The curve $\underline{\alpha}$ is called the *generating curve* of S , and the x^3 axis is the *rotation axis* of S . The circles swept out by the points of $\underline{\alpha}$ are called the *parallels* of S , and the various placements of $\underline{\alpha}$ on S are called the *meridians* of S .

Let $\cos(u^2)$ be abbreviated as c and $\sin(u^2)$ as s . Then $x_1 = (x_1c, x_1s, z_1)$ and $x_2 = (-x_1s, xc, 0)$. The first fundamental coefficients are then

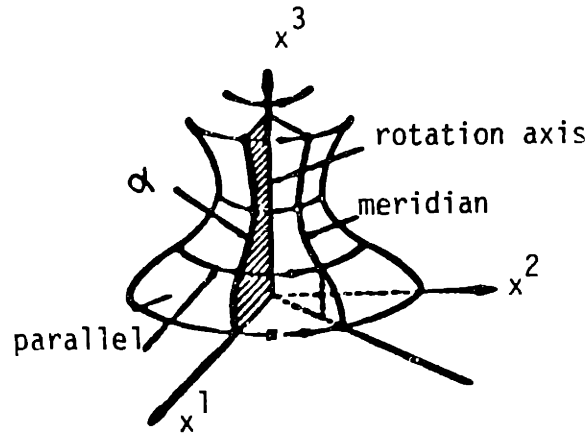
$$g_{ij} = x_i \cdot x_j = \begin{pmatrix} x_1^2 + z_1^2 & 0 \\ 0 & x_1^2 \end{pmatrix}$$

The surface normal is

$$N = \frac{x_1 \times x_2}{|x_1 \times x_2|} = \frac{(x_1c, x_1s, -x_1)}{\sqrt{x_1^2 + z_1^2}}$$

If we let u^1 be arc length along α then $\sqrt{x_1^2 + z_1^2} = 1 = g_{11}$ and

Figure 3. Surface of revolution



$$\mathbf{N} = (x_1 c, x_1 s, -x_1)$$

The second fundamental coefficients are

$$b_{ij} = \mathbf{x}_{ij} \cdot \mathbf{N} = \begin{pmatrix} x_{11}x_1 - x_1x_{11} & 0 \\ 0 & -x_1 \end{pmatrix}$$

Since $g_{12} = b_{12} = 0$ the principal curvatures of a surface of revolution are

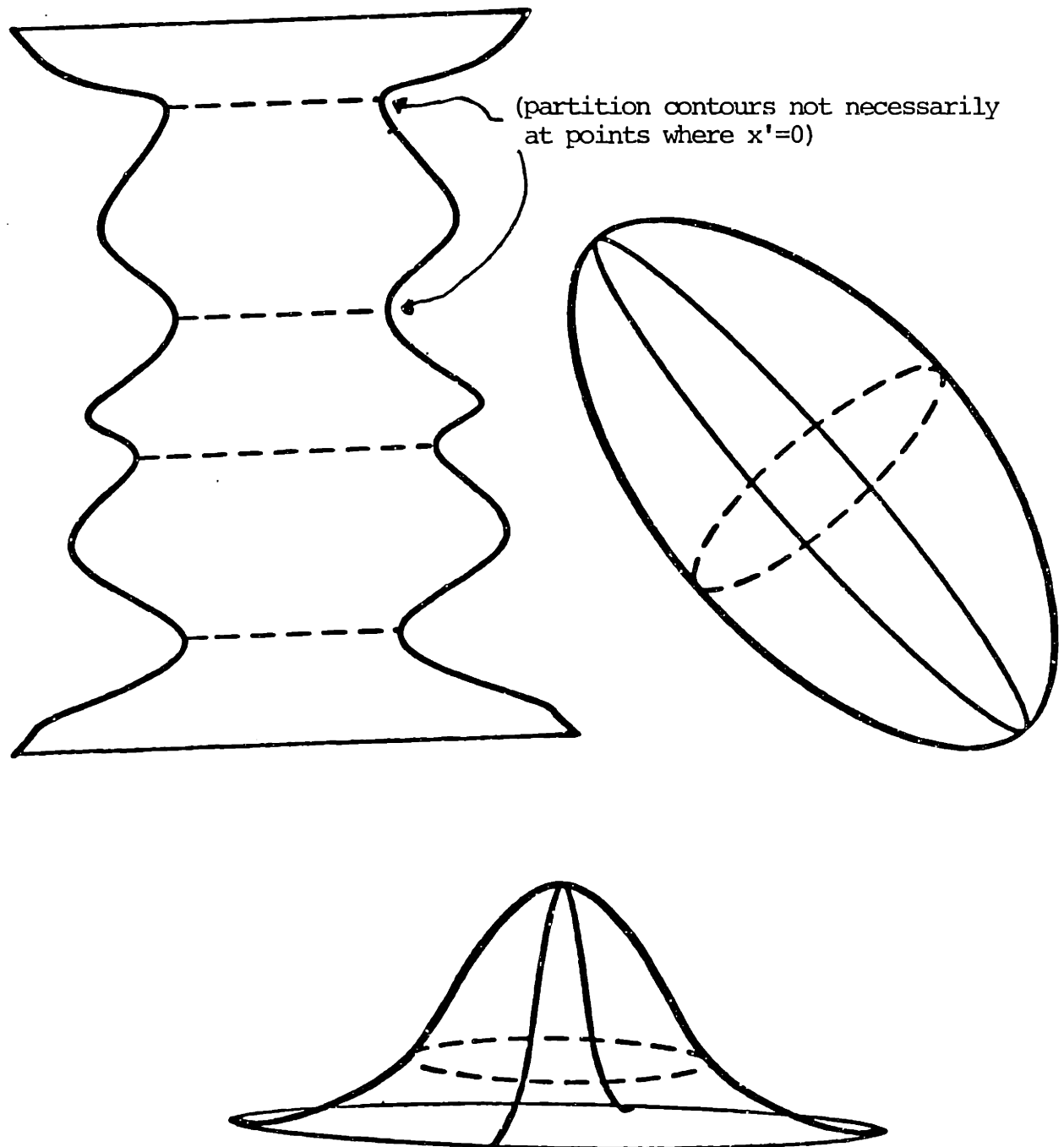
$$k_1 = b_{11}/g_{11} = x_{11}x_1 - x_1x_{11}$$

$$k_2 = b_{22}/g_{22} = -x_1/x$$

The expression for k_1 is identical to the expression for the curvature along α . In fact the meridians (the various positions of α on S) are lines of curvature, as are the parallels. The curvature along the meridians is given by the expression for k_1 and the curvature along the parallels is given by the expression for k_2 . The expression for k_2 is simply the curvature of a circle of radius x multiplied by the cosine of the angle that the tangent to α makes with the axis of rotation.

Observe that the expressions for k_1 and k_2 depend only upon the parameter u^1 , not u^2 . In particular, since k_2 is independent of u^2 there are no extrema or inflections of the normal curvature along the parallels. The parallels are circles. Consequently no segmentation contours arise from the lines of curvature associated with k_2 . Only the minima of k_1 along

Figure 4. Segmentation contours for surfaces of revolution.



the meridians are used for segmentation. Figure 3-4 shows several surfaces of revolution with the minima of curvature along the meridians marked. The resulting segmentation contours appear quite natural to human observers.

Figure 3-5 illustrates that reversing the orientation (of the surface normals) of a surface of revolution causes us to carve the same surface differently. The top and bottom figures are identical, except that one is rotated 180 degrees from the other. The dotted circular lines in the top figure are the segmentation contours according to our rule for surfaces. Note that they lie in the valleys of the top figure. In the bottom figure they no longer lie in the valleys but on the peaks. By reversing the field of surface normals the signs of the principal curvatures everywhere have reversed. Contours of minima become contours of maxima and vice versa. Consequently the part boundaries are not invariant under an orientation reversal.

3.8. Segmentation of the torus

A torus in \mathbb{R}^3 is the surface in \mathbb{R}^3 , shown in figure 3-6, which is obtained by revolving a circle about a line not passing through the circle. A convenient parametrization for the torus is

$$\mathbf{x}(u^1, u^2) = ((b + a \sin u^2)(\cos u^1), (b + a \sin u^2)(\sin u^1), a \cos u^2) \quad b > a$$

Let $\sin u^i$ be abbreviated s^i and $\cos u^i$ be abbreviated c^i . Then the first partials are $\mathbf{x}_1 = (-(b + as^2)c^1, (b + as^2)c^1, 0)$ and $\mathbf{x}_2 = (ac^2c^1, ac^2s^1, -as^2)$. The metric tensor is

$$g_{ij} = \mathbf{x}_i \cdot \mathbf{x}_j = \begin{pmatrix} (b + as^2)^2 & 0 \\ 0 & a^2 \end{pmatrix}$$

The surface normal is

$$\mathbf{N} = \frac{\mathbf{x}_1 \times \mathbf{x}_2}{|\mathbf{x}_1 \times \mathbf{x}_2|} = (-c^1s^2, -s^1s^2, -c^2)$$

The second fundamental coefficients are

$$b_{ij} = \mathbf{x}_{ij} \cdot \mathbf{N} = \begin{pmatrix} (b + as^2)s^2 & 0 \\ 0 & a \end{pmatrix}$$

Figure 5. Part boundaries shift when surface orientation reverses

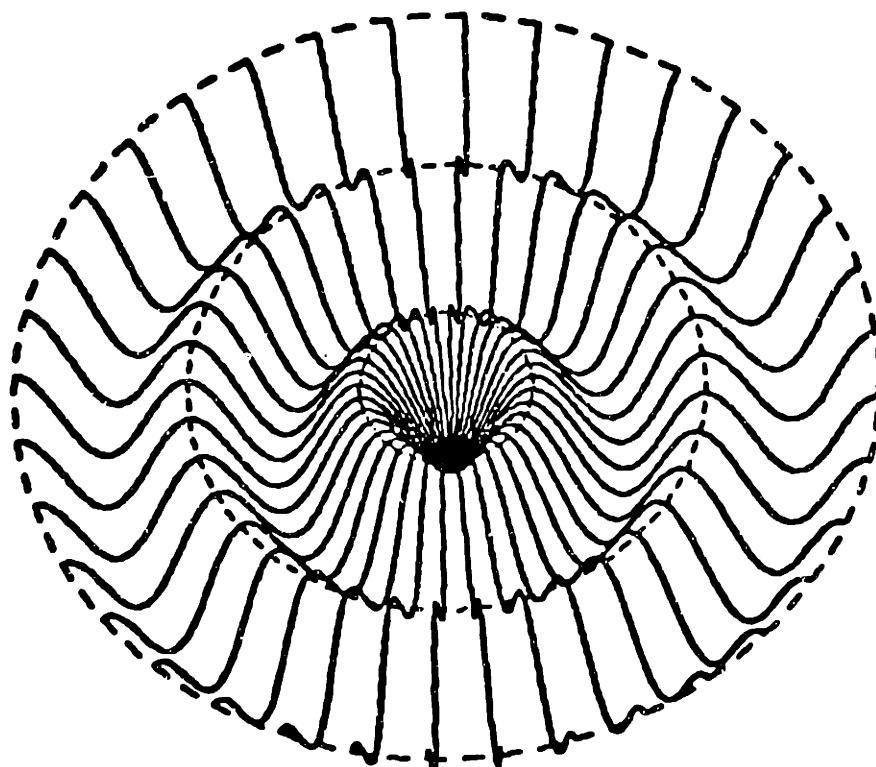
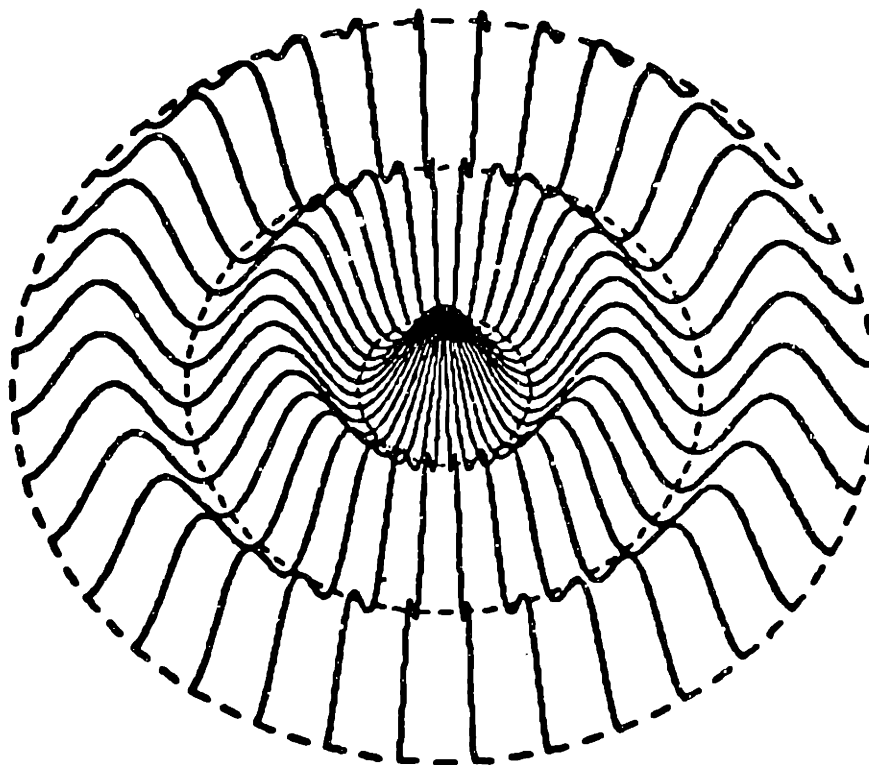
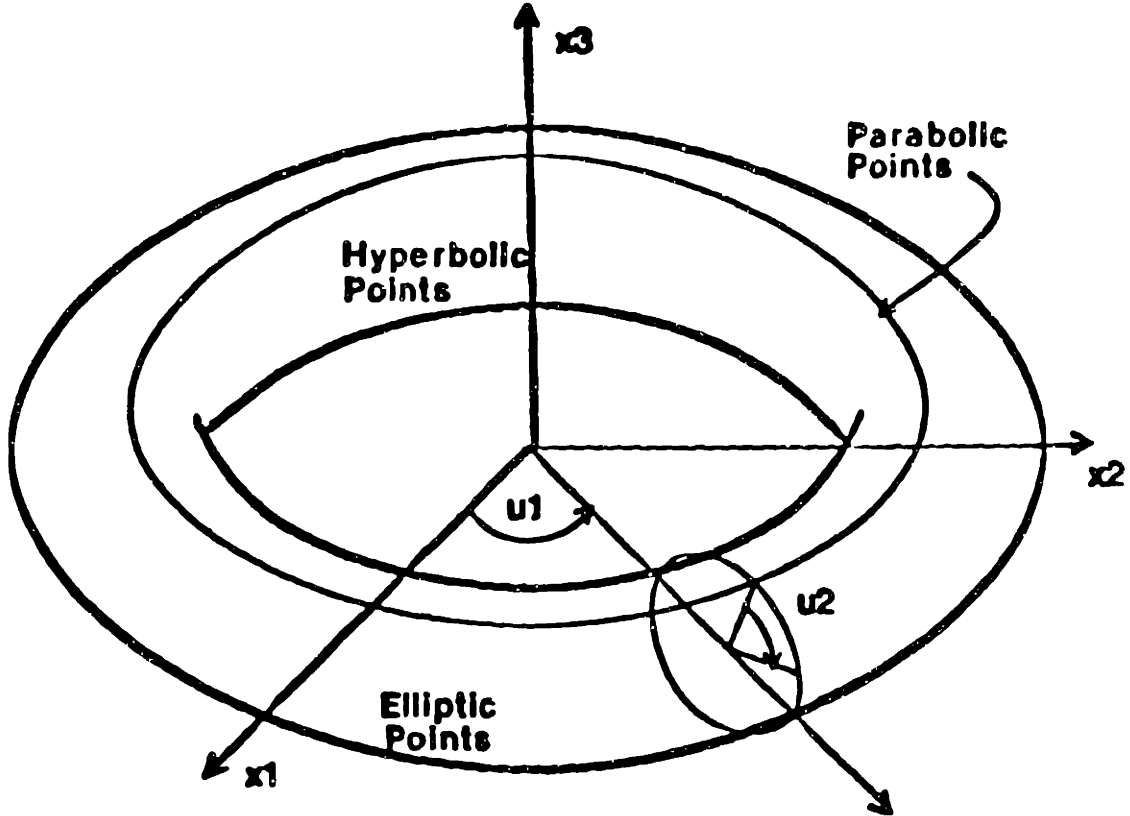


Figure 6. A torus



Since $g_{12} = b_{12} = 0$ the u^1 and u^2 parameter curves are lines of curvature and the principal curvatures of a torus are

$$k_1 = b_{11}/g_{11} = s^2/(b + as^2)$$

$$k_2 = b_{22}/g_{22} = a^{-1}$$

The principal curvature k_1 is associated with the u^1 parameter curves and k_2 with the u^2 parameter curves. k_2 is a constant so the torus is not segmented using the u^2 parameter curves. k_1 is not a constant, but it is independent of u^1 . Therefore the torus is not segmented using the u^1 parameter curves either. The conclusion is that the torus is one indivisible unit based on our segmentation rule.

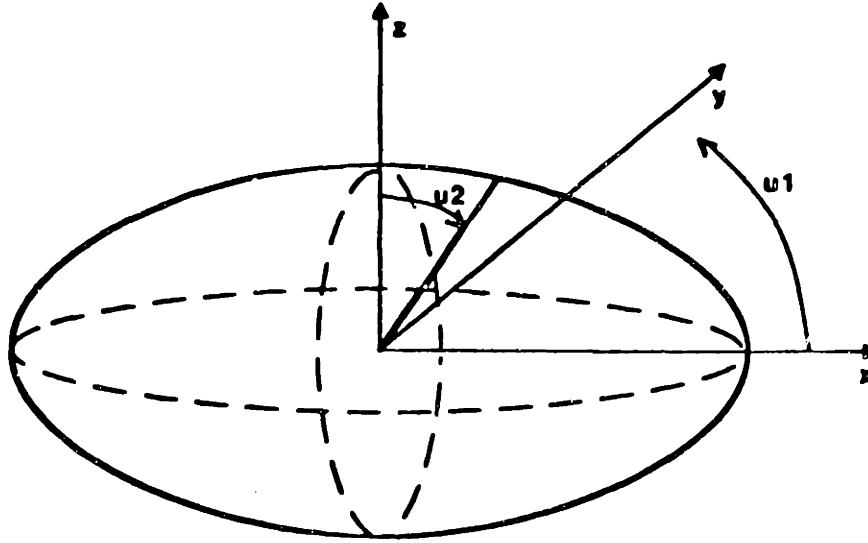
This is a different result than would be obtained using a segmentation rule proposed by Koenderink and Van Doorn (1975, 232). They suggest that hyperbolic points of a surface be used to divide it into parts. The hyperbolic points of a torus are indicated in figure 3-6.

3.9. Segmentation of the general ellipsoid

Let the ellipsoid be given in spherical coordinates by (figure 3-7)

$$x(u^1, u^2) = (a \sin u^2 \cos u^1, b \sin u^2 \sin u^1, \cos u^2) \quad a > b > 1$$

Figure 7. General ellipsoid: dotted contours are partitions



Let subscripted s 's and c 's indicate sines and cosines of the parameters. Then

$$\frac{\partial \mathbf{x}}{\partial u^1} = \mathbf{x}_1 = (-as_1s_2, bs_2c_1, 0)$$

$$\frac{\partial \mathbf{x}}{\partial u^2} = \mathbf{x}_2 = (ac_1c_2, bc_2s_1, -s_2)$$

The metric tensor is

$$g_{ij} = \mathbf{x}_i \cdot \mathbf{x}_j = \begin{pmatrix} s_2^2(a^2s_1^2 + b^2c_1^2) & s_1c_1s_2c_2(b^2 - a^2) \\ s_1c_1s_2c_2(b^2 - a^2) & c_2^2(a^2c_1^2 + b^2s_1^2) + s_2^2 \end{pmatrix}$$

The surface normal is

$$\mathbf{N} = \frac{\mathbf{x}_1 \times \mathbf{x}_2}{|\mathbf{x}_1 \times \mathbf{x}_2|} = (-bs_2c_1, -as_1s_2, -abc_2)/d$$

where

$$d = \sqrt{b^2s_2^2c_1^2 + a^2s_2^2s_1^2 + a^2b^2c_2^2}$$

The second partials are

$$\mathbf{x}_{11} = (-as_2c_1, -bs_1s_2, 0)$$

$$\mathbf{x}_{12} = (-ac_2s_1, bc_1c_2, 0)$$

$$\mathbf{x}_{22} = (-as_2c_1, -bs_1s_2, -c_2)$$

So the second fundamental coefficients are

$$b_{ij} = \mathbf{x}_{ij} \cdot \mathbf{N} = \begin{pmatrix} abs_2^2/d & 0 \\ 0 & ab/d \end{pmatrix}$$

Since $g_{12} = g_{21} \neq 0$ the u^1 and u^2 -parameter curves are not in general lines of curvature on the ellipsoid. The differential equation for the lines of curvature on any surface is (Lipschutz 1969)

$$(g_{11}b_{12} - b_{11}g_{12})du^1 du^1 + (g_{11}b_{22} - b_{11}g_{22})du^1 du^2 + (g_{12}b_{22} - g_{22}b_{12})du^2 du^2 = 0$$

which for an ellipsoid becomes quite unwieldy. Rather than solve a differential equation for the lines of curvature and then try to find the equations for the principal curvatures along the lines of curvature, we exploit the symmetry of the ellipsoid to determine the contours of minima of normal curvature along the associated lines of curvature. We will show that the curves $u^2 = \pi/2$ and $u^1 = 0$ are lines of curvature. We will then show that the minima of normal curvature on these curves all lie on the curve $u^1 = \pi/2$. Then by the symmetry of the ellipsoid it will follow that all such minima must lie on the curve $u^1 = \pi/2$, making this curve one of the two partitioning contours determined by our rule. A similar approach will show that the curve $u^1 = 0$ is the second partitioning contour.

We note first that for the parameter curve $u^2 = \pi/2$ the metric tensor becomes

$$g_{ij} = \mathbf{x}_i \cdot \mathbf{x}_j = \begin{pmatrix} a^2 s_1^2 + b^2 c_1^2 & 0 \\ 0 & 1 \end{pmatrix}$$

and the second fundamental coefficients become

$$b_{ij} = \mathbf{x}_{ij} \cdot \mathbf{N} = \begin{pmatrix} ab(b^2 c_1^2 + a^2 s_1^2)^{-1/2} & 0 \\ 0 & ab(b^2 c_1^2 + a^2 s_1^2)^{-1/2} \end{pmatrix}$$

Since $g_{12} = 0$ and $b_{12} = 0$ the curve $u^2 = \pi/2$ is a line of curvature. The associated principal curvature is

$$\kappa = b_{11}/g_{11} = ab(a^2 s_1^2 + b^2 c_1^2)^{-3/2}$$

The extrema of curvature occur when $\partial\kappa/\partial u^1 = 0$.

$$\frac{\partial\kappa}{\partial u^1} = -\frac{3}{2}ab(a^2 s_1^2 + b^2 c_1^2)^{-5/2}(2a^2 c_1 s_1 - 2b^2 s_1 c_1) = 0$$

This implies that

$$b^2 s_1 c_1 = a^2 c_1 s_1$$

which occurs when $u^1 = n\pi/2$, $n = 0, 1, 2, 3$. Thus the extrema of normal curvature along this particular line of curvature occur at the four points where the line of curvature intersects the x and y axes. The points of intersection on the x axis are the images of the parameter points $(u^1, u^2) = (0, \pi/2)$ and $(u^1, u^2) = (\pi, \pi/2)$. The points of intersection on the y axis are the images of the parameter points $(u^1, u^2) = (\pi/2, \pi/2)$ and $(u^1, u^2) = (-\pi/2, \pi/2)$. At the points of intersection with the x axis the relevant principal curvature is $\kappa = a/b^2$. At the points of intersection with the y axis $\kappa = b/a^2$. Since $a > b > 1$ then $a/b^2 > b/a^2$ and the minima are the points on the y axis.

Next we consider the curve $u^1 = 0$. Along this curve the metric tensor simplifies to

$$g_{ij} = \mathbf{x}_i \cdot \mathbf{x}_j = \begin{pmatrix} b^2 s_2^2 & 0 \\ 0 & a^2 c_2^2 + s_2^2 \end{pmatrix}$$

The second fundamental coefficients simplify to

$$b_{ij} = \mathbf{x}_{ij} \cdot \mathbf{N} = \begin{pmatrix} a s_2^2 / \sqrt{s_2^2 + a^2 c_2^2} & 0 \\ 0 & a / \sqrt{s_2^2 + a^2 c_2^2} \end{pmatrix}$$

Since g_{12} and b_{12} are both zero the curve $u^1 = 0$ is a line of curvature. The associated normal curvature along this line of curvature is

$$\kappa = b_{22}/g_{22} = a(s_2^2 + a^2 c_2^2)^{-3/2}$$

with the partial derivative

$$\frac{\partial \kappa}{\partial u^2} = -\frac{3}{2} a (a^2 c_2^2 + s_2^2)^{-5/2} (2s_2 c_2 - 2a^2 c_2 s_2) = 0$$

This implies that

$$a^3 c_2 s_2 = a s_2 c_2$$

which occurs when $u^2 = n\pi/2$, $n = 0, 1, 2, 3$. Thus the extrema of the normal curvature associated with this line of curvature occur where the line of curvature intersects the x and z axes. At the intersection with the x axis the relevant principal curvature is $\kappa = a$. At the

intersection with the z axis $\kappa = a^{-2}$. Thus the minima occur along the z axis at the images of the parameter points $(u^1, u^2) = (0, 0)$ and $(u^1, u^2) = (0, \pi)$.

Combining this result with the minima from the curve $u^2 = \pi/2$ we find that both pairs of minima lie on the curve $u^1 = \pi/2$. By the symmetry of the ellipsoid it is clear that all the other minima of normal curvature from this family of lines of curvature must also lie on the curve $u^1 = \pi/2$. Thus this curve, which is the intersection of the yz plane with the ellipsoid, is a partition of the ellipsoid as defined by our rule.

Next we consider the curve $u^1 = \pi/2$. Along this curve the metric tensor simplifies to

$$g_{ij} = \mathbf{x}_i \cdot \mathbf{x}_j = \begin{pmatrix} a^2 s_2^2 & 0 \\ 0 & b^2 c_2^2 + s_2^2 \end{pmatrix}$$

The second fundamental coefficients simplify to

$$b_{ij} = \mathbf{x}_{ij} \cdot \mathbf{N} = \begin{pmatrix} b s_2^2 / \sqrt{s_2^2 + b^2 c_2^2} & 0 \\ 0 & b / \sqrt{s_2^2 + b^2 c_2^2} \end{pmatrix}$$

Since g_{12} and b_{12} are both zero the curve $u^1 = \pi/2$ is a line of curvature. The normal curvature associated with this line of curvature is

$$\kappa = b_{22}/g_{22} = b(s_2^2 + b^2 c_2^2)^{-3/2}$$

with the partial derivative

$$\frac{\partial \kappa}{\partial u^2} = -\frac{3}{2} b (b^2 c_2^2 + s_2^2)^{-5/2} (2s_2 c_2 - 2b^2 c_2 s_2) = 0$$

This implies that

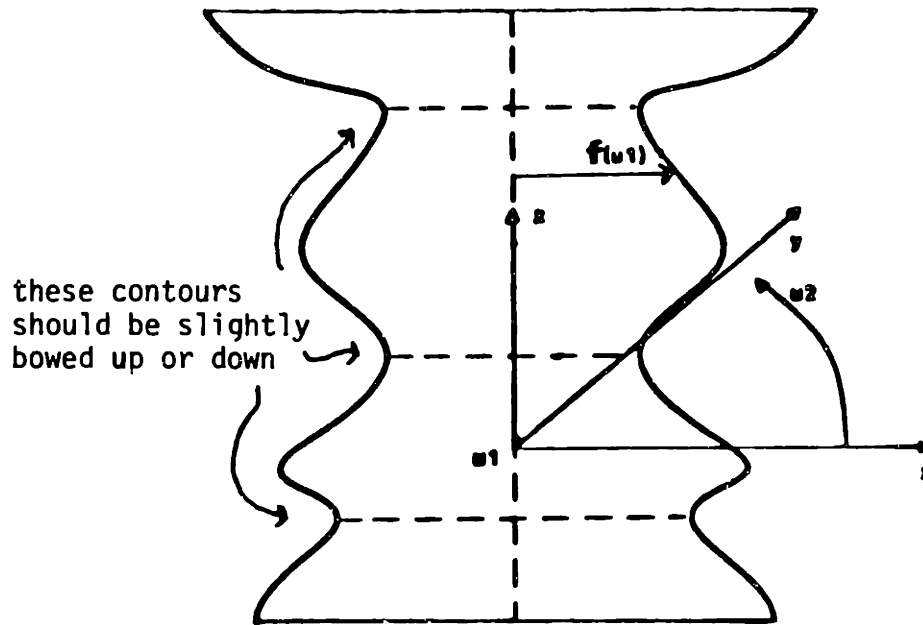
$$b^3 c_2 s_2 = b s_2 c_2$$

which occurs when $u^2 = n\pi/2$, $n = 0, 1, 2, 3$. Thus the extrema of the normal curvature associated with this line of curvature occur where the line of curvature intersects the y and z axes. At the intersection with the z axis the relevant principal curvature is $\kappa = b^{-2}$. At the intersection with the y axis $\kappa = b$. Thus the minima occur along the z axis, at the images of the parameter points $(u^1, u^2) = (\pi/2, 0)$ and $(u^1, u^2) = (\pi/2, \pi)$. These minima both lie on the curve $u^1 = 0$. By the symmetry of the ellipsoid it is clear that all the other minima associated with this family of lines of curvature must also lie on this curve. Thus the second partitioning contour defined by our rule is the curve $u^1 = 0$ (actually without the

two points where this contour intersects the x axis). This is the contour of intersection of the ellipsoid with the xz plane. The first partitioning contour was the intersection with the yz plane. These contours may be imagined better by looking at figure 3-7.

Should these two partitioning contours be treated independently, or should they be taken together to define the part boundaries? Treated independently they each divide the ellipsoid into halves. Taken together they divide the ellipsoid into quadrants. The division into halves seems the more natural, suggesting that the contours be treated independently. The fact that for some surfaces only one family of lines of curvature gives rise to partitioning contours also argues that the partitioning contours from the two families of lines of curvature be treated independently.

Figure 8. Flattened SOR: dotted contours are partitions



3.10. Flattened surfaces of revolution

Let a flattened surface of revolution be given in cylindrical coordinates by (figure 3-8)

$$\mathbf{x}(u^1, u^2) = (f(u^1) \cos(u^2), a f(u^1) \sin(u^2), u^1) \quad 0 < a < 1$$

Let s 's and c 's indicate sines and cosines of u^2 . Let $f(u^1)$ be abbreviated to f and let dots over the f 's indicate derivatives with respect to u^1 . Then the metric tensor is

$$g_{ij} = \mathbf{x}_i \cdot \mathbf{x}_j = \begin{pmatrix} \dot{f}^2(c^2 + a^2s^2) + 1 & f\dot{f}sc(a^2 - 1) \\ f\dot{f}sc(a^2 - 1) & f^2(s^2 + a^2c^2) \end{pmatrix}$$

The second fundamental form is

$$b_{ij} = \mathbf{x}_{ij} \cdot \mathbf{N} = \begin{pmatrix} -a\ddot{f}/d & 0 \\ 0 & a\dot{f}/d \end{pmatrix}$$

where $d = \sqrt{a^2c^2 + s^2 + a^2\dot{f}^2}$

Since $\mathbf{x}_1 \cdot \mathbf{x}_2 \neq 0$ in general, the parameter curves are not in general lines of curvature. However when $\dot{f} = 0$ then $\mathbf{x}_1 \cdot \mathbf{x}_2 = 0$ so that contours where this holds are lines of curvature. These contours are elliptical cross sections of the flattened surface of revolution, cross sections having either the greatest or least major axis locally.

Along these lines of curvature the associated principal curvature is

$$\kappa = b_{22}/g_{22} = af^{-1}(s^2 + a^2c^2)^{-3/2}$$

Its extrema occur when

$$\partial\kappa/\partial u^2 = -3.5af^{-1}(a^2c^2 + s^2)^{-5/2}(-2a^2cs + 2cs) = 0$$

which happens when $a^2sc = sc$. This implies that $u^2 = n\pi/2$, $n = 0, 1, 2, 3$. For n even $\kappa = a^{-2}f^{-1}$ and for n odd $\kappa = af^{-1}$. Thus the minima occur when u^2 is $\pi/2$ or $3\pi/2$. All such points lie on the intersection of the yz plane with the flattened surface of revolution. Since the yz plane is a plane of symmetry of the surface it follows that the two contours where it intersects the surface constitute one partition defined by the surface partitioning rule. This is a "weak" partition, one where the value of the principal curvature is always positive at its minimum.

To determine the other family of partitioning contours we begin by noting that when u^2 is $n\pi/2$ the metric tensor becomes

$$g_{ij} = \mathbf{x}_i \cdot \mathbf{x}_j = \begin{pmatrix} \dot{j}^2 + 1 & 0 \\ 0 & f^2 a^2 \end{pmatrix}$$

for n even, and

$$g_{ij} = \mathbf{x}_i \cdot \mathbf{x}_j = \begin{pmatrix} \dot{j}^2 a^2 + 1 & 0 \\ 0 & f^2 \end{pmatrix}$$

for n odd. Thus the u^1 -parameter curves given by $u^2 = n\pi/2$ are lines of curvature. These curves are also the intersection of the flattened surface of revolution with the xz plane or the yz plane. For n even the associated principal curvature is

$$\kappa = b_{11}/g_{11} = -\dot{j}(1 + \dot{j}^2)^{-3/2}$$

and for n odd it is

$$\kappa = b_{11}/g_{11} = -a\dot{j}(1 + a\dot{j}^2)^{-3/2}$$

The extrema of these two curvatures do not, in general, occur at the same values of u .⁶ Thus the partitioning contours on the flattened surface of revolution are not, in general, planar. So as a surface of revolution is flattened, the partitioning contours which are at first circles become more elliptical and usually bow either up or down slightly.

⁶I am grateful to Alan Yuille for pointing this out to me, and for being of great assistance in several of the derivations in this chapter. Yuille also disproved the interesting conjecture that a contour defined by the surface partitioning rule always lies on a line of curvature from the opposite family of lines of curvature. He found the right conoid to be a counterexample.

3.11. Surfaces with discontinuities

Our partitioning rule is easily extended to surfaces with discontinuities of the tangent plane. These discontinuities come in two basic classes, concave and convex. Intuitively, a concave discontinuity points into the object and a convex discontinuity points out of the object. Then by the argument advanced in chapter 2 (figure 2-2) a surface should be broken into parts at concave discontinuities, not at convex discontinuities. The argument is that when two objects are intersected their line of intersection is always (with probability one) a concave discontinuity regardless of the shapes of the two objects.

3.12. Elbows

An apparent problem for the partitioning rules we have discussed in this chapter are "elbows" (figure 3-9a). The problem with elbows is that they prevent a closed partitioning contour. Consequently the partition is always incomplete. This is true not only for surfaces with discontinuities of the tangent plane but also for smooth surfaces with elbows.

As can be seen in figure 3-9b, however, there is good reason for the rule to only specify part of the segmentation contour – the appropriate way to continue the segmentation is inherently ambiguous. Each of the three divisions shown in figure 3-9b is equally good (or bad).

We can augment our rule to handle this circumstance by suggesting that the best contour of division is the one with the smallest arc length. This eliminates the second possibility shown in figure 3-9b. This also explains why we prefer only one of the possible divisions in figure 3-9c.

Elbows may also occur on entirely smooth surfaces. For example, a torus which has been scaled along one axis has two elbows. The following derivation will show that the surface partitioning rule gives rise to two open semicircular contours, one on the inside of each elbow.

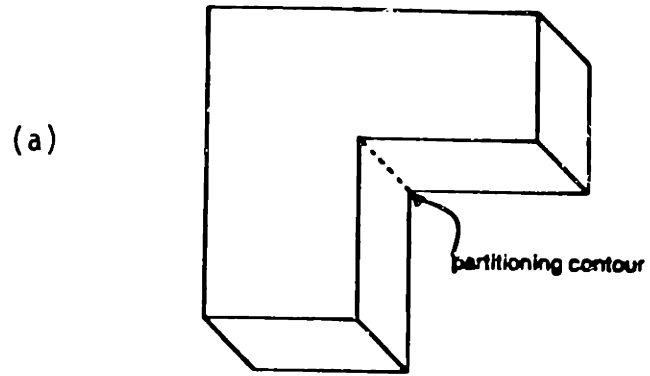
Let the torus be given by the parametrization

$$\mathbf{x}(u^1, u^2) = ((b + a \sin u^2)(\cos u^1), d(b + a \sin u^2)(\sin u^1), a \cos u^2) \quad b > a, d > 1$$

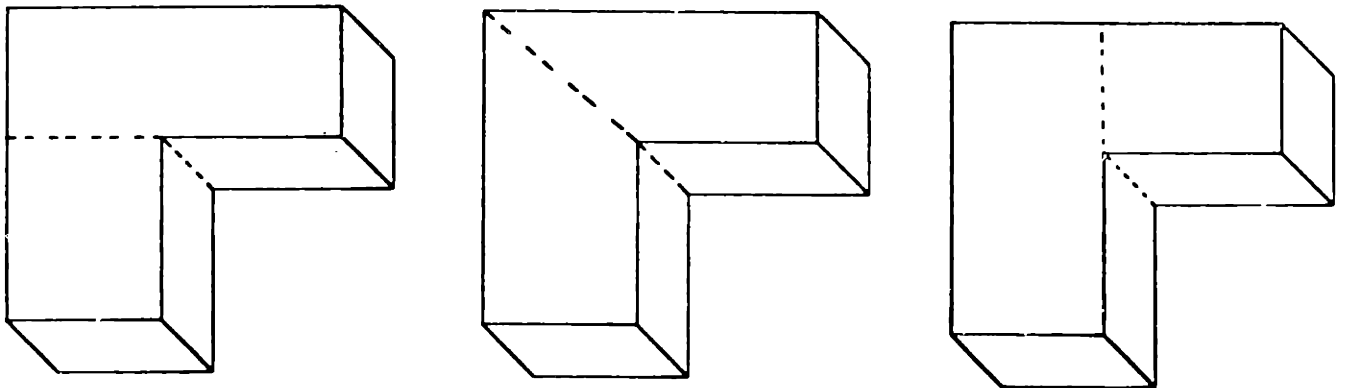
This corresponds in figure 3-6 to expanding the torus along the x^2 -axis. Let subscripted s 's and c 's indicate sines and cosine of the parameters. Then the first partials are $\mathbf{x}_1 = (-(b + as_2)s_1, d(b + as_2)c_1, 0)$ and $\mathbf{x}_2 = (ac_2c_1, adc_2s_1, -as_2)$.

The metric tensor is

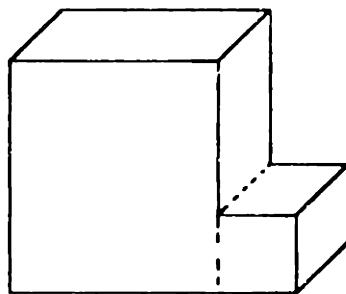
Figure 9. Segmentation of elbows.



(b)



(c)



$$g_{ij} = \mathbf{x}_i \cdot \mathbf{x}_j = \begin{pmatrix} (b + as_2)^2(s_1^2 + d^2c_1^2) & ac_2c_1s_1(b + as_2)(d^2 - 1) \\ ac_2c_1s_1(b + as_2)(d^2 - 1) & a^2(c_2^2c_1^2 + d^2c_2^2s_1^2 + s_2^2) \end{pmatrix}$$

The surface normal is

$$\mathbf{N} = \frac{\mathbf{x}_1 \times \mathbf{x}_2}{|\mathbf{x}_1 \times \mathbf{x}_2|} = (-ds_2c_1, -s_1s_2, -dc_2)/f$$

where

$$f = \sqrt{d^2s_2^2c_1^2 + s_1^2s_2^2 + d^2c_2^2}$$

The second fundamental coefficients are

$$b_{ij} = \mathbf{x}_{ij} \cdot \mathbf{N} = \begin{pmatrix} ds_2(b + as_2)/f & 0 \\ 0 & ad/f \end{pmatrix}$$

Since $g_{12} \neq 0$ the u^1 - and u^2 -parameter curves are not in general lines of curvature. However, along the curve $u^2 = \pi/2$ we have $c_2 = 0$, $s_2 = 1$, and

$$g_{ij} = \mathbf{x}_i \cdot \mathbf{x}_j = \begin{pmatrix} (b + a)^2(s_1^2 + d^2c_1^2) & 0 \\ 0 & a^2 \end{pmatrix}$$

implying that this is a line of curvature. The second fundamental coefficients are

$$b_{ij} = \mathbf{x}_{ij} \cdot \mathbf{N} = \begin{pmatrix} d(b + a)/h & 0 \\ 0 & ad/h \end{pmatrix}$$

where

$$h = \sqrt{d^2c_1^2 + s_1^2}$$

The principal curvature along this line of curvature is

$$\kappa = b_{11}/g_{11} = d(b + a)^{-1}h^{-3}$$

The extrema of curvature occur where $\partial\kappa/\partial u^1 = 0$.

$$\frac{\partial\kappa}{\partial u^1} = -1.5d(b + a)^{-1}(d^2c_1^2 + s_1^2)^{-5/2}(2s_1c_1 - 2d^2s_1c_1) = 0$$

Since $d > 1$ this implies that $d^2 c_1 s_1 = c_1 s_1$, which occurs for $u^1 = n\pi/2$ and n a natural number. Maxima of curvature occur when n is odd, minima when n is even.

A similar analysis shows that the contour $u^2 = -\pi/2$ is a line of curvature whose extrema of curvature occur for $u^1 = n\pi/2$ where n is a natural number. The difference is that maxima of curvature occur when n is even, minima when n is odd.

Finally, at the parameter point $(\pi/2, 0)$ we have that $s_1 = 1, c_1 = 0, s_2 = 0, c_2 = 1$ and find that the metric tensor is

$$g_{ij} = \mathbf{x}_i \cdot \mathbf{x}_j = \begin{pmatrix} b^2 & 0 \\ 0 & a^2 d^2 \end{pmatrix}$$

implying that at this point the u^1 - and u^2 -parameter curves are in principal directions. The second fundamental form is

$$b_{ij} = \mathbf{x}_{ij} \cdot \mathbf{N} = \begin{pmatrix} 0 & 0 \\ 0 & ad \end{pmatrix}$$

Hence $\kappa_1 = b_{11}/g_{11} = 0$. By symmetry this also holds for κ_1 at the parameter points $(-\pi/2, 0), (-\pi/2, \pi), (\pi/2, \pi)$. At each of the two elbows then we have found that the innermost point of the elbow is a minima of κ_1 , the outermost is a maxima, the uppermost and lowermost points have $\kappa_1 = 0$. By symmetry, then we have that the two partitioning contours at the elbows are the open semicircles $u^1 = \pi/2, \pi < u^2 < 0$ and $u^1 = -\pi/2, \pi < u^2 < 0$. Again, a rule which says to complete unclosed partitioning contours with the shortest path would give the appropriate result.

3.13. Summary

In this chapter I propose the following rule for partitioning smooth surfaces into parts: *Divide a surface into parts at minima, along lines of greatest curvature, of the greatest principal curvature. Or divide a surface into parts at minima, along lines of least curvature, of the least principal curvature.*

I argue this rule is useful for the following reasons: 1) the parts defined by this rule do not vary with changes in viewing geometry, 2) when two objects are intersected their (smoothed) contour of intersection corresponds to the contour defined by this rule (when the minima of normal curvature are negative) and 3) the parts defined by this rule over all surfaces analyzed thus far look quite natural.

Much is left to do to obtain a representation of surfaces for recognition. It would be desirable to obtain an exhaustive classification of all possible surface parts, as the five

codon types are for plane curve parts. (In the case of surfaces of revolution the possible surface parts are in fact direct extensions of the five codon types for plane curves. This is not true for more general surfaces.) It would then be desirable to construct a vocabulary to state the spatial relationships between the parts. However, a clear and well motivated rule for defining surface parts is an important first step.

4. Discovering natural scales

In chapter 2 we used some basic tools of differential geometry to develop a representation for recognition of smooth and piece-wise smooth plane curves. Of particular importance in this development were the notions of the tangent vector and the curvature at a point on a curve. Codon part boundaries, for instance, were defined using minima of curvature.

We also noted that the codon description of a curve depends upon the resolution at which the curve is examined. This lead us to postulate nested hierarchies of codon descriptions for a single object.

Now the notion of nested hierarchies of codon descriptions, of codons nested within larger codons, is strictly nonsense from the point of view of differential geometry. If codon part boundaries are defined as being at minima of curvature, then given some analytic curve there is one and only one set of minima of curvature along that curve and therefore only one level of codon description. There can be no hierarchy. The reason, of course, is that differential geometry defines the tangent and curvature at a point entirely in terms of properties of the curve in a vanishingly small neighborhood of the point, leading to a unique answer at each point.¹ As long as the tangent and curvature are uniquely defined at each point then there is really only one level of codon description.

Our visual systems, however, can assign more than one tangent to a single point on a curve. Figure 4-1, for instance, shows an ellipse with a high frequency sinusoidal modulation added. If we inspect the curve closely the tangents we assign are largely due to the high frequency sinusoid. If we inspect the curve less carefully the tangents we assign are predominated by the ellipse. At any one point on this curve, then, we can assign two independent tangents. In some cases these tangents are identical, but at most points these two tangents are different. In fact one can easily construct examples where the two tangents are orthogonal.

From this demonstration we conclude that the rules our eyes use to assign tangents to points on a curve are not the same rules used in traditional differential geometry. In particular, our eyes assign tangents in a manner which depends upon the scale of resolution. Further, our eyes are not capricious in their choice of scales. Rather, the choice of scales and the subsequent assignment of tangents is quite consistent across observers. Perhaps, then, our use of the word "assign" is misleading as a description of how we choose tangents. "Discover" is closer to the truth.

¹Actually the tangent and curvature for a plane curve are unique up to a change in orientation of the curve.

In this chapter we implement the theory of chapter 2. That is, we develop a set of algorithms which begin with a bitmap representation of a curve and deliver a codon description. To do this we need more theory, a theory of "natural scale" that allows us to define what we mean by the tangent and curvature on curves where the standard definitions cannot apply. The standard definitions of tangent and curvature require a curve to be, respectively, once and twice continuously differentiable. Our bitmap curves, however, are nowhere differentiable. Their nondifferentiability arises from two sources. First, the contours in the world which give rise to the image contours are *fractals* (Mandelbrot 1977 and 1982, also Witkin 1981). Fractals are nondifferentiable functions. Second, the imaging process fractures curves into small segments in order to represent them on a discrete grid.

After developing definitions for the tangents and curvature on nondifferentiable curves we turn to the problem of integrating the local tangents and curvature along a curve into an analytic and more global description. From this description we compute the codon description.

The implementation is written in zeta-lisp and runs on a Lisp Machine with an attached four plane color monitor. Some example contours and the codon descriptions computed by the lisp code are examined. The lisp code itself is presented in the appendix.

4.1. Differential quantities on fractal curves

4.1.1. Tangents on fractal curves

It is helpful to review how the tangent is defined for differentiable curves to see how best to adapt the notion of tangent to the case of nondifferentiable curves.

The tangent direction of a curve $\alpha: [a, b] \mapsto \mathbb{R}^2$ at any time t is the direction of the limit of lines through $\alpha(t)$ and $\alpha(t')$ as $t' \rightarrow t$. This limit exists if $\alpha'(t)$ exists and is non-zero. If α' is continuous at t , then this limit can be described as the limit, as $t_1, t_2 \rightarrow t$, of the direction of the line through $\alpha(t_1)$ and $\alpha(t_2)$ since this line is parallel to the tangent through $\alpha(\zeta)$ for some ζ between t_1 and t_2 (see figure 4-1a).

If one plots the orientation of the line between $\alpha(t_1)$ and $\alpha(t_2)$ as the two points approach $\alpha(t)$ one expects a curve such as that shown in figure 4-1b. The curve wanders unpredictably when the points are distant from $\alpha(t)$. As the points draw near to $\alpha(t)$ the curve asymptotes to the tangent direction.

If a similar plot is constructed for a fractal curve, the random behavior of the plot persists regardless how close the two points approach the point of interest (see figure 4-1c).

The tangent simply does not exist in the classical sense. One can specify the tangent meaningfully only if one simultaneously specifies the *scale*.²

For digitized fractal curves, the curves of interest in our application, there is a lower bound on the minimum scale at which a tangent can be computed. The two points used to determine a tangent can get no closer to the pixel of interest than its two neighboring pixels.³ Digitized fractal curves exhibit the wandering tangent illustrated in figure 4-1c. At smaller scales, however, the randomness in tangent angle is due more to the quantization of the curve than to the fractal nature of the curve itself.

4.1.2. Curvature on fractal curves

A story similar to the one for tangents also applies to curvature.

The curvature of a curve $\alpha: [a, b] \mapsto \mathbb{R}^2$ at any time t is the inverse of the radius of the osculating circle to $\alpha(t)$. The osculating circle is the limit of the circles through $\alpha(t_1)$, $\alpha(t_2)$ and $\alpha(t_3)$ as $t_1, t_2, t_3 \rightarrow t$ (see figure 4-2).

If one plots the curvature of the circle defined by the three points as they approach $\alpha(t)$ one obtains a graph having an asymptote like figure 4-1b. If a similar plot is constructed for a fractal, the result is like figure 4-1c, having no asymptotic value. Again the curvature simply does not exist in the classical sense. One specifies the curvature at a point meaningfully only in conjunction with the simultaneous specification of a scale.

4.2. Natural scale

The tangent angle and curvature at a point on a nondifferentiable curve are fundamentally contingent upon scale. Yet they are critical for the codon representation of curves. It seems then that the codon scheme, developed as it was for well behaved smooth curves, must give multiple (a euphemism for "a very large number of") descriptions for nondifferentiable curves *even when the viewing geometry does not vary*. This would be quite unfortunate since any real imaging system delivers only discretized curves. A hierarchy of codon descriptions is desirable, a continuum of descriptions is not.

²The scale of resolution discussed here should not be confused with the overall scaling of a curve mentioned in the previous chapter. The codon description is invariant under an overall scaling of a curve because it is based on extrema and inflections of curvature. The codon description is not invariant under changes in scale of resolution since at different scales of resolution different extrema and inflections become accessible to the observer.

³Pixel is short for "picture element".

Figure 1. Tangents on differentiable and fractal curves

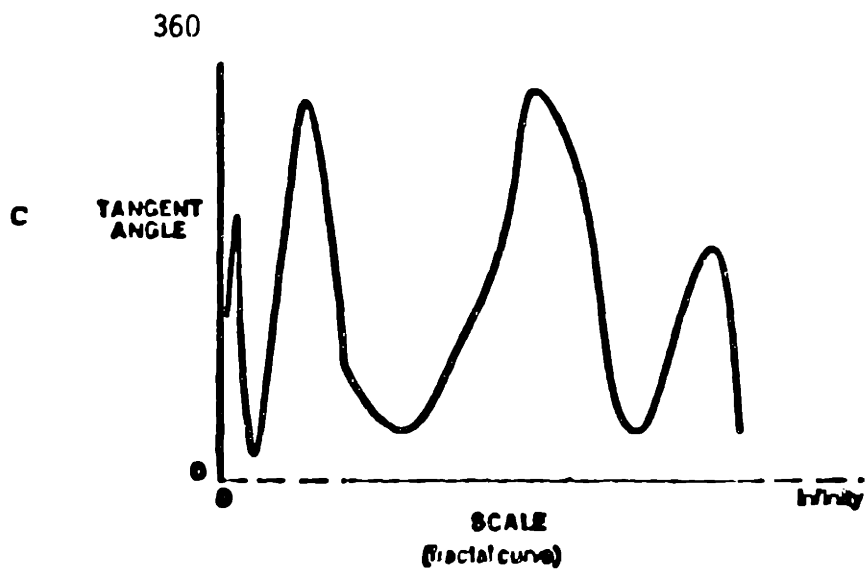
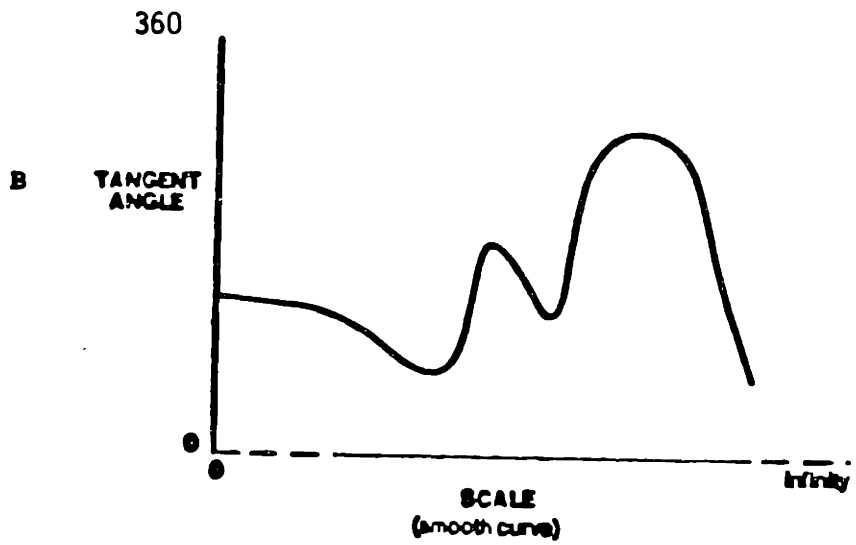
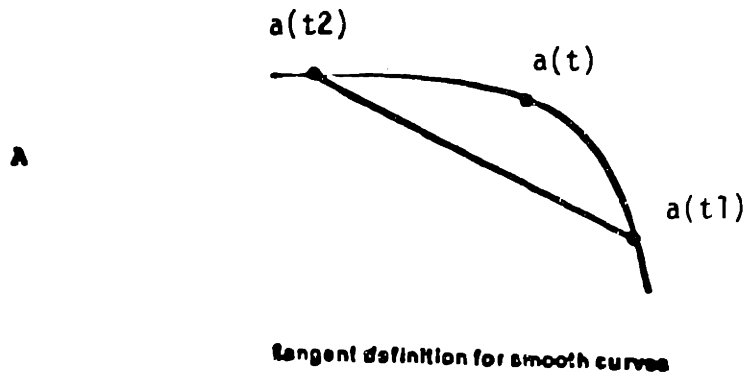
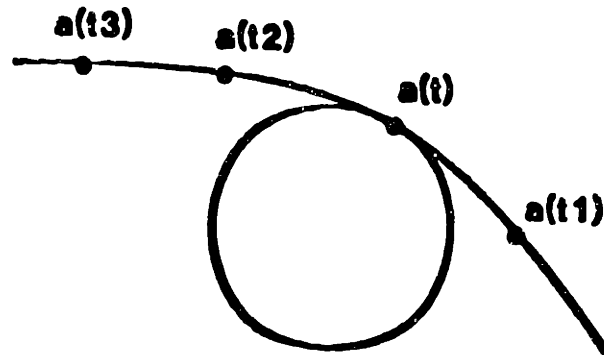


Figure 2. Curvature on differentiable curves



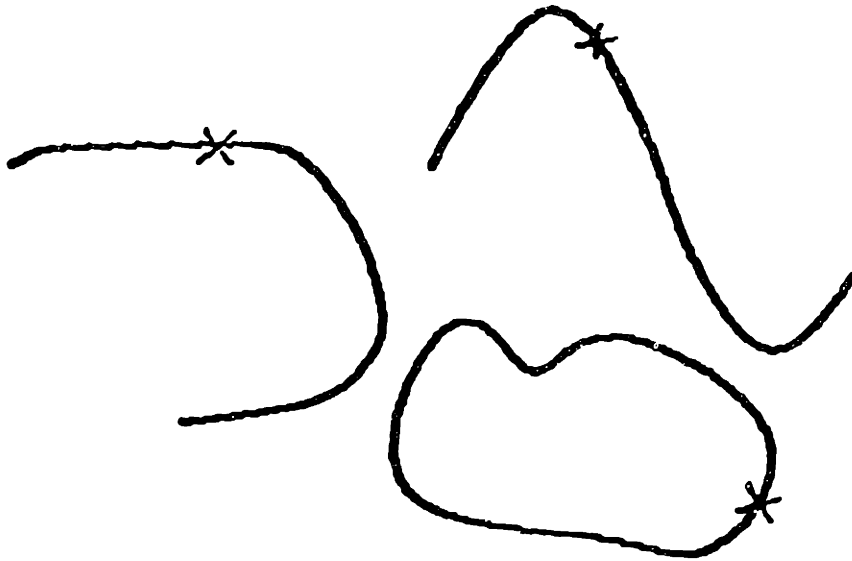
A simple observation about the human perception of fractal curves suggests that there is a way out of this problem (if we can find it). This observation is illustrated in figure 4-3. The figure shows several fractal curves with small x's marked on certain points. The question is, What is the tangent to each curve at the points marked by the x's? We have no difficulty answering. Indeed, the answers across observers are quite consistent. Yet, because the curves are fractals, there are literally an infinite number of "right" answers. Why the concensus among human observers?

Apparently there are "natural" scales to the human observer. Simply by inspecting a curve we all consistently discover the same natural scale(s). Note that the natural scales of a curve seem to be discovered, not imposed, by our visual systems. Were they imposed, the scale that seemed natural would not vary from curve to curve, as it does in the figure. There is a sense in which the natural scale is imposed. We can only see a finite range of scales at any one time, so any natural scales we discover must be within that range. This range, however, is quite large, between four and five orders of magnitude.⁴ In this section a method for discovering natural scale tangents and curvatures at points on (quantized) fractal curves will be proposed.

First we need some assumptions and terminology. Assume that the image plane has a hexagonal tessellation. Assume that the discretized curve is only one pixel thick at every point. Then a convenient analogue for the arc length between two points on a smooth curve

⁴This estimate comes from dividing our field of view, about 180 degrees, by our best spatial resolution, about two seconds of arc.

Figure 3. Some fractal curves

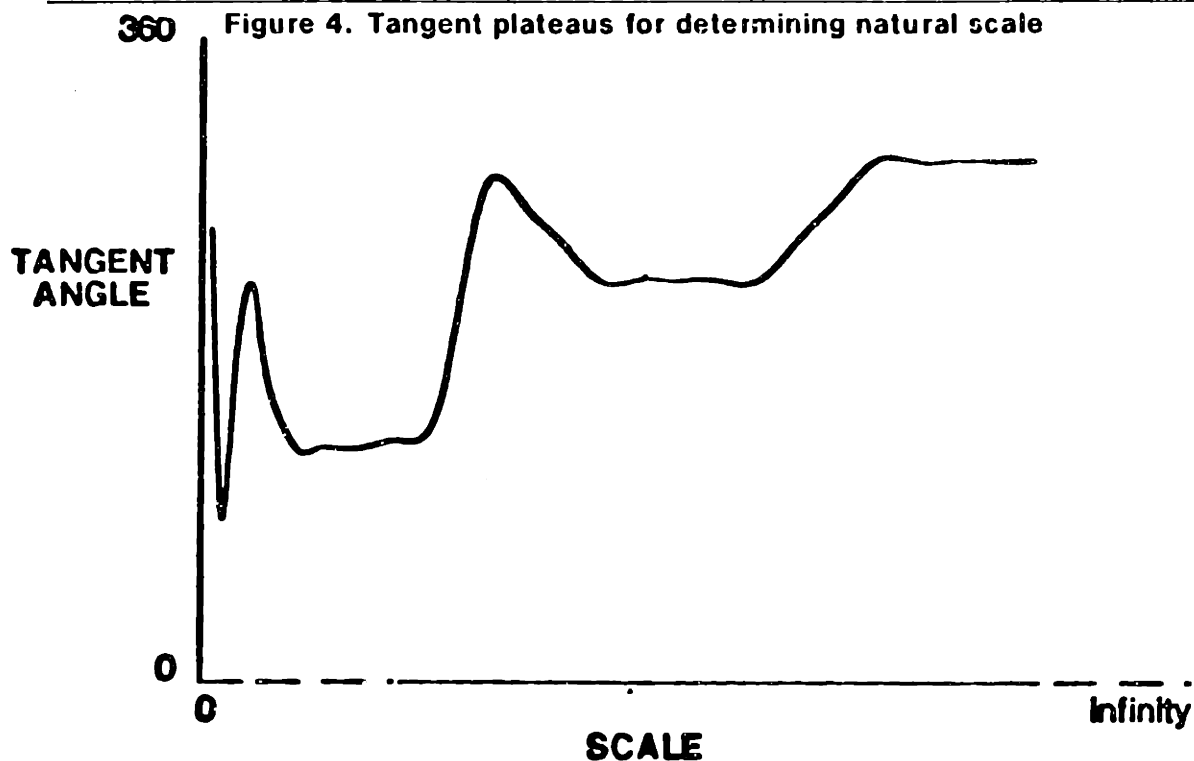


is the number of pixels between two points on a discretized curve. The "pixel arc length" will be denoted by s and curves by c .

Let us plot the angle of the chord between two points $c(s_1)$ and $c(s_2)$ as $s_1, s_2 \rightarrow s$ such that $c(s_1)$ and $c(s_2)$ are always equal pixel arc length away from $c(s)$ and on opposite sides of $c(s)$. Suppose that, instead of the consistent randomness of figure 4-1c, a plot such as figure 4-4 is obtained having "tangent plateaus". These tangent plateaus are regions of the plot where the chord angle does not vary much over a wide range of scales. Tangent plateaus of sufficiently long extent and sufficiently low variance of the chord angle are likely candidates for determining natural scale tangents to fractal curves. If there are no such plateaus, if the plot of the chord angle is truly random over all scales, then the choice of a set of scales as "natural" is surely arbitrary.

Some obvious problems for this definition of natural scale are the following: 1) How many tangents should go into the variance computation at each scale, 2) how low should the variance be to qualify as a natural scale, and 3) how long should the plateau be to qualify as a natural scale.

On the first problem, if an arbitrarily fixed number of tangents, say twenty, is picked as the window for the variance computation then the whole scale issue is begged. A window of two tangents would always be sensitive to the curve quantization even at large scales, certainly an undesirable result. A huge window of fixed size would be too insensitive for the smaller scales. Thus we need a window whose size varies with the scale at which the variance is to be computed. This is the approach taken in the implementation (see figure

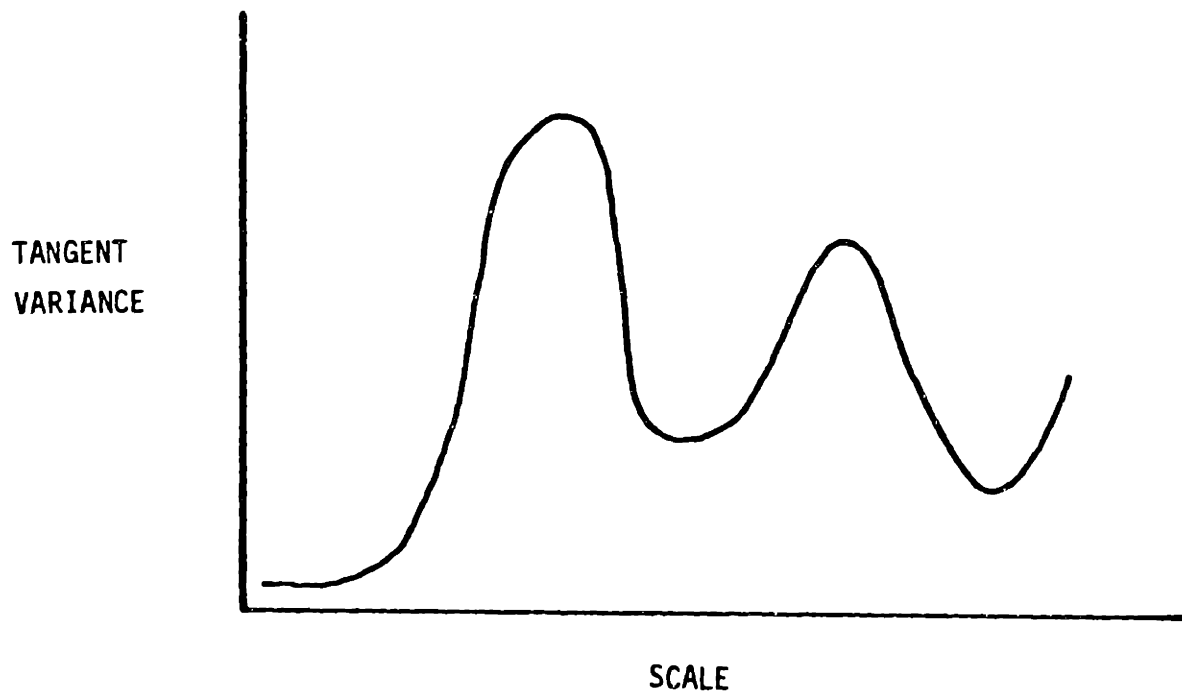
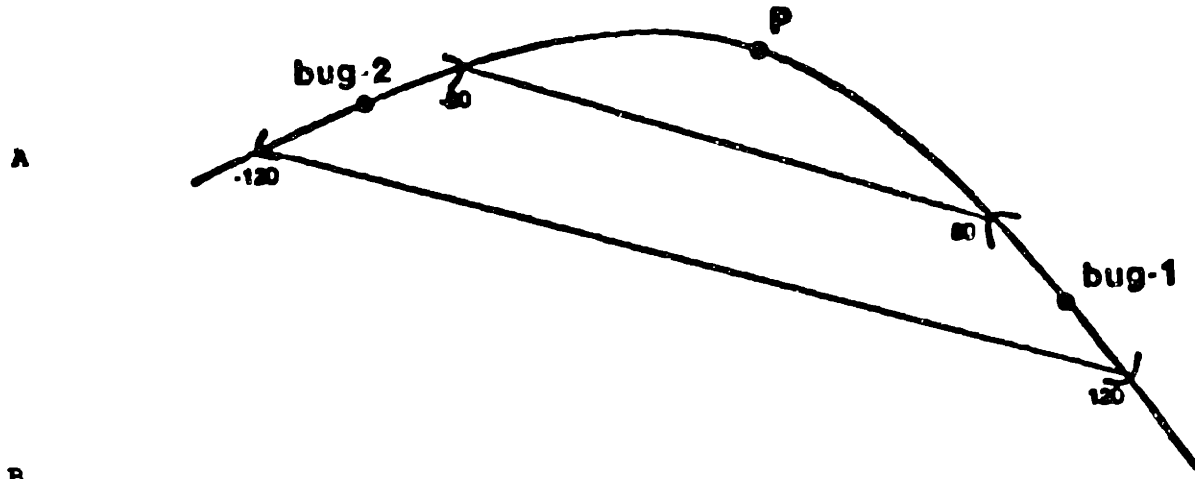


4-5a). To find the natural scale tangent(s) at some point P on a curve we send two "bugs" walking away from P at equal speeds along the curve. After each bug takes a step two new windows are formed, whose sizes are proportional to the distance of the bugs from P , with each bug in the center of its window. Suppose, for example, the proportionality constant is 0.4. Then when the bugs are ten pixels away from P the windows are four pixels wide, two pixels in front of the bugs and two behind. When the bugs are 100 pixels from P the windows are forty pixels wide. If we assign positive values to pixel distances to the right of P and negative values to distances to the left of P , then the window for the bug to the right of P would extend from pixel 80 to pixel 120 while the window for the bug to the left of P would extend from pixel -80 to pixel -120.

The tangents within the windows are defined as the angles of the lines between corresponding pixels in the two windows. Corresponding pixels are pixels of the same absolute value but opposite sign, such as pixels 80 and -80. Once all the tangents are determined within the windows, the variance of the tangents is computed. Each such variance computation is represented by one point in the variance plot of figure 4-5b. The bugs take one step forward and the whole procedure is repeated, leading ultimately to the entire plot of figure 4-5b. Minima of this variance plot are candidate natural scales for the tangent at P . Of course some minima are lower than others, with the consequence that some natural scales may be more natural than others.

Figure 4-6a shows that, as one expects, there can be more than one natural scale tangent at a point on a curve. The curve in the figure is an ellipse with some high frequency,

Figure 5. Variance computation for natural scale tangents



low amplitude, sinusoidal modulation of the y -coordinate. The X's in the figure indicate the points about which the natural scale computation was performed. The straight lines indicate both the angle and scale of the natural scale tangents found about the X. In some cases the tangents at the smaller and larger scales can be almost mutually orthogonal.

Figure 4-6b shows the result when the tangents at the two scales are computed for every point on the ellipse, stored in two separate lists, and integrated (in a manner to be explained shortly) to give two scale dependent analytic representations of the curve. The two resulting curves correspond nicely to our perception.

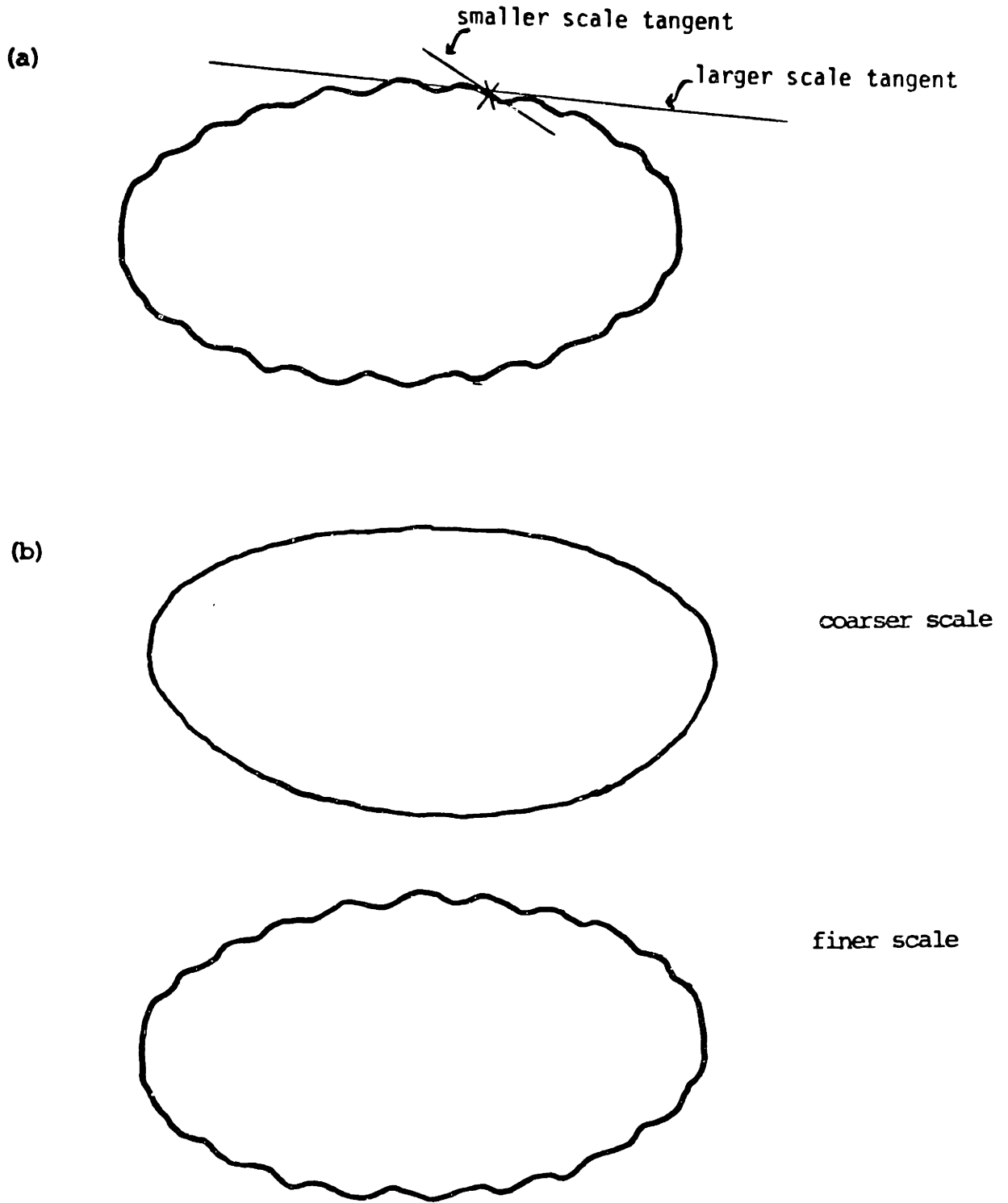
Finding natural scale tangents is simply a means to an end as far as the codon theory is concerned. What the codon approach really needs is the natural scale *curvature* since the codon part boundaries and part descriptions are based upon extrema of curvature. Unfortunately the straightforward approach of finding tangent plateaus that works well for discovering natural scale tangents does not seem to generalize for the case of curvature. An algorithm was designed to find curvature plateaus in a manner analogous to that for tangent plateaus. The resulting variance plots did not show well defined minima. In retrospect this makes sense. One would expect to find a curvature plateau at a scale only if the curve is an arc of a circle at that scale.

An alternative approach is to compute an analytic approximation of the curve based on the natural scale tangents and then to use the analytic representation to compute the codon description. Ideally such an approximation should satisfy several criteria:

- It should be able to approximate every continuous function on some interval $[a,b]$ with arbitrary accuracy. Otherwise there might be classes of curves for which no reasonable codon description could be given simply because of the approximation used.
- It should approximate well the derivatives of a function, in addition to the function itself. For instance, it should not oscillate about the function. Otherwise the codon description, which relies heavily on these derivatives, will be inaccurate.
- It should be at least twice continuously differentiable everywhere on the interval. Our visual systems are not sensitive to higher order discontinuities.
- It should require no iteration for its computation, or very few iterations. Otherwise the computations become too expensive and time consuming.
- It should require only local support for its computation. This means that the approximation at one point should not depend on the entire function being approximated.

A particularly simple class of approximating functions which satisfies these criteria is the class of cubic b-splines (Schumaker 1981). For plane curves, a cubic b-spline is a piece-wise polynomial approximation of the form (Foley & Van Dam 1982)

Figure 6. Two natural scales discovered by the algorithm for one curve



(redrawn from a computer driven color display)

$$\beta(t) = (x(t), y(t))$$

where

$$\begin{aligned} x(t) &= a_x t^3 + b_x t^2 + c_x t + d_x, \\ y(t) &= a_y t^3 + b_y t^2 + c_y t + d_y, \quad 0 \leq t \leq 1 \end{aligned}$$

In matrix notation this is simply

$$\begin{aligned} x(t) &= TC_x, \\ y(t) &= TC_y \end{aligned}$$

where

$$\begin{aligned} T &= [t^3 \ t^2 \ t \ 1] \\ C_x &= [a_x \ b_x \ c_x \ d_x]^T \\ C_y &= [a_y \ b_y \ c_y \ d_y]^T \end{aligned}$$

The restriction on the parameter t does not limit the generality of the b-spline since it is a segment by segment approximation.

To determine the four coefficients in the vectors C_x and C_y four *control points* are needed. Let the control points be $P_i = (x_i, y_i)$, $i = -1, 0, 1, 2$. Then the coefficients of the b-spline from near P_0 to near P_1 are given by

$$\begin{aligned} C_x &= MG_x \\ C_y &= MG_y \end{aligned}$$

where

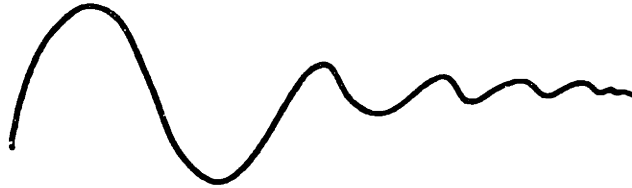
$$M = \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$$

and

$$\begin{aligned} G_x &= [x_{-1} \ x_0 \ x_1 \ x_2]^T \\ G_y &= [y_{-1} \ y_0 \ y_1 \ y_2]^T \end{aligned}$$

M is called the spline matrix, G_x and G_y the geometry vectors.

Figure 7. A curve whose natural scale changes gradually



From this formulation of cubic b-splines it is clear that their computation is non-iterative and requires only local control, i.e., the four points in the geometry vector. A different geometry vector is used for each spline segment.

If we are to use b-splines as our approximating functions we must first answer an important question: How should the control points be chosen? If we want the b-spline to be a natural scale approximation then our choice of control points must somehow be faithful to the tangents we have discovered at a natural scale. And how can we talk about the tangents (plural) at one natural scale along the curve? Consider the curve of figure 4-7. The natural scales at each end of the curve appear to be quite different.

To talk about the tangents at one natural scale along a curve we must turn our attention from the image curve back to the world. Shapes in the world exhibit various natural scales because they are often composed of various processes. The outline of a tree, for instance, incorporates at least two major processes, the branches and the leaves. We should expect, then, that the image of the tree's outline would display corresponding natural scales. These natural scales in the image should be approximately as spatially uniform as their corresponding processes in the world. Consequently one should expect to find natural scale tangents whose scale does not vary much over that part of the curve under the influence of one process. Since more than one process may influence a given segment of curve one might find more than one set of natural scale tangents, each set varying little in scale.

In practice one finds that the plot of the tangent variance as a function of scale (as in figure 4-5b) does not change radically from one point to the next along a curve. It is thus quite easy to put the minima from one variance plot in correspondence with the minima

from the plots of neighboring points. In this manner it is possible to collect the natural scale tangents of each natural scale into the correct global lists.

How should we choose the control points for one natural scale? It appears that we must somehow relate our choice of control points to the list of natural scale tangents at that scale. Otherwise why would the b-spline approximation warrant the label "natural"?

Unfortunately I have no tight theory of how best to choose the control points given the natural scale tangents along a curve. What follows is a description of how I in fact did so, its only merit being, other than a few justifications here and there, that it works well. Here it is.

The natural tangents are computed at each point along a curve and stored in lists, one list for each natural scale. Let us restrict our attention to one such list. Every time the natural tangents along the curve rotate by twenty degrees, a control point is placed on the curve. When this has been done for the entire curve, the control points are then more evenly spaced along the curve using a local averaging and a cubic b-spline fit through them. Some examples of the resulting curves are shown in figure 4-6b.

Now some justifications. First, why choose the number of control points so that on average the tangent rotates twenty degrees between them? Why not choose five degrees or ninety degrees? This spacing problem is important. If the control points are too closely spaced then the resulting b-spline will be influenced too much by scales smaller than the scale of interest. If they are spaced too far apart then the spline will be influenced by larger scales, possibly to the exclusion of the scale of interest.

A rough upper bound on the spacing can be set by asking what is the minimum number of control points needed to approximate a circle such that the approximation is not easily distinguished visually from a true circle.⁵ Four is certainly insufficient. The resulting spline looks more like a rounded square than a circle. The minimum for a pleasing circle is about eight control points, or one every forty five degrees. To be conservative, let us choose ten control points as the minimum and thus about thirty five degrees as the upper bound on the spacing between control points.

The lower bound on the spacing between control points must be sufficiently large that the b-spline washes out the smaller scale effects. If forty five degrees is the absolute maximum spacing to capture a scale, then two or three times that spacing, say 110 degrees, should be enough to wash out a lot of a scale. Consequently, if our lower bound on control point spacing for the desired scale is such that on average it works out to a spacing of 110 degrees or more for the smaller scale then we should be able to eliminate most smaller scale effects. If, on average, the smaller scale tangents rotate at least between five and

⁵A circle is chosen because it is the simplest closed plane curve.

ten times faster than the tangents of the scale of interest, this would imply a lower bound on control point spacing of between twenty two and eleven degrees respectively. Thus our optimal spacing for capturing one scale while ignoring both larger and smaller scales is likely to be somewhere between eleven degrees and thirty five degrees.

After placing control points every twenty degrees along the curve why space the points more evenly? First, this puts more points on the segments of curve which are straight, leading to a better approximation. This also spaces out the control points in high curvature segments of the curve, reducing the magnitude of the discontinuity of the third derivative at the joins between spline segments and thereby reducing the probability of finding false extrema of curvature at these points.⁶ Why use a local averaging window, rather than a global one, to space the control points? A local window doesn't require an integration of information over large regions of space. In addition, a local window allows a reasonable spacing of points for curves such as the one shown in figure 4-7, whereas a global window would not.

We are now in a better position to appreciate the significance of figure 4-6. This figure illustrates that a given curve can have more than one natural scale, and that the approach outlined here can discover them. Again the key is that the natural scales must be discovered, not imposed. However, the particular algorithm presented in this chapter is not likely to be the one our visual systems use. Our visual systems seem to rely on banks of filters of various sizes for early processing ($\nabla^2 G$ filters according to Marr and Hildreth 1980, and Marr and Ullman 1980). A biologically plausible algorithm for determining natural scales would probably need to be able to discover the natural scale tangents by looking at the outputs of these filters of various sizes.

After obtaining the b-spline approximation at a natural scale we find extrema of curvature which are maxima of the absolute value of curvature. This implies that all extrema of curvature are found except positive minima and negative maxima. Positive minima and negative maxima are difficult to localize directly because they are points having a relatively low signal to noise ratio. However their presence, and consequently the complete codon top level description, can be inferred entirely from the pattern of the signs of the other extrema of curvature. This is done using the finite state machine illustrated in figure 4-8. The circles with Q's in them are states of the machine. The starting state is the circle labelled Q0. Each state has two arcs leaving it, one labelled + and one labelled -. These arcs indicate which state to go into next depending upon whether the sign of the next extrema of curvature is positive or negative. Some arcs have a codon label attached to them. This indicates that as an arc is traversed its codon label should be added to the growing codon description.

⁶S. Ullman has pointed out that this might be done more effectively using quartic b-splines rather than cubics.

Figure 8. Finite state machine which computes codon top level description

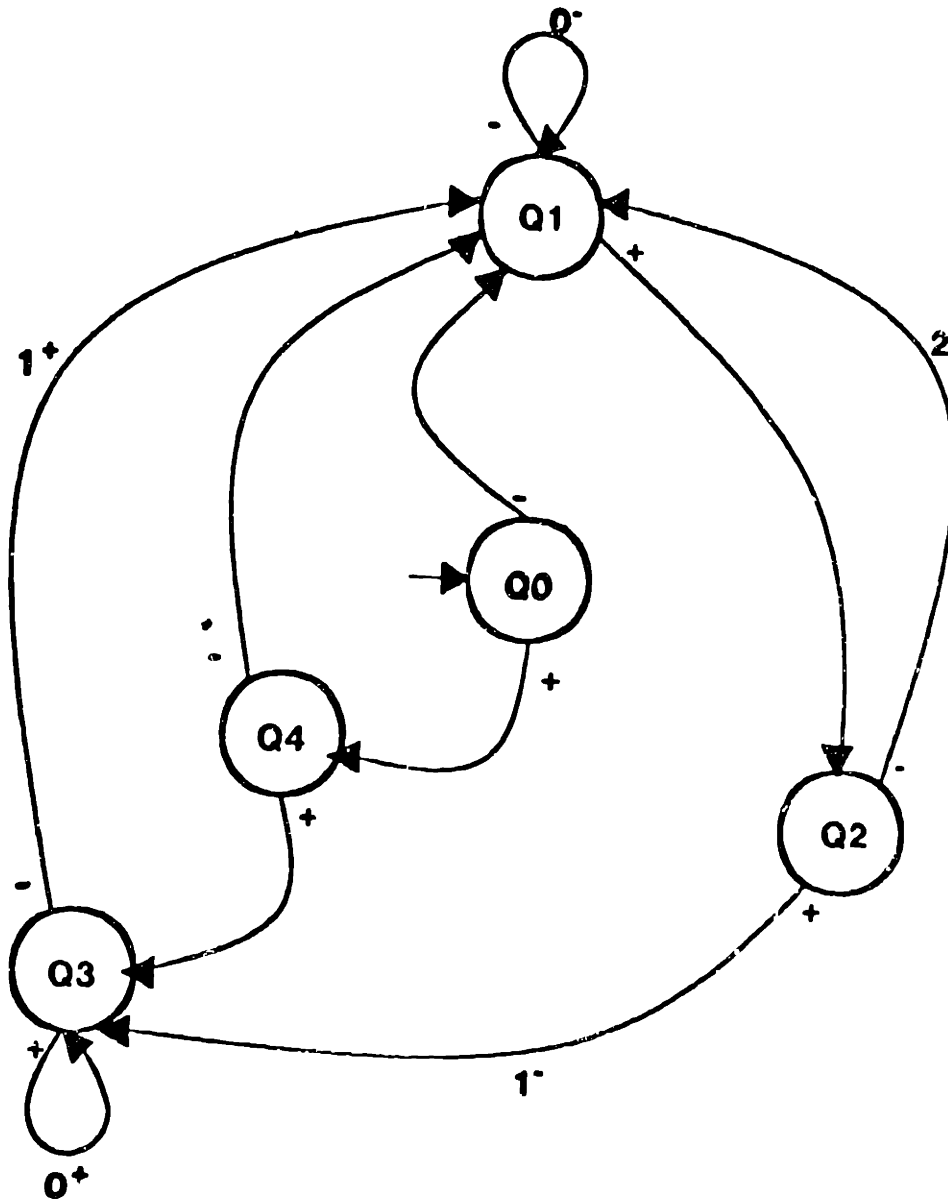


Figure 4-9 shows the extrema and codon descriptions computed in this manner for one curve. Since the curve has two visible natural scales the algorithm shows two separate sets of extrema and codon descriptions. This figure illustrates that the codon description one obtains for a curve depends upon the natural scale at which one inspects the curve. An ability to decouple natural scales in this manner is a prerequisite to satisfying the stability/sensitivity criterion mentioned in the first chapter.

4.3. Natural scale and effective dimension

In his book *The fractal geometry of nature* Mandelbrot (1982) discusses several ways of defining the dimension of a curve or surface. Some of these are novel in that they allow non-integer values for the dimension. Another, which he calls the *effective dimension*, is unusual in that he claims it should not be defined precisely, but rather left as an intuitive notion. Since his concept of effective dimension has some interesting connections with our concept of natural scale we will here note this relationship.

Mandelbrot explains what he means by effective dimension with an example of a ball of thread:

"... a ball of 10 cm diameter made of a thick thread of 1 mm diameter possesses (in latent fashion) several distinct effective dimensions.

To an observer placed far away, the ball appears as a zero-dimensional figure: a point. (Anyhow, it is asserted by Blaise Pascal and by medieval philosophers that on a cosmic scale our whole world is but a point!) As seen from a distance of 10 cm resolution, the ball of thread is a three-dimensional figure. At 10 mm, it is a mess of one-dimensional threads. At 0.1 mm, each thread becomes a column and the whole becomes a three-dimensional figure again. At 0.01 mm, each column dissolves into fibers, and the ball again becomes one-dimensional, and so on, with the dimension crossing over repeatedly from one value to another. When the ball is represented by a finite number of atomlike pinpoints, it becomes zero-dimensional again. An analogous sequence of dimensions and crossovers is encountered in a sheet of paper.

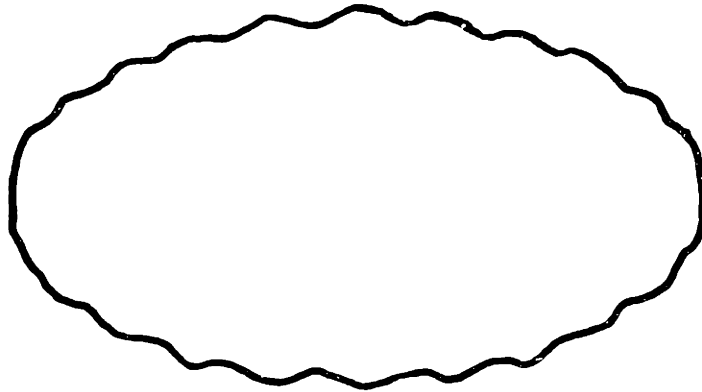
The notion that a numerical result should depend on the relation of object to observer is in the spirit of physics in this century and is even an exemplary illustration of it."

The effective dimension of a shape depends on the relation of object and observer. In particular it depends upon the rules used by the visual system of the observer to organize, structure, and represent his images of the object.⁷ These are the rules we have tried to motivate and to state explicitly in our discussion of natural scale. The theory of natural

⁷Of course, these rules are worthwhile only to the extent that they capture useful aspects of the structure of the observer's visual world. The structure imposed by a visual system is useful only if it bears a non-arbitrary relationship to the structure of the world. Vision is a process of veridical hallucination.

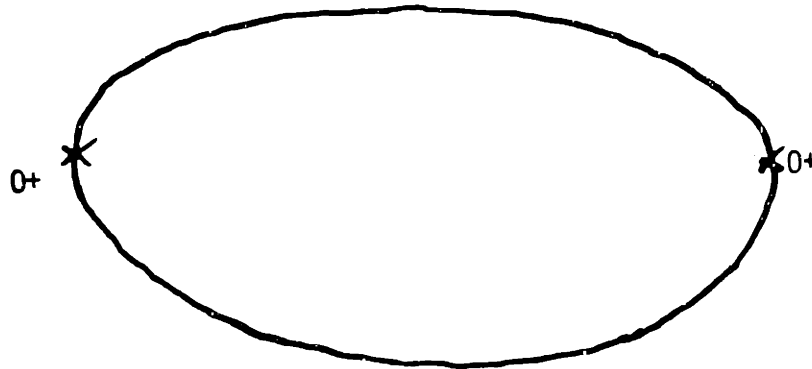
Figure 9. Extrema of curvature, and resulting codon description

(a)

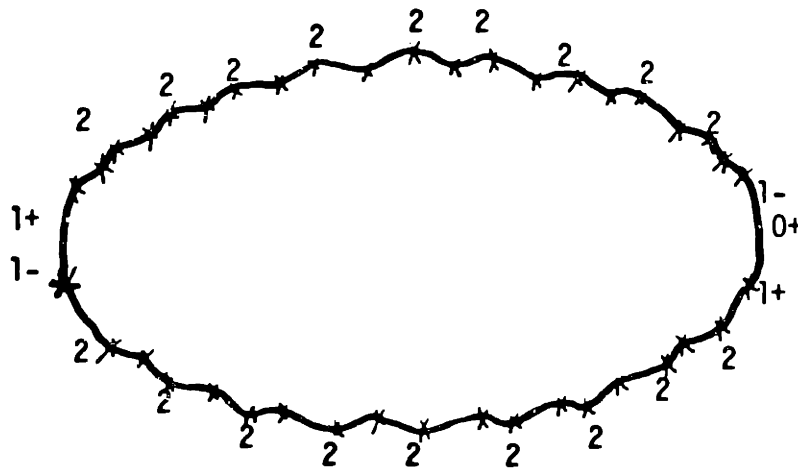


original curve

(b)



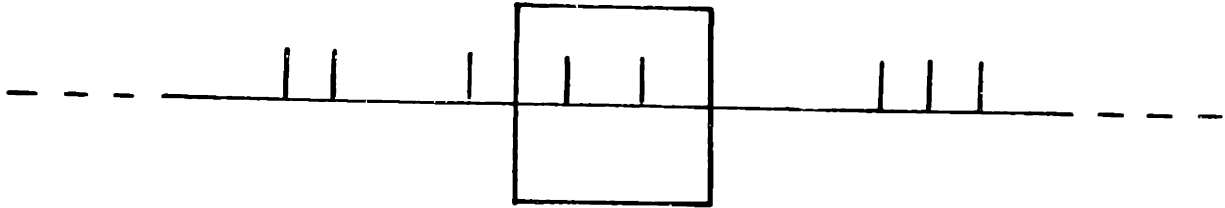
coarser scale



finer scale

(redrawn from a computer driven color display)

Figure 10. Natural scales visible at one time



scales can explain why the effective dimension of an object changes with scale, and the algorithm presented for computing natural scales can explain how this change of dimension is accomplished.

Mandelbrot's discussion of effective dimension is potentially misleading on one point. His example of the ball of string might seem to imply that at most one effective dimension can be seen at any one time. However, as illustrated earlier by our wiggly ellipse, an object may reveal several different natural scales simultaneously. A helpful way to think about this is shown in figure 4-10. The long straight line in the figure represents the continuum of scales at which some object can be considered. Movement to the left along the line is movement to smaller scales, movement to the right is to larger scales. The spikes along the curve indicate natural scales of the object as defined by the tangent plateau rule. The box indicates the window of scales available to the visual system of the observer at any one time, about four or five orders of magnitude for human observers as noted earlier. By looking at an object through a microscope or simply getting closer the observer can slide the box along the line to its left. By stepping back the observer can shift the box to the right. The number of spikes in the box varies as the box shifts left or right along the line and needn't be just one, indicating that the number of natural scales visible to the observer at one time is variable and needn't be just one.

4.4. Summary

The intent of this chapter and the lisp code in the appendix is to show that it is possible to compute a codon representation for real imaged curves. To do this, however, requires redefining several of the properties of the local geometry of plane curves – particularly the tangent at a point. Our redefinition involves the notion of “natural scale”, the notion that our visual systems discover useful scales at which to describe the geometry of curves (and surfaces). The concept of natural scale is still only vaguely understood. It must eventually be made more precise and related to the real world processes which govern the shapes of objects in the visual world.

5. Relation to human perception

5.1. Predictions

The theory of shape representation presented in the previous chapters, although by no means complete, makes several clear predictions about human perception and memory for shapes. The clearest predictions come from the rule for dividing a plane curve into parts and from the rule for dividing a surface into parts. For ease of reference we restate these two rules.

Segmentation rules:

1. Plane curves: Divide a plane curve into parts at minima of signed curvature and concave cusps.
2. Surfaces: Divide a surface into parts at minima, along lines of greatest curvature, of the greatest principal curvature. Or else divide a surface into parts at minima, along lines of least curvature, of the least principal curvature.

Recall that when figure and ground are reversed for a curve, the sign of the curvature everywhere along the curve also flips. In particular minima of curvature, which define part boundaries, become maxima of curvature and maxima become minima. Similarly, recall that when figure and ground are reversed for a surface the signs of the normal curvatures at each point also reverse. This follows from a convention linking the choice of figure and ground with the choice of the field of surface normals, and from the following equation for normal curvature:

$$\kappa_n = \kappa \cdot \mathbf{N}$$

where κ_n is the normal curvature, κ is the curvature vector of an arbitrary curve on the surface at the point of interest, and \mathbf{N} is the surface normal at the point. When figure and ground reverse, the surface normal \mathbf{N} changes direction by 180 degrees and the resulting scalar product for the normal curvature must change sign. Of interest here is the consequence that minima of the greatest principal curvature along lines of greatest curvature must become maxima of the least principal curvature along lines of least curvature when figure and ground reverse. In other words, the partitioning rule for surfaces predicts that the same surface should be cut into parts differently when figure and ground reverse.

For convenient reference I state these and other predictions that follow from the partitioning rules.

Predictions:

1. When a plane curve undergoes a figure-ground reversal, the visual system should segment it into parts differently (since the positions of the minima of curvature change).
2. When a surface undergoes a figure-ground reversal, the visual system should segment it into parts differently (since the positions of the minima of normal curvature along lines of curvature change).
3. Recognition memory for parts of plane curves cut at minima of curvature should be better than any other cuts, better, in particular, than cuts at maxima of curvature.
4. Parts of plane curves cut at minima of curvature should appear more natural to observers than cuts at maxima, inflections or some combination of these.
5. Parts defined by minima of curvature should be the parts to which we most naturally assign verbal labels.
6. The mirror image of a plane curve should be judged more similar to the original curve than the figure-ground reversal of the original curve (since the division into parts should be the same for the mirror reversal but not for the figure-ground reversal).

In this chapter I compare these predictions with human performance and find an interesting correspondence. First I present three experiments designed to test predictions 3, 4, and 6. Then I examine nine perceptual demonstrations, some mentioned earlier in chapters 2 and 3. Finally, in an epilogue I discuss in outline how a theory of the visual interpretation of faces might benefit from the rules and representations proposed in this thesis.

5.2. Experiments


In this section I examine three experiments designed to test some of the six previously stated predictions. The first experiment tests the prediction that parts defined by minima should appear more natural to human observers than parts defined by maxima, inflections or some combination of these (prediction 4). The second experiment tests the prediction that the mirror image should be more similar to the original curve than the figure-ground reversal (prediction 6). The last experiment tests whether recognition memory for parts of plane curves cut at minima is better than for other cuts (prediction 3).

5.2.1. Experiment 1: Naturalness

Subjects in this experiment were read the following instructions:

"I will give you a pile of 8 cards. On the front of each card is a black object on a white background. On the back of each card are several candidate 'parts' of the black object.

Choose the one part which is the most natural part of the black object. Then write down the number that appears on the front of the card and the letter that appears next to the part you have chosen. You may flip the card back and forth as often as you wish. There are no time constraints.

“What is a natural part? Consider the sentence ‘The boy hit the ball’. A natural way to break this sentence would be to group the words ‘the boy’ and ‘the ball’. An unnatural grouping would be the phrase ‘hit the’. Consider also this figure:  A natural grouping might be into a circle and an x. An unnatural grouping might be:

“Once again, be sure to choose what you feel are natural parts of the black object – not of the white background. And be sure to judge each card independently – do not let the black objects on other cards you have already seen affect your judgement of the natural parts of the black object you see next.”

The front and back of an example card are shown in figure 5-1. Each card was four by six inches. The parts of the curve shown on the backs of the cards were cut either at minima of curvature, maxima of curvature, minima and maxima, minima and inflections, or maxima and inflections. On no card were more than one quarter of the parts cut at minima. Nine subjects were run, all students and staff of the MIT psychology department. On average subjects choose parts cut at minima of curvature 95.8% of the time, which is significantly above chance, $t(8) = 8.01$, $p < .001$.

Some of the cards were simply figure-ground reversals of each other. Subjects consistently choose parts cut at minima of curvature, even though this meant cutting the same curve differently when figure and ground were reversed. Thus it appears that parts defined by minima of curvature seem more natural to observers than parts defined by maxima of curvature or any of the other options mentioned above.

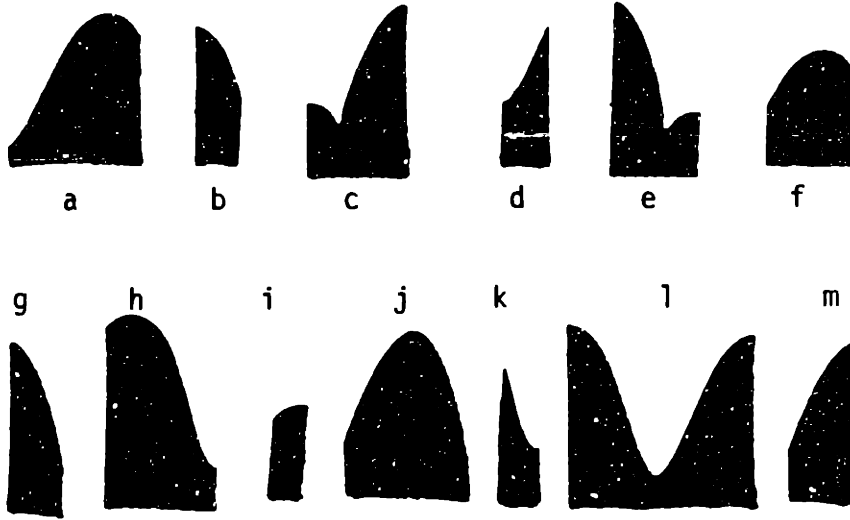
5.2.2. Experiment 2: Mirror versus figure-ground reversal

Subjects were read the following instructions:

“This is a curve similarity test. I will show you one wiggly curve for four seconds. Then I will cover it up. Finally I will show you two wiggly curves. Choose the one wiggly curve that you feel is most similar to the first wiggly curve.”

In all, twenty trials were presented to each of ten subjects. An example triple of shapes is shown in figure 5-2. The cards were four by six inches and were presented about two feet from the subject using a cardboard apparatus which allowed the experimenter to manually cover and uncover the cards. On the trials of interest the two shapes shown together were both simple transforms of the original shape. One of the two was a mirror reversal of the

Figure 1. A shape and various parts



Parts a, h, and j are all cut at minima of curvature.
Subjects chose part j most often as the most natural.

original. The other of the two was a figure-ground reversal of the original. These trials comprised half (ten) of the trials. The other ten were catch trials in which the second two shapes were not related to the first by either a mirror reversal or a figure-ground reversal.

By showing the curves to subjects as displayed in figure 5-2 it is clear that the bounding contour of the figure-ground reversed curve is identical to the original, just translated a little and rotated less than about 30 degrees. The mirror reversed curve, however, is not only translated and rotated slightly, it is also flipped in a manner that cannot be undone by any rotation in the plane. Thus if the similarity judgements are related to the number and complexity of the transformations needed to bring each curve into congruence with the original, one would predict that the figure-ground reversed curve, requiring the fewest transformations, should be most similar to the original. On the other hand, the codon theory espoused here predicts that the mirror reversals should appear more similar to the originals than do the figure-ground reversals. This is predicted because the segmentation rule carves the original and the mirror reversal into the same set of parts, with only the order of the parts reversed. However the segmentation rule carves the original and figure-ground reversal into entirely different parts. Assuming that parts figure prominently in judgements of similarity and identity of objects, it follows that the mirror reversal, having the same parts as the original, should be more similar to the original than the figure-ground reversal, which does not have the same parts.

The codon prediction was confirmed by the subjects, who picked the mirror reversal as most similar to the original 94% of the time on average, which is significantly greater than chance, $t(9) = 3.48, p < .005$. In figure 5-2, for instance, the bottom left half moon is mirror reversed from the half moon on the right, and appears more similar to it than does the half moon in the upper left which is figure ground reversed.

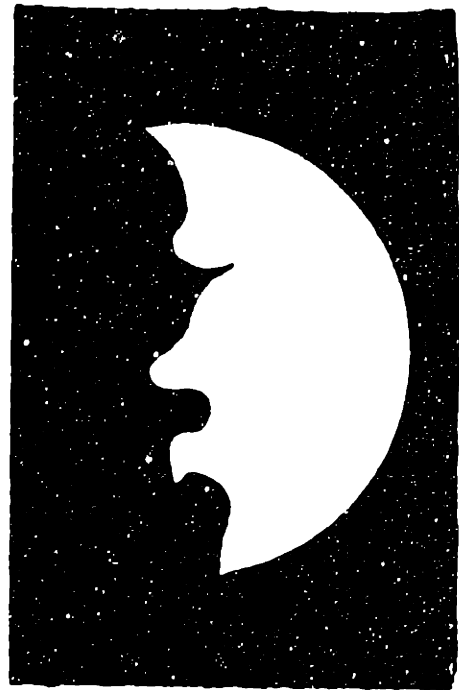
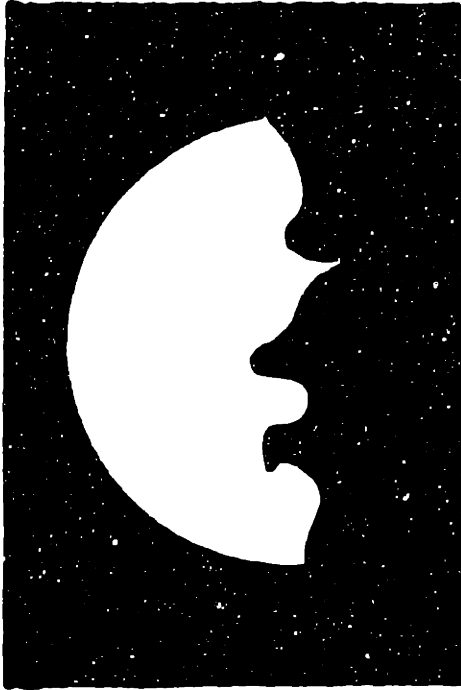
Ten other subjects were run on this experiment with the stimulus figures placed horizontally, with the semi-circular sections of the curves below. Subjects picked the mirror reversal 91% of the time on average, again significantly greater than chance.

5.2.3. Experiment 3: Recognition memory for parts

This experiment failed, but in an interesting way. Here is how it was intended to proceed. Subjects were shown a small white semi-circular region with a wiggly bounding contour joining the ends of the semi-circle (similar to the shapes in figure 5-2). The white region was presented foveally (within two degrees from the point of fixation) against a black background in a tachistoscope for 150 milliseconds. Then a pattern was flashed for 50 milliseconds to erase any icon. Then a small dark wiggly curve against a white background was presented. Subjects were to indicate as quickly as possible, by pressing one of two

Figure 2. Example stimuli for experiment 2

figure-ground reversed



reference



mirror reversed

keys, whether or not the small wiggly curve was a part of the bounding contour of the original white region.

There were twenty trials of interest. In fact only ten different bounding contours were shown in the twenty trials. The twenty trials were constructed from ten bounding contours by using each contour twice but with opposite sides of the contour being the white figure. Thus subjects saw the same bounding contour twice during the course of the twenty trials, but the second presentation was figure-ground reversed from the first. For both presentations of the same bounding contour the small wiggly curve used as a probe was identical, and was, in fact, a part of the original bounding contour. However the wiggly curve was chosen so that for one presentation of the bounding contour the wiggly curve went from one minima of curvature to the successive minima. Necessarily this means that for the presentation of the figure-ground reversal of the same bounding contour this wiggly probe went from a maxima of curvature to the successive maxima.

The codon theory makes a clear prediction here. Subjects should correctly identify the wiggly probe as a part of the bounding contour more often when the probe runs between minima than when it runs between maxima. This is because the natural parts stored in memory, according to the theory, are parts defined by minima of curvature.

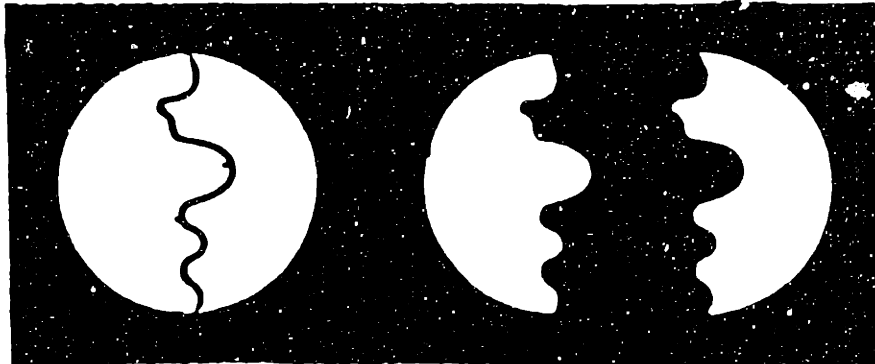
The experiment failed because subjects did not always see the small white region as figure against a dark background. Often subjects saw different figure-ground relations on different portions of the same bounding curve, implying that in these brief flashes the visual system does not seem to require global consistency in the assignment of figure and ground. Figure and ground seemed to be determined locally by which assignment would give the "best" local part. Color and texture cues did not seem to help.

One subject reported consistently seeing the black surrounding region as figure. Subsequent analysis of his data showed that he never made errors for probes which were between minima for the black figure, but made many errors for probes which were between maxima. It is likely, then, that the experiment will work if a method can be devised for forcing the figure-ground choice in brief exposures of contours. Perspective cues, motion cues, or perhaps even stereo cues are possible candidates.

5.3. Demonstrations

Sometimes the construction of explanatory theories in a field can rely in large part on the careful examination of what everyone already knows. Linguistic theories are a prime example. By careful analysis of grammaticality judgements linguists can construct theories of extraordinary depth and explanatory power. Grammaticality judgements are empirical

Figure 3. The reversing circle



data to the linguist, data with the same status as data obtained from elaborate experiments, but with one big advantage – they are obtained with much greater ease.

Visual demonstrations can serve the same purpose for theories of vision as grammaticality judgements do for theories of language. Visual demonstrations are empirical data, data with the same status as data obtained from more elaborate visual experiments. Careful analysis of visual demonstrations may provide empirical confirmation for aspects of an existing theory and suggest needed modifications to the theory. It is with this motivation that I examine the nine demonstrations of this section.

5.3.1. Reversing circle

Figure 5-3 is a demonstration first described by Fred Attneave (1971). Regarding this figure Attneave says:

... you can make a perfectly good reversing figure by scribbling a meaningless line down the middle of a circle. The line will be seen as a contour or a boundary, and its appearance is quite different depending on which side of the contour is seen as the inside and which as the outside. The difference is so fundamental that if a person first sees one side of the contour as the object or figure, the probability of his recognizing the same contour when it is shown as part of the other half of the field is little better than if he had never seen it at all ... The point of basic interest in figure-ground reversal is that one line can have two shapes ... The perceptual representation of a contour is specific to which side is regarded as the figure and which as the ground. Shape may be invariant over a black-white reversal, but it is not invariant over an inside-outside reversal.

Attneave's point is quite important. The same curve looks so different when figure-ground reversed because the visual system builds an entirely different representation for

the curve. In consequence, for any theory of shape representation to be psychologically plausible it is necessary that the theory (1) predict two widely divergent descriptions of the same curve, one for each choice of figure and ground, and (2) make this prediction in a principled manner. The codon theory satisfies this condition by (1) defining the fundamental parts of a shape by a rule which necessarily gives different parts when figure and ground reverse, and (2) motivating the partitioning rule by a powerful natural constraint. The partitioning rule is stated above. The motivating constraint is the fact that when two arbitrarily shaped parts are joined to create a composite shape, concave cusps will always (with probability one) be created in the imaged silhouette (figure 2-2). By smoothing the silhouette we have that the parts always meet each other at minima of curvature, which is precisely the partitioning rule.

Not all conceivable schemes have this property of yielding different parts when figure and ground reverse. If one defines parts by inflections of curvature (Marr 1977, Hollerbach 1975) or by maxima and minima (Brady 1982a) then the part boundaries will not move when figure and ground reverse.

This reversing circle illustrates one further important point made by Attneave (1971). Our visual assignment of differing descriptions to the same curve does not depend on any previous familiarity with the curve. This implies the existence of a descriptive system whose rules pertain only to the geometry of the curve, not to top down knowledge of the identity of the object. The codon scheme, with its minima rule for defining parts, is consistent with this observation.

5.3.2. The face-goblet demonstration

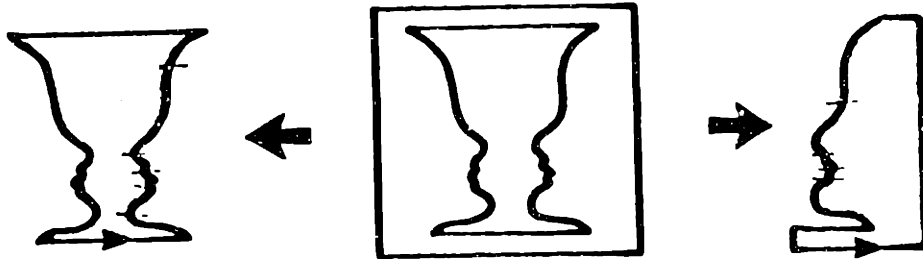
One of Edgar Rubin's most striking examples of visual reversal is the face-goblet shown in figure 5-4. Of this figure Rubin says:

The reader has the opportunity not only to convince himself that the ground is perceived as shapeless but also to see that a meaning read into a field when it is figure is not read in when the field is seen as ground.

In other words, at any moment one can either see the goblet or one can see the faces, but not both simultaneously. Apparently the visual system is exploiting another powerful regularity about the visual world: *At most one side of a visible bounding surface can be filled with a solid at any moment.*

The face-goblet demonstration shows clearly that the minima rule cuts the same curve into different parts when figure and ground reverse (part boundaries are indicated by dashes across the curves). For the face-like curve on the left the rule divides the curve into a forehead, nose, upper lip, lower lip, and chin. For the goblet-like curve the rule divides

Figure 4. The face-goblet demonstration



the curve into a base, two parts of a stem, and a bowl. As one can verify by looking, the part boundaries have actually moved around, in agreement with prediction 1. It is interesting that the parts defined by the minima rule are easily assigned verbal labels in this demonstration, such as nose, chin, base, and bowl (prediction 5). This observation should be pursued further to see if indeed there is a parallel here to Rosch's (1977) observation that "basic colour terminology appears to be universal and that perceptually salient focal colours appear to form natural prototypes for the development of colour terms. Contrary to initial ideas, the colour space appears to be a prime example of the influence of underlying perceptual-cognitive factors on linguistic categories."

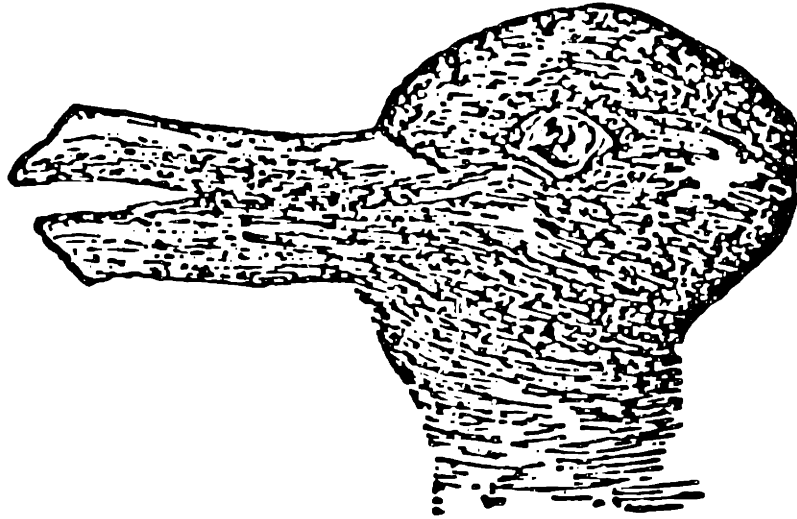
5.3.3. The rabbit-duck figure

Figure 5-5 shows the rabbit-duck figure first used in 1900 by psychologist Joseph Jastrow. Of this figure Attneave (1971) says:

Some of the most striking and amusing ambiguous figures are pictures (which may or may not involve figure-ground reversal) that can be seen as either of two familiar objects, for example a duck or a rabbit ... What is meant by "familiar" in this context is that the visual inputs can be matched to some acquired or learned schemata of classes of objects. Just what such class schemata consist of - whether they are like composite photographs or like lists of properties - remains a matter of controversy. In any case the process of identification must involve some kind of matching between the visual input and a stored schema. If two schemata match the visual input about equally well, they compete for its perceptual interpretation; sometimes one of the objects is seen and sometimes the other. Therefore one reason ambiguity exists is that a single input can be matched to different schemata.

The important observation for our purposes is that the two interpretations of the rabbit-duck figure do not involve a figure-ground reversal. According to the minima rule, then,

Figure 5. The rabbit-duck figure



the two interpretations must arise not because the figure is cut up in different ways but because the same parts are assigned different roles. That is, different parts obtain for the same curve by the minima rule only if the curve is subjected to a figure-ground reversal. The rabbit-duck figure does not involve a figure-ground reversal, just a reinterpretation of the same figure. Consequently both interpretations must have parts which start and stop at the same points according to the minima rule.

Inspection of the figure reveals that this is indeed the case. For example, the same section of the bounding contour which is an ear under the rabbit interpretation becomes the upper half of the bill in the duck interpretation. The part boundaries have not physically shifted on the contour, as they do in the face-goblet demonstration. Instead the same physical parts simply get relabelled. Another example of this is the hawk-goose demonstration shown in figure 5-6.

5.3.4. Escher's symmetry drawings

In figure 5-7 is one of Escher's many captivating tessellations of the plane. Here a switch of attention from the white birds and fish to the black birds and fish necessarily involves a figure-ground reversal. Consequently the minima of curvature rule predicts that a segment of curve which is a single part for one assignment of figure and ground should be split among parts in the other assignment. To verify that this occurs, consider the black fish in the center. The minima rule carves out its tail fin as a single part. When the white

Figure 6. The hawk-goose demonstration

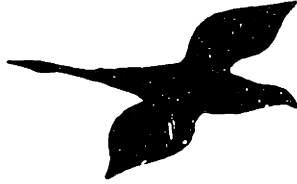
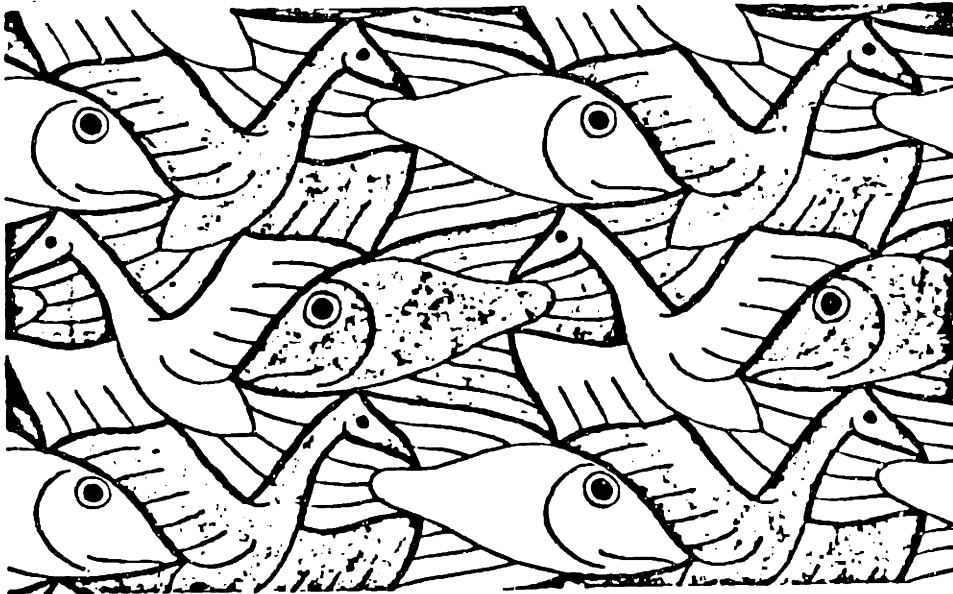


Figure 7. A tessellation of the plane by Escher

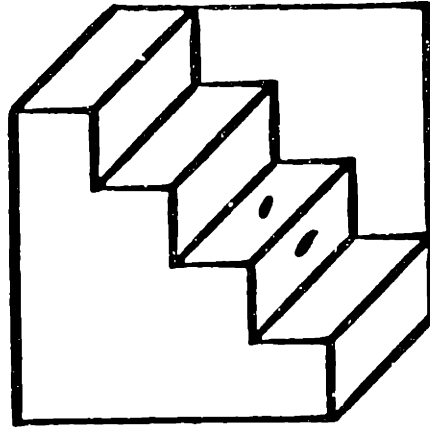


bird behind it is seen as figure the same contour participates in three separate parts: the beak, neck, and wing. Once again it is interesting to note that a piece of contour receives a simple verbal label if it starts and stops at successive minima, but the same segment of contour does not when figure and ground are reversed.

5.3.5. The cosine surface

Figure 5-8 shows two identical surfaces, each rotated 180 degrees from the other in

Figure 9. The reversible Schroder staircase



the plane of the page. In this demonstration the preference of the visual system for an interpretation which places the object below rather than overhead causes the visual system to assign different figure-ground relations to the two presentations of this surface. Now according to prediction 2 this figure-ground reversal should be accompanied by a different division of the surface into parts. It is quite clear that the division into rings is different between the two presentations of the surface. The dotted circular contour which falls in the middle of a ring in the bottom presentation of the surface falls between two rings in the top one. To verify that no tricks are being played here, one can stare at one of the surfaces and watch what happens when the page is turned upside down.

5.3.6. The Schroder staircase

Figure 5-9 shows the well known reversible Schroder staircase. I have placed a dot on two adjacent step faces of the staircase. Observe that sometimes the two dotted faces are organized together into one step. When figure and ground reverse these same two dotted faces become organized as faces on two different steps. Again a figure-ground reversal induces a change in the way the same shape is organized into parts, and does so in a way predicted by the surface partitioning rule.

5.3.7. The stacked cubes figure

Somewhat more complicated than the Schroder staircase, but illustrating the same

Figure 8. The cosine surface

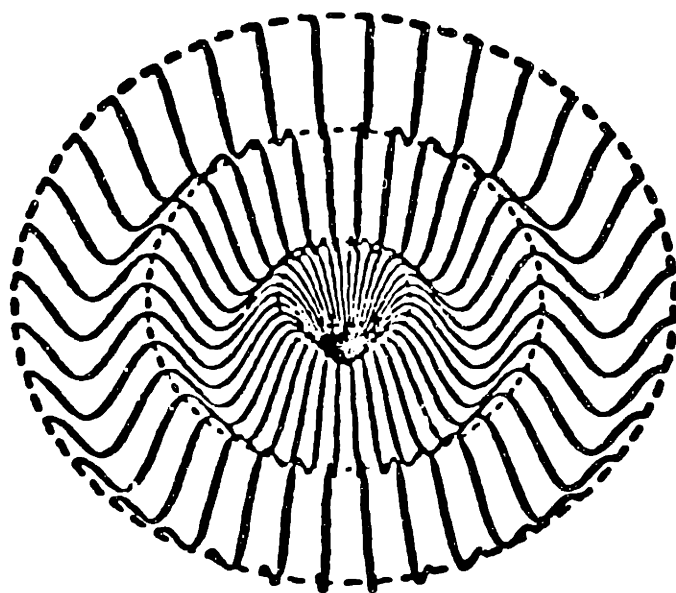
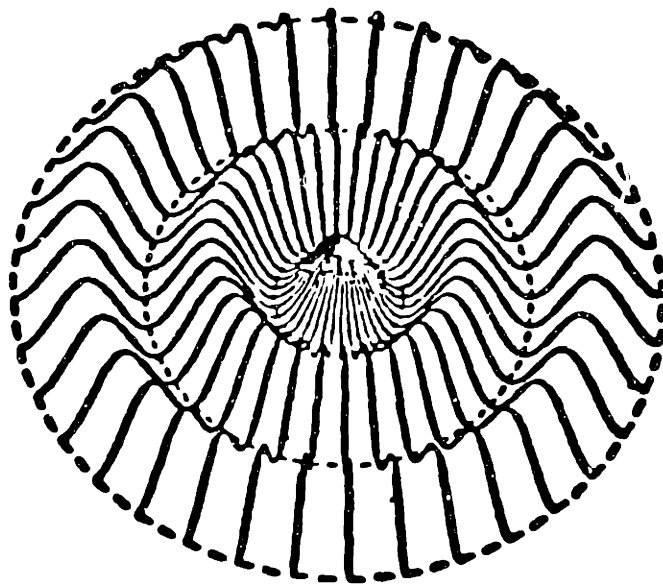
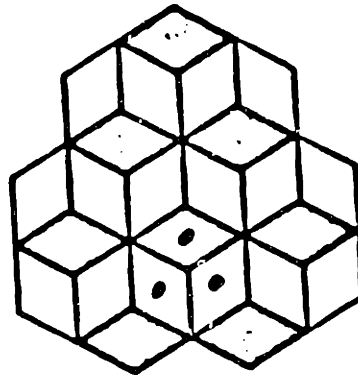


Figure 10. The stacked cubes figure



effects of the surface partitioning rule, is the well known stacked cubes demonstration of figure 5-10. I have placed three dots on the cube in the center, one on each of its visible faces. When figure and ground reverse the same three faces become members of three different cubes. Once again we see that the division of a surface into parts changes when figure and ground reverse, as predicted by the surface partitioning rule.

5.3.8. The crater illusion

The final in my parade of demonstrations that illustrate the approach taken in this thesis is the crater illusion (figure 5-11). If this figure is turned upside down one notices that bumps turn into depressions and vice-versa. The explanation most often given is that we are accustomed to an overhead light source and will choose an interpretation consistent with overhead illumination (Frisby 1979, Schiffman 1976, Gregory 1966).

This explanation is unlikely on two counts. First, the direction of light source plays no role in similar depth reversals such as the ones discussed earlier in this chapter (Kaufman 1974 also notes this). That is, shading is unnecessary for the perception of illusory depth reversals. Second, natural scenes often have more than one effective light source, making the light source direction a weak basis for an explanatory theory of this phenomenon.

What seems more likely is that the visual system prefers an interpretation which places an object below rather than overhead. When a shaded image is turned upside down, the visual system must reverse figure and ground, and consequently the field of surface normals,

Figure 11. The crater illusion



to keep the object from ending up overhead. This reversal implies an attendant flip in the sign of normal curvature everywhere, making bumps into depressions and vice-versa.

5.4. Summary

In this chapter I test a few clear perceptual predictions of the codon theory using both new experiments and familiar perceptual demonstrations. These predictions follow from the claim that parts of curves and surfaces are defined by minima of curvature, and that therefore the parts should be different when figure and ground reverse. I find that some tests of judgements of shape similarity and part naturalness are indeed consistent with the codon theory. In addition a memory probe test, though not yet properly working, also gives promise of supporting some predictions of the theory.

The experiments reported in this chapter are just a beginning. More detailed predictions of similarity based on the metrical properties of codons need to be checked. More detailed studies of memory for shape need to be performed. The conclusion that part boundary strength is related to the magnitude of curvature at minima of curvature needs to be tested. Both theoretical and experimental work is needed to discover the relation between boundary based schemes, such as codons, and region based schemes, such as generalized cylinders. For example, can the predictions of similarity of shapes using the two schemes be pitted

against each other to see under which conditions each predominates? Both theoretical and experimental work is needed to discover how surface occlusions are exploited in determining surface parts. Ultimately the theory and experiments need to reintroduce contextual information and other top down information that must inevitably play an important role in understanding the problem of visual recognition in all its complexity. The problem is a big one, and the further theoretical and experimental exploration required should keep many researchers tantalized and intrigued for years to come.

Epilogue: Face interpretation

Initially this thesis was to be a theory of face recognition and interpretation. Unsatisfied with extant theories of face perception which relied on isolated features, such as hair or eye color, or which mentioned but never really explored configural properties, I decided that I would try to develop a theory of face perception which focussed on the shapes of the surface of the face. I speculated that, given an appropriate representational scheme, the shape of a face might yield such secrets as the age, sex, race, emotions, and identity of the individual. A quick literature search would reveal a representation of shape that, with little remodelling, would be appropriate for faces.

I still think the shapes of faces can be made to betray interesting secrets. I don't think there is a representation of shape in the literature which is suited for the task. Using generalized cylinders to represent faces, for example, is comparable to forcing square pegs into round holes. Consequently, this thesis represents some haltering steps in the direction of a representation with pretenses of being appropriate for the shapes of faces. A rigorous test of the segmentation rules proposed here will come when they are applied to real depth maps of more complex natural surfaces, such as faces. Issues like scale, quantization, and noise will come to the fore and will have to be dealt with in a principled manner.

But suppose that an elaborated version of the schemes presented here, or some other scheme, turns out to be appropriate for representing the surfaces of faces. How can properties like age, race, sex, and mood be inferred from the shape of the face?

Once again the role of natural constraints in explanatory theories of visual inferences cannot be overemphasized. In the case of faces many of the constraints will ultimately have their source in genetics. For example, systematic differences in faces due to sex and race must at root be genetic. On the other hand, systematic differences in faces due to age likely have genetic and environmental components.

Whatever the underlying source of the constraints, I suggest that they are all expressed at one or more of three basic levels of face structure: (1) the *deep structure*, which includes the bones and cartilage of the face, (2) the *middle structure*, which includes the muscle and fat of the face, and (3) the *surface structure*, which includes the skin and facial hair.

A concrete example is helpful: the deep structure of the face differs systematically between the two sexes. According to Gray's anatomy, the female skull has thinner walls, less strongly marked muscular ridges, less prominent glabella and superciliary arches, smaller air sinuses, a sharper upper margin of the orbit, a more vertical forehead, more prominent frontal and parietal eminences, a more flattened cranial vault, a more rounded contour to the face, smoother facial bones, a smaller maxilla and mandible, and smaller teeth. Not all of

these differences in deep structure between the sexes are reflected in the surface structure of the face, but those that are can often be exploited to determine the sex of the individual, thereby cutting the the search space for recognition in half. Gray, however, makes the following disclaimer about these skull differences (p. 151), "A well marked male or female skull can be easily recognized as such, but in some cases the respective characteristics are so indistinct that the determination of the sex may be difficult or impossible".

To a large extent the deep structure determines what is unique or characteristic of a given face. From the deep structure alone it is possible to infer the race, age, and as just mentioned above, the sex. The deep structure largely determines whether the face is square, rounded, narrow, or wide. The shape of the jaw, nose, upper cheek, and forehead all reflect the structure of the underlying bone. In addition, the deep structure remains relatively invariant for long periods of time. This facilitates recognition despite changes in hair style, skin tone, and other changes in surface structure.

Quantitative studies of the deep structure of faces performed by dentists are one potential source of useful characterizations (Enlow 1975). Dentists recognize three basic types of facial profiles: orthognathic, retrognathic, and prognathic. These categories depend upon the spatial relationship between the mandible and the maxilla. A retrusive mandible and a convex profile are characteristic of the retrognath. The prognath, in contrast, has a protrusive mandible and a concave profile. The orthognathic profile strikes a happy medium between these two extremes.

The mandible is also an important indicator of age. As one grows from youth to adulthood the angle between the ramus and corpus of the mandible decreases from about 140 to 110 degrees (Gray 1977). The mandible takes a more prominent role in the overall structure of the face. As one approaches old age the angle again increases and the mandible becomes edentulous and less prominent.

The two nasal bones are an informative part of the deep structure in that their shape varies considerably from individual to individual and between races (Enlow 1975), and in that the overlying skin reflects quite faithfully the curvature of these bones. Generally the nasal bones have negative Gaussian curvature. The absolute magnitude of this curvature is much less in orientals than europeans. The curvature of the nasal bones increases from infancy to adulthood.

The point of all this is simply to show that the deep structure of a face provides a potentially powerful locus of constraints for the tasks of segmentation, identification, and interpretation. A fuller understanding of these constraints may come from an examination of the literature in the fields of dentistry, plastic surgery, art, perception, and basic anatomy.

I have tried to give some idea of the role of the deep structure in determining the shape of the visible surfaces of a face. Similar accounts can be developed for the middle and surface structures. Knowledge of the constraints embodied at each of these three levels should make possible reliable inferences about age, sex, race, identity, and even mood, from the surface shapes of a face.

References

1. Agin G, 1972 "Representation and description of curved objects" *Stanford Artificial Intelligence Memo* 173
2. Attneave F, 1954 "Some informational aspects of visual perception" *Psychological Review* 61 183-193
3. Attneave F, 1971 "Multistability in perception" *Scientific American* 225 6 63-71
4. Ballard D, Brown C, 1982 *Computer vision* (New Jersey: Prentice-Hall)
5. Binford T, 1971 "Visual perception by computer" *IEEE Conference on Systems and Control, Miami*
6. Blum H, 1973 "Biological shape and visual science, part 1." *Journal of Theoretical Biology* 38 205-287
7. Bower G, Glass A, 1976 "Structural units and the redintegrative power of picture fragments" *Journal of Experimental Psychology: Human Learning and Memory* 2 456-466
8. Brady M, 1982a "Parts description and acquisition using vision" *Proceedings of the Society of Photo-optical and Instrumentation Engineers*
9. Brady M, 1982b "Smoothed local symmetries and local frame propagation" *Proceedings of the IEEE Conference on Pattern Recognition and Image Processing Las Vegas*
10. Brooks R, 1981 "Symbolic reasoning among 3-D models and 2-D images" *Artificial Intelligence* 17 285-348.
11. Bruss A, 1981 "Shape from shading and bounding contour" *PhD Thesis, MIT*
12. Chomsky N, 1980 *Rules and representations* (New York: Columbia University Press)
13. Clows M, 1971 "On seeing things" *Artificial Intelligence* 2 79-112
14. Dennett D, 1978 "Intentional Systems" In *Brainstorms* (Montgomery Vermont: Bradford Books)
15. Do Carmo M, 1976 *Differential geometry of curves and surfaces* (New Jersey: Prentice-Hall)
16. Duda R, Hart P, 1973 *Pattern classification and scene analysis* (New York: Wiley)
17. Enlow D, 1975 *Handbook of facial growth* (Philadelphia: Saunders)
18. Falk G, 1972 "Interpretation of imperfect line data as a three-dimensional scene" *Artificial Intelligence* 4 2 101-144
19. Fodor J, 1982 *Modularity of mind: An essay on faculty psychology* (Cambridge: MIT Press)
20. Foley J, Van Dam A, 1982 *Fundamentals of interactive computer graphics* (Reading, Massachusetts: Addison-Wesley)
21. Frisby J, 1979 *Seeing* (Oxford: Oxford University Press)
22. Gati I, Tversky A, 1982 "Representations of qualitative and quantitative dimensions" *Journal of Experimental Psychology: Human Perception and Performance* 8 325-340
23. Goodman N, 1955 *Fact, Fiction and Forecast* (Cambridge: Harvard University Press)
24. Gray H, 1977 *Gray's anatomy* (New York: Crown Publishers)
25. Gregory R, 1966 *Eye and brain* (New York: McGraw-Hill)
26. Grimson E, 1981 *From images to surfaces: A computational study of the human early visual system* (Cambridge: MIT Press)

27. Guzman A, 1968 "Computer recognition of three-dimensional objects in a visual scene" *MIT AI-TR-228*
28. Hildreth E, 1980 "A computer implementation of a theory of edge detection" *MIT AI-TR-579*
29. Hoffman D, 1982 "Inferring local surface orientation from motion fields" *Journal of the Optical Society of America* 72 888-892
30. Hoffman D, Flinchbaugh B, 1982 "The interpretation of biological motion" *Biological Cybernetics* 42 197-204
31. Hollerbach J, 1975 *Hierarchical shape description of objects by selection and modification of prototypes* (Massachusetts Institute of Technology AI-TR-346) 47-51
32. Horn B, 1973 "The Binford-Horn LINEFINDER" *MIT AI Memo* 285
33. Horn B, 1975 "Obtaining shape from shading information" In *The psychology of computer vision* Ed P Winston (New York: McGraw-Hill)
34. Ikeuchi K, Horn B, 1981 "Numerical shape from shading and occluding boundaries" *Artificial Intelligence* 15 141-184
35. Kaufman L, *Sight and mind* (New York: Oxford)
36. Koenderink J, van Doorn A, 1976 "Local structure of movement parallax of the plane" *Journal of the Optical Society of America* 66 717-723
37. Levi I, 1980 *The enterprise of knowledge* (Cambridge: MIT Press)
38. Lipschutz M, 1969 *Differential Geometry* (New York: McGraw-Hill)
39. Longuet-Higgins M, Prazdny K, 1980 "The interpretation of moving retinal images" *Proceedings of the Royal Society of London, Series B* 208 385-397
40. Mandelbrot B, 1977 *Fractals - form, chance and dimension* (San Francisco: W.H. Freeman and Co)
41. Mandelbrot B, 1982 *The fractal geometry of nature* (San Francisco: W.H. Freeman and Co)
42. Marr D, Poggio T, 1977 "From understanding computation to understanding neural circuitry" *Neurosciences Research Progress Bulletin* 15 470-488
43. Marr D, 1977 "Analysis of occluding contour" *Proceedings of the Royal Society of London, Series B* 197 441-475
44. Marr D, Nishihara H K, 1978 "Representation and recognition of the spatial organization of three-dimensional shapes" *Proceedings of the Royal Society of London, Series B* 200 269-294
45. Marr D, Poggio T, 1979 "A computational theory of human stereo vision" *Proceedings of the Royal Society of London, Series B* 204 301-328
46. Marr D, Hildreth E, 1980 "Theory of edge detection" *Proceedings of the Royal Society of London, Series B* 207 187-217
47. Marr D, 1982 *Vision: a computational investigation into the human representation and processing of visual information* (San Francisco: Freeman)
48. Minsky M, 1975 "A framework for representing knowledge" In *The psychology of computer vision* Ed P Winston (New York: McGraw-Hill)
49. Nevatia R, 1974 "Structured descriptions of complex curved objects for recognition and visual memory" *Stanford AI Memo* 250
50. Nishihara H K, 1977 "Representation of the spatial organization of three-dimensional shapes for visual recognition" *PhD Thesis, MIT*
51. Palmer S, 1977 "Hierarchical structure in perceptual representation" *Cognitive Psychology* 9 441-474

52. Pentland A, 1982 "The visual inference of shape: computation from local features" *PhD Thesis, MIT*
53. Persoon E, Fu K, 1974 "Shape discrimination using Fourier descriptors" *Proceedings of the Second International Joint Congress on Pattern Recognition* 126-130
54. Prazdny K, 1980 "Egomotion and relative depth from optical flow" *Biological Cybernetics* 36 87-102
55. Reed S, 1974 "Structural descriptions and the limitations of visual images" *Memory and Cognition* 2 329-336
56. Richards W, Nishihara H K, Dawson B, 1983 "Cartoon: A biologically motivated edge detection algorithm" MIT AI Memo 660
57. Rock I, 1974 "The perception of disoriented figures" *Scientific American* 230 1 78-85
58. Rock I, 1974 *Orientation and form* (New York: Academic Press)
59. Rosch E, 1977 "Linguistic relativity" In *Thinking* Ed P Johnson-Laird & P Wason (Cambridge: Cambridge University Press)
60. Rubin E, 1958 "Figure and ground" in *Readings in perception* Eds D C Beardslee, M Wertheimer (New York: Van Nostrand)
61. Schiffman H, 1976 *Sensation and perception* (New York: Wiley)
62. Schumaker L, 1981 *Spline functions: basic theory* (New York: Wiley)
63. Smith E, Medin D, 1981 *Categories and Concepts* (Cambridge: Harvard University Press)
64. Spivak M, 1970 *Differential Geometry, Volume 2* (Berkeley: Publish or Perish)
65. Stevens K, 1981 "The information content of texture gradients" *Biological Cybernetics* 42 95-105
66. Tversky A, 1977 "Features of similarity" *Psychological Review* 84 327-352
67. Tversky A, Gati I, 1982 "Similarity, separability, and the triangle inequality" *Psychological Review* 89 123-154
68. Ullman S, 1979 *The interpretation of visual motion* (Cambridge: MIT Press)
69. Vatan P, 1976 "Segmentation of figure after separation from ground" Bachelor's thesis, Dept. of Electrical Engineering and Computer Science, MIT
70. Winston P, 1975 "Learning structural descriptions from examples" In *The psychology of computer vision* Ed P Winston (New York: McGraw-Hill)
71. Witkin A, 1981 "Recovering surface shape and orientation from texture" *Artificial Intelligence* 17 17-45
72. Zucker S, Hummel R, Rosenfeld A, 1977 "An application of relaxation labeling to line and curve enhancement" *IEEE Transactions on computers* C-26 4 394-403

;-*- Mode:LISP ; Fonts:CPTFONT,TR10I,CPTFONTB,TR10IB,HL10; Base:10-+-

```

::: MIT Artificial Laboratory
::: File: oz:ps:<ddhoff>thesis.lisp
::: Creation date: 7-82
::: Last major update: 9-25-82
::: Main goal: compute a codon description of a plane curve
::: Original programmer: Donald D. Hoffman
::: Other files needed: oz:ps:<kent>disp.qfasl
:::                   oz:ps:<kent>trig.qfasl
:::                   oz:ps:<kent>paral.qfasl
:::                   oz:ps:<bradst>biceps.qfasl
::: Conventions: tr10i for comments
:::               cfontb upper case for new definitions
:::               cfontb for special (global) variables and own function calls
:::               hl10 upper case for keywords
:::

```

```

::: Constants defined in this file are:
::: degrees-to-radians

```

```

::: Global variables defined in this file are:
::: global-right-points
::: global-left-points
::: global-scale-ratio
::: b-spline-matrix-flag
::: spline-matrix
::: codon-transition-table
::: codon-transition-table-initialized
::: codon-string-table

```

```

::: Functions defined in this file are:
::: a-demo
::: b-spline-coeffs
::: codon-top-level-description
::: codon-transition-table-init
::: compute-crvtr-variances
::: compute-tangent-variances
::: dig-curve-list
::: display-spline
::: disp-list
::: draw-circle
::: draw-x
::: fill-in-list
::: find-circle
::: find-parabola
::: graph-spline-curvature
::: init-bs-matrix
::: line-angle
::: make-knots
::: make-parabola
::: mouse-curve
::: nat-tans
::: norm-list
::: rotate-list
::: smooth-list
::: spline-coords
::: spline-curvature
::: spline-curvature-extrema
::: spline-curvature-extrema-between-knots
::: spline-pts-to-coefficients
::: spline-unit-tangent

```

```

:::
:::
:::

```

```

*****
*  CONSTANTS and GLOBALS  *
*****

```

```

(defconst DEGREES-TO-RADIANS 0.0174533
  ":: Converts degrees to radians ")

(defvar GLOBAL-RIGHT-POINTS nil
  ":: A list of the points to the right of the point being analyzed
  ::: by compute-tangent-variances ")

(defvar GLOBAL-LEFT-POINTS nil
  ":: A list of the points to the left of the point being analyzed
  ::: by compute-tangent-variances ")

(defvar GLOBAL-SCALE-RATIO nil
  ":: Fraction of scale to be used for window size in variance
  ::: computations by compute-tangent-variances ")

(defvar B-SPLINE-MATRIX-FLAG nil
  ":: flag is nil if spline-matrix is not set up.
  ::: 1 if it is ")

(defvar SPLINE-MATRIX nil
  ":: b spline matrix ")

(defvar CODON-TRANSITION-TABLE nil
  ":: transition table for a finite state machine which converts
  ::: lists of signs of curvature of points of maximum absolute
  ::: value of curvature into a codon string description")

(defvar CODON-TRANSITION-TABLE-INITIALIZED nil
  ":: flag is nil if codon-transition-table is not
  ::: initialized.")

(defvar CODON-STRING-TABLE nil
  ":: codon print names")

(defvar LP-ARRAY nil
  ":: array of points left of the point being analyzed by
  ::: compute-tangent-variances")

(defvar RP-ARRAY nil
  ":: array of points right of the point being analyzed by
  ::: compute-tangent-variances")

(defvar TV-ARRAY nil
  ":: array of tangent variances as a function of scale about a point")

(defvar PTS-ARRAYS-INITED nil
  ":: flag is nil if the right and left point arrays are not created")

(defvar TVLIST nil
  ":: list of tangent variances as a function of scale about a point")

```



```

:
:
:
:

```

```

*****
* FUNCTIONS *
*****

```

```

(defunp A-DEMO ()
  (prog (:tln natural-tangents-list t1 s1 knot-list-tmp equal-spaced-knot-list
        knot-pt pt-coords tangent-angle knot-x pi-coords-list knot-y
        x-coefficient-list y-coefficient-list unequal-spaced-knot-list)
    (setq tln (norm-list (setq t1 (dig-curv-list 200 0.035)) 3))
    (setq tln (norm-list (setq t1 (dig-curv-cusp)) 3))
    (clear)
    (disp-list tln)
    (setq natural-tangents-list nil
          pt-coords-list nil)
    (do i 15 (1+ i) (= i (- (length tln) 15))
      (multiple-value (nil nil) (compute-tangent-variances tln i nil))
      (multiple-value (tangent-angle pt-coords) (nat-tans))
      (push tangent-angle natural-tangents-list)
      (push pt-coords pt-coords-list))
    (setq natural-tangents-list (nreverse natural-tangents-list)
          pt-coords-list (nreverse pt-coords-list))
    (setq s1 (smooth-list natural-tangents-list 6))
    (multiple-value (unequal-spaced-knot-list nil equal-spaced-knot-list)
      (make-knots s1 pt-coords-list 0 nil 5))
    (setq knot-list-tmp (copylist equal-spaced-knot-list))
    (dotimes (i (length equal-spaced-knot-list))
      (setq knot-pt (pop knot-list-tmp)
            knot-x (car knot-pt)
            knot-y (cadr knot-pt))
      (draw-x knot-x knot-y))
    (breep)
    (print "type a space to see the spline")
    (terpri)
    (prin1-then-space "type a c to clear screen first")
    (if (= (tyi) 99)
      (clear))
    (multiple-value (x-coefficient-list y-coefficient-list)
      (spline-pts-to-coefficients equal-spaced-knot-list))
    (display-spline x-coefficient-list y-coefficient-list)
    (dreep)
    (print "type a space to see the spline curvature")
    (tyi)
    (graph-spline-curvatura x-coefficient-list y-coefficient-list)
    (setq extrema-list
      (spline-curvature-extrema x-coefficient-list y-coefficient-list))
    (multiple-value (codon-string codon-list)
      (codon-top-level-description extrema-list))
    (print "codon string is")
    (print codon-string)
    (nreep)
    (print "type a space to see the extrema")
    (tyi)
    (clear)
    (display-spline x-coefficient-list y-coefficient-list)
    (dolist (item extrema-list)
      (draw-x (car (cadr item))(cadr (cadr item))))
    (breep)
    (print "type a space to see the natural tangents plot")
    (tyi)
    (clear)
    (disp-list (norm-list natural-tangents-list 100))
    (beep-rise)
    (print "type a space to see the smoothed natural tangents plot")
    (tyi)
    (clear))

```

```
(disp-list (norm-list s1 100))
(spoob)
(return x-coefficient-list y-coefficient-list s1 natural-tangents-list)))
```

```
(defun B-SPLINE-COEFFS (p1 p2 p3 p4)
```

```
  ...
  ::: b-spline-coeffs: takes four x coordinates or four y coordinates and returns
  ::: a list of four b-spline coefficients
  ...
  ..
```

```
(prog (geometry-vector b-coeffs bcoef-list)
  (setq geometry-vector (make-array '(4 4) ':TYPE 'ART-FLOAT)
        b-coeffs      (make-array '(4 4) ':TYPE 'ART-FLOAT))

  (aset p1 geometry-vector 0 0) ::: initialize geometry vector
  (aset p2 geometry-vector 1 0)
  (aset p3 geometry-vector 2 0)
  (aset p4 geometry-vector 3 0)

  (if (null b-spline-matrix-flag) ::: be sure spline-matrix is set up
      (init-bs-matrix))

  (math:multiply-matrices spline-matrix geometry-vector b-coeffs)
  (setq bcoef-list nil)
  (push (//$ (aref b-coeffs 3 0) 6.0) bcoef-list)
  (push (//$ (aref b-coeffs 2 0) 6.0) bcoef-list)
  (push (//$ (aref b-coeffs 1 0) 6.0) bcoef-list)
  (push (//$ (aref b-coeffs 0 0) 6.0) bcoef-list)
  (return bcoef-list)))
```

```
(defun CODON-TOP-LEVEL-DESCRIPTION (curvature-extrema-list
                                     &AUX
                                     (cel curvature-extrema-list)
                                     (state 0) (codon-list nil)
                                     (new-codon nil) (codon-string " "))
```

```
(if (not codon-transition-table-initialized)
    (codon-transition-table-init))
(dolist (current-extremum cel)
  (if (>= (car current-extremum) 0)
      (setq new-codon (aref codon-transition-table state 0)
            state     (aref codon-transition-table state 1))
      (setq new-codon (aref codon-transition-table state 2)
            state     (aref codon-transition-table state 3)))
  (if (not (null new-codon))
      (setq codon-list (append codon-list (list new-codon)))))

(dolist (new-codon codon-list)
  (setq codon-string (string-append codon-string
                                     (aref codon-string-table new-codon))))
(values codon-string codon-list))
```

```
(defun CODON-TRANSITION-TABLE-INIT ()
  (setq codon-transition-table (make-array '(5 4) ':TYPE 'ART-Q)
        codon-transition-table-initialized t)
```

```
(aset 4 codon-transition-table 0 1)
(aset 1 codon-transition-table 0 3)
```

```
(aset 2 codon-transition-table 1 1)
(aset 0 codon-transition-table 1 2)
```

```

(aset 1 codon-transition-table 1 3)

(aset 2 codon-transition-table 2 0)
(aset 3 codon-transition-table 2 1)
(aset 4 codon-transition-table 2 2)
(aset 1 codon-transition-table 2 3)

(aset 1 codon-transition-table 3 0)
(aset 3 codon-transition-table 3 1)
(aset 3 codon-transition-table 3 2)
(aset 1 codon-transition-table 3 3)

(aset 3 codon-transition-table 4 1)
(aset 1 codon-transition-table 4 3)

(setq codon-string-table (make-array 5 'TYPE 'ART-Q ))
(aset "0- " codon-string-table 0)
(aset "0+ " codon-string-table 1)
(aset "1- " codon-string-table 2)
(aset "1+ " codon-string-table 3)
(aset " 2 " codon-string-table 4)

(return t))

(defun COMPUTE-CRVTR-VARIANCES
  (list-of-points &OPTIONAL
    (point (// (length list-of-points) 2.) point-given-by-user)
    (closed-curve t) (scale-ratio 0.5 scale-ratio-by-user))
  "
  " ::: compute-crvtr-variances: tried to find natural scale curvatures, but failed
  "
  (prog (length-of-list temp-list-pts new-center left-points right-points
        crawl-limit s-plus-dels s-minus-dels sum-crvtrs sum-sqd-crvtrs
        crv-stack crv-list crv-sqd-stack crv-sqd-list mean-crv-list
        crv-var-list s-plus-dels-new s-minus-dels-new crawl-s lpt rpt
        crvtr j-dig-2 crv-sqd mean-crv crv-var crl mcl cvl dels
        pt-coords pt-x-coord pt-y-coord curvature)
    (setq length-of-list (length list-of-points)
          temp-list-pts (copylist list-of-points))

    (if point-given-by-user ::: verify user supplied point is in range
      (if (or (signp 1 (- length-of-list point))(signp 1 point))
        (ferror nil "user given point for computing crvtrs is not in list")))

    (if scale-ratio-by-user ::: check scale ratio between 0 and 1
      (if (or (signp 1 scale-ratio) (signp 1 (-$ .999999 scale-ratio)))
        (ferror nil "scale ratio not between 0 and 1")))

    (if closed-curve ::: for closed curves put point in middle
      (setq temp-list-pts
            (rotate-list temp-list-pts (- point (// length-of-list 2.)))
            new-center (// length-of-list 2.))
      (setq new-center point))

    (setq left-points nil) ::: put points in right and left list (from center)
    (dotimes (i new-center)
      (push (pop temp-list-pts) left-points))
    (setq right-points (cdr temp-list-pts)
          global-right-points (copylist right-points) ::: for displaying natural crvs
          global-left-points (copylist left-points)
          global-scale-ratio scale-ratio)

    (setq pt-coords (pop right-points)
          pt-x-coord (pop pt-coords)
          pt-y-coord (pop pt-coords))

    (setq crawl-limit ::: distance to crawl along curve from center point!

```

```

    (fixr (-$ (//$ (min (length left-points) (length right-points))
                    (+$ 1.0 (//$ scale-ratio 2.0))) 1.0)))

  (setq s-plus-dels 0    :: location of end of variance window
        s-minus-dels 1  :: location of start of variance window
        sum-crvtrs 0    :: sum of crvtr values in window
        sum-sqd-crvtrs 0 :: sum of squared crvtr vales in window
        crv-stack nil   :: all crvtrs in current window
        crv-list nil    :: all crvtrs from point to crawl-limit
        crv-sqd-stack nil :: all squared crvtrs in current window
        crv-sqd-list nil :: all squared crvtrs from point to crawl-limit
        mean-crv-list nil :: all mean crvtr values from point to crawl-limit
        crv-var-list nil :: all crvtr variance values from point to crawl-limit)

  (dotimes (s crawl-limit) :: loop from point to crawl-limit
    (setq dels (//$ (*$ (+$ s 1.0) scale-ratio) 2.0) :: window size/2
          s-plus-dels-new (fixr (+$ s 1.0) dels) :: new window end
          s-minus-dels-new (fixr (-$ (+$ s 1.0) dels)) :: new window start
          crv-stack (nreverse crv-stack)
          crv-sqd-stack (nreverse crv-sqd-stack)
          crawl-s s)

    (dotimes (j (- s-minus-dels-new s-minus-dels)) :: new sum of crvtrs in window
      (setq sum-crvtrs (-$ sum-crvtrs (pop crv-stack))
            sum-sqd-crvtrs (-$ sum-sqd-crvtrs (pop crv-sqd-stack))))

    (setq crv-stack (nreverse crv-stack)
          crv-sqd-stack (nreverse crv-sqd-stack)
          s-minus-dels s-minus-dels-new)

    (dotimes (j (- s-plus-dels-new s-plus-dels)) :: add new values to window
      (setq lpt (pop left-points) :: lpt = (x y)
            rpt (pop right-points) :: rpt = (x y)
            crvtr (find-circle (- (pop lpt) pt-x-coord)
                               (- (pop lpt) pt-y-coord)
                               (- (pop rpt) pt-x-coord)
                               (- (pop rpt) pt-y-coord))
                  curvature (car crvtr)
                  crv-sqd (*$ curvature curvature)
                  sum-crvtrs (+$ sum-crvtrs curvature)
                  sum-sqd-crvtrs (+$ sum-sqd-crvtrs crv-sqd)
                  j-dig-2 j)
      (push curvature crv-list)
      (push curvature crv-stack)
      (push crv-sqd crv-sqd-list)
      (push crv-sqd crv-sqd-stack))

    (setq mean-crv (//$ sum-crvtrs (length crv-stack))
          crv-var (-$ (//$ sum-sqd-crvtrs (length crv-stack))
                  (*$ mean-crv mean-crv))
          s-plus-dels s-plus-dels-new)

    (push mean-crv mean-crv-list)
    (push crv-var crv-var-list))

  (return (list (setq cvl (nreverse crv-var-list))
                (setq mcl (nreverse mean-crv-list))
                (setq crl (nreverse crv-list)))))

```

```
(defun COMPUTE-TANGENT-VARIANCES
```

```

  (list-of-points &OPTIONAL
    (point (// (length list-of-points) 2.) point-given-by-user)
    (closed-curve t) (scale-ratio 0.4 scale-ratio-by-user))

```

```
...
```

```

::: (compute-tangent-variances list) takes a list of positions of points
::: on a curve specified relative to the first point in the list and
::: computes tangent angle variances as a function of scale. One can
::: specify a point P about which the tangents are to be taken, otherwise
::: the middle point of the list is the default. If P is specified then
::: the curve is assumed closed unless otherwise stated in the argument list.
::: Returns multiple values.

```

```

(prog (length-of-list temp-list-pts new-center left-points right-points
      crawl-limit s-plus-dels s-minus-dels sum-tangents sum-sqd-tangents
      tan-stack tan-list tan-sqd-stack tan-sqd-list mean-tan-list
      s-plus-dels-new s-minus-dels-new crawl-s lpt rpt
      tangent j-dig-2 tan-sqd mean-tan tan-var dels)

  (setq length-of-list (length list-of-points)
        temp-list-pts (copylist list-of-points))

  (if point-given-by-user
      ::: verify user supplied point is in range
      (if (or (signp 1 (- length-of-list point))(signp 1 point))
          (ferror nil "user given point for computing tangents is not in list")))

  (if scale-ratio-by-user
      ::: check scale ratio between 0 and 1
      (if (or (signp 1 scale-ratio) (signp 1 (-$ .999999 scale-ratio)))
          (ferror nil "scale ratio not between 0 and 1")))

  (if closed-curve
      ::: for closed curves put point in middle
      (setq temp-list-pts
            (rotate-list temp-list-pts (- point (/ length-of-list 2.)))
            new-center (/ length-of-list 2.))
      (setq new-center point))

  (setq left-points nil)
  ::: put points in right and left list (from center)
  (dotimes (i new-center)
    (push (pop temp-list-pts) left-points))
  (setq right-points (cdr temp-list-pts)
        global-right-points (copylist right-points) ::: for displaying natural tans
        global-left-points (copylist left-points)
        global-scale-ratio scale-ratio)

  (setq crawl-limit ::: distance to crawl along curve from center point
        (fixr (-$ (/ $ (min (length left-points) (length right-points))
                          (+$ 1.0 (/ $ scale-ratio 2.0))) 1.0)))

  (setq s-plus-dels 0 ::: location of end of variance window
        s-minus-dels 1 ::: location of start of variance window
        sum-tangents 0 ::: sum of tangent values in window
        sum-sqd-tangents 0 ::: sum of squared tangent values in window
        tan-stack nil ::: all tangents in current window
        tan-list nil ::: all tangents from point to crawl-limit
        tan-sqd-stack nil ::: all squared tangents in current window
        tan-sqd-list nil ::: all squared tangents from point to crawl-limit
        mean-tan-list nil ::: all mean tangent values from point to crawl-limit
        tvlist nil) ::: all tangent variance values from point to crawl-limit

  (dotimes (s (min crawl-limit (fixr (*$ 0.15 length-of-list))))
    ::: loop from point to crawl-limit
    (setq dels (/ $ (*$ (+$ s 1.0) scale-ratio) 2.0) ::: window size/2
          s-plus-dels-new (fixr (+$ s 1.0 dels)) ::: new window end
          s-minus-dels-new (fixr (-$ (+$ s 1.0) dels)) ::: new window start
          tan-stack (nreverse tan-stack)
          tan-sqd-stack (nreverse tan-sqd-stack)
          crawl-s s)

    (dotimes (j (- s-minus-dels-new s-minus-dels))
      ::: new sum of tangents in window
      (setq sum-tangents (-$ sum-tangents (pop tan-stack))
            sum-sqd-tangents (-$ sum-sqd-tangents (pop tan-sqd-stack))
            mean-tan-list (-$ mean-tan-list (pop mean-tan-list))
            tvlist (-$ tvlist (pop tvlist))
            tan-stack (pop tan-stack)
            tan-sqd-stack (pop tan-sqd-stack)
            mean-tan-list (pop mean-tan-list)
            tvlist (pop tvlist))
    )
  )

```

```

sum-sqd-tangents (-$ sum-sqd-tangents (pop tan-sqd-stack))))

(setq tan-stack      (nreverse tan-stack)
  tan-sqd-stack      (nreverse tan-sqd-stack)
  s-minus-dels       s-minus-dels-new)

(dotimes (j (- s-plus-dels-new s-plus-dels))      ;; add new values to window
  (setq lpt          (pop left-points)           ;; lpt = (x y)
    rpt             (pop right-points)          ;; rpt = (x y)
    tangent         (line-angle (pop lpt)(pop lpt)(pop rpt)(pop rpt))
    tan-sqd         (*$ tangent tangent)
    sum-tangents    (+$ sum-tangents tangent)
    sum-sqd-tangents (+$ sum-sqd-tangents tan-sqd)
    j-dig-2         j)
  (push tangent      tan-list)
  (push tangent      tan-stack)
  (push tan-sqd      tan-sqd-list)
  (push tan-sqd      tan-sqd-stack))

(setq mean-tan      (//$ sum-tangents (length tan-stack))
  tan-var (-$ (//$ sum-sqd-tangents
    (length tan-stack)
    (*$ mean-tan mean-tan))
  s-plus-dels       s-plus-dels-new)

(push mean-tan      mean-tan-list)
(push tan-var tvlist)
(setq tvlist (nreverse tvlist))

(return (nreverse mean-tan-list)
  (nreverse tan-list)))

(defun DIG-CURV-CUSP (&OPTIONAL (steps 300))
  (prog ()
    (setq pts-list nil)
    (dotimes (i steps)
      (setq x (+$ -5.999 (*$ 12.0 (//$ (float i) (float steps))))))
      (cond ((< x -2.0)
        (setq y (^ (+$ x 4.0) 2)))
        ((< x 2.0)
        (setq y (^ x 2)))
        (t
        (setq y (^ (-$ x 4.0) 2))))
      (push (list x y) pts-list))
    (return (nreverse pts-list)))

(defun DIG-CURV-LIST (&OPTIONAL (steps 100.)(size 0.05)(start 0.0))
  ;;
  ;;
  ;; dig-curv-list creates a list of points on a digitized analytic
  ;; function. The list members are lists of the form (x y).
  ;; 7/29/82
  (prog (curv-list scale x arg y-new ydif yabs ytmp ysign dig-point y)
    (setq curv-list nil
      scale (//$ 1.0 size))

    (dotimes (j steps)
      (setq x (fix (+$ start j))
        arg (+$ start (*$ j size))
        y-new (fix (*$ scale (sin arg)))
        y-new (fix (*$ scale (+$ (sin arg)
          (*$ 0.1 (sin (*$ 10.0 arg)))))))

      (cond ((signp g j)
        (setq ydif (- y-new y)
          yabs (abs ydif))
        (if (signp e yabs)

```

```

      (setq ysign 1.0)
      (setq ysign (//$ ydif yabs)))
    (dotimes (l (+ 1 yabs))
      (setq ytmp (fix (+$ y (*$ ysign 1)))
            dig-point (list x ytmp))
      (push dig-point curv-list))

    (t (setq dig-point (list x y-new))
      (push dig-point curv-list))

    (setq y y-new))
  (return (nreverse curv-list)))

```

```
(defun DISPLAY-SPLINE (xcoefficient-list ycoefficient-list &OPTIONAL (res 50))
```

```

  "
  ...
  ... display-spline: takes two lists. one of coefficients of powers for the x coordinate
  ... and the other of coefficients of powers for the y coordinate of a b-spline, and displays
  ... the spline on the lisp machine screen
  ...
  "

```

```

  (prog (num-coeffs xc yc resolution-scaling new-x-coeffs new-y-coeffs
        ax bx cx dx ay by cy dy tp x-coord y-coord)
    (setq num-coeffs (length xcoefficient-list)
          xc (copylist xcoefficient-list)
          yc (copylist ycoefficient-list)
          resolution-scaling (//$ 1.0 res))

    (dotimes (i num-coeffs)
      (setq new-x-coeffs (pop xc)
            new-y-coeffs (pop yc)
            ax (car new-x-coeffs)
            bx (cadr new-x-coeffs)
            cx (caddr new-x-coeffs)
            dx (caddr new-x-coeffs)
            ay (car new-y-coeffs)
            by (cadr new-y-coeffs)
            cy (caddr new-y-coeffs)
            dy (caddr new-y-coeffs))

      (dotimes (j res)
        (setq tp (*$ (float j) resolution-scaling)
              x-coord (+$ dx (*$ tp (+$ cx (*$ tp (+$ bx (*$ tp ax)))))) ::horner's
              y-coord (+$ dy (*$ tp (+$ cy (*$ tp (+$ by (*$ tp ay)))))) ::rule
              (if (and (= i 0) (= j 0))
                  (disp-setpos x-coord y-coord)
                  (disp-drawline x-coord y-coord))))

      (return num-coeffs))

```

```
(defun DISP-LIST (dsplst &OPTIONAL (x-flg nil)
                 (x-pt (fixr (//$ (float (length dsplst)) 2.0))))
```

```

  "
  ...
  ... disp-list: displays a list of points on the lisp machine
  ... joined by line segments. The list of points is a list of
  ... lists, each of the form (x y). 7/30/82 (optionally the user
  ... can specify that an x be drawn through one of the points.
  ...
  "

```

```

  (prog (dsplst-tmp pt lstlen xval yval)
    (setq dsplst-tmp (copylist dsplst)
          pt (pop dsplst-tmp))
    (disp-setpos (pop pt) (pop pt))
    (setq lstlen (length dsplst-tmp))
    (dotimes (m lstlen)

```

```

    (setq pt (pop dsplst-tmp))
    (setq xval (pop pt)
          yval (pop pt))
    (if x-flg
        (if (signp e (- x-pt (1+ m)))
            (draw-x xval yval)))
        (disp-drawline xval yval))
    (return lstlen))

```

```

(defun DRAW-CIRCLE (x0 y0 r)

```

```

    "
    ...
    ... draw-circle: draws a circle of given x, y center and radius
    ...
    "

```

```

    (prog (theta newx newy)
          (disp-setpos (fixr (+ $ x0 r)) (fixr y0))
          (dotimes (i 72)
                (setq theta (* $ 5.0 degrees-to-radians (+ $ 1.0 (float i)))
                      newx (+ $ x0 (* $ r (cos theta)))
                      newy (+ $ y0 (* $ r (sin theta))))
                (disp-drawline newx newy)))

```

```

(defun DRAW-X (xval yval &OPTIONAL (x-size 16.))

```

```

    "
    ...
    ... draw-x: draws an x through the specified point
    ...
    "

```

```

    (disp-setpos (+ xval x-size) (+ yval x-size))
    (disp-drawline (- xval x-size) (- yval x-size))
    (disp-setpos (- xval x-size) (+ yval x-size))
    (disp-drawline (+ xval x-size) (- yval x-size))

```

```

(defun FILL-IN-LIST (pts-1st)

```

```

    "
    ...
    ... fill-in-list: takes a list of points which are not necessarily
    ... spatially contiguous and returns a list which fills in between
    ... the points in a linear fashion.
    ...
    "

```

```

    (prog (out-list old-pt old-x old-y new-x new-y yd xd xabs xsign apt
              slope ytm ytm-new ytm-dif ytm-abs xtm ytm-sgn ytt
              tmplst yabs ysign ytmp)
          (setq tmplst (copylist pts-1st)
                out-list nil
                old-pt (pop tmplst))
          (push old-pt out-list)
          (setq old-x (pop old-pt)
                old-y (pop old-pt))
          (dolist (new-pt tmplst)
                (setq new-x (pop new-pt)
                      new-y (pop new-pt)
                      yd (- new-y old-y)
                      xd (- new-x old-x)
                      yabs (abs yd)
                      xabs (abs xd))
                (if (signp e yabs)
                    (setq ysign 1.0)
                    (setq ysign (// $ yd yabs))))

```



```

(if (signp e xabs)
    (setq xsign 1.0)
    (setq xsign (//$ xd xabs)))
(cond ((signp e xabs)
      (dotimes (i yabs)
        (setq ytmp (+ old-y (* (1+ i) ysign))
              apt (list old-x ytmp))
        (push apt out-list)))
      (t
       (setq slope (//$ yd xd)
              ytm old-y)
       (dotimes (i xabs)
         (setq ytm-new (fix (+$ old-y (*$ (1+ i) slope xsign)))
               ytm-dif (- ytm-new ytm)
               ytm-abs (abs ytm-dif)
               xtm (+ old-x (* (1+ i) xsign)))
         (if (signp e ytm-abs)
             (setq ytm-sgn 1.0)
             (setq ytm-sgn (//$ ytm-dif ytm-abs)))
         (dotimes (l (1+ ytm-abs))
           (setq ytt (fix (+$ ytm (*$ ytm-sgn l)))
                 apt (list xtm ytt))
           (push apt out-list))
         (setq ytm ytm-new))))
      (setq old-x new-x
            old-y new-y))
(return (setq out-list (nreverse out-list))))

```

```

(defun FIND-CIRCLE (x1 y1 x2 y2)

```

```

:::
:::
::: (find-circle x1 y1 x2 y2) finds the circle through the given points and the point 0,0.
:::
:::

```

```

(prog (a1 b1 c1 a2 b2 c2 x0 y0 xdenom ydenom curvature)
      (setq a1 (*$ x1 -2.0)
            b1 (*$ y1 -2.0)
            c1 (+$ (*$ x1 x1)(*$ y1 y1))
            a2 (*$ x2 -2.0)
            b2 (*$ y2 -2.0)
            c2 (+$ (*$ x2 x2)(*$ y2 y2))
            xdenom (-$ (*$ a1 b2)(*$ a2 b1))
            ydenom (-$ (*$ a2 b1)(*$ a1 b2)))
      (if (or (signp e xdenom) (signp e ydenom))
          (setq x0 nil
                y0 nil
                curvature 0)
          (setq x0 (//$ (-$ (*$ b1 c2)(*$ b2 c1))
                       xdenom)
                y0 (//$ (-$ (*$ a1 c2)(*$ a2 c1))
                       ydenom)
                curvature (//$ 1.0 (sqrt (+$ (*$ x0 x0) (*$ y0 y0))))))
      (return (list curvature x0 y0)))

```

```

(defun FIND-PARABOLA (x1a y1a x2a y2a x3a y3a)

```

```

::: find-parabola:
::: This routine takes the x,y coordinates of three points and fits
::: a general parabola through them and the point 0,0.
:::
::: The graph of the equation:

```

```

...:  $ax^2 + bxy + cy^2 + dx + ey + f = 0$ 
...:
...: is a parabola only if
...:
...:  $b^2 - 4ac = 0$ 
...:
...: We assume that the point (0,0) is one of the points, so that  $f = 0$ .
...: We then divide through by  $c$ . So we need only find  $a, b, c, d$ .
...: Since in general there are two solution sets, par returns a list
...: of two lists each containing, in order, the four elements  $a b c d$ .
...: ( $c = 1, f = 0$ )
...:
...:

```

```

(prog (x1 y1 x2 y2 x3 y3 h1 h2 h3 h4 h5 h6 h7 h8 k1 k2 k3 dc c1 c2 b1 b2
      a1 a2 v1 v2 dcs fg d1 d2)
  (setq x1 (float x1a)
        y1 (float y1a)
        x2 (float x2a)
        y2 (float y2a)
        x3 (float x3a)
        y3 (float y3a)
        h1 (-$ (*$ x2 x1 x1) (*$ x1 x2 x2))
        h2 (-$ (*$ x1 y1 x2) (*$ x2 y2 x1))
        h3 (-$ (*$ x2 y1 y1) (*$ y2 y2 x1))
        h4 (-$ (*$ x2 y1) (*$ x1 y2))
        h5 (-$ (*$ x3 x1 x1) (*$ x1 x3 x3))
        h6 (-$ (*$ x1 y1 x3) (*$ x3 y3 x1))
        h7 (-$ (*$ y1 y1 x3) (*$ y3 y3 x1))
        h8 (-$ (*$ x3 y1) (*$ x1 y3))
        k1 (-$ (*$ h1 h8) (*$ h4 h5))
        k2 (-$ (*$ h2 h8) (*$ h4 h6))
        k3 (-$ (*$ h3 h8) (*$ h4 h7))
        dc (-$ (*$ k2 k2) (*$ k1 k3)))
  (cond ((signp e k1)
        (setq c1 0.0
              c2 0.0
              b1 0.0
              b2 0.0)
        (cond ((and (signp e h1) (signp e h5))
              (setq a1 nil
                    a2 nil))
              ((signp e h1)
               (setq a1 (/$ (-$ 0.0 h8) h5)
                     a2 a1))
              (t (setq a1 (/$ (-$ 0.0 h4) h1)
                       a2 a1))))
        ((signp e k2)
         (setq a1 0.0
               a2 0.0
               b1 0.0
               b2 0.0)
         (cond ((and (signp e h3) (signp e h7))
               (setq c1 nil
                     c2 nil))
               ((signp e h3)
                (setq c1 (/$ (-$ 0.0 h8) h7)
                      c2 c1))
               (t (setq c1 (/$ (-$ 0.0 h4) h3)
                        c2 c1))))
        (t (cond ((signp 1 dc)
                  (setq dcs 0.0
                        fg t))
              (t (setq fg nil
                      dcs (sqrt dc))))
          (setq v1 (*$ 2.0 (/$ (-$ dcs k2) k1))
                v2 (*$ 2.0 (/$ (-$ 0.0 dcs k2) k1))
                c1 (/$ (*$ -4.0 h4) (+$ (*$ v1 v1 h1) (*$ 4.0 v1 h2) (*$ 4.0 h3)))
                c2 (/$ (*$ -4.0 h4) (+$ (*$ v2 v2 h1) (*$ 4.0 v2 h2) (*$ 4.0 h3)))
                h1 (*$ v1 c1))

```

```

      b2 (*$ v2 c2))
      (if (signp e h5)
          (setq a1 (//$ (+$ (*$ b1 h2) (*$ c1 h3) h4) (*$ -1.0 h1))
                a2 (//$ (+$ (*$ b2 h2) (*$ c2 h3) h4) (*$ -1.0 h1)))
          (setq a1 (//$ (+$ (*$ b1 h6) (*$ c1 h7) h8) (*$ -1.0 h5))
                a2 (//$ (+$ (*$ b2 h6) (*$ c2 h7) h8) (*$ -1.0 h5))))))
      (setq d1 (//$ (+$ (*$ a1 x1 x1) (*$ b1 x1 y1) (*$ c1 y1 y1) y1) (*$ -1.0 x1))
            d2 (//$ (+$ (*$ a2 x1 x1) (*$ b2 x1 y1) (*$ c2 y1 y1) y1) (*$ -1.0 x1)))
      (return (list (list a1 b1 c1 d1) (list a2 b2 c2 d2))))

```

```

...

```

```

... The following test cases have been passed by the find-parabola routine:(7/26/82)

```

```

... (find-parabola 1.0 1.0 2.0 4.0 -1.0 1.0) = (-1.0 0.0 0.0 0.0)
... (find-parabola 1.0 -1.0 2.0 -4.0 -1.0 -1.0) = (1.0 0.0 0.0 0.0)
... (find-parabola 1.0 2.0 2.0 6.0 -1.0 0.0) = (-1.0 0.0 0.0 -1.0)
... (find-parabola 1.0 0.38196601 2.0 1.0 3.0 1.69722436) = (-1.0 2.0 -1.0 0.0)
... (find-parabola -1.0 -2.18614066 -4.0 -6.5894542 -10.0 -14.2291182)
... = (-2.0 4.0 -2.0 -5.0)
... (find-parabola 1.0 4.0 2.0 28.0 3.0 72.0) = (-10.0 0.0 0.0 6.0)
... (find-parabola -2.0 1.0 -6.0 2.0 0.0 -1.0) = (0.0 0.0 1.0 1.0)
...

```

```

(defun GRAPH-SPLINE-CURVATURE (x-coefficient-list y-coefficient-list)
  (prog (spline-curvature-list xc yc curvature-list)
    (setq spline-curvature-list nil
          xc (copylist x-coefficient-list)
          yc (copylist y-coefficient-list))
    (dotimes (i (length x-coefficient-list))
      (multiple-value (nil nil nil nil curvature-list)
        (spline-curvature-extrema-between-knots (pop xc) (pop yc)))
      (setq spline-curvature-list (append spline-curvature-list curvature-list)))
    (clear)
    (disp-list (norm-list spline-curvature-list 12000))
    (return t spline-curvature-list)))

```

```

(defun INIT-BS-MATRIX ()
  (setq spline-matrix (make-array '(4 4) :TYPE 'ART-FLOAT)
        b-spline-matrix-flag t)

  (aset -1 spline-matrix 0 0)
  (aset 3 spline-matrix 0 1)
  (aset -3 spline-matrix 0 2)
  (aset 1 spline-matrix 0 3)

  (aset 3 spline-matrix 1 0)
  (aset -6 spline-matrix 1 1)
  (aset 3 spline-matrix 1 2)

  (aset -3 spline-matrix 2 0)
  (aset 3 spline-matrix 2 2)

  (aset 1 spline-matrix 3 0)
  (aset 4 spline-matrix 3 1)
  (aset 1 spline-matrix 3 2)
  (return t))

```

```

(defun LINE-ANGLE (x11 y11 x12 y12)

```

```

...
... line-angle: takes the coordinates of 2 2-d points and returns
... the angle of the line between them. Gives a value between 0

```

```

::: and 2*pi in radians. (verified 7/29/82)
:::
:::

```

```
(return (atan2 (-$ y12 y11) (-$ x12 x11))))
```

```
(defun MAKE-KNOTS (tan-list pts-list &OPTIONAL (offset 3) (closed nil)
                  (window 5) ::: assumed to be odd (its the number of deltas - not knots)
                  )
  (prog (tarray len parray last-tan last-crd knot-list tmp-tan k
        equal-arc-length-knot-list)
    (setq tarray (make-array (setq len (length pts-list)) ':leader-list '(offset t))
          parray (make-array len ':leader-list '(0 t)))

    (fillarray tarray tan-list)
    (fillarray parray pts-list)

    (setq last-tan (aref tarray offset)
          last-crd (aref parray 0)
          knot-list nil
          last-index 0
          deltas-list nil
          equal-arc-length-knot-list nil)
    (push last-crd knot-list)
    :::
    ::: unequally spaced knots
    :::
    (do i (1+ offset)(1+ i)(= i (1- len))
      (setq tmp-tan (aref tarray i))
      (if (greaterp (abs (-$ last-tan tmp-tan)) 0.35) :::knot every 20 degrees
        (setq last-tan tmp-tan
              last-crd (aref parray i)
              k (push (- i last-index) deltas-list) ::: for locally equal spacing of knots
                last-index i
                k (push last-crd knot-list))))
    (setq knot-list (nreverse knot-list)
          k (/ (length pts-list)(length knot-list)))
    :::
    ::: equally spaced knots
    :::
    (dotimes (i (length knot-list))
      (push (aref parray (fix (*$ (float i) k)))
            equal-arc-length-knot-list))
    (push (aref parray (1- (length pts-list))) equal-arc-length-knot-list)
    (cond (closed ::: append the first two knots to the end of the knot list
          (push (aref parray (fix k)) equal-arc-length-knot-list)
          (push (aref parray (fix (* 2 k))) equal-arc-length-knot-list)))
    :::
    ::: locally equally spaced knots
    :::
    (cond (closed
          (dotimes (i (1+ (/ window 2)))
            (setq tmp-lst (rotate-list deltas-list (- 0 i)))
            (push (car tmp-lst) deltas-list)))
          (setq deltas-list (nreverse deltas-list)
                del-array (make-array (length deltas-list) ':TYPE 'ART-FLOAT)
                locally-equal-deltas nil
                locally-equal-knot-list nil)
          (fillarray del-array deltas-list)
          (setq first-time-through-loop t)
          (do i (1- window) (1+ i) (= i (1- (length deltas-list)))
            (setq del-sum 0)
            (do j 1 (1- j) (= j (- i window))
              (setq del-sum (+ del-sum (aref del-array j))))
            (if first-time-through-loop
              (setq first-knot-index (fixr (*$ (/ window 2)
                                             (/ $ (float del-sum)

```

```

                                (float window))))
      first-time-through-loop nil))
    (push (//$ (float del-sum) (float window)) locally-equal-deltas))
  (setq locally-equal-deltas (nreverse locally-equal-deltas))
  (if (not closed)
      (setq first-knot-index (+ first-knot-index 15)))
  (push (aref parray first-knot-index) locally-equal-knot-list)
  (setq next-knot-index first-knot-index)
  (dolist (item locally-equal-deltas)
    (setq next-knot-index (fixr (+$ next-knot-index item)))
    (push (aref parray next-knot-index) locally-equal-knot-list))
  (return knot-list equal-arc-length-knot-list locally-equal-knot-list)))

```

```
(defun MAKE-PARABOLA (a b c d x)
```

```

  ;; make-parabola:
  ;; on parabola of form  $ax^2 + bxy + cy^2 + dx + y + 0 = 0$ 
  ;; this routine computes the y value for the given x value
  ;; and returns it. routine is for generating
  ;; data to check the find-parabola routine.
  ;; Do not use this routine if  $c=0$ . its simple enough to do by hand
  ;; if  $c=0$  anyway.

```

```

(prog (bp cp an1 an2 dis dc)
  (setq dis (-$ (*$ b b) (*$ 4.0 a c)))
  (cond ((signp e dis)
        (setq bp (+$ 1.0 (*$ b x))
              cp (+$ (*$ a x x) (*$ d x))
              dc (sqrt (-$ (*$ bp bp) (*$ 4.0 c cp)))
              an1 (//$ (-$ dc bp) (*$ 2.0 c))
              an2 (//$ (-$ 0.0 bp dc) (*$ 2.0 c))))
        (t (setq an1 nil
                 an2 nil))))
  (return (list an1 an2)))

```

```
(defun MOUSE-CURVE (&OPTIONAL (num-pts 200.))
```

```

  ;;
  ;; mouse-curve: reads the mouse movements and stores a specified number of
  ;; points in a list
  ;;

```

```

(prog (old-mouse-x old-mouse-y mouse-list new-mouse-x new-mouse-y)
  (clear)
  (setq old-mouse-x (disp-x sys:mouse-x)
        old-mouse-y (disp-y sys:mouse-y)
        mouse-list nil)
  (disp-setpos old-mouse-x old-mouse-y)
  (dotimes (i num-pts)
    (setq new-mouse-x (disp-x sys:mouse-x)
          new-mouse-y (disp-y sys:mouse-y))
    (cond ((or (signp n (- old-mouse-x new-mouse-x))
               (signp n (- old-mouse-y new-mouse-y))))
          (push (list new-mouse-x new-mouse-y) mouse-list)
          (disp-drawline new-mouse-x new-mouse-y))
    (t (setq i (1- i))))
  (setq old-mouse-x new-mouse-x
        old-mouse-y new-mouse-y)))

```

```
(defun NAT-TANS (&OPTIONAL
  (var-thrsh 0.0005)
```

```
(sr global-scale-ratio))
```

```

...
... nat-tans: takes a tangent variance list computed by
... compute-tangent-variances and finds the most natural
... tangents. It returns multiple values.
...

```

```

(prog (garbage min-limit nat-scl-1st new-nat-scl
      pt-index l-point r-point
      npl lx ly rx ry num-nat-scales
      tlnlst tmp ydif xdif tangent-angle tangent-angle-list)
  (if (not pts-arrays-inited)
      (setq lp-array (make-array 500 :TYPE 'ART-Q)
            rp-array (make-array 500 :TYPE 'ART-Q)
            tv-array (make-array 500 :TYPE 'ART-FLOAT)
            pts-arrays-inited t))
    (fillarray lp-array global-left-points)
    (fillarray rp-array global-right-points)
    (fillarray tv-array tvlist)
    (setq min-limit 4.
          nat-scl-1st nil ;; list of natural scales
          new-nat-scl nil) ;; possible new natural scale
    (dotimes (i (length tvlist))
      (if (signp 1 (- var-thrsh (aref tv-array i))) ;; is var-val > var-thrsh?
          (if (and (not (null new-nat-scl)) (signp 1e (- min-limit i))) ;; yes
              (setq garbage (push new-nat-scl nat-scl-1st)
                    new-nat-scl nil)
              (setq new-nat-scl i ;; no
                    min-limit (+ i 10 (fixr (*$ (float i) sr))))))
          (if (not (null new-nat-scl)) ;; in case one runs out of list.
              (push new-nat-scl nat-scl-1st))
          (setq num-nat-scales (length nat-scl-1st)
                npl nil
                tangent-angle-list nil
                tlnlst nil)
          (dotimes (i num-nat-scales)
            (setq tmp (pop nat-scl-1st)
                  pt-index (fixr (+$ tmp 1.0 (//$ (*$ (+$ tmp 1.0) sr) 2.0)))
                  pt-index (fixr tmp)
                  l-point (aref lp-array pt-index)
                  r-point (aref rp-array pt-index)
                  lx (car l-point)
                  ly (cadr l-point)
                  rx (car r-point)
                  ry (cadr r-point)
                  xdif (-$ rx lx)
                  ydif (-$ ry ly)
                  tangent-angle (line-angle lx ly rx ry))
              (push tangent-angle tangent-angle-list)
              (push (sqrt (+$ (*$ xdif xdif)
                             (*$ ydif ydif))) tlnlst) ;; lengths of natural tangents
              (push (list l-point r-point) npl) ;; endpoints of natural tangents
              (return tangent-angle (aref rp-array 0) tangent-angle-list tlnlst npl)))
    ;

```

```
(defun NORM-LIST (nrmlst &OPTIONAL (y-scaling 1.0)
                 (xlower 5.0)(xupper 1000.0)(ylower 5.0)(yupper 1000.0))
```

```

...
...
... norm-list: scales a list of points to fit between the upper
... and lower values specified
... one can give either a list where each member of the list is itself
... a list of the form (x y), or one can give a list of y values where
... x values are implicitly to go from 0 to the length of the list

```

```
... 7/30/82
```

```
...
...
"
```

```
(prog (nrmlst-tmp xmax xmin ytmp ymax ymin nrmlen npt xtmp xrange yrange
      xrange-allowed yrange-allowed xratio yratio scalef outlst
      newx newy)
  (setq nrmlst-tmp (copylist nrmlst)
        xmax -999999.0
        xmin 9999999.0
        ymax xmax
        ymin xmin
        nrmlen (length nrmlst))

  (dotimes (i nrmlen)
    (setq npt (pop nrmlst-tmp))
    (if (listp npt)
        (setq xtmp (pop npt)
              ytmp (*$ y-scaling (pop npt)))
        (setq ytmp (*$ npt y-scaling)
              xtmp (float i)))
    (setq xmax (max xmax xtmp)
          xmin (min xmin xtmp)
          ymax (max ymax ytmp)
          ymin (min ymin ytmp)))

  (setq xrange (-$ xmax xmin)
        yrange (-$ ymax ymin)
        xrange-allowed (-$ xupper xlower)
        yrange-allowed (-$ yupper ylower)
        xratio (//$ (float xrange) (float xrange-allowed))
        yratio (//$ (float yrange) (float yrange-allowed))
        scalef (//$ 1.0 (float (max xratio yratio)))
        nrmlst-tmp (copylist nrmlst)
        outlst nil)

  (dotimes (i nrmlen)
    (setq npt (pop nrmlst-tmp))
    (if (listp npt)
        (setq xtmp (pop npt)
              ytmp (*$ y-scaling (pop npt)))
        (setq ytmp (*$ npt y-scaling)
              xtmp (float i)))
    (setq newx (+$ xlower (*$ scalef (-$ xtmp xmin)))
          newy (+$ ylower (*$ scalef (-$ ytmp ymin))))
    (push (list newx newy) outlst))

  (return (nreverse outlst)))
```

```
(defun ROTATE-LIST (lst &OPTIONAL (times 1))
```

```
...
... rotate-list: takes the car of a list and puts it at the end
... of the list. If a number is provided, the list is rotated
... the number of times specified. If the number is negative
... the list is rotated backwards.
... (verified 7/29/82)
"
```

```
(prog (tmplst tmp)
  (setq tmplst (copylist lst))
  (if (signp ge times)
      (dotimes (i times)
        (setq tmp (pop tmplst)
              tmplst (nreverse tmplst)
              tmplst (push tmp tmplst)
              tmplst (nreverse tmplst)))
      (dotimes (i (- times))
```

```

      (setq tmpist (nreverse tmp1st)
            tmp (pop tmp1st)
            tmp1st (nreverse tmp1st)
            tmp1st (push tmp tmp1st)))
    (return tmp1st))

```

```

(defun SMOOTH-LIST (alist &OPTIONAL (window-size 5))
  (prog (ll atlst tempsum window sl ws temp tvalue)
    (setq ll (length alist)
          atlst (copylist alist)
          tempsum 0.0
          window nil
          sl nil
          ws (float window-size))
    (dotimes (i (1- window-size))
      (setq temp (pop atlst)
            tempsum (+$ tempsum temp))
      (push temp window))
    (dotimes (i (- ll window-size))
      (setq temp (pop atlst)
            tempsum (+$ tempsum temp)
            tvalue (//$ tempsum ws);
            (push tvalue sl)
            (push temp window)
            (setq window (nreverse window)
                  temp (pop window)
                  tempsum (-$ tempsum temp)
                  window (nreverse window)))
      (setq sl (nreverse sl))
    (return sl)))

```

```

(defun SPLINE-COORDS (x-coeffs y-coeffs tp)
  (let* ((ax (car x-coeffs))
        (bx (cadr x-coeffs))
        (cx (caddr x-coeffs))
        (dx (caddr x-coeffs))
        (ay (car y-coeffs))
        (by (cadr y-coeffs))
        (cy (caddr y-coeffs))
        (dy (caddr y-coeffs)))
    (values (+$ dx (+$ tp (+$ cx (+$ tp (+$ bx (+$ tp ax)))))) ::horner's
            (+$ dy (+$ tp (+$ cy (+$ tp (+$ by (+$ tp ay)))))) ::rule

```

```

(defun SPLINE-CURVATURE (x-coeffs y-coeffs tp)
  (let* ((ax (car x-coeffs))
        (bx (cadr x-coeffs))
        (cx (caddr x-coeffs))
        (ay (car y-coeffs))
        (by (cadr y-coeffs))
        (cy (caddr y-coeffs))
        (n2 (-$ (+$ 6.0 ay bx)(+$ 6.0 ax by)))
        (n1 (-$ (+$ 6.0 ay cx)(+$ 6.0 ax cy)))
        (n0 (-$ (+$ 2.0 by cx)(+$ 2.0 bx cy)))
        (d4 (+$ (+$ 9.0 ay ay)(+$ 9.0 ax ax)))
        (d3 (+$ (+$ 12.0 ay by)(+$ 12.0 ax bx)))
        (d2 (+$ (+$ 6.0 ay cy)(+$ 6.0 ax cx)
                (+$ 4.0 by by)(+$ 4.0 bx bx)))
        (d1 (+$ (+$ 4.0 by cy)(+$ 4.0 bx cx)))
        (d0 (+$ (+$ cy cy)(+$ cx cx)))
        (//$ (+$ (+$ n2 tp tp)(+$ n1 tp) n0)
              (^ (sqrt (+$ (+$ d4 tp tp tp tp)(+$ d3 tp tp tp)
                          (+$ d2 tp tp)(+$ d1 tp) d0)) 3.00))))

```

```

(defun SPLINE-CURVATURE-EXTREMA (x-coefficient-list y-coefficient-list

```



```

&OPTIONAL (steps 50)(closed nil)
&AUX (xc (copylist x-coefficient-list))
      (yc (copylist y-coefficient-list))
      (spline-curvature-array :: i know its wasteful
        (make-array (* 2 (length xc)) :TYPE 'ART-Q))
      curvature-list
      (x-coeff-arr
        (make-array (* 2 (length xc)) :TYPE 'ART-Q))
      (y-coeff-arr
        (make-array (* 2 (length xc)) :TYPE 'ART-Q))
      : (angle-array
        : (make-array (length xc) :TYPE 'ART-FLOAT))
        tangent-x-0 tangent-y-0
        tangent-x-1 tangent-y-1 tmp tmp-crv-1st)

(setq angle-array (make-array (* 2 (length xc)) :TYPE 'ART-FLOAT))
(setq offset-array (make-array (* 2 (length xc)) :TYPE 'ART-8B)
  num-zeroes 0)
(dotimes (i (length xc))
  (multiple-value (tangent-x-0 tangent-y-0)
    (spline-unit-tangent (car xc)(car yc) 0))
  (multiple-value (tangent-x-1 tangent-y-1)
    (spline-unit-tangent (car xc)(car yc) 0.9999))
  (setq inflection (spline-zeroes (car xc)(car yc))) ::: 0 < inflection < 1 if there's an inflection
  (aset (car xc) x-coeff-arr (+ i num-zeroes))
  (aset (car yc) y-coeff-arr (+ i num-zeroes))
  (multiple-value (nil nil nil nil curvature-list)
    (spline-curvature-extrema-between-knots (pop xc)(pop yc) steps))
  (cond ((< inflection 0.0) ::: no inflections in spline segment
    (aset curvature-list spline-curvature-array (+ i num-zeroes))
    (aset (*$ 57.29577866
      (atan2 (-$ (*$ tangent-x-0 tangent-y-1) ::: sin of angle
        (*$ tangent-x-1 tangent-y-0))
      (+$ (*$ tangent-x-0 tangent-x-1) ::: cos of angle
        (*$ tangent-y-0 tangent-y-1))))
      angle-array (+ i num-zeroes))
    (aset 0 offset-array (+ i num-zeroes)))
    (t ::: spline segment has inflections
    (aset (setq tmp (fixr (*$ 50.0 inflection))) offset-array (+ 1 i num-zeroes))
    (aset 0 offset-array (+ i num-zeroes))
    (aset (aref x-coeff-arr (+ i num-zeroes)) x-coeff-arr (+ i 1 num-zeroes))
    (aset (aref y-coeff-arr (+ i num-zeroes)) y-coeff-arr (+ i 1 num-zeroes))
    (multiple-value (tan-x tan-y)
      (spline-unit-tangent (aref x-coeff-arr (+ i num-zeroes))
        (aref y-coeff-arr (+ i num-zeroes)) inflection))
    (aset (*$ 57.29577866
      (atan2 (-$ (*$ tangent-x-0 tan-y)
        (*$ tan-x tangent-y-0))
      (+$ (*$ tangent-x-0 tan-x)
        (*$ tangent-y-0 tan-y))))
      angle-array (+ i num-zeroes))
    (aset (*$ 57.29577866
      (atan2 (-$ (*$ tan-x tangent-y-1)
        (*$ tangent-x-1 tan-y))
      (+$ (*$ tangent-x-1 tan-x)
        (*$ tangent-y-1 tan-y))))
      angle-array (+ 1 i num-zeroes))
    (setq tmp-crv-1st nil)
    (dotimes (j tmp)
      (push (pop curvature-list) tmp-crv-1st))
    (aset (nreverse tmp-crv-1st) spline-curvature-array (+ i num-zeroes))
    (aset curvature-list spline-curvature-array (+ 1 i num-zeroes))
    (setq num-zeroes (1+ num-zeroes))))))
(cond (closed ::: if curve is closed allow wrap-around at ends of list
  (aset (aref angle-array 0)
    angle-array
    (setq tmp (+ (length x-coefficient-list) num-zeroes)))
  (aset (aref angle-array 1) angle-array (+ tmp 1))
  (aset (aref spline-curvature-array 0) spline-curvature-array tmp)
  (aset (aref spline-curvature-array 1) spline-curvature-array (+ tmp 1))

```

```

(aset (aref x-coeff-arr 0) x-coeff-arr tmp)
(aset (aref x-coeff-arr 1) x-coeff-arr (+ tmp 1))
(aset (aref y-coeff-arr 0) y-coeff-arr tmp)
(aset (aref y-coeff-arr 1) y-coeff-arr (+ tmp 1))
(setq num-zeroes (+ 2 num-zeroes)))

(loop FOR i FROM 1 TO (- (+ num-zeroes (length x-coefficient-list)) 2)
      WHEN
      (and (or (and (> (*$ (aref angle-array i)
                          (aref angle-array (1- i)))) 0.0)
             (> (abs (aref angle-array i))
                 (abs (aref angle-array (1- i)))))
           (< (*$ (aref angle-array i)
                (aref angle-array (1- i))) 0.0))
           (or (and (> (*$ (aref angle-array i)
                          (aref angle-array (1+ i)))) 0.0)
                (> (abs (aref angle-array i))
                    (abs (aref angle-array (1+ i)))))
                (< (*$ (aref angle-array i)
                     (aref angle-array (1+ i))) 0.0))
              (> (abs (aref angle-array i)) 15.0))

      COLLECT
      (let* ((abs-curv-1st (mapcar 'abs (aref spline-curvature-array i)))
             (max-curv (apply 'max abs-curv-1st))
             (max-index (find-position-in-list-equal max-curv abs-curv-1st))
             (param (//$ (float max-index) (float steps))))
            (list (spline-curvature (aref x-coeff-arr i)(aref y-coeff-arr i)
                                     param)
                  (multiple-value-list
                   (spline-coords (aref x-coeff-arr i)(aref y-coeff-arr i)
                                   param))
                  (aref x-coeff-arr i)(aref y-coeff-arr i)
                  i))
      INTO
      extrema-list
      FINALLY
      (return extrema-list)))

(defun SPLINE-CURVATURE-EXTREMA-BETWEEN-KNOTS (x-coefficient-list y-coefficient-list
                                              &OPTIONAL (steps 50))
  "
  ...
  ... spline-curvature-extrema-between-knots: takes two lists, each of length 4, one for x b-spline coefficients
  ... and the other for y b-spline coefficients. It returns a value between 0 and 50 (resolution
  ... can be changed optionally by the user) giving the extremum of curvature of the spline
  ... or nil if there is no extremum other than the parameter boundaries.
  ...
  ...
  (prog (ax bx cx ay by cy n2 n1 n0 d4 d3 d2 d1 d0
        curvature-max curvature-min max-index min-index
        curvature-list step-size tp)
        (setq ax (car x-coefficient-list)
              bx (cadr x-coefficient-list)
              cx (caddr x-coefficient-list)
              ay (car y-coefficient-list)
              by (cadr y-coefficient-list)
              cy (caddr y-coefficient-list)
              n2 (-$ (*$ 6.0 ay bx)(*$ 6.0 ax by))
              n1 (-$ (*$ 6.0 ay cx)(*$ 6.0 ax cy))
              n0 (-$ (*$ 2.0 by cx)(*$ 2.0 bx cy))
              d4 (+$ (*$ 9.0 ay ay)(*$ 9.0 ax ax))
              d3 (+$ (*$ 12.0 ay by)(*$ 12.0 ax bx))
              d2 (+$ (*$ 6.0 ay cy)(*$ 6.0 ax cx)
                  (*$ 4.0 by by)(*$ 4.0 bx bx))
              d1 (+$ (*$ 4.0 by cy)(*$ 4.0 bx cx))
              d0 (+$ (*$ cy cy)(*$ cx cx))

```

```

    curvature-list nil
    step-size (// $ 1.0 steps))
  (dotimes (i steps)
    (setq tp (*$ (float i) step-size))
    (push (// $ (+$ (*$ n2 tp tp)(*$ n1 tp) n0)
      (^ (sqrt (+$ (*$ d4 tp tp tp tp)(*$ d3 tp tp tp)
        (*$ d2 tp tp)(*$ d1 tp) d0)) 3.00))
      curvature-list))
  (setq curvature-list (nreverse curvature-list)
    curvature-max (apply 'max curvature-list)
    max-index (find-position-in-list-equal curvature-max curvature-list)
    curvature-min (apply 'min curvature-list)
    min-index (find-position-in-list-equal curvature-min curvature-list))
  (return max-index curvature-max min-index curvature-min curvature-list)))

```

```
(defun SPLINE-PTS-TO-COEFFICIENTS (control-list)
```

```

  "
  ...
  ... spline-pts-to-coefficients: takes a list of x,y control points of length
  ... at least four and multiple value returns two lists. The first is a list
  ... of coefficients for the x coordinate of the b-spline. Each member of the
  ... list is itself a list of four coefficients for the powers of the spline
  ... parameter. The second list is a list of coefficients for the y coordinate
  ... of the b-spline.
  ...
  "

```

```

  (prog (ctl cp1 cp2 cp3 xcoeff-list ycoeff-list cp4 xcoeffs ycoeffs)
    :: make sure there are at least four points for spline
    (if (< (length control-list) 4)
      (ferror nil "need at least four points for b-splines"))
    (setq ctl (copylist control-list)
      cp1 (pop ctl)
      cp2 (pop ctl)
      cp3 (pop ctl)
      xcoeff-list nil
      ycoeff-list nil)
    (dotimes (i (length ctl))
      (setq cp4 (pop ctl)
        xcoeffs (b-spline-coeffs (car cp1)(car cp2)(car cp3)(car cp4))
        ycoeffs (b-spline-coeffs (cadr cp1)(cadr cp2)(cadr cp3)(cadr cp4))
        cp1 cp2
        cp2 cp3
        cp3 cp4)
      (push xcoeffs xcoeff-list)
      (push ycoeffs ycoeff-list))
    (return (nreverse xcoeff-list) (nreverse ycoeff-list)))

```

```
(defun SPLINE-UNIT-TANGENT (xcoefficients ycoefficients
  &OPTIONAL (parameter-value 0.0 parameter-supplied)
  &AUX tangent-x tangent-y tan-magnitude)
```

```

  "
  ... spline-unit-tangent: takes two lists of length four. One list contains the coefficients for
  ... the x component of a cubic b-spline segment, the other contains the coefficients for the y
  ... component. Optionally one can specify a parameter value between 0 and 1 inclusive.
  ... The unit tangent to the spline at the parameter value is returned. Multiple value return
  ... is used for the x and y components of the tangent.
  "

```

```

  (if parameter-supplied
    (if (or (< parameter-value 0.0)(> parameter-value 1.0))
      (ferror nil "parameter must be between 0 and for for spline-unit-tangent")))
  (setq tangent-x (+$ (caddr xcoefficients)
    (*$ 2.0 (cadr xcoefficients) parameter-value)
    (*$ 3.0 (car xcoefficients) parameter-value parameter-value))
    tangent-y (+$ (caddr ycoefficients)
    (*$ 2.0 (cadr ycoefficients) parameter-value)
    (*$ 3.0 (car ycoefficients) parameter-value parameter-value))
    tan-magnitude (sqrt (+$ (*$ tangent-x tangent-x)

```

```

                                (*$ tangent-y tangent-y)))
(values (//$ tangent-x tan-magnitude)
        (//$ tangent-y tan-magnitude)))

(defun SPLINE-ZEROES (xcoefficients ycoefficients)
  (prog (a b c d param p1 p2)
    (setq a (*$ 3.0
              (-$ (*$ (car ycoefficients)(cadr xcoefficients))
                   (*$ (car xcoefficients)(cadr ycoefficients)))))
    b (*$ 3.0
       (-$ (*$ (car ycoefficients)(caddr xcoefficients))
            (*$ (car xcoefficients)(caddr ycoefficients))))
    c (-$ (*$ (caddr xcoefficients)(cadr ycoefficients))
         (*$ (caddr ycoefficients)(cadr xcoefficients)))
    d (-$ (*$ b b)(*$ 4.0 a c))
    (cond ((or (= a 0.0)(< d 0.0))
           (setq param -1.0))
          (t
           (setq p1 (//$ (-$ (sqrt d) b) (*$ 2.0 a))
                 p2 (//$ (-$ 0.0 b (sqrt d)) (*$ 2.0 a))
                 param -1.0)
           (if (and (< p1 1.0)(> p1 0.0))
               (setq param p1))
           (if (and (< p2 1.0)(> p2 0.0))
               (setq param p2))))
    (return param)))

```

```
;-*- Mode:LISP ; Fonts:CPTFONT,TR101,CPTFONTB,TR10IB,HL10; Base:10-*-
```

```
(defconst RED 1)
(defconst GREEN 2)
(defconst BLUE 3)
(defconst YELLOW 4)
(defconst MAGENTA 5)
(defconst CYAN 6)
(defconst WHITE 7)
(defconst PINK 8)
(defconst PASTEL-BLUE 9)
(defconst ORANGE 10)
(defconst BLACK 11)

(defunp COLOR-CLEAR ()
  (funcall color:color-screen ':clear-screen))

(declare (special phi cp sp ip theta ct st rho x y z jp yp xold yold))
(defunp CSRF-COLOR (&optional (xrot 30) (xyscale 12.0) (zscale 35.0) (xcent 288) (ycent 180))
  (funcall color:color-screen ':clear-screen)
  (setq phi (*$ xrot 0.0174533)
        cp (cos phi)
        sp (sin phi))
  (dotimes (i 36)
    (setq ip (+$ 4.0 (*$ (float i) 10.0))
          theta (*$ ip 0.0174533)
          ct (cos theta)
          st (sin theta)
          xold 0
          yold 0)
    (dotimes (j 900)
      (setq rho (*$ (float j) 0.0174533)
            x (*$ rho ct xyscale)
            y (*$ rho st xyscale)
            z (*$ (cos rho) zscale)
            yp (-$ (+$ (*$ y cp) (*$ z sp))))
      (color:color-draw-line (fix (+$ xold xcent))
                            (fix (+$ yold ycent))
                            (fix (+$ xcent x))(fix (+$ ycent yp)) green)

      (setq xold x
            yold yp))
    (dotimes (i 3)
      (setq ip (+$ 180.0 (*$ (float i) 360.0))
            rho (*$ ip 0.0174533)
            xold 0
            yold 0)
      (dotimes (j 144)
        (setq jp (+$ 4.0 (*$ (float j) 2.5))
              theta (*$ jp 0.0174533)
              ct (cos theta)
              st (sin theta)
              x (*$ rho ct xyscale)
              y (*$ rho st xyscale)
              z (*$ (cos rho) zscale)
              yp (-$ (+$ (*$ y cp) (*$ z sp))))
        (if (signp e j)
            (+ i 1)
            (if (oddp (fix j))
                (color:color-draw-line (fix (+$ xold xcent))(fix (+$ yold ycent))
                                       (fix (+$ xcent x))(fix (+$ ycent yp)) yellow)
                (color:color-draw-line (fix (+$ xold xcent))(fix (+$ yold ycent))
                                       (fix (+$ xcent x))(fix (+$ ycent yp)) black)))

        (setq xold x
              yold yp))
      (color:color-princ "Cosine surface - Dotted lines are minima" 100 400 orange))

(defunp BLUE-BACKGROUND (&optional (brightness 80))
  (color:write-color-map 0 0 0 brightness))

(defunp RED-BACKGROUND (&optional (brightness 50))
```

```

(color:write-color-map 0 brightness 0 0))

(defunp GREEN-BACKGROUND (&optional (brightness 40))
  (color:write-color-map 0 0 brightness 0))

(defunp YELLOW-BACKGROUND (&optional (brightness 30))
  (color:write-color-map 0 brightness brightness 0))

(defunp CANDY-COLOR-MAP ()
  (blue-background)
  (color:write-color-map 1 255 0 0)           ;red
  (color:write-color-map 2 0 255 0)         ;green
  (color:write-color-map 3 0 0 255)         ;blue
  (color:write-color-map 4 255 255 0)       ;yellow
  (color:write-color-map 5 255 0 255)       ;magenta
  (color:write-color-map 6 0 255 255)       ;blue-green (cyan)
  (color:write-color-map 7 255 255 255)     ;white
  (color:write-color-map 8 200 90 150)      ;pink
  (color:write-color-map 9 50 200 200)      ;pastel blue
  (color:write-color-map 10 200 50 0)       ;orange
  (color:write-color-map 11 0 0 0)         ;black
)

(defun DISP-LIST-COLOR (dsplst &OPTIONAL (list-color green) (x-flg nil)
  (x-pt (fixr (// $ (float (length dsplst)) 2.0)))
  (x-color red))
  "
  ...
  ...
  ... disp-list: displays a list of points on the lisp machine
  ... joined by line segments. The list of points is a list of
  ... lists, each of the form (x y). 7/30/82 (optionally the user
  ... can specify that an x be drawn through one of the points.
  ...
  ...
  (prog (dsplst-tmp pt xold yold lstlen xval yval)
    (setq dsplst-tmp (copylist dsplst)
          pt (pop dsplst-tmp)
          lstlen (length dsplst-tmp)
          xold (fixr (pop pt))
          yold (fixr (pop pt)))
    (dotimes (m lstlen)
      (setq pt (pop dsplst-tmp))
      (setq xval (fixr (pop pt))
            yval (fixr (pop pt)))
      (if x-flg
          (if (signp e (- x-pt (1+ m)))
              (draw-x-color xval yval x-color)))
          (color:color-draw-line xold yold xval yval list-color))
      (setq xold xval yold yval))
    (return lstlen)))

(defun NORM-LIST-COLOR (nrmlst &OPTIONAL (y-scaling 1.0)
  (xlower 115.0)(xupper 475.0)(ylower 0.0)(yupper 300.0))
  "
  ...
  ...
  ... norm-list: scales a list of points to fit between the upper
  ... and lower values specified
  ... one can give either a list where each member of the list is itself
  ... a list of the form (x y), or one can give a list of y values where
  ... x values are implicitly to go from 0 to the length of the list
  ... 7/30/82
  ...
  ...
  (prog (nrmlst-tmp xmax xmin ytmp ymax ymin nrmlen npt xtmp xrange yrange
    xrange-allowed yrange-allowed xratio yratio scalef outlst
    newx newy)

```

```

(setq nrmlst-tmp (copylist nrmlst)
  xmax -999999.0
  xmin 9999999.0
  ymax xmax
  ymin xmin
  nrmlen (length nrmlst))

(dotimes (i nrmlen)
  (setq npt (pop nrmlst-tmp))
  (if (listp npt)
    (setq xtmp (pop npt)
          ytmp (*$ y-scaling (pop npt)))
    (setq ytmp (*$ npt y-scaling)
          xtmp (float i)))
  (setq xmax (max xmax xtmp)
        xmin (min xmin xtmp)
        ymax (max ymax ytmp)
        ymin (min ymin ytmp)))

(setq xrange (-$ xmax xmin)
      yrange (-$ ymax ymin)
      xrange-allowed (-$ xupper xlower)
      yrange-allowed (-$ yupper ylower)
      xratio (//$ (float xrange) (float xrange-allowed))
      yratio (//$ (float yrange) (float yrange-allowed))
      scalef (//$ 1.0 (float (max xratio yratio)))
      nrmlst-tmp (copylist nrmlst)
      outlst nil)

(dotimes (i nrmlen)
  (setq npt (pop nrmlst-tmp))
  (if (listp npt)
    (setq xtmp (pop npt)
          ytmp (*$ y-scaling (pop npt)))
    (setq ytmp (*$ npt y-scaling)
          xtmp (float i)))
  (setq newx (+$ xlower (*$ scalef (-$ xtmp xmin)))
        newy (+$ ylower (*$ scalef (-$ ytmp ymin))))
  (push (list newx (-$ 360.0 newy)) outlst))

(return (nreverse outlst)))

(defun DRAW-X-COLOR (xval yval &OPTIONAL (x-color red) (x-size 8.))
  "
  ...
  ... draw-x-color: draws an x through the specified point
  ...
  "
  (color:color-draw-line (fixr (+ xval x-size))(fixr (+ yval x-size))
                        (fixr (- xval x-size))(fixr (- yval x-size))
                        x-color)
  (color:color-draw-line (fixr (- xval x-size))(fixr (+ yval x-size))
                        (fixr (+ xval x-size))(fixr (- yval x-size))
                        x-color))

(defun GRAPH-SPLINE-CURVATURE-COLOR (x-coefficient-list y-coefficient-list
                                     &OPTIONAL (spline-color pink))
  (prog (spline-curvature-list xc yc curvature-list)
    (setq spline-curvature-list nil
          xc (copylist x-coefficient-list)
          yc (copylist y-coefficient-list))
    (dotimes (i (length x-coefficient-list))
      (multiple-value (nil nil nil nil curvature-list)
        (spline-curvature-extrema-between-knots (pop xc)(pop yc)))
      (setq spline-curvature-list (append spline-curvature-list curvature-list)))
    (disp-list-color (norm-list-color spline-curvature-list 3000) spline-color)
    (return t spline-curvature-list)))

(defun DISPLAY-SPLINE-COLOR (xcoefficient-list ycoefficient-list

```

&OPTIONAL (spline-color pink)
(res 50))

...
... display-spline: takes two lists, one of coefficients of powers for the x coordinate
... and the other of coefficients of powers for the y coordinate of a b-spline, and displays
... the spline on the lisp machine screen
...
...

```
(prog (num-coeffs xc yc resolution-scaling new-x-coeffs new-y-coeffs tmp
      ax bx cx dx ay by cy dy tp x-coord y-coord)
  (setq num-coeffs (length xcoefficient-list)
        xc (copylist xcoefficient-list)
        yc (copylist ycoefficient-list)
        resolution-scaling (//$ 1.0 res))

  (dotimes (i num-coeffs)
    (setq new-x-coeffs (pop xc)
          new-y-coeffs (pop yc)
          ax (car new-x-coeffs)
          bx (cadr new-x-coeffs)
          cx (caddr new-x-coeffs)
          dx (caddr new-x-coeffs)
          ay (car new-y-coeffs)
          by (cadr new-y-coeffs)
          cy (caddr new-y-coeffs)
          dy (caddr new-y-coeffs))
    (dotimes (j res)
      (setq tp (*$ (float j) resolution-scaling)
            x-coord (+$ dx (*$ tp (+$ cx (*$ tp (+$ bx (*$ tp ax)))))) ;;horner's
            y-coord (+$ dy (*$ tp (+$ cy (*$ tp (+$ by (*$ tp ay)))))) ;;rule
            (if (and (= i 0)(= j 0))
                (setq xold x-coord yold y-coord)
                (setq tmp (color:color-draw-line (fixr xold)(fixr yold)
                                                (fixr x-coord)(fixr y-coord) spline-color)
                      xold x-coord yold y-coord))))
      (return num-coeffs)))
```

```
(defun A-DEMO-COLOR (&OPTIONAL (closed nil))
  (prog (:tln natural-tangents-list t1 s1 knot-list-tmp equal-spaced-knot-list
        knot-pt pt-coords tangent-angle knot-x pt-coords-list knot-y codon-string
        x-coefficient-list y-coefficient-list unequal-spaced-knot-list
        codon-list extrema-list)
    )
  (if closed
      (setq tln (norm-list-color (setq t1 (dig-curv-list-closed))))
      (setq tln (norm-list-color (setq t1 (dig-curv-list 200 0.035)) 3)))
  (color-clear)
  (candy-color-map)
  (disp-list-color tln)
  (color:color-princ "ORIGINAL QUANTIZED CURVE" 175 375 orange)
  (print "type a space to continue")
  (tyi)
  (setq natural-tangents-list nil
        pt-coords-list nil)
  (if closed
      (dotimes (i (length tln))
        (multiple-value (nil nil) (compute-tangent-variances tln i closed))
        (multiple-value (tangent-angle pt-coords) (nat-tans))
        (push tangent-angle natural-tangents-list)
        (push pt-coords pt-coords-list))
      (do i 15 (1+ i) (= i (- (length tln) 15)) ;; don't do ends of open curve
        (multiple-value (nil nil) (compute-tangent-variances tln i closed))
        (multiple-value (tangent-angle pt-coords) (nat-tans))
        (push tangent-angle natural-tangents-list)
        (push pt-coords pt-coords-list)))
  (setq natural-tangents-list (nreverse natural-tangents-list)
        pt-coords-list (nreverse pt-coords-list))
  (setq s1 (smooth-list natural-tangents-list 6))
```



```

(multiple-value (unequal-spaced-knot-list equal-spaced-knot-list)
 (make-knots s1 pt-coords-list 0 closed))
(setq knot-list-tmp (copylist equal-spaced-knot-list))
(color-clear)
(disp-list-color tln)
(dotimes (i (length equal-spaced-knot-list))
 (setq knot-pt (pop knot-list-tmp)
        knot-x (car knot-pt)
        knot-y (cadr knot-pt))
 (draw-x-color knot-x knot-y))
(color:color-princ "EQUALLY SPACED KNOTS AT NATURAL SCALE" 105 375 orange)
(breep)
(print "type a space to see the spline")
(terpri)
(print-then-space "type a c to clear screen first")
(if (= (tyi) 99)
 (color-clear))
(multiple-value (x-coefficient-list y-coefficient-list)
 (spline-pts-to-coefficients equal-spaced-knot-list))
(display-spline-color x-coefficient-list y-coefficient-list)
(color:color-princ "NATURAL SCALE B-SPLINE" 150 375 orange)
(dreep)
(print "type a space to see the spline curvature")
(tyi)
(color-clear)
(graph-spline-curvature-color x-coefficient-list y-coefficient-list)
(setq extrema-list
 (spline-curvature-extrema x-coefficient-list y-coefficient-list 50 closed))
(color:color-princ "B-SPLINE CURVATURE" 220 375 orange)
(nreep)
(print "type a space to see the extrema")
(tyi)
(color-clear)
(display-spline-color x-coefficient-list y-coefficient-list)
(dolist (item extrema-list)
 (draw-x-color (car (cadr item))(cadr (cadr item))))
(multiple-value (codon-string codon-list)
 (codon-top-level-description-color extrema-list closed))
(print "codon string is")
(print codon-string)
(color:color-princ "CURVATURE EXTREMA AND CODON DESCRIPTION" 140 375 orange)
(breep)
(print "type a space to see the natural tangents plot")
(tyi)
(color-clear)
(disp-list-color (norm-list-color natural-tangents-list 100))
(color:color-princ "UNSMOOTHED NATURAL TANGENTS PLOT" 120 375 orange)
(beep-rise)
(print "type a space to see the smoothed natural tangents plot")
(tyi)
(color-clear)
(disp-list-color (norm-list-color s1 100))
(color:color-princ "SMOOTHED NATURAL TANGENTS PLOT" 115 375 orange)
(speob)
(return x-coefficient-list y-coefficient-list s1 natural-tangents-list)))

(defun A-DBL-DEMO-COLOR (&OPTIONAL (closed nil))
 (prog ( ;tln natural-tangents-list t1 s1 knot-list-tmp equal-spaced-knot-list
        ; knot-pt pt-coords tangent-angle knot-x pt-coords-list knot-y codon-string
        ; x-coefficient-list y-coefficient-list unequal-spaced-knot-list
        ; codon-list extrema-list)
 )
 (if closed
 (setq tln (norm-list-color (setq c1 (dig-curv-list-closed 500.))))
 (setq tln (norm-list-color (setq t1 (dig-curv-list 200 0.035)) 3)))
 (color-clear)
 (candy-color-map)
 (disp-list-color tln)
 (color:color-princ "ORIGINAL QUANTIZED CURVE" 175 375 orange)
 (print "type a space to continue")

```

```

(tyi)
(setq natural-tangents-list nil
  pt-coords-list nil
  nat-tan-1st-2 nil)
(if closed
  (dotimes (i (length tln))
    (multiple-value (nil nil) (compute-tangent-variances tln i closed))
    (multiple-value (nil pt-coords tan-ang-1st nil tan-pts-1st) (nat-tans))
    (push (car tan-ang-1st) natural-tangents-list)
    (if (> (length tan-ang-1st) 1)
      (push (cadr tan-ang-1st) nat-tan-1st-2)
      (push (car tan-ang-1st) nat-tan-1st-2))
    (push tangent-angle natural-tangents-list)
    (push pt-coords pt-coords-list))
  (do i 15 (1+ i) (= i (- (length tln) 15)) ::: don't do ends of open curve
    (multiple-value (nil nil) (compute-tangent-variances tln i closed))
    (multiple-value (tangent-angle pt-coords) (nat-tans))
    (push tangent-angle natural-tangents-list)
    (push pt-coords pt-coords-list)))
(setq natural-tangents-list (nreverse natural-tangents-list)
  pt-coords-list (nreverse pt-coords-list)
  nat-tan-1st-2 (nreverse nat-tan-1st-2))
(setq s1 (smooth-list natural-tangents-list 6)
  s12 (smooth-list nat-tan-1st-2 6))

;
;

(dotimes (p 2)
  (multiple-value (unequal-spaced-knot-list equal-spaced-knot-list)
    (make-knots s1 pt-coords-list 0 closed))
  (setq knot-list-tmp (copylist equal-spaced-knot-list))
  (color-clear)
  (disp-list-color tln)
  (dotimes (i (length equal-spaced-knot-list))
    (setq knot-pt (pop knot-list-tmp)
      knot-x (car knot-pt)
      knot-y (cadr knot-pt))
    (draw-x-color knot-x knot-y))
  (color:color-princ "EQUALLY SPACED KNOTS AT NATURAL SCALE" 105 375 orange)
  (breep)
  (print "type a space to see the spline")
  (terpri)
  (prin1-then-space "type a c to clear screen first")
  (if (= (tyi) 99)
    (color-clear))
  (multiple-value (x-coefficient-list y-coefficient-list)
    (spline-pts-to-coefficients equal-spaced-knot-list))
  (display-spline-color x-coefficient-list y-coefficient-list)
  (color:color-princ "NATURAL SCALE B-SPLINE" 150 375 orange)
  (dreep)
  (print "type a space to see the spline curvature").
  (tyi)
  (color-clear)
  (graph-spline-curvature-color x-coefficient-list y-coefficient-list)
  (setq extrema-list
    (spline-curvature-extrema x-coefficient-list y-coefficient-list 50 closed))
  (color:color-princ "B-SPLINE CURVATURE" 220 375 orange)
  (nreep)
  (print "type a space to see the extrema")
  (tyi)
  (color-clear)
  (display-spline-color x-coefficient-list y-coefficient-list)
  (dolist (item extrema-list)
    (draw-x-color (car (cadr item))(cadr (cadr item))))
  (multiple-value (codon-string codon-list)
    (codon-top-level-description-color extrema-list closed))
  (print "codon string is")
  (print codon-string)
  (color:color-princ "CURVATURE EXTREMA AND CODON DESCRIPTION" 140 375 orange)
  (breep)
  (print "type a space to see the natural tangents plot")

```

```

(tyi)
(color-clear)
(disp-list-color (norm-list-color natural-tangents-list 100))
(color:color-princ "UNSMOOTHED NATURAL TANGENTS PLOT" 120 375 orange)
(beep-rise)
(print "type a space to see the smoothed natural tangents plot")
(tyi)
(color-clear)
(disp-list-color (norm-list-color s1 100))
(color:color-princ "SMOOTHED NATURAL TANGENTS PLOT" 115 375 orange)
(spoob)
:
:
(setq s1 s12))
(return x-coefficient-list y-coefficient-list s1 natural-tangents-list))

```

```

(defun CODON-TOP-LEVEL-DESCRIPTION-COLOR (curvature-extrema-list
&OPTIONAL (closed nil)
&AUX
(cel curvature-extrema-list)
(state 0) tmp (codon-list nil)
(new-codon nil) (codon-string " "))

(if (not codon-transition-table-initialized)
(codon-transition-table-init))

(cond (closed :: allow wraparound for closed curves
(setq cel (append cel (list (car cel))))
(if (>= (car (car cel)) 0) :: if first extremum is >0 then we need to append two extrema
(setq cel (append cel (list (cadr cel)))))))

(dolist (current-extremum cel)
(if (>= (car current-extremum) 0)
(setq new-codon (aref codon-transition-table state 0)
state (aref codon-transition-table state 1))
(setq new-codon (aref codon-transition-table state 2)
state (aref codon-transition-table state 3)))
(if (not (null new-codon))
(setq codon-list (append codon-list (list new-codon))
tmp (color:color-princ (aref codon-string-table new-codon)
(fixr (+$ (car (cadr current-extremum)) 9.0))
(fixr (+$ (cadr (cadr current-extremum)) 9.0))
yellow))))

(dolist (new-codon codon-list)
(setq codon-string (string-append codon-string
(aref codon-string-table new-codon)))
(values codon-string codon-list))

```

```

(defun DIG-CURV-LIST-CLOSED (&OPTIONAL (steps 300.) (scaling 60.0))

```

```

:
:
: dig-curv-list creates a list of points on a digitized analytic
: function. The list members are lists of the form (x y).
: 7/29/82
:

```

```

(prog (curv-list x arg y-new size)
(setq curv-list nil
size (//$ 6.31 steps))

(dotimes (j steps)
(setq arg (*$ j size)
x (*$ 2.0 scaling (cos arg))
y-new (*$ scaling (+$ (sin arg)
(*$ 0.04 (sin (*$ 25.0 arg))))))
(push (list x y-new) curv-list))
(return (nreverse curv-list)))

```

```

(defun TV-DEMO (pt-index closed)
  (prog ()
    (color-clear)
    (candy-color-map)
    (if closed
      (setq tln (norm-list-color (setq tl (dig-curv-list-closed 600.))))
      (setq tln (norm-list-color (setq tl (dig-curv-list 200 0.035)) 2)))
    (disp-list-color tln green t pt-index red)
    (color:color-princ "NATURAL SCALE TANGENTS ABOUT THE X" 140 375 yellow)
    (multiple-value (nil nil) (compute-tangent-variances tln pt-index closed))
    (multiple-value (tan-ang tan-pos tan-ang-list tan-len-list end-pts-list)
      (nat-tans))
    (setq epl (copylist end-pts-list))
    (dotimes (i (length end-pts-list))
      (setq pt (pop epl))
      xo (car (car pt))
      yo (cadr (car pt))
      x1 (car (cadr pt))
      y1 (cadr (cadr pt)))
      (color:color-draw-line (fixr xo)(fixr yo)(fixr x1)(fixr y1) (+ i 5)))
    (disp-list-color (norm-list-color tvlist 2.5 115.0 475.0 200.0 300.0))
    (return (*$ tan-ang 57.29578) tan-pos tan-ang-list tan-len-list end-pts-list)))

```