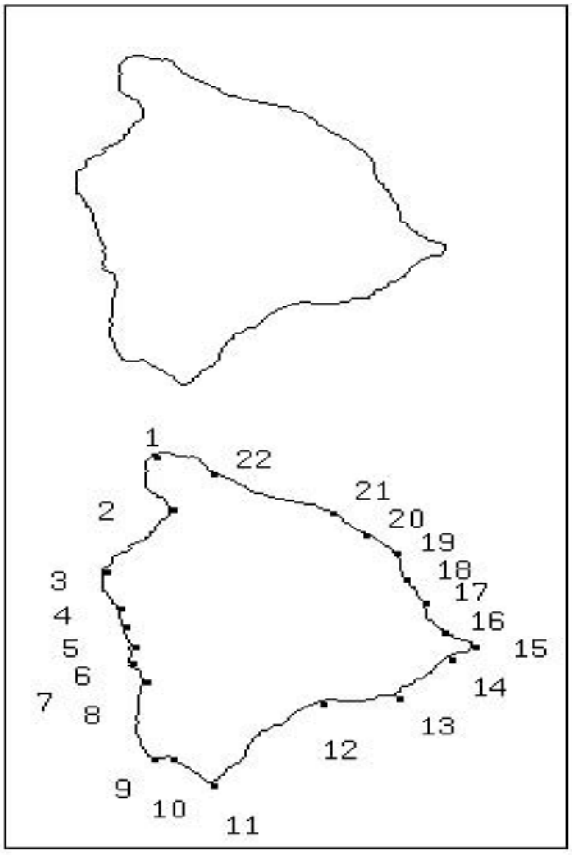
Aarhus University
Mathematical Institute
Computer Science Department
Ny Munkegade
8000 Aarhus C

Process Grammar and Process History for 2D Objects

(Daimi PB 586)



Done by:
Thomas W. Larsen, Aarhus University, 1991
Translated by Professor Brian Mayoh, 2008

Foreword

(This report is a translation of DAIMI IR-115 written by Thomas W. Larsen in Danish)
This project is the written report for the course in Picture Processing as part of the computer science master's degree at the University of Aarhus. The starting point for my project is an article of Michael Leyton in Artificial Intelligence 34 1988 "A process grammar for shape" [1]. The article describes how it is possible to derive the process history for an object from its state at two stages in its development. The aim in this project is to describe and test an algorithm for deriving the process history of an object from its state at two different stages. First I give a short summary of Leyton's article and describe his method. After this there is a description of an implemented algorithm and a system that incorporates it. The system extracts the information for the algorithm in an interactive environment. All my ideas (good or bad) have been implemented and tested. All parts of my testing are programmed in C and the user interface is built in Sun-View (Sun's window system). The images used for testing (not all of which are in this report) are scanned in a Macintosh and then ftp-ed to a Sun.

I will freely use terms from image processing and computer graphics without defining and explaining them. As this is a course project some parts and peripheral topics will be treated superficially. Some will be described, some named and some omitted. A few concepts that are directly connected to the problems or their solutions will be treated. The implementation and testing of the methods will take up a major part of the report.

Finally I would like to thank my adviser Brian Mayoh who gave me the idea for this project and always had time to answer questions and give criticism.

Thomas W. Larsen, 1991

Index

1 Background

Interest in knowing and understanding the conditions, mechanisms and rules that control events has always been great. It is not enough to know that an event happens; we also want to know why and preferably when.

Earthquakes are facts. We often observe earthquakes, we investigate what causes them, we generalize and set up rules for how an earthquake develops and we try to predict earthquakes from these rules. Plants that grow are also a development process we are interested in. Islands that appear in the oceans are a third example. We look at cloud formations to predict weather and much else.
For all these events we are interested in explaining conditions and development. The rules we develop from our observations describe explain how processes develop and the conditions explain when.

There can be situations where several of the rules we have derived can be used (ambiguity). An ambiguity does not necessarily lead to an incorrect end result, but to use one rule rather than another can describe a less likely history.
Therefore it may be necessary to choose between rules. These choices are based on the knowledge and information one has in the given situation.

We will look at some of this in what follows, using Leyton's process grammar as our starting point.

1.1 "A Process Grammar for Shape"

Leyton presents his theory in two sections. The first is concerned with the derivation of the process history for a single object (i.e. which processes have influenced this object through time?). This derivation uses two rules. The other section looks at how it is possible to connect two successive stages of an object with a process history (i.e. which processes have played a role in the intervening period?). This is done by introducing a "process grammar". The use of the word "object" refers to "some object described by a simple, closed, planar, curve".

1.1.1 Process history for a single object

Extremes of curvature of an object play a central role in the derivation of the process history of the object. The transition between curvature extremities and processes is given by axes of symmetry. Leyton defines these using differentials.
As shown in figure 1.1 the line cI_2 reflects the tangent in point "a" to the tangent in point "b". By pushing the circle along the two sides of a curve (that represents a section of the object) and maintaining contact in two points, one can define different types of differential symmetry axes as a trace of some kind of middle point of the circle. The Symmetry Axis Transform, SAT, defines the symmetry axes as the trace of the center of the circle. Smooth Local Symmetry, SLS, defines the symmetry axes as the trace of I_1while Leyton defines the symmetry axes as the trace of I_2. Leyton calls this form for symmetry Process Inferring Symmetry Analysis, PISA.

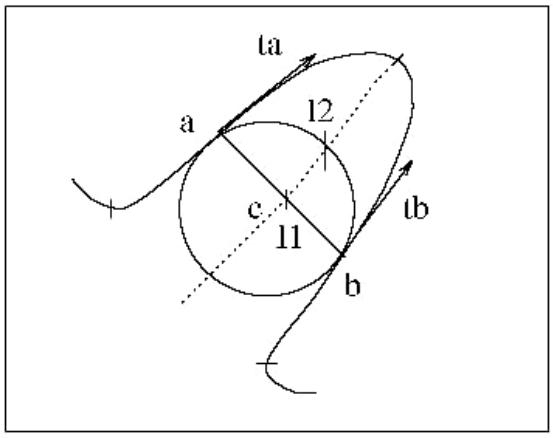


Fig. 1.1: Illustration of axes of symmetry

For any of these symmetries it is shown in one of Leyton's articles [2] that a curvature extremity, maximum or minimum, lying on a curve segment between two extremities of the other type forces a uniquely determined symmetry axis that ends in that curvature extremity. Leyton calls this theorem the **Symmetry-Curvature-Duality** theorem. Figure 1.1 illustrates also this.

The connection between symmetry axes and processes gives what Leyton calls the **Interaction Principle**: Symmetry axes for an object are understood as the directions where processes probably have worked or will work. Preconditions for this principle are discussed in other articles of Leyton[3].
By combining these two rules, **Symmetry-Curvature-Duality** and the **Interaction Principle**, we get the rule: a curvature extremity determines a process whose trace is given by the unique symmetry axis that is produced by and ends in that extremity.

On the curves we look at there can be four types of curvature extremities. A plot of curvature as a function of curve length for the curve in figure 1.2 is shown in figure 1.3. Here the four types can be seen. **M** denotes a local maximum, **m** denotes a local minimum while + and – show whether they are positive or negative. Figure 1.4 shows a curve with its curvature extremities and the directions of the corresponding symmetry axes. This is called a process diagram. One can group these process diagrams according to the number of curvature extremities along the curve. If one compares process diagrams with the way one classifies processes, one sees that a purely syntactic (structural) characterization of curvature extrema gets a semantic meaning from the process that they indicate have been working:

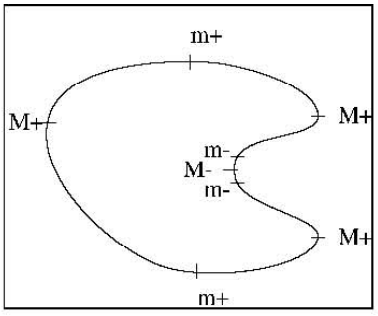


Fig. 1.2 Curve with the four extremum types

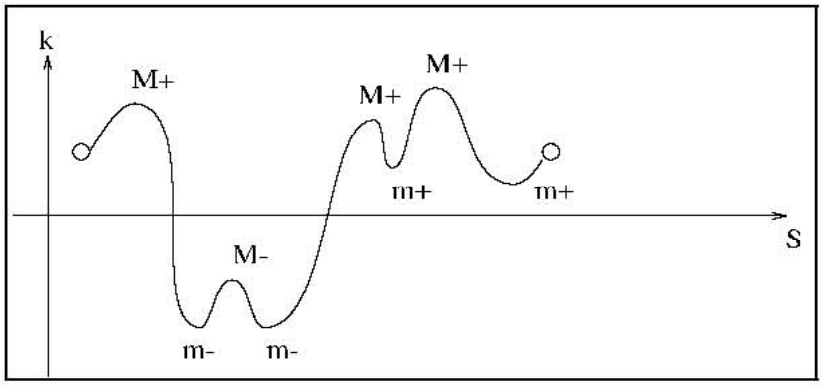


Fig. 1.3 *k(s)* for the curve

M+: Bulging out
M-: Inner resistance
m+: Compression
m-: Sink

1.1.2 Intermediate process history

Given two objects as representatives of two successive stages of an object, it is possible to describe what has happened in the period between the stages. The later stage is explained in terms of the earlier stage.
The evolution of processes can be divided into two groups:

- Continuous
- Bifurcation.

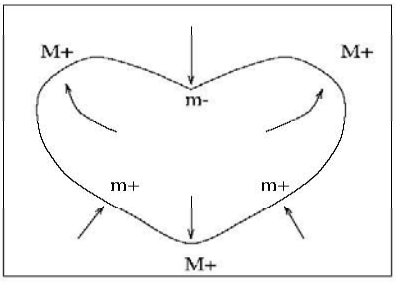


Fig. 1.4 Process diagram for an object

Since the process traces in a given process diagram end in an extremum, the problem reduces to looking at the four types of extremes and what happens to them with the different process developments. A continuous process is called **Cm** and a bifurcation process is called **Bm**, where **m** is one of the four extremity types.

A bifurcation of a process can be considered as a bifurcation of the associated extremity. This result in an extremity of the opposite type is introduced between these. The continuation of a process can be considered as a (continued) push on the curve in the direction of the process, so no new extremity is introduced. Initially we have the following rules:

C1: **Cm+ : m+ -> m-**

C2: **Cm- : m- -> m-**

C3: **CM- : M- -> M+**

C4: **CM+ : M+ -> M+**

B1: **BM+ : M+ -> M+m+M+**

B2: **BM+ : M+ -> M+m-M+**

B3: **BM- : M- -> M-m-M-**

B4: **Bm- : m- -> m-M-m-**

B5: **Bm- : m- -> m-M+m-**

B6: **Bm+ : m+ -> m+M+m+**

We note that for purely mathematical reasons we can not have $\mathbf{Bm}^+ : \mathbf{m}^+ \rightarrow \mathbf{m}^+\mathbf{m}\,\mathbf{m}^+$ where **m** is negative and $\mathbf{BM}^- : \mathbf{M}^- \rightarrow \mathbf{M}^-\mathbf{m}\,\mathbf{M}^-$ where **m** is positive. The above rules can be simplified. From a structural viewpoint C2 and C4 are identities. This is also in agreement with the semantic interpretation: a bulge continues to bulge and a sink continues to sink. Using rule B2 can be considered as using rule B1 then C1. Similarly using rule B5 can be considered as using B4 then C3. Thus we are left with six rules: C1, C3, B1, B3, B4, B6.

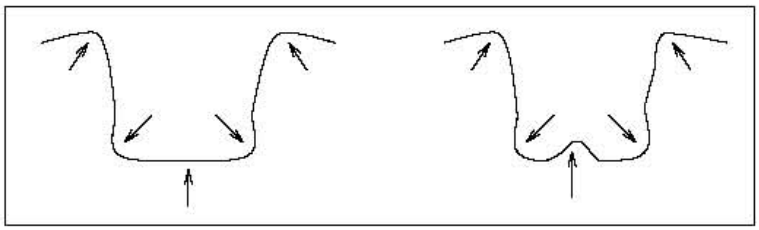


Fig. 1.5 Illustration of $\mathbf{CM}^-$

An example of the continuous development of a process is shown in figure 1.5. Initially we have $\mathbf{M}^-$ at the bottom of a sink.
The development of the process presses the curve up until it bulges. $\mathbf{M}^-$ is replaced by $\mathbf{M}^+$ at the top of the bulge. The semantic interpretation of this is that the inner resistance increases until it causes the bulge.
In figure 1.6 we see an example of a bifurcation development of a process. We have a process that ends in $\mathbf{M}^+$.
If this process bifurcates there will come processes on its left and right. The endpoints for these processes are still $\mathbf{M}^+$. A minimum is introduced (a mathematical consequence) between the two $\mathbf{M}^+$. The semantic interpretation of this could be that unevenness becomes a sink.

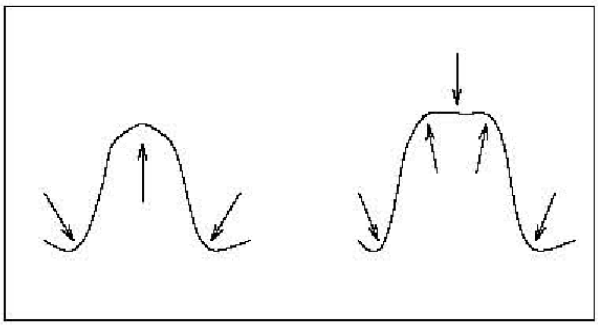


Fig. 1.6 Illustration of $\mathbf{BM}^+$

It is now possible to generate all possible process developments with the six rules. In other words we can explain the connection between two arbitrary process diagrams by repeated use of the rules for process development.

As mentioned earlier the number of curvature extremes gives a partition of objects into classes or levels. The first class consists of objects with four curvature extremities; the next class consists of objects with six curvature extremities, the next of objects with eight etc.
The result of using a continuous process rule on an object is an object in the same class. The result of using a bifurcation process rule on an object is an object in the next class. Thus there is a highly ordered structure between these levels, when we connect objects with all possible applications of the grammar rules. From this structure it is apparent that there can be ambiguous derivations of a given process history. Leyton says that the grammatical operations commute. Figure 1.7 shows an example.

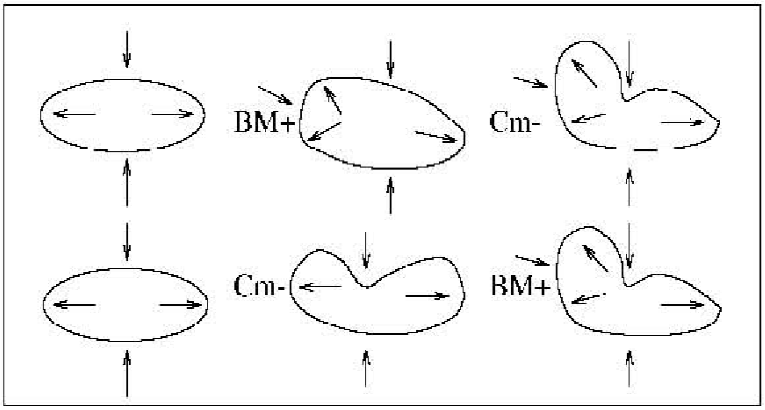


Fig. 1.7 Illustration of commuting operations in the grammar

To alleviate this Leyton suggests a heuristic: size-is-time, the younger a process the less remarkable it is.

This says that younger processes have worked in less time and have had smaller effects.

2 Image Processing and Process Grammar

In this chapter we look at different properties of objects and symmetry axes. We look at what information one can use to derive a process history for an object. Also we characterize a process grammar.

2.1 Information and Image Processing

In image processing and analysis it is important that one can extract information from a given scene[1]. It often suffices to resolve the picture into components (objects). Sometimes one wants to know about the individual components in the scene. This knowledge is information about the components form, color and position in the scene.

There are many techniques, each with their aim. Techniques that extract information about the form of a component can be described as information preserving or not information preserving. If it is possible to reconstruct an object from the extracted description it is information preserving.

When certain information or a description based on this information is extracted from an object, one usually "transforms" the object and uses this (more compact) form in the future. The choice of a technique depends on which information one wants to use in the future analysis or which conclusions one wants to reach.

An example could be to decide how many objects are in a scene. Here it is enough to determine the number of connected components. If one also wants to decide if the objects represent previously known objects, one needs to analyze further. One needs to describe the unknown objects in the scene in the same format as the description of the known objects (e.g. Chain encoding, graph of the skeleton), and then try to match using this information.

2.1.1 Information

The methods used in "recognition" can be divided in two categories. The first type looks at the global structure of the object and tries to divide it into simple structures. The second type looks at the local properties of the object along its contour. Most methods of the first type are directed towards inner properties (e.g. contour filling, thinning), while methods of the second type are directed towards outer properties (e.g. contour tracing, polygon approximation). The choice of method depends on which properties one wants to determine the presence of or which structural information one should use. The aims of methods are different but there are similarities. In what follows I will look at the actual problem and what information is required by Leyton's theory.
Abstractly described our goal is to explain the connection between two subjects (or two states of the same object) that is to give a semantic explanation of what we see. There are (luckily) some limits to this. We have chosen to limit the object domain to consist of objects that (or whose projection in one or another direction) can be described by a simple closed and plane curve. We are also limited by the connection we want to derive in that objects are described by syntactic elements (with a semantic explanation) and a finite set of rules (with a semantic explanation) is used on these descriptions.
Leyton claims that from the process diagrams for two objects one can determine the intervening process history. One can by "reducing" the information for the two objects to the description of their process diagrams form the information needed to derive the process history.
The process diagram for an object consists of the curvature extremes for the curves that represent the object and the associated symmetry axes. From the **Symmetry-Curvature-Duality** theorem we know that the curvature extremes "generate" the symmetry axes. This implies that the symmetry axes can be derived from the curve and its curvature extremes. Our first step is thus to transform the implicit representation of an object (in the form of an image) to an explicit representation of a curve. As discussed in chapter 3.2 there a different ways of representing a curve. The representation of a curve is based on the outline of an object. We can now derive the two components that comprise a process diagram.

Curvature

From [11] we know that the curvature for a $\mathbf{R}^2$ curve is given by

$$k(t) = (X'(t)\,Y''(t) - X''(t)\,Y'(t)) / (X'(t)^2 + Y'(t)^2)^{3/2}$$

In other words the curvature at any point can be determined, independently of the other points on the curve if the curve is given in the form a(t) = (x(t) , y(t)) . When the outline of an object is not smooth, the use of this rule assumes that the curve is represented by a parametrised

approximation. Whatever the chosen curve representation (almost) one can calculate an approximation to the curvature. In these calculations the curvature at a point depends on a segment of the curve on both sides of the point. Various methods are discussed in chapter 3.3.
To determine the local extremes along a curve we do not need to calculate the absolute curvature at every point. It is enough to be able to decide if a point is a local maximum or local minimum (i.e. it is enough with a function k' such that $k'(t1) > k'(t2) \Leftrightarrow k(t1) > k(t2)$.
Using this function we can determine the position and type of every extreme along a curve.

Symmetry axes and skeletonising

When we have a curve and the positions of the curvature extremes we can find the symmetry axes. The procedure was described in chapter 1.1.1. The symmetry axes are the traces of a midpoint of a circle that touches the curve twice. The **Symmetry-Curvature-Duality** theorem tells us that each curvature extreme determines a symmetry axis and the nature of the process ensures that some of these axes are external to the curve. We know that the circles that define the symmetry axes are the same for the different methods (PISA, SAT, SLS). However Leyton shows that only PISA gives symmetry axes for the four types of extrema that matches theory.
For the position of symmetry axes to agree with the semantic explanation we use both internal and external circles on the curve segments that determine symmetry axes.
The only case when all three methods give axes that agree with the theory is for curve segments with local positive maxima (i.e. axes generated by inner circles.
We represent objects by plane closed curves and the object is the set of points enclosed by the curves. A point in such a set is a skeleton or multiple point if the point has more than one nearest neighbor on the curve.
The set of multiple points is the skeleton (Medial Axis Transform [15]) of the original set. Thus the skeleton is a proper subset of the object. From the skeleton and the distances to the nearest neighbors one can reconstruct the original figure. The connection between the skeleton and the outline of an object is: if one has the outline one can find the skeleton and vice-versa.
We see that if a point P is multiple, then P is the centre of a circle with maximal radius completely within the set. This also means that P is the point generated by SAT and it is on the symmetry axis. We remember that symmetry axes are defined using differentials, but when we use inner circles this coincides with the definition of multiple points (Note: The difference between a symmetry axis and skeleton "branch" is that in the end point for the skeleton one continues with minimize the radius (SAT) while the circle touches the extremum. This way one makes the axis exactly terminate in the extremum).

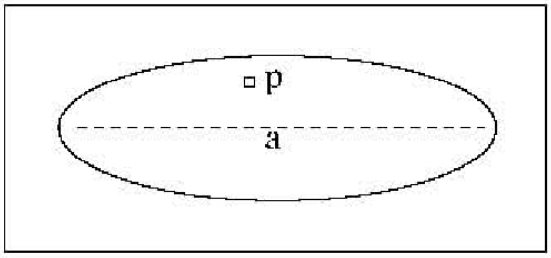


Fig. 2.1: No point outside a can be multiple

If we look at a local negative maximum or a local positive minimum on a curve, we cannot find the symmetry axes by thinning (skeletonising), which ends in an extremity (or more correctly in the centre of the curvature circle of the point). The reason is that no point p outside axis a – see figure 2.1- can be a multiple point. This shows why it is not enough to use inner circles to determine symmetry axes.

If we now consider "circumscribing" circles, we see that both SAT and SLS fail. The reason for SAT can be seen in figure 2.2. Circle C centre c is the intersection of the normals n_a and n_b to the tangents t_a and t_b at a and b [16] and c determines the symmetry axis. This implies that the axes are neither finite nor connected.

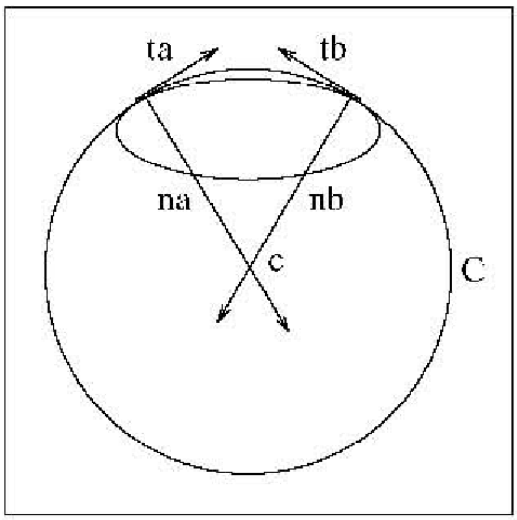


Fig. 2.2: SAT fails with circumscribed circles

For SLS the symmetry axes are determined by the midpoint P of the line between a and b.See figure 2.3. P is center of line "l" between "a" and "b". Line "l" has the same angle to t_a and t_b. We have determined a interior symmetry axis, contradicting our semantic interpretation of the extremum: An exterior force.

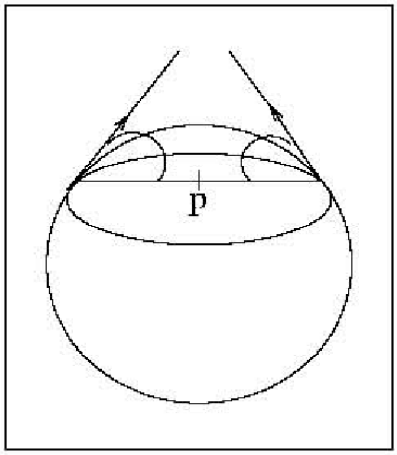


Fig. 2.3: SLS fails with circumscribed circles

By choosing the midpoint of the arc between a and b PISA avoids these situations: with a circumscribing circle forces the symmetry axis outside the curve and this midpoint is the point on the circumscribing circle that is closest to the extremity-see figure 2.4.

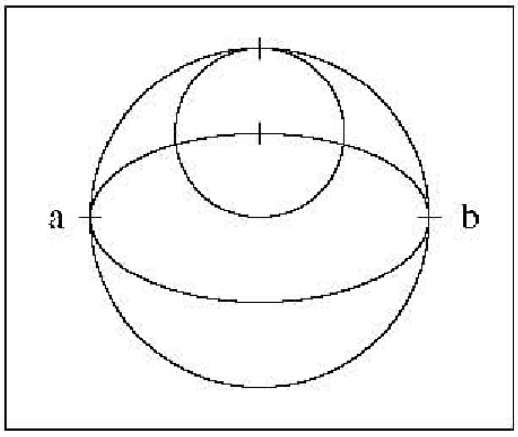


Fig. 2.4: PISA chooses the midpoint between a and b

For me the symmetry axes for an object are more "fundamental" than its skeleton. When one has the skeleton and a radius for each of its points, one can reconstruct the figure. One can say that we can grow it from the inside. A figure whose outline is without discontinuities, i.e. smooth, can be scaled by scaling radii up to a certain interval without changing the structural properties of the figure. One can say that there is a form of equivalence connected with this. One can not do this for the symmetry axes of an object. This is because the axes terminate at the extremes, not at points distant from them by the curvature radius, so they indicate a discontinuity for the reconstruction. This means that it is impossible to scale the figure.

In conclusion it should be said that I have not implemented the derivation of the symmetry axes. In chapter 3 there is a description of methods to compute curvatures.

2.2 Process Grammars

We consider objects as represented by plane curves. We let the alphabet Σ be given by $\Sigma = \{\mathbf{m}^+, \mathbf{M}^+, \mathbf{m}^-, \mathbf{M}^-\}$. By a Characteristic String for a curve C we mean a string $S_c \in \Sigma^*$ such that S_c is a listing of the curvature extremes one meets starting somewhere on the curve C and moving along it in a predetermined direction. It does not matter where on C one starts.

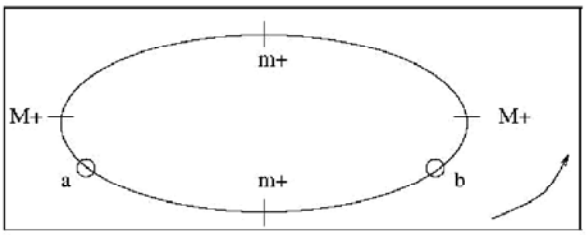


Fig. 2.5: Determining the characteristic string

Not all strings in Σ^* are characteristic strings e.g. $\mathbf{M}^+ \mathbf{m}^-$, while several strings can characterize the same curve. If we look at figure 2.5 we get $\mathbf{m}^+ \mathbf{M}^+ \mathbf{m}^+ \mathbf{M}^+$ by starting at point a and $\mathbf{M}^+ \mathbf{m}^+ \mathbf{M}^+ \mathbf{m}^+$ by starting at point b. Both strings characterize the curve, so the characteristic string is not uniquely defined. We therefore define a "rotational" equivalence on Σ.

2.2.1 Redundancy in the Grammar

Using the rotational equivalence from the last section gives equivalence classes that agree with Leyton's partition of curves into levels according to how many curvature extremities they have. (Implicitly Leyton has used this rotational equivalence).
The goal now is to derive from an object at two different stages in its development, what has happened in the intervening period. We want to deduce a discrete process history for an object. In order to use the operations in a grammar it is enough to know the characteristic strings for the curves that represent the object at two different stages in its development. At this time the absolute curvature has no influence on our understanding of the curve; only the type of the extremities and their relative locations is significant.
We can define a context-free grammar that generates all characteristic strings. We assume terminal symbols for the nonterminals that symbolize the four extremum types. The set of non-terminals is $\mathbf{V = \{ S, B, M^+, m^+, M^-, m^- \}}$ and the rules are:

```
1. S  -> M+BM+B | e
2. B  -> m-  | m+
3. m- -> m-  | m-M-m-
4. m+ -> m-  | m+M+m+
5. M+ -> M+  | M+m+M+
6. M- -> M+  | M-m-M-
```

Rule 3 corresponds to process grammar operations $\mathbf{Cm^-}$ and $\mathbf{Bm^-}$, rule 4 to $\mathbf{Cm^+}$ and $\mathbf{Bm^+}$, rule 5 to $\mathbf{CM^+}$ and $\mathbf{BM^+}$, and rule 6 to $\mathbf{CM^-}$ and $\mathbf{BM^-}$.
If we now let two strings S_1 and S_2 represent two stages in the development of an object, it is clear that the derivation $\mathbf{S_1 =>^* S_2}$ is not always unique. These ambiguities arise because of redundancies in the grammar and the rules can commute. To illustrate redundancy consider an occurrence of $\mathbf{m^-M^-m^-}$ in a string $\mathbf{S}$.
Consider the occurrence $\mathbf{m^-M^-m^-}$ isolated from the rest of the string. One can use $\mathbf{Bm^-: m^-M^-m^- -> m^-M^-m^-M^-m^-}$. Instead one can use $\mathbf{BM^-: m^-M^-m^- -> m^-M^-m^-M^-m^-}$.
One cannot decide if $\mathbf{Bm^-}$ or $\mathbf{BM^-}$has been used in such a derivation. The reason for this is that we describe a discrete development of an object: we see the object before and after a bifurcation without regard to the extremity size or other information such as the size of the extrema or the symmetry axes. Thus we cannot decide which of the two rules should be used. The same is true for $\mathbf{Bm^+}$ or $\mathbf{BM^+}$, when it is possible to simulate $\mathbf{Bm^+}$ by $\mathbf{BM^+}$.

One could avoid the redundancy of the grammar by reducing the number of the rules. However this cuts off the possibility of distinguishing between developments where a process bifurcates because of inner resistance and those where a process bifurcates because of compression. All these are now considered as a bifurcation of a process representing an exterior force, and a bifurcation of a process representing interior force (bulge). In chapter 3.4 we describe an algorithm that derives a process history, using characteristic strings as its only source of information, which is based on an implicit reduction of the grammar rules. The algorithm cannot be used to derive the real process history; it can only answer the question of whether one object can become another object in time. Thus a reduction of rules means that not all process developments can be described satisfactorily.

Redundancy arises because there are situations where we cannot decide which of the possible rules is used (i.e. which is the most likely to be used), while commutativity of rules is because we have no time aspects (i.e. which rule is used first). We can illustrate the latter by:

M+m+M+m+ -> M+m+m+Cm+ ->

M+m+M+m- -> M+m+BM+m- ->

M+m+M+m+M+m-

Using the two rules in the opposite order gives the same result; this is shown in figure 1.7.

3 Implementation of methods

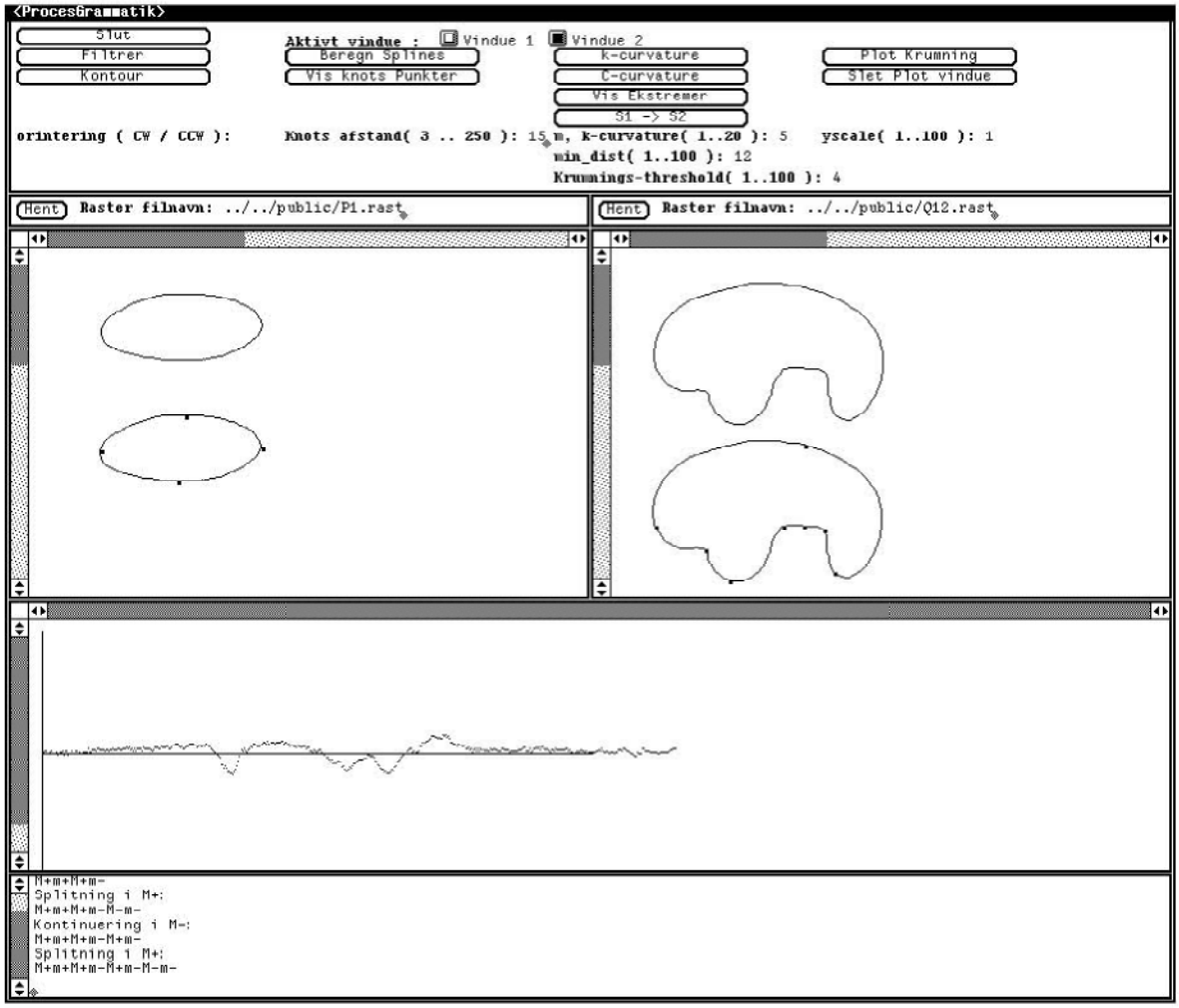


Fig 3.1 User interface for the program

For testing the different algorithms I have built a window-based system. The objects, used by the algorithms are represented by binary raster images. There is some preprocessing of these images – filtering, derivation and transformation of information in the images- so that the algorithms can work satisfactorily with them. The system makes it possible to visualize the results of the algorithms and to give parameters interactively. The tools used in this project are not chosen after philosophical considerations. I had two requirements: The programs should be fast and be programmed flexibly, and there should be a graphic interface that gave a consistent interface to the programming language. Therefore the programs were implemented in C with interface to

SunView and tested on Sun workstations[1]. I will briefly describe the methods I used for this and the information I derived from the images. In figure 3.1 shows the interface for the program. The way the program works is shown in figure 3.2. This shows the steps.

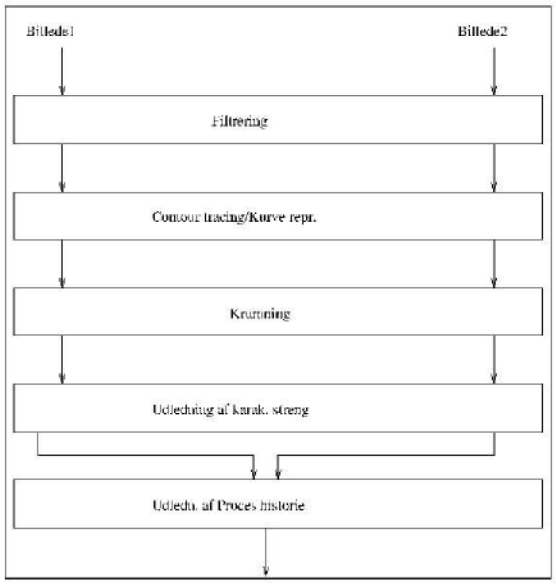


Fig.3.2 Program components

Filtering is part of the image processing since pictures often have “noise”. Filtering can be omitted. Contour tracing derives the necessary information from an object for further work. Then we compute the curvature along the curve. Based on the curvature we derive the characteristic string for the curve. From the characteristic strings from object1 and object2 we can now decide whether object1 can or cannot develop into object2. A more detailed description of the separate steps will be given now.

3.1 Filtering

The purpose of filtering is to remove salt, pepper noise from the image. I have tried different methods, all based on 3x3 matrices [17]. Filtering is not a central part of the program and it will not be further discussed.

3.2 Contour tracing and Curve representation

To find the variation in the curve that represents the object we must thin it. By thinning we get an approximation to a simple plane closed curve with the topological properties that such curves have (e.g. zero area) and at the same time keep the topological properties from the original curve (e.g. connectedness).

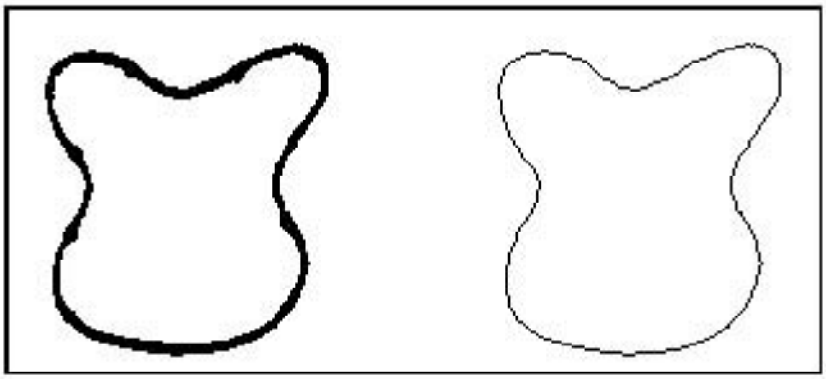


Fig. 3.3: a) Original image b) contour of a)

For thinning I have used a contour tracing algorithm taken from [6]. An example can be seen in figure 3.3. In all its simplicity the algorithm is:

1. Choose either clockwise or anticlockwise
2. Find a point on the contour (left-right top-bottom scan)
3. For each point find a transition from white to black and make it the new actual point
4. Repeat 3 until we are home again.

By running along the boundary of an image in this way, one can gather a Chain code or a list of (x,y)-pairs for the curve. I chose the latter in what follows. Another and very elegant method to find the contours of image components (if there are many) is described in [8]. The method has two parts. In the first part inner points are removed until only contour points are left. The second part is basically a contour tracing algorithm but it can eliminate noise and small divergences from the contour by remembering k previous points. Every gathered point is deleted so any remaining points in the image belong to contours that are not yet traced. However I used the algorithm from [6], partly because it is quick and partly because my images are almost without noise and divergences. Also I only need one contour from my images.

The result of contour tracing is a list of (x,y)-pairs. This gives an explicit representation of a curve. There are advantages and disadvantages with this. One immediately has y = f(x) for all values of x and it is simple to redraw the curve anywhere. On the other hand one has no smoothing, if there is much noise along the curve and some desirable mathematical properties are missing i.e. scalability, rotation and smoothness (C^n), see [9,10,20].
As I want to derive curvatures I was interested in finding a general expression for the curvature along a curve, see chapter 3.3. By using cubic B-splines we can reduce the number of data points needed for a parametric approximation to a curve with the C^2 property. From [11] we then know that the curvature is given by the length of the double derivative in $\mathbf{R}^3$ and by the equation given in chapter 2.1.1 in $\mathbf{R}^2$. By using cubic uniform B-splines it is possible to compute the curve very efficiently [18,19]. In figure 3.4 we see the spline approximation to the curve in figure 3.3.

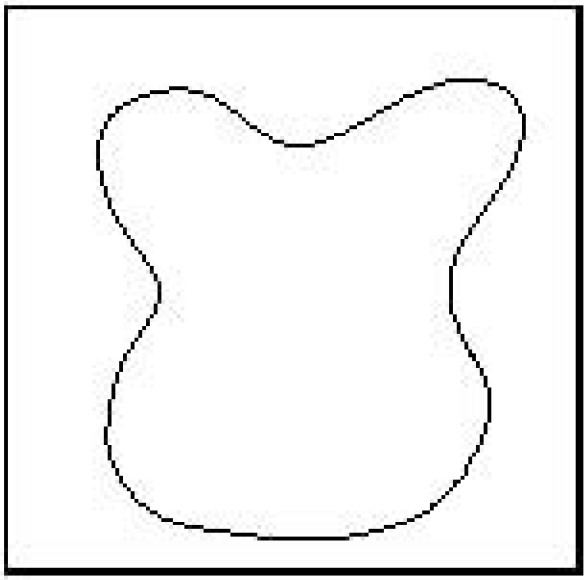


Fig. 3.4: Spline approximation

As mentioned data points must be chosen so the spline approximation can be interpolated i.e the approximation must go through the points. Otherwise one has no control over the spline curve, other than it is within the convex hull of the control points [12,20]. In normal interactive interpolation the control points are given first. This is not so here. In my case the control points are found by Gauss-Seidel iteration, a method taken from [12].

By taking points over intervals of the same length one risks losing information. In [21] a method is described that avoids this. One gathers points that satisfy one of the following criteria:

1. Absolute curvature is greater than a given minimum
2. Curvature is a local minimum or maximum
3. Distance from the last point is greater than the interval length.

The method assumes a calculation of the curvature so it takes longer to find an approximation to the curve.

I have discussed B-splines. There are other representations of curves. Two of these, Hermite curves and Bezier curves, are also cubic parametrisations. Unlike B-splines, both are based on both control points and tangent vectors. A Bezier curve's tangent vector is implicitly defined by four control points. In contrast to cubic B- splines, which have the $C^{(2)}$ property at the endpoints, both Bezier and Hermite curves have only the $C^{(1)}$ property. Furthermore one has to specify two tangent vectors for a Hermite curve. Some graphics programs use these two types of curves, but I have not found use for them.

3.3 Calculating Curvatures

When we have a plane closed curve, we know that the curvature is defined at every point, if the curve is smooth. For a curve in $\mathbf{R}^2$ we have, as mentioned in chapter 2.1.1,

$$k(t) = (X'(t)\,Y''(t) - X''(t)\,Y'(t)) / (X'(t)^2 + Y'(t)^2)^{3/2}$$

provided we have a parameterization of the curve. For a smooth curve one can compute the curvature at a point p as 1/r, where r is the radius of curvature (to be completely exact: r is calculated from the curvature). If one can find the radius of curvature, then one can find the curvature.

When one digitalizes a smooth curve, there is no longer anything that is smooth. One gets a sampled version of the curve, a discrete set of points in $\mathbf{R}^2$. There is no immediate way to use mathematics. There two ways one can go from here. As described in chapter 3.2 one can either derive a parameter representation/Fourier transform, a smoothing, or one can consider the given digitalized curve of (x,y)-pairs as smooth. Either way will give an estimate for the curvature. I have chosen to discuss the second way as the first is more or less given by the parameter representation.

There are many ways of computing curvatures in the discrete plane. There are also many articles on this subject. In this document I have chosen to describe only a few methods that I have tried. The most important is to be clear about when one can use what and how one does.

3.3.1 Calculating Curvatures from point-pairs

Radius of curvature

I have implemented an algorithm that finds an approximation to the circle of curvature for every point along a contour. We want to compute the curvature for the white pixel by the arrow in figure 3.5. Mathematically the curvature is defined by a small region around the point. Here I cannot use this, so I approximate. In the figure one sees to the right and left of the white pixel a pixel with a white edge.

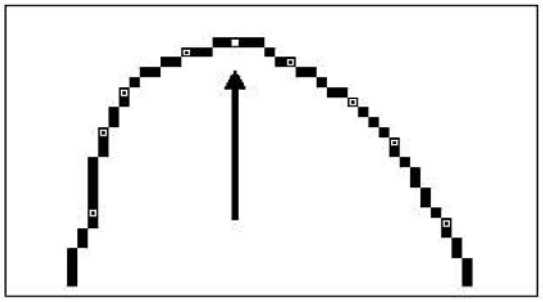


Fig. 3.5 Computing the curvature

From these three points one determines the radius of the circumcircle. I quickly discovered that this method was too uncertain. Those points, that visually had higher curvatures than others, had computed curvatures that were lower than the others because of the slight noise along the curve.
To smooth not the curve but the curvature I weighted the curvatures within a little bigger region around the actual point. In figure 3.5 there are four pixels with a white edge to both left and right (a little unclear). I take the average of the radii of the four circumscribing circles. Maybe one could weight the averages differently. By testing I got the most reliable results by placing the four point-pairs, which with the actual point determine the circles, at distances 6, 12, 16 and 24 pixels from the actual point. A plot of the curvature for the curve in figure 3.3 is shown in figure 3.6. I call this method of computation c-curvature.

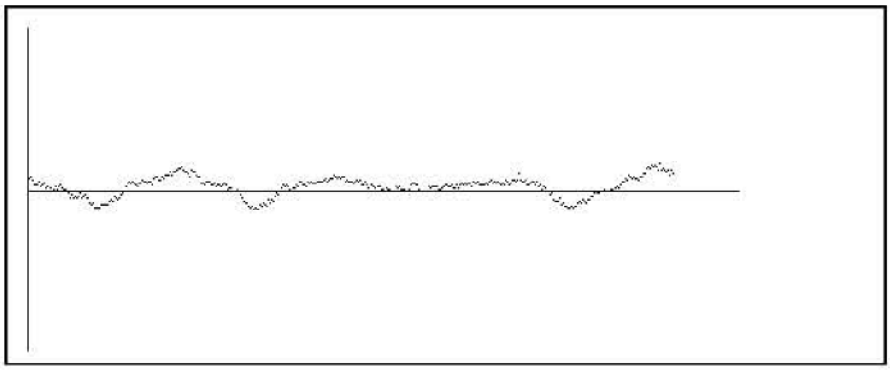


Fig. 3.6 c-curvature

Another method which is similar to ours is described in [24]. It is not for estimating curvatures but an iterative process for estimating a circle arc from a given set of points. The use of this method is limited as the algorithm requires too large arcs to converge quickly enough. Another approximation to the radius of curvature (perhaps more correct) is given in [7].

Angles

The curvature in $\mathbf{R}^2$ is a measure of the change in the tangent along a curve. We will find an expression for the curvature at the point p_i on the curve segment in figure 3.7. By using the angle between a_{ik} and b_{ik} we get a measure for the change in the tangents at the point p_i. We have:

$$cos(v_{ik}) = (a_{ik}b_{ik})/|a_{ik}||b_{ik}|.$$

If we choose p_{i-k} and p_{i+k} as the previous and next points (i.e. k = 1), one has only 8 different angles between a_{ik} and b_{ik}. Thus it is important to choose a k that gives smaller angle changes. How to do this is described in [23]. For every point p_i one computes $cos(v_{ik})$ for $k = 1,\ldots m$, where m is about 1/10 of the curve length. For h where

$cos(v_{im}) < cos(v_{im-1}) < \ldots < cos(v_{ih}) > cos(v_{ih-1})$

We use $cos(v_{ih})$ as the measure for angle change. I call this measure k-curvature.

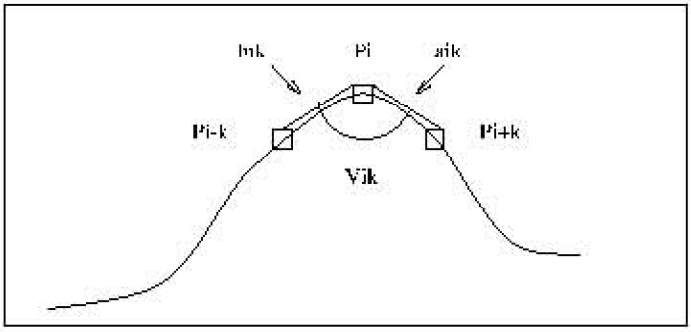


Fig. 3.7 Computing Curvature

In figure 3.8 these values are plotted for the curve in figure 3.3. The top plot is computed with $m = 1/20$ of the curve length and the bottom plot is computed with $m = 1/10$ of the curve length. It is quite clear that k-curvature smoothes the actual curvature. This is particularly clear on the part of the plots that correspond to the long flat part at the bottom of the curve in figure 3.3. If one looks at this curve segment locally it is just flat as in the upper plot, but if one looks at the whole curve segment it is slightly curved as in the lower plot.

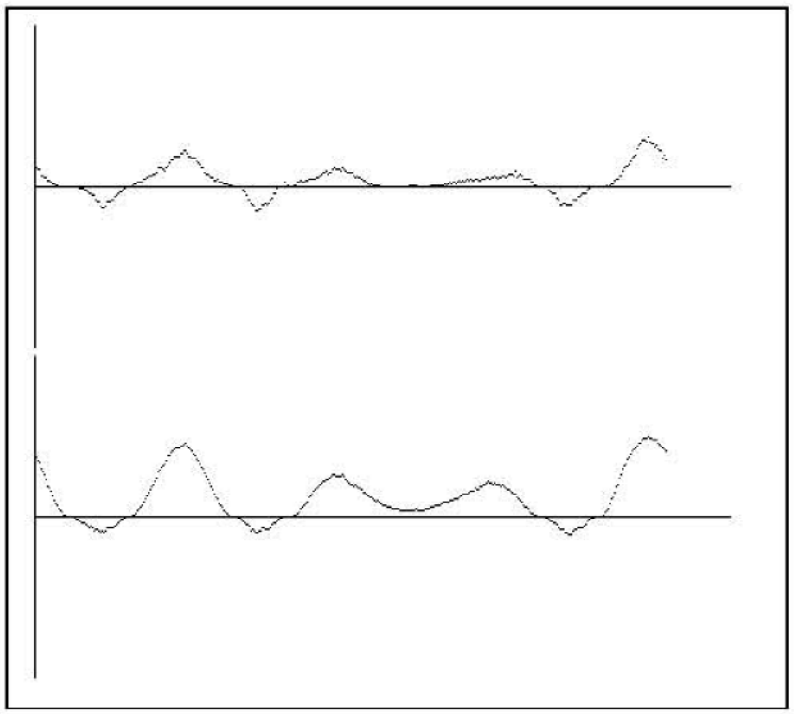


Fig. 3.8 k-curvature

One must say that this method gives a good result. However the method is not sensitive enough to find discontinuities, particularly if there is noise along the contour. More noise, the larger m and more smoothing. Look at the plot in figure 3.10 of figure 3.9.

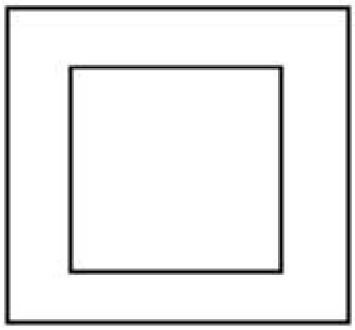


Fig. 3.9 Discontinuities

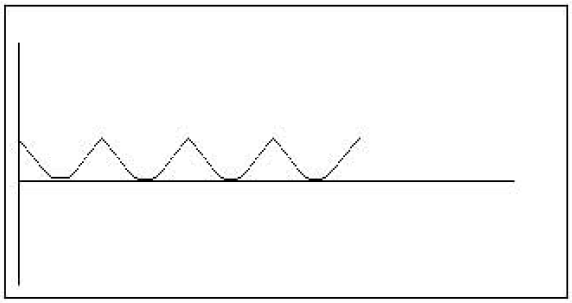


Fig.3.10 Discontinuities become points with high curvature

In this connection we should mention that [4] describes an extension of Leyton's process grammars so they also describe discontinuities, i.e. points with infinite curvature. The extension is also based on the similarity-curvature duality theorem. The only difference between the original symmetry axes and the symmetry axis in a discontinuity is that one no longer has a unique tangent. However the problem is solved by arbitrary choice among the set of tangents between the two half-tangents of the discontinuity. We will not discuss this further but we should mention that [22] discusses very elegant a method of finding discontinuities, based on B-splines, and [8] discusses a method based on tangent differences.

3.4 Methods to reveal process development

In chapter 2.2 we described the characteristic string of a curve and the equivalence class of such a string. The method, which is described now, uses the characteristic string as its only information about a curve. Chapter 3.3 describes how one calculates the curvature along a curve.

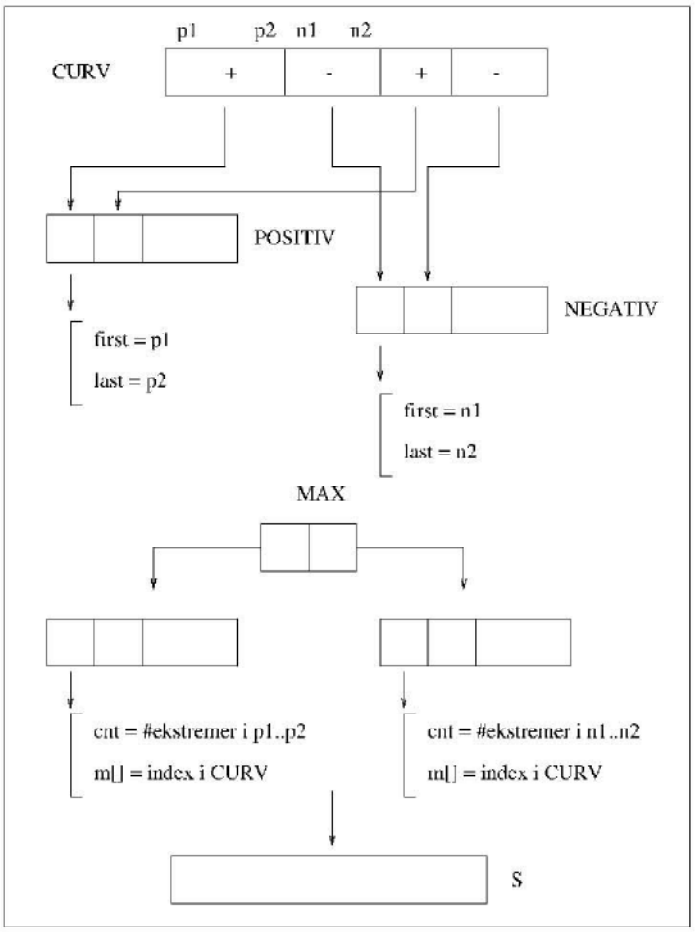


Fig. 3.11 Data structures for deriving the characteristic string

From this list of values the characteristic string is derived. I will not describe in detail how this is done, just give the overall outline. Look at figure 3.11.

- **CURV** holds the curvature along the curve
- from **CURV** we get sequences of positive and negative curvatures. The start and end indices of these are saved in **POSITIV** and **NEGATIV**
- for every sequence in **POSITIV**(**NEGATIV)** the number of extremities and their index in **CURV** is stored in **MAX**[0] (**MAX**[1])
- the information in **MAX**[0] and **MAX**[0] is converted to the symbols $\mathbf{m}^+$, $\mathbf{M}^+$, $\mathbf{m}^-$, $\mathbf{M}^-$.

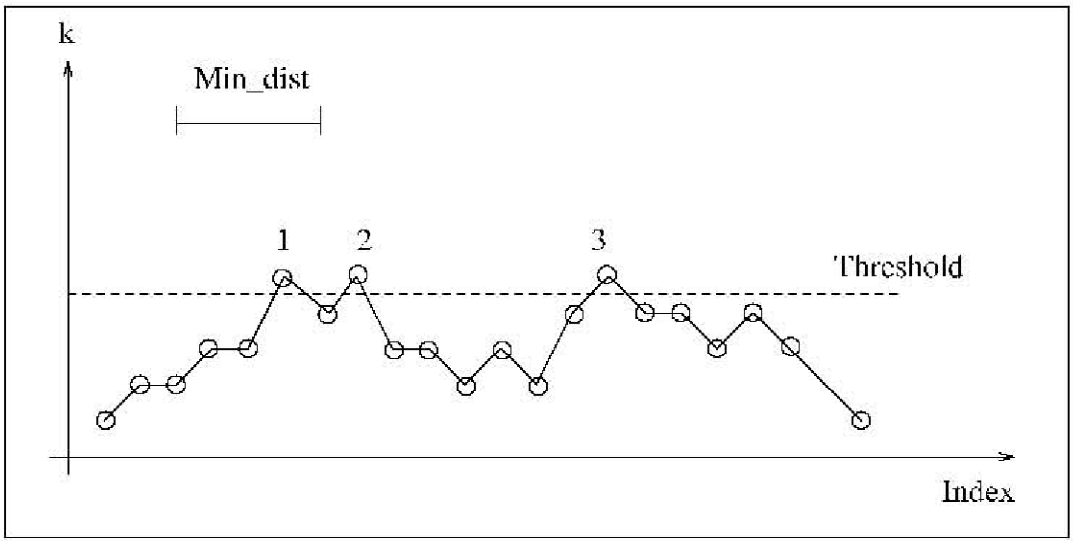


Fig. 3.12: Curvature for every index in **CURV**

Of course there is no clear transition between positive and negative sequences. The algorithm that makes this distinction looks at the tendency in **CURV** (via indices) to decide if it now collects positive or negative sequences. When all these sequences are collected, it is still necessary to look at tendencies within each sequence to decide where there are local extremes (note: the procedure described here is done in one loop through CURV, but the steps are separated for clarity's sake).

To check the tendencies we require k successive values of a type (positive or negative) for a sequence. If we get these k values, we change from gathering positive (negative) to gathering negative (positive) and remember these k values. To determine extrema within each sequence there are two matters to consider. Figure 3.12 illustrates this (the situation for negative sequences is analogous).

One must consider if the curvature is large enough to be a potential extreme and if the distance to the next extreme is large enough. In figure 3.12 all points over the dashed line are extrema candidates, but only 2 and 3 are local maxima. The distance between 1 and 2 is not large enough. **Min_dist** gives the least allowed difference between extrema.
For the ith point we compute $level_i = level_{i-1} + (\mathbf{CURV}[i] - \mathbf{CURV}[i-1])$. This is compared with **Threshold** and the previous index j where $level_j >$ **Threshold**, if any. If $level_i >$ **Threshold** and $|i-j| >$ **Min_dist** , then j is declared an index of a local maximum and we look for the next. Between j and the next local maximum k we remember a local minimum. Both **Threshold** and **Min_dist** are dynamic.

The result string $\mathbf{S_1}$ from the above is delivered with the corresponding string $\mathbf{S_2}$ to the algorithm in figure 3.13. The algorithm derives the process history from the object represented by $\mathbf{S_1}$ to the object represented by $\mathbf{S_2}$. Remarks on the algorithm follow:

- We rotate $\mathbf{S_1}$ until we get a representative in the equivalence class for $\mathbf{S_1}$ that has $\mathbf{S_1}[0]=\mathbf{M}^+$
- With **Match** we find the maximal match between $\mathbf{S_1}$ and $\mathbf{S_2}$ i.e. we get *mcnt* = max(k such that $\mathbf{S_1}$(i) = **S**(i) for i=0,..,k-1 for some **S** equivalent to $\mathbf{S_2}$) and *t* contains the member of the equivalence class for $\mathbf{S_2}$ that gives the maximal match. The idea is borrowed from [13].
- In the repeat loop GENTAG the partial string $\mathbf{S_1}$[*mcnt .. len1*] is pumped to $\mathbf{S_2}$[*mcnt .. len2*] using the rules that are possible in the given situation. It is always the actual ith sign in $\mathbf{S_1}$ that determines the next action.
- **Insert**($\mathbf{S_1}$, *buf*) saves this version of $\mathbf{S_1}$until we know that $\mathbf{S_1} \Rightarrow * \mathbf{S_2}$. *buf* thus contains the process history.

The implicit reduction of the grammatical rules, mentioned in chapter 2.2.1, appears in the ELSE IF(*len1* < *len2*) block in the algorithm. In this sentence block the actual bifurcation is inserted. There can be other bifurcations in the algorithm but they do not lead to a reduction in the rules. Apparently all four possible bifurcations can occur, but actually there are only two possible rules that can be used. Upon consideration one sees that the shown configurations for $\mathbf{S_1}$ and **t**, where we have focused on the (i-1)-th and i-th places, are the only configurations that can occur in this block of the algorithm:

```
S1 : * * ... * M+m- ? ? ... ?
 t : * * ... * M+m+ ? ? ..... ?
Bifurcation BM+ is inserted
S1 : * * ... * m-M+ ? ? ... ?
 t : * * ... * m-M- ? ? ..... ?
Bifurcation Bm- is inserted
```

We see that the rules used are those we found in chapter 2.2.1 could simulate the other two rules. In this way we eliminate redundancy from the grammar but unfortunately we sacrifice a "prioritized" explanation of the process.

```
Algorithm ProcHist_ver_1(S1, S2)
    len1 = Len(S1);
    len2 = Len(S2);
    IF(len1 > len2) ->
        Print("S1 cannot derive S2");
    REPEAT(S1[0] <> M+) ->
        Rotate(S1);
    Match(S1,S2,t,mcnt);
    "t contains  S2 rotated, such that"
    "a maximal match between S1 and t as achieved"
    i = j = mcnt - 1;
    Insert(S1, buf);
    stop = 0;
    REPEAT(stop = 0) ->
        IF(S1[i] = t[j]) ->
            IF(i = len1 - 1) ->
                IF(j >= len2 - 1) ->
                    stop = 1;
                ELSE "find bifurcations for the rest"
                    S1[i + 1] = Bifurcation[t[j]][1];
                    S1[i + 2] = Bifurcation[t[j]][2];
                    len1 = len1 + 2;
                    Insert(S1,buf);
```

```
            ELSE
                i = i + 1;
                j = j + 1;
        ELSE IF(Continuation([S1[i]]) = t[j]) ->
            S1[i] = Continuation([S1[i]]);
            Insert(S1,buf);
        ELSE IF(len1 < len2) ->
            "Insert bifurcation of t[j-1] in S1 between S1[i-1] and S1[i]"
            "In next loop we will find the potential continuations"
            "of the new elements in S1"
            Insert(S1,buf);
            len1 = len1 + 2;
        ELSE
            stop = 2;
    END "REPEAT"
    IF(stop = 2) ->
        Print("S1 cannot derive S2");
    ELSE
        Print(buf);
END
```

Fig 3.13 Algorithm for deriving a process history

4 Testing

As mentioned earlier the method described in chapter 3.4 gives only one process history between two states of an object. There can be many others. The emphasis in this report has been laid elsewhere.

As a practical application the method was tested on two islands in the Hawaii group, Kauai and Hilo. We know that Kauai, fig.4.1, is younger than Hilo, fig.4.2. The islands' form changes all the time because of the geographic activity (volcanoes etc.). This process is very slow, so we do not have many observations of these changes. Therefore it is difficult to test theories of this process. That is why we are interested in the description of the islands' development by Leyton's grammars.

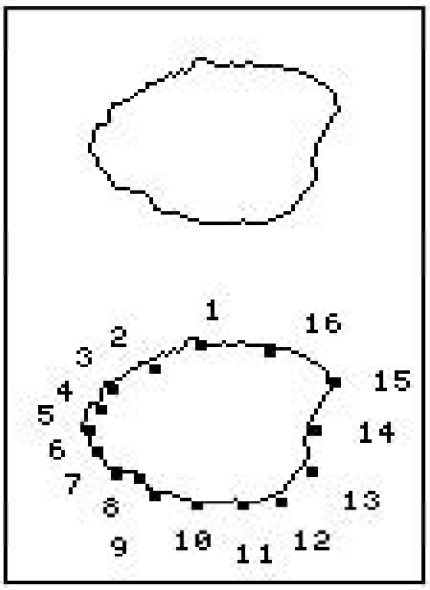


Fig. 4.1 Contour of Kauai

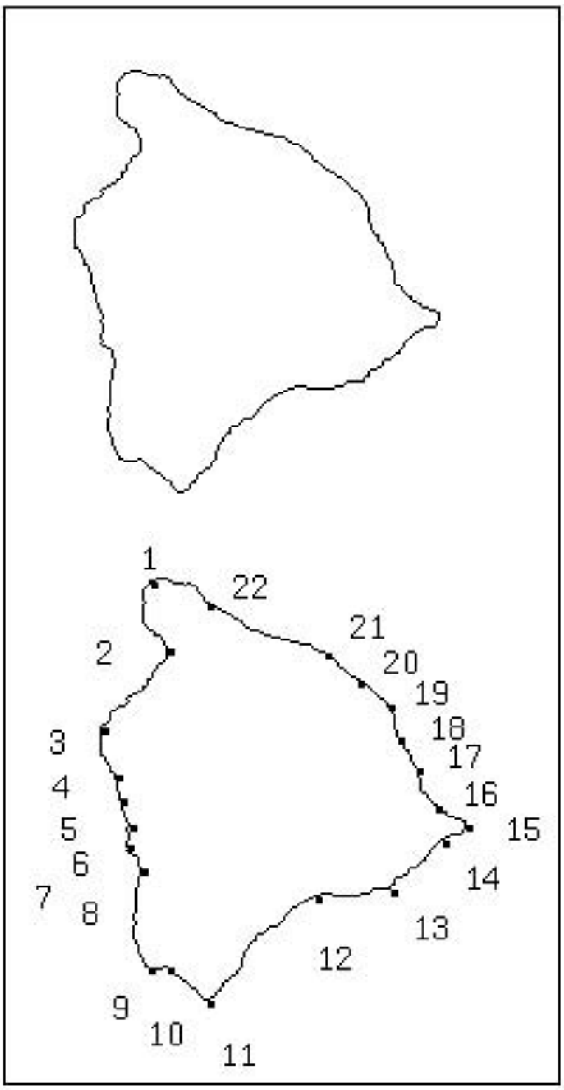


Fig. 4.2 Contour of Hilo

In figures 4.1 and 4.2 we see two islands. The extremes found by our algorithm are marked by a black dot. The counterclockwise ordering is shown by the numbering of the exrema. The characteristic string for Kauai is:

$M^+m^+\ M^+m^+\ M^+m^+\ M^+m^+\ M^+m^-\ M^+m^+\ M^+m^+\ M^+m^-$

And for Hilo:

$M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^+\ M^+m^-$

The algorithm starts by making a maximal match between the two characteristic strings. Hilo's string is rotated to give:

$M^+m^+\ M^+m^-\ M^+m^-\ M^+m^-\ Mm^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-$

It runs 19,20,…,18. We have matched 1, 2, 3 from Kauai to 19, 20, 21 from Hilo. We see that 22 from Hilo is m^- while 4 from Kauai is m^+ so we use $\mathbf{Cm^+}$ on 4 in Kauai. In this way we run through the string for Kauai and the result of different grammar operations is put in the string for Kauai. The entire run is:

$M^+m^+\ M^+m^+\ M^+m^+\ M^+m^+\ M^+m^-\ M^+m^+\ M^+m^+\ M^+m^-$

Continuation in m^+

$M^+m^+\ M^+m^-\ M^+m^+\ M^+m^+\ M^+m^-\ M^+m^+\ M^+m^+\ M^+m^-$

Continuation in m^+

$M^+m^+\ M^+m^-\ M^+m^-\ M^+m^+\ M^+m^-\ M^+m^+\ M^+m^+\ M^+m^-$

Continuation in m^+

$M^+m^+\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^+\ M^+m^+\ M^+m^-$

Bifurcation in m^-

$M^+m^+\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^+\ M^+m^+\ M^+m^-$

Continuation in m^+

$M^+m^+\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^+\ M^+m^-$

Continuation in m^+

$M^+m^+\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-$

Bifurcation in M^+

$M^+m^+\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-Mm^-$

Continuation in M^-

$M^+m^+\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-M^+m^-$

Bifurcation in M^+

$M^+m^+\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-M^+m^-Mm^-$

Continuation in M^-

$M^+m^+\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-\ M^+m^-M^+m^-M^+m^-$

As we see, we have "edited" Kauai to Hilo. In other words Hilo was once like Kauai. If the development of Kauai keeps to the rules it can take the same form as Hilo in some million years.

As a comment on the above example we should mention that the islands are shown to scale. I tried to make Kauai about as large as Hilo and ran the algorithm again. There was too much noise on the enlarged Kauai contour for it to find sensible extrema. I tried also with a spline approximation to the enlarged Kauai. Here the results were much better. Perhaps there should be only eight extrema: 1, 2, 5, 8, 11, 14, 15, 16. We get this by changing the choice-criteria in the algorithm and if one uses k-curvature by adjusting m.

5 Conclusion

Our experience shows the utility and strength of Leyton's process grammars. With very few and simple rules we have a tool for determining the development of a dynamic entity. We can relate any two of its states; we divide the development into stages and identify each stage as the effect of one of four possible processes.

As mentioned in chapter 1.1.2 we get a very ordered structure in our "object universe", using various rules. As a very important point we note that there is a unique direction of movement through this structure, because we are describing irreversible processes. In other words we are describing development in the word's real meaning, not just "executing" rules. This means that the use of process grammars in their present form is not universal.

Each of the four processes has a semantic explanation; one could say meta-explanation. When one wants to use the process grammar on a concrete category of objects, say the Hawaiian islands, one can identify the four processes with existing "phenomena" or "factors" (e.g. oceans eroding coasts, lava streams) whose effects give the same development as the corresponding process in the process grammar. In this way we have a transition between the real world and the object universe of the process grammar. A somewhat unfortunate consequence of this is a tendency to consider an idealized version of the world. Chance events that are not covered by the real world's phenomena or factors are falsely explained by them. A perhaps extreme example could be that one of the islands suffered an earthquake that changed its coastline appreciably. The new bay would be explained as "continued impounding from the ocean".

I will now very briefly describe various expansions that could be useful.

More Information

We mentioned earlier in chapter 2.2.1 that there could be ambiguous (redundant and commutative) use of the rules in the grammar. This arises because of the missing temporal aspect or, said in another way, missing measure for how probable the use of a given rule would be in a given situation. Independently of the "meaning" one chooses, one has to base oneself on more information in the given situation, e.g. symmetry axes, and more knowledge of the nature of the concrete objects.

A technique that possibly could be used with advantages to capture process grammars and administer them is to use L-systems as described in [14]. In this context we can also suggest the future development of an object.

3D

Another aspect of process grammars is their extension to 3 dimensions. Leyton describes two possible ways of doing this. One can use a three dimensional version of the Symmetry-Curvature-Duality theorem where the symmetry axes are now symmetry planes. On this basis one gets completely equivalent rules in three dimensions. The other possibility is to use the known two dimensional rules directly on the 3D objects (generalized cylinders). This is done by slicing the object by planes and using the rules on the resulting 2D objects. In [18,19] a spline representation is described, that can be used for precisely this.

Animation

Another thing I will mention is to improve the visual part of the process explanation. One could use a variation of "Inbetweening" in [25] to give an animated version of the process development, concurrently with the algorithm. This would make it easier, more attractive and much quicker to run through a process development.

Shape Matching

As a last remark in [5] there is a description of a shape matching algorithm which uses process grammars. Instead of extrema they consider curve segments (concave or convex) separated by "zero crossings". The matching itself is done by "editing" both objects' curves using dynamic programming. I first found [5] very late in the project so it has not given me any inspiration. Also it is a completely different approach from the one I have used.

As we see there are many ways one can go and many aspects that require careful treatment. In any case we have started on part of an area that can and will find serious applications in image processing and pattern recognition. By agreement with Brian Mayoh this report has no program documentation, apart from the module that implements the process history. All source text is in my university file system.

Aarhus University, Sep 9 1991, Thomas W. Larsen

```
#include     "pg.h"
#include     <suntool/textsw.h>

static char  continuation[5] = {
                  0, Mplus, mminus, mminus, Mplus
             };

static char  bifurcation[5][3] ={
                  { 0, 0, 0 },
                  { Mplus, mplus, Mplus },
                  { mplus, Mplus, mplus },
                  { mminus, Mminus, mminus },
                  { Mminus, mminus, Mminus }
             };

static char  convert[5][2] = {
                  "  ", "M+", "m+", "m-", "M-"
             };

/********** Routines to match S1 with S2 ************************/
static void out_s( s1, textsw, no )
    char    *s1;
    Textsw  textsw;
    int     no;
{
    int     k;
    char    msg[200];

    if( no == 1 ){
       sprintf( msg, "S1 kan ikke derivere S2\n" );
    }
    else if( no == 2){
       sprintf( msg, "S1 er laengere end S2\n" );
```

```
    }
    else{
       for( k = 0 ; k < strlen(s1) ; k ++ ){
           msg[2 * k] = convert[ s1[k] ][0];
           msg[2 * k + 1] = convert[ s1[k] ][1];
       }
       msg[2 * k] = '\n';
       msg[2 * k + 1] = '\0';
    }
    textsw_insert( textsw, msg, strlen(msg) );
}

static void rotate(s)
    char *s;
{
    char t[MAX_STR_LEN];

    strncpy( &t[1], s, strlen(s) - 1 );
    t[strlen(s)] = '\0';
    t[0] = s[strlen( s ) - 1];
    strcpy(s, t);
}

static void s_match(s1, s2, t, match_length)
    char *s1, *s2, *t;
    int  *match_length;
{
    int  c, max_match, match, len1, len2, i;

    c = 0; max_match = 0;
    len1 = strlen( s1 );
    len2 = strlen( s2 );
    if( len1 > len2 ) len1 = len2;

    do{
```

```
        match = 0; i = 0;
        while( i < len1 ){
            if( s1[i] != s2[i] ) break;
            i ++;
        }
        match = i;
        if( max_match < match ){
            strcpy( t, s2 );
            max_match = match;
        }
        rotate( s2 );
        c ++;
    }while( c < len2 );

    *match_length = max_match;
}

int process_history( q, r, textsw )
    char     *q, *r;
    Textsw   textsw;
{
    int      len1, len2, match, i, j, k;
    char     t[MAX_STR_LEN], s1[MAX_STR_LEN],
             s2[MAX_STR_LEN], s1_1[MAX_STR_LEN];
    char     msg[200];

    strcpy( s1, q );
    strcpy( s2, r );
    len1 = strlen( s1 );
    len2 = strlen( s2 );

    if( len1 > len2 ){
        /* stop */
        out_s( s1, textsw, 2 );
        return 0;
```

```
}
while( s1[0] != Mplus ) rotate( s1 );
s_match(s1, s2, t, &match);

/** from now on, t is the rotated version of s2 **/
i = j = match - 1;
out_s( s1, textsw, 0 );
do{
   if( s1[i] == t[j] ){
      if( i == len1 - 1 ){
        if( j >= len2 - 1 ) return 1;
        else{ /** find bifurcations for the rest **/
           s1[i + 1] = bifurcation[t[j]][1];
           s1[i + 2] = bifurcation[t[j]][2];
           s1[i + 3] = '\0';
           len1 += 2;
           out_s( s1, textsw, 0 );
        }
      }
      else{
        i += 1;
        j += 1;
      }
   }
   else if( continuation[ s1[i] ] == t[j] ){
       /** save Cont.[s1[i]] **/
       s1[i] = continuation[ s1[i] ];
       out_s( s1, textsw, 0 );
   }
   else if( len1 < len2 ){
      /** insert  bifur. of t[j-1] between s1[i-1] and s1[i]  **/
      /** find cont. in the next round and save these as well **/
      strcpy( s1_1, s1 );
      s1[i] = bifurcation[t[j - 1]][1];
      s1[i + 1] = bifurcation[t[j - 1]][2];
```

```
            strcpy( &s1[i + 2], &s1_1[i] );
            out_s( s1, textsw, 0 );
            len1 += 2;
         }
         else{
            /** ERROR , s1 doesn't match t **/
            out_s( s1, textsw, 1 );
            return 0;
         }
      }while( 1 );
}
```

Bibliography

[1] A Process Grammar for Shape
M. Leyton
AI vol. 34 1988.

[2] Symmetry Curvature Duality
M. Leyton
CVGIP vol. 38 1987.

[3] Generative Systems of Analyzers
M. Leyton
CVGIP vol. 31 1985.

[4] Processes at Discontinuities
P. J. Hayes, M. Leyton
IJCAI-89 vol.2 1989.

[5] Shape Matching Using Curvature Processes
E. E. Milios
CVGIP vol. 47 1989.

[6] Neighborhood Coding of Binary Images for Fast
Contour following and General Array Processing
I. Sobel
CGIP vol. 8 1978.

[7] Algorithms for Graphics and Image Processing
T. Pavlidis
Computer Science Press 1982.

[8] Automatic Representation of Binary Images
C. A. Cabrelli, U. M. Molter
PAMI vol. 12 1990.

[9] Elliptic Fourier Features of a Closed Contour
F. P. Kuhl, C. R. Giardina
CGIP vol. 18 1982.

[10] Fourier Descriptors for Plane Closed Curves
C. T. Zahn, R. Z. Roskies
Trans. Comp. vol. 3 1972.

[11] Differential Geometry of Curves and Surfaces
M. P. DoCarmo
Prentice Hall 1976.

[12] Elementary Nummerical Analysis, an Algorithmic Approach
S. D. Conte, C. deBoor
McGraw Hill 1972.

[13] Attributed String Matching with Merging for Shape Recognition
W. H. Tsai, S. S. Yu
PAMI vol. 4 1985.

[14] The Algorithmic Beauty of Plants
P. Prusinkiewicz, A. Lindenmayer
Springer Verlag 1990.

[15] Structural Pattern Recognition
T. Pavlidis
Springer Verlag 1977.

[16] Shape Description Using Weighted Symmetry Axis Features
H. Blum, R. N. Nagel
PR vol. 10 1978.

[17] A width-Independent Fast Thinning Algorithm
C. Arcelli, G. S. Baja
PAMI vol. 7 1985.

[18] An Interactive Computer Graphics Approach to Surface Representation
S. C. Wu, J. F. Abel, D. P. Greenberg
CACM vol. 20 1977.

[19] Splines as Embeddings for Generalized Cylinders
U. Shani, D. H. Ballard
CVGIP vol. 27 1984.

[20] Fundamentals of Interactive Computer Graphics
J. D. Foleys, A. vanDam
Addison Wesley Publishing Company 1982.

[21] Reconstruction of 3D Medical Images: A nonlinear
Technique for Recognition of 3D Medical Images
L. W. Chang, H. W. Chen, J. R. Ho
CVGIP/BMIP vol. 53 1991.

[22] Corner Detection and Curve Representation Using Cubic B-splines
G. Medioni, Y. Yasumoto
CVGIP vol. 39 1987.

[23] Angle Detection on Digital Curves
A. Rosenfeld, E. Johnston
Trans. Comp. vol. c-22 1973.

[24] Estimation of a Circular Arc Center and it's Radius
U. M. Landau
CVGIP vol. 38 1987.

[25] Design, transformation and animation of human faces
N. M. Thalmann, H. T. Minh, M. d. Angelis, D. Thalmann
Visual Computer vol. 5 1989.