stichting
mathematisch
centrum

MC

AFDELING INFORMATICA
(DEPARTMENT OF COMPUTER SCIENCE)

IW 96/78 MEI

P. KLINT & H.J. SINT

A FRAMEWORK FOR THE INTEGRATION OF GRAPHICS AND PATTERN RECOGNITION

Preprint

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O).

AMS(MOS) subject classification scheme (1978): 68A45

Computing Reviews 3.63, 8.2

A framework for the integration of graphics and pattern recognition.*)

by

Paul Klint & Marleen Sint

ABSTRACT

A model is presented that describes a symmetric input/output function in a computer Graphics System. This model is based on the construction of a symmetric input function by means of pattern recognition. Both theoretical and implementational aspects of this model are discussed.

KEY WORDS & PHRASES: *Computer Graphics, Pattern Recognition, Symmetric input/output.*

*) This report will be submitted for publication elsewhere.

1. Informal outline.

Symmetry of input and output operations is not only aesthetically pleasing but has also advantages in many computer applications. Sometimes such a symmetry is difficult to achieve, sometimes it is not even clear what it means.

In the case of Computer Graphics one can envisage a high level graphics language that produces as output programs in some machine independent intermediate language, and accepts as input programs in the same intermediate language. Input from a drawing machine should somehow produce programs in the intermediate language. In this paper we will try to refine this 'somehow' somewhat.

First we need some terminology: 'A picture is defined as a description of some object such that a visible image can be obtained from this description in a uniform way. The description may include both geometrical (shape, size) and non-geometrical (colour, weight) properties of the object' [1].

This definition states that there exists a function that maps pictures onto visible images. This function is an output function; it maps an internal representation (the picture) onto an external representation (the image, i.e. the line-drawing, photograph or video image, produced by some output device). Pictures are structured objects, but images are not. By going from a picture to its image the structure is lost. Consequently, the inverse mapping from images to pictures will not be a function: the same image can be mapped onto more pictures since there is no way to structure an image uniquely. Consider the following pictures (the notation from [1] is used):

```
    pict Picture1 {Triangle1; Triangle2}.
subpict Triangle1 LINE([0,0], [1,0], [1,1], [0,0]).
subpict Triangle2 WITH ROTATE 180 AROUND ([.5,.5]) DRAW Triangle1.
```

```
pict Picture2 {Square; LINE([0,0], [1,1])}.
subpict Square  LINE([0,0], [1,0], [1,1], [0,1], [0,0]).

pict Picture3 {Square; LINE([1,1], [0,0])}.
```

When Picture1, Picture2 and Picture3 are drawn, they all lead to the image shown in figure 1. Picture2 differs from Picture3 only in the drawing direction of the diagonal.

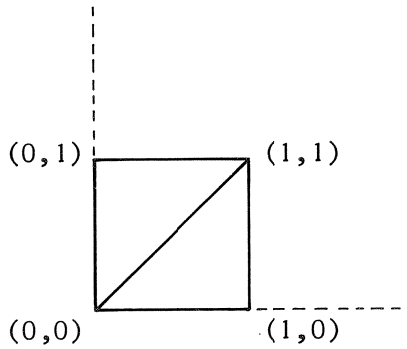

Figure 1.

The inverse mapping, from images to sets of pictures, is an input mapping, from an external representation (the image, i.e. what you draw on some input device) to a set of internal representations (pictures). If one succeeds in constructing such an input mapping as the reverse of the output mapping (for a given device which is capable to do both input and output) then symmetry of input and output is obtained for that device.

Suppose we want an input function instead of an input mapping, how could that be achieved? A picture combines 'content' and 'structure' while in an image only the 'content' is retained. (The terms content and structure are somewhat vague, but are probably intuitively clear. They will only be used to clarify underlying ideas, and are not used as

technical terms.) To obtain a unique mapping from images to pictures, we need structure descriptions and a process to combine these with the images to yield pictures. For example, structure descriptions corresponding to 'two triangles sharing one side' and 'a square with one diagonal' would allow us to map the image of figure 1 back to Picture1 when combined with the first description, and to Picture2 (or Picture3) when combined with the second. This process is nothing else than a pattern recognition process; the structure descriptions will be called patterns.

In the next section we will present a formalization of these ideas. Everything between square brackets is comment on, not part of the model. In section 3 we pay some attention to implementational aspects of our model and try to identify problem areas.

2. A model for graphics input/output functions.

2.1. Description of the model.

[Our goal can be stated as follows: given a graphical output function, construct an input function, using patterns and pattern recognition, such that this input function is symmetric (in some sense) with the output function.]

Given are a set PICT [of pictures] and a set IM [of images]. Given is also an equivalence relation ^ on IM, dividing IM into a set of equivalence classes denoted by IM^.
[It is not at all obvious when two images are the same; should for instance, drawing order be considered part of the image? For the moment we content ourselves with stating that there is some equivalence relation on IM, dividing IM into classes of images indiscernible for the input process. In the comment we will keep using the word 'image' instead of 'equivalence class of images'.]

Also given is a surjective function DRAW: PICT -> IM^ .
[DRAW is the graphical output function. We are only interested in images

which can be drawn, hence the assumption that DRAW is surjective. In section 2.3. we will justify the choice of having IM^ rather than IM as the range of DRAW. For people who object that a graphical output function always maps a picture to an image and not to an equivalence class of images, we can add one step to the definition:

Given a function

OUT: PICT -> IM, let

DRAW: PICT -> IM^ be defined by

DRAW(p) = im^ <=> OUT(p) $\in$ im^.

The situation is as shown in figure 2a; DRAW is surjective but not necessarily injective. Next, we want to formalize our notion of symmetric input/output.]

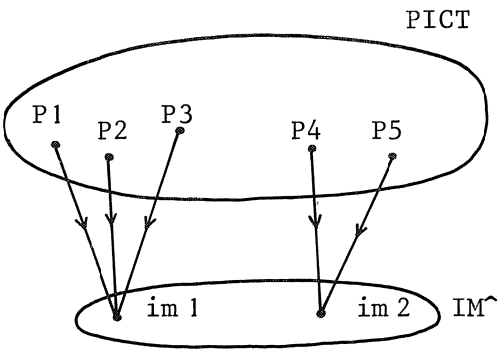

Figure 2a.

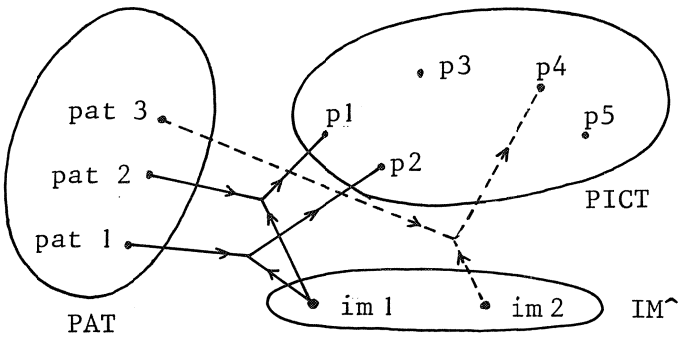

Figure 2b.

Output - example:

DRAW (p1) = DRAW (p2) = DRAW (p3) = im1.

DRAW (p4) = DRAW (p5) = im2.

Input - examples (for the same pictures and images as in figure 2a):

REC(im1,pat1) = p1,

REC(im1,pat2) = p2,

REC(im2,pat3) = p4.

p3 and p5 are not in the range of REC.

A function I: IM^ -> $\mathcal{P}$(PICT)
will be called an input function symmetric with DRAW iff
(1) $\forall p \in PICT\ \forall im^ \in IM^ (p \in I(im^) \Rightarrow DRAW(p) = im^)$
and
(2) $\forall im^ \in IM^ (I(im^) \neq \emptyset)$

[We appoint as the range of the input function the powerset of PICT and not PICT itself, because we do not want to exclude the possibility that the input function is really the inverse of the output function. Requirement (1) states that the input function must be consistent with the output function in the following sense: if the input function maps an image onto some picture, this picture should, when drawn, yield that same image. Requirement (2) states that the input function must be complete in the sense that it can map each image to at least one picture. It seems not possible to formulate weaker requirements and still obtain an input function which can reasonably be called symmetric with the given output function. The arrow in (1) is one-sided; we do not require that the input function maps an image back onto all pictures which, when drawn, yield that image. Whether that requirement is necessary as well, is merely a matter of taste. In section 2.2. we will briefly review this stronger form of symmetry.

Next we show how the function I can be realized by introducing a set of patterns and a recognition function.]

Given an additional set PAT and a function
REC: PAT x IM^ -> PICT $\cup$ {fail}, where fail $\notin$ PICT.

[PAT contains the patterns, the structure descriptions which can be considered as abstractions of pictures from their content. The recognition function REC (re)combines images with patterns. There are two possible outcomes when the recognition process is comparing an image and a pattern. First, it is possible to structure the image in the way the pattern prescribes, in which case the result is a picture. Second, the

image can fail to match the pattern. To allow for this case and yet define REC as a complete function, the element 'fail' is added to the range of REC.]

We now formulate consistency (3) and completeness (4) requirements for PAT and REC, such that we will be able to construct a symmetric input function:

(3) $\forall im^\wedge \in IM^\wedge \; \forall \; pat \in PAT \; (REC(im^\wedge,pat) = fail \vee DRAW(REC(im^\wedge,pat)) = im^\wedge)$

[This embodies the consistency requirement: a match using an image and a pattern should either fail or yield a picture, which can produce the same image.]

(4) $\forall im^\wedge \in IM^\wedge \; \exists \; pat \in PAT \; (REC(im^\wedge,pat) \neq fail)$

[This embodies the completeness requirement: each image can be combined with at least one picture without failing.
Figure 2b shows an example of the behaviour of REC.]

If the properties (3) and (4) hold, then the function:

INP: $IM^\wedge \rightarrow \mathcal{P}(PICT)$

with

(5) $INP(im^\wedge) = \{p \in PICT \mid \exists \; pat \in PAT \; (REC(im^\wedge,pat) = p)\}$

is an input function symmetric with DRAW.

Proof:

1. $p \in INP(im^\wedge) \Rightarrow DRAW(p) = im^\wedge$:

   $p \in INP(im^\wedge)$ => there is some pattern, call it patc, with

   $p = REC(im^\wedge,patc)$ by definition of INP.

$p \neq fail$ (as $p \in PICT$ and $fail \notin PICT$), hence
$REC(im\hat{},patc) \neq fail$. Combined with (3) this yields
$DRAW(REC(im\hat{},patc)) = im\hat{} \Rightarrow$
$DRAW(p) = im\hat{}$.

2. $\forall im\hat{} \in IM\hat{}\ (INP(im\hat{}) \neq \emptyset)$:

$\forall im\hat{} \in IM\hat{}\ \exists\ pat \in PAT\ (REC(im\hat{},pat) \neq fail)$
(according to (4)), hence
$\exists p \in PICT\ (REC(im\hat{},pat) = p)$, hence
$p \in INP(im\hat{})$, which is consequently not empty.

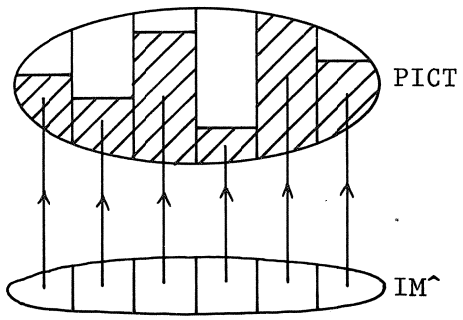

Figure 3a. INP with weak symmetry requirements.

[What did we achieve so far? First, the notion of i/o symmetry is formalized. Second, we showed how a symmetric input function can be constructed by means of pattern recognition. Now we will look a little closer at the relation between DRAW, INP and the equivalence relation which is induced by $\hat{}$ in PICT. PICT can be divided into equivalence

classes, putting all pictures mapped onto the same image by DRAW into one class. INP maps an image uniquely into a subset of the equivalence class generated by that image. Figure 3a shows the idea: equivalence classes are separated by vertical bars; the arrows point at the values of INP(im^), which are shown as shaded areas. We have used $\mathcal{P}$(PICT) as the range of INP, but it turns out that the induced equivalence classes in PICT are sufficient. An auxiliary function INVDRAW is introduced to formalize these ideas. It turns out that INVDRAW acts as an upperbound on the behaviour of a symmetric input function. See equation (6) below.]

The equivalence relation $\hat{=}$ on PICT, dividing PICT into a set of equivalence classes denoted by PICT$\hat{=}$, is defined as follows:

p1 $\hat{=}$ p2 <=> DRAW(p1) = DRAW(p2).

[p1 $\hat{=}$ p2 if they map to equivalent images].

The function

INVDRAW: IM^ -> $\mathcal{P}$(PICT)

with

INVDRAW(im^) = {p $\in$ PICT | DRAW(p) = im^}

is a bijection from IM^ to PICT$\hat{=}$.

[i.e. we will show that each equivalence class is generated by one element from IM^]

Proof:

1. $\forall$im^ $\in$ IM^ (INVDRAW(im^) $\in$ PICT$\hat{=}$):

INVDRAW(im^) is not empty, as DRAW is surjective.

p1 $\in$ INVDRAW(im^) $\wedge$ p2 $\in$ INVDRAW(im^) <=>

DRAW(p1) = im^ $\wedge$ DRAW(p2) = im^ <=>

DRAW(p1) = DRAW(p2) <=>

p1 $\hat{=}$ p2

2. INVDRAW is injective:

$im1^\wedge \neq im2^\wedge \Rightarrow INVDRAW(im1^\wedge) \neq INVDRAW(im2^\wedge)$.

Suppose not:

$\exists\ im1^\wedge, \exists\ im2^\wedge \in IM^\wedge\ (im1^\wedge \neq im2^\wedge \wedge INVDRAW(im1^\wedge) = INVDRAW(im2^\wedge))$

Then, as $INVDRAW(im^\wedge)$ is never empty,

$\exists\ p \in PICT\ (p \in INVDRAW(im1^\wedge) \wedge p \in INVDRAW(im2^\wedge))$ hence

$DRAW(p) = im1^\wedge \wedge DRAW(p) = im2^\wedge$ hence

$im1^\wedge = im2^\wedge$, contrary to assumption.

3. INVDRAW is surjective:

$\forall\ \underline{p^\wedge} \in \underline{PICT^\wedge}\ \exists\ im^\wedge \in IM^\wedge\ (INVDRAW(im^\wedge) = \underline{p^\wedge})$

namely, the element from $IM^\wedge$ such that

$\forall\ p \in \underline{p^\wedge}\ (DRAW(p) = im^\wedge)$.

[Back to the input function at last: each image is mapped into a subset of the equivalence class generated by INVDRAW:]

(6) $\forall\ im^\wedge \in IM^\wedge\ (INP(im^\wedge) \subset INVDRAW(im^\wedge))$

because $p \in INP(im^\wedge) \Rightarrow DRAW(p) = im^\wedge$,

as a consequence of the fact that INP is symmetric, see (1).

2.2. An alternative model.

Lets look once more at our definition of i/o symmetry. We required that each image can be mapped back onto at least one picture, but possibly more, and this finally resulted in the situation depicted in fig. 3a. Some pictures (in the white areas) can never be the result of applying the recognize function, though they can be drawn. Looking back at the example in section 1, we allow the possibility that there is only

one pattern, specifying a 'square with one diagonal'. We will now (without giving any proofs) give results for a stronger requirement on symmetric i/o:

A function I: IM^ -> $\mathcal{P}$(PICT)
is called strongly symmetric with the output function DRAW iff
(1') $\forall$ p $\in$ PICT $\forall$ im^ $\in$ IM^ (p $\in$ I(im^) <=> DRAW(p) = im^)
There is no need for a second requirement; (2) immediately follows from (1'). We have to change the requirements on REC accordingly, (3) and (4) are now replaced by one stronger requirement:

(3') $\forall$ p $\in$ PICT $\forall$ im^ $\in$ IM^ (DRAW(p) = im^ <=>
$\exists$ pat $\in$ PAT (REC(im^,pat) = p))

In other words, a picture is mapped by DRAW onto some image if and only if there is a pattern, which, when matched against that image, yields the picture. The right arrow represents the stronger completeness requirement, the left arrow represents the consistency requirement.

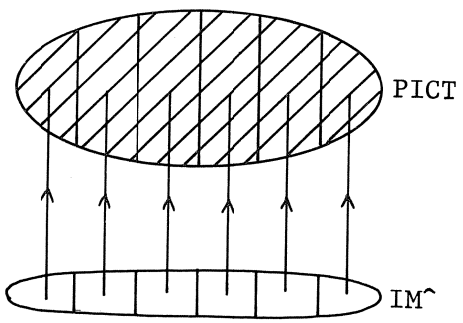

Figure 3b. INP with strong symmetry requirements.

If (3′) holds, then INP is strongly symmetric with DRAW. Moreover, (5) can be changed to INP(im^) = INVDRAW(im^). The situation is as shown in figure 3b. Note that these stronger requirements are not requirements on the recognize function, but on the set of patterns: now each picture must have a corresponding pattern.

2.3. Notes on the model.

note 1. We will have to think some more about the role of the equivalence relation ^ on images. Interesting cases for pattern recognition often include fuzzy matching: A hand drawn (and hence imperfect) square should ideally be recognized and be drawn as a (perfect) square. In the model as presented here, this effect can be achieved by a proper choice of the equivalence relation on images: all nearly-squares should go into the same equivalence class. However, this solution has the serious disadvantage that we cannot utilize our patterns and recognition function for inexact matching. If we do use the patterns and recognition function for that purpose, then the consistency requirement forbids to both recognize the imperfect square and draw the perfect one. To solve this difficulty, we can either relax the consistency requirement or add a layer of pattern recognition to the model.

For exact pattern matching, a reasonable candidate for the equivalence relation seems: 'two images are equivalent iff they coincide when superimposed'. This definition of equivalence covers device dependency: pictures leading to the same image on a black-and-white device can lead to different images on a colour device. In our model this dependency is hidden in the dependency of $\hat{}$ on DRAW.

note 2. In our formalization we assume the existence of a set of pictures and an output mapping and state as our goal the construction of an input mapping. We made this asymmetric choice because we have the feeling that there is a reasonable understanding what pictures and output functions should look like. This choice had consequences for the model. We assumed that pictures contain more information than images

(again, conforming to common practice) and excluded all images that are not internally represented by pictures. Hence, the set of pictures is at least as large as the set IM^. Since ^ is not specified, one can not say the same about PICT and IM.

If we had started from the pattern recognition point of view we should probably have made the reverse choice. In that case, one has an input function mapping images to pictures, and looses information on the way. For example, in cluster analysis and contour finding different images may well lead to the same picture. We can still handle this case by choosing the equivalence relation on images and the recognize function in such a way, that all equivalence classes of pictures contain only one picture, and that all images mapping onto the same picture are in one class. In this case an output function could never map pictures to images, but it can still map pictures to equivalence classes of images (probably drawing the contours that have been determined by the input function). This is the reason why we choose IM^ rather than IM as the range of DRAW - we wanted, so to speak, minimize the consequences of our asymmetric treatment of symmetric i/o.

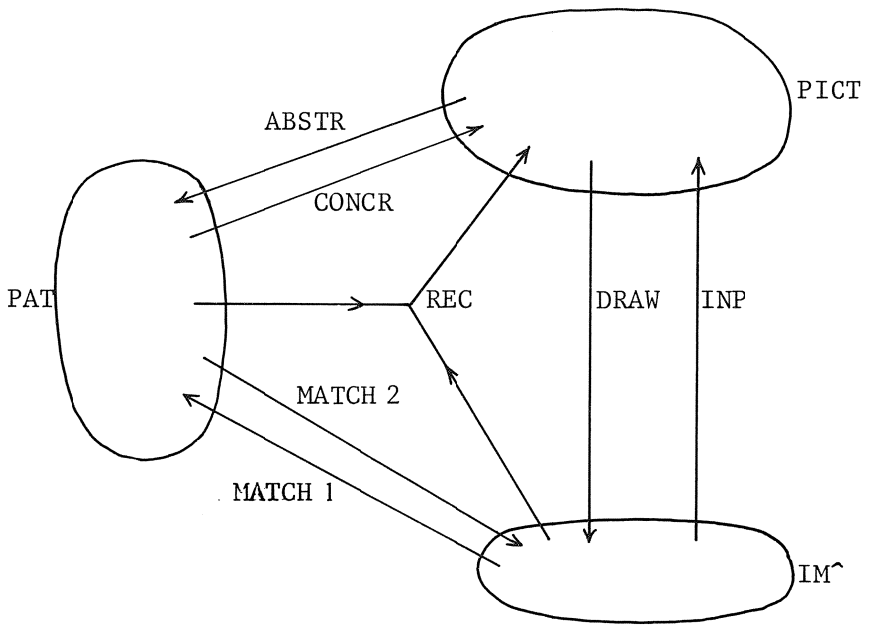

Figure 4: Functions between PICT, PAT and IM^.

note 3. (see figure 4). Other functions between PICT, PAT and IM^ with a reasonable interpretation are:

- MATCH1: IM^ -> $\mathcal{P}$(PAT), with
  MATCH1(im^)={pat $\in$ PAT | $\exists$ p $\in$ PICT (REC(im^,pat)=p)
  MATCH1 searches all patterns which match a given image.

  MATCH2: PAT -> $\mathcal{P}$(IM^), with
  MATCH2(pat)={im^ $\in$ IM^ | $\exists$ p $\in$ PICT (REC(im^,pat)=p)}
  MATCH2 searches images matching a given pattern. There is some relation between MATCH1 and bottom up parsing, and between MATCH2 and top down parsing. Constructing these two MATCH functions could well turn out to be the real difficult task when implementing the model.

- ABSTR: PICT -> $\mathcal{P}$(PAT) with
  ABSTR(p)={pat $\in$ PAT | $\exists$ im^ $\in$ IM^ (REC(im^,pat)=p)}
  ABSTR is the function which maps pictures into structure descriptions. With the weaker symmetry requirement, ABSTR(p) can be empty; with the stronger it always contains at least one element. It can contain more. In section 3.4. we pay more attention to abstraction.

- CONCR: PAT -> $\mathcal{P}$(PICT) with
  CONCR(pat)={P $\in$ PICT | $\exists$ im^ $\in$ IM^ (REC(im^,pat)=p)}.
  CONCR maps a pattern to all pictures from which that pattern is an abstraction; from 'square with one diagonal' to a whole set of individual squares with individual diagonals.

3. From Model to Implementation

In section 2 we introduced a very general model for the input/output behaviour of a Computer Graphics system. Given an output function DRAW, an input function was constructed with certain symmetry properties. For this construction we needed an additional set of structure descriptions (PAT) and a recognition function (REC). Their external behaviour was defined, but it was left unspecified how such a behaviour can be achieved. In this section we attempt to derive some properties of PAT and REC. These considerations are a first step towards an implementation of our abstract model.

One should bear in mind that we do not want to (re)invent techniques and algorithms for the solution of certain recognition tasks, but rather want to provide a framework in which existing methods can be incorporated.

3.1. Pictures

We assume the existence of a set of pictures and an output function which maps pictures on images. Their precise form is not relevant though the discussions are based on tree-like structured picture descriptions such as ILP[1]. ILP pictures can describe objects of any dimension. Implementation of the model can be simplified by restricting pictures to two dimensions. It will be clear that any implementation must choose its method of picture description and that this choice will influence the various parts of the system, especially the form of patterns and the recognition function.

3.2. Patterns.

Patterns can be defined in several ways:

1. Patterns are an integral part of the recognition program. We will call this type of patterns fixed.

2. Patterns can be defined by the user of the program. The user selects (in some way) a picture, which is converted to a pattern by the system. We will call this type of patterns user defined. In this case patterns are created as abstractions from pictures, in other words, the function ABSTR (1.5, note 3) must be part of the Graphics System.

3. Patterns are created by the system for the recognition of frequently occurring subpictures. We will call this type of patterns implicit.

Pattern primitives can best best be chosen the same as picture primitives (point, line, arc, curve). In that case, patterns and pictures are of a comparable level of complexity and the mapping from pictures to patterns is simplified. The patterns from the ESP language [2] come close to what we envisage.

3.3. Images.

Images can be characterized in several ways, of which the following two seem important to us. The first characterization concerns the form of the data of which the image consists:

1. An image is a list of coordinate pairs (points). Subsequent points in the list may be connected or not. Such coordinate values are typically obtained as the result of an input action on a sampled device (tablet).

2. The image consists of higher level primitives. This kind of information can either be obtained directly as the result of input actions on event devices or be the result of processing data of the previous type. Possible primitives are lines, circles and curves. There is a relation between this division of images according to the complexity of primitives and the decomposition problem to be discussed in 3.5..

3. The image consists of a digitized photograph, i.e. a two-dimensional array in which each of the elements specifies the intensity and/or colour at that point. This case will be ignored as presenting too much difficulties to be handled in a uniform and application independent way. It also presents problems on the output side, since our proposed output function is based on ILP, which is heavily oriented towards line drawings.

We consider the first possibility as the most interesting, but do not regard this as a form suitable to define a recognition process on. Hence, our images will consist of data of the second kind. Preprocessing may be needed to transform an image of the first kind into an image of the second kind. Such a preprocessing phase can be based on well-known techniques [3].

A second characterization of images concerns the meaning of the drawing order of primitives. Availability of the drawing order can be used in two ways, either as a heuristic for the recognition task or as a part of the structure description. In the latter case two images are different if they were drawn in a different way, even if they are furth-

er identical. Restrictions of this kind lead to inflexibility, hence we think that (obliged) drawing order may only be required for a small set of hand written symbols.

3.4. Abstraction

The pattern recognition function induces an equivalence relation on pictures, putting all pictures which are recognized by means of the same pattern into one class. It is clear that the form of such an equivalence relation heavily influences both pattern definitions and recognition function. The definition of an equivalence relation requires that one disregards some aspects of a picture. Closely related with the definition of such an equivalence relation is the problem how pictures can be abstracted to patterns. In section 2.2. it was mentioned that there is no unique way to abstract from the 'contents' of a picture. In principle there is a whole range of possibilities, from considering the picture as content only to considering the picture as structure only. The extreme cases do not seem very useful, but the following four intermediate forms make sense:

1. weak topological abstraction: considers only the connectivity of components in a picture as structure, but disregards shape. This type of abstraction can be used for electrical network design.

2. strong topological abstraction: considers only connectivity, with the restriction that interior and exterior components can never lead to the same abstraction. This type of abstraction is invariant under arbitrary continuous transformations.

3. geometrical abstraction: takes both connectivity and shape into account. Two pictures lead to the same abstraction if their images can be made to coincide by

means of a linear transformation. This is a special case of 2. above.

4. object abstraction: two pictures lead to the same abstraction if their images have exactly the same shape and orientation, though these images may be located in different parts of the enclosing image.

The above abstraction types have some relation with graphical properties of the pictures under consideration. Other abstraction types may be induced by the application program: a traffic analysis program will manipulate 'vehicles' as abstraction, though the various sorts of vehicles (bicycles, cars, boats) have no graphical properties in common. This type of abstraction requires that patterns allow the enumeration of graphically unrelated pictures and that patterns can contain application dependent predicates on pictures.

There are two places where application independent abstraction and picture equivalence can be introduced: in the pattern definitions and in the recognition function. In the first case, different pattern primitives are needed for the various abstraction types, i.e. CONNECTED_TO and LINE_TO, which are used by one (not parameterized) recognition process. This approach has the disadvantage that patterns must be modified if one wants to switch to another abstraction type. In the second case, recognition must be performed in a topological, geometrical or object recognition mode, but patterns remain unchanged.

The second possibility seems more elegant but for practical reasons we prefer the first. If the type of abstraction depends on the mode of the recognition process, then one is obliged to define patterns which can handle the lowest abstraction type. Working in object or geometrical recognition mode is for example impossible if different patterns for a circle and a square are not available. Many applications though will not need object abstraction, and will hence suffer from needless redundancy in their patterns. Furthermore, application dependent abstrac-

tions must be included in the pattern definitions, since the recognition function should not be made aware of application programs. Hence we decided to introduce abstraction and equivalence of pictures by means of the pattern definitions.

The main difficulty in the implementation of the ABSTR function lies in the necessity to make relations explicit which in the picture definition are contained only implicitly. A picture describing two touching circles will in general contain no reference to this topological important fact.

3.5. Recognition.

Methods which are based on syntactic pattern recognition seem for our goal the most suitable. (See [4] for an overview).

Two tasks must be solved by a recognition process:

1. Given an image and a set of patterns, find the pattern matched by the complete image. In this case the task of finding out where one primitive ends and the next one begins is not part of the recognition task. We will call this process identification.

2. Given an image and a set of patterns, find the combination of patterns matched by the image. We will call this process decomposition. This case is the most interesting but by far the most difficult one.

The decomposition problem exists on all levels of the recognition process. This is made clear by the choice of a tree-structured picture description method. In such a description complex pictures are subdivided into simpler ones. Each subdivision in the picture description is reflected in a pattern definition and presents a new decomposition

problem to the recognition process.

In all levels of the decomposition hierarchy one can choose between top down and bottom up recognition techniques. These two approaches are represented in the formal model by the functions MATCH1 and MATCH2. A different amount of a priori knowledge is available on each level. It is extremely important to use as much a priori information as possible. In that way one can avoid combinatorial explosions caused by exhaustive searching. The description of the recognition process in terms of a decomposition hierarchy does neither imply that the decomposition steps are sequentially ordered in time, nor that they are mutually independent. We envisage a system architecture as used in the speech recognition system HEARSAY [5].

In 2.2. we divided images according to the level of complexity of their primitives. The transition from raw coordinates to lines and arcs can be seen as a decomposition step. But there is a difference. We already committed ourselves to a certain kind of pictures and patterns. Raw coordinates are data of a lower level than these primitives. Hence, the transition can never be handled by the recognition process itself, and if raw coordinates are allowed as input one needs a separate process for that transition. This does not necessarily mean preprocessing - a certain interaction between recognition process and primitive identification process (as we will call it) is very well possible. This is an example of the dependency between various decomposition levels. The following approach seems sensible: At first, the primitive identification process identifies straight lines in the image. It tries to find the exact form of curves only on request of the recognition process - which will in general be able to provide extra information (look for a circle, a sinoid etc.) which is not available to the primitive identification process itself.

4. Conclusion.

An attempt was made to provide both a theoretical and pragmatic basis for the integration of input/output functions in a Computer Graphics System. The following conclusions were reached:

1. input/output symmetry can be defined in such a way that the input function is (almost) the reverse of the output function.

2. A symmetrical input function can be constructed by means of pattern matching. This requires a set of patterns (PAT) and a recognition function (REC). In this way 'structure' and 'content' of pictures can be separated.

3. Tree structured picture descriptions are well suited to define a recognition process on.

4. Pattern primitives can best be chosen the same as picture primitives.

5. Patterns should allow the enumeration of graphically unrelated pictures.

6. It should be possible that application dependent predicates on pictures are part of patterns.

7. Several types of abstraction and equivalence can be defined on pictures. These should be part of the pattern definitions.

8. Pattern matching should use as much a priori knowledge of the problem as possible.

9. Our model wants to provide a terminology for describing and thinking about input functions which incorporate pattern matching. In the present state of the art it is not feasible to build a Graphics System which includes an input function with the same power as conventional output functions. In a first implementation of the model we will have to make severe restrictions; for instance, only try to handle identification of 2-dimensional images. With the insight gained in realizing this attainable goal, one can go on to the more interesting cases and add decomposition and preprocessing.

5. References

[1] T. Hagen, P.J.W. ten Hagen, P. Klint & H. Noot, 'The intermediate language for pictures', IFIP Congress Proceedings, 1977.

[2] L.G. Shapiro & R.J. Baron, 'ESP3: A language for pattern description and a system for pattern recognition', IEEE transactions on software engineering, vol. SE-3, no. 2, 1977.

[3] H.J. Freeman, 'Computer processing of line drawing images', Computing Surveys, Vol. 6, no. 1, 1974.

[4] M. Nagao, 'A survey of pattern recognition and picture processing', Artificial Intelligence and Pattern Recognition in Computer Aided Design, IFIP W.G. 5.2 Working Conference, 1978.

[5]. V.R. Lesser & L.D. Erman, 'A retrospective view of the HEARSAY-II architecture', Proceedings of 5th International Joint Conference on Artificial Intelligence, 1977.