Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

P.J.W. ten Hagen, C.G. Trienekens

Pattern representation

Department of Computer Science Report CS-R8602 January

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

Pattern Representation

P.J.W. ten Hagen, C.G. Trienekens
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

This paper introduces a new method of representing area oriented picture primitives. The representation aims at efficient conversion of these primitives to framebuffers for raster displays. The representation and the conversion to raster can be the basis for area generators being comparable in speed to vector generators used in conventional vector displays. The representation is independent of the type and resolution of raster hardware. As a result interactive graphics on raster display can be of higher quality because of fast responses involving picture change without compromising picture quality. This paper concentrates on the treatment of the area's domain, e.g., definition and manipulation of domains. The generation of the texture in the domains will be dealt with in a subsequent paper.

1980 Math. Subject Classification : 69K31, 69K33, 69K36.
1983 CR Categories : I.3.1, I.3.3, I.3.6.
Key Words & Phrases : Computer graphics, raster displays, pattern representation, scanconversion.

Note : These investigations were supported by the Netherlands Technology Foundation (STW)

1. Introduction

This paper discusses the representation of patterns given certain requirements. Patterns are area oriented picture elements. The requirements are that the picture making devices can use raster technology and that the applications using the pictures are highly interactive, making intensive use of real-time picture change.
It is a well known fact that in the general case small picture changes require a complete regeneration of the raster image in order to make this change visible. This situation causes interaction on raster devices being either unacceptably slow or highly restricted because the kind of changes that can be efficiently supported is limited.
Recently introduced methods originating from image processing techniques have provided more elegant methods for raster image representation. Examples are quadtrees, octrees, run length encoding schemes and other hierarchical representations. However, at this moment all of these methods are insufficient for interaction support.
Graphics devices using vector technology are essentially low quality devices for representing area-oriented pictures. Their ability to support real-time picture change and therefore interaction stems from the fact that they support a structured display file, which on the one hand can be changed locally and on the other hand can be traversed very efficiently generating the new screen image.
Efficient picture change obviously can be realized by combining the ability to make local changes and the ability to quickly regenerate the image from the display file.

Report CS-R8602
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

The representation for area-oriented pictures proposed here on the one hand allows the primitives to be part of a structured display file, and on the other hand represents each picture primitive in such a way that fast rasterization (comparable to fast vector generation) is possible. Being part of a structured display file among other things requires that picture elements can be retransformed or differently coloured while being displayed.
Fast rasterization requires that the individual picture elements can be scan converted quickly and in the right order. The most time consuming step in the rasterization process (2D or 3D) is the sorting of the points that make up an area.
The strategy being followed to realize picture representation for interactive use on raster displays, includes a representation in a structured display file which has already performed most, if not all, of the sorting.

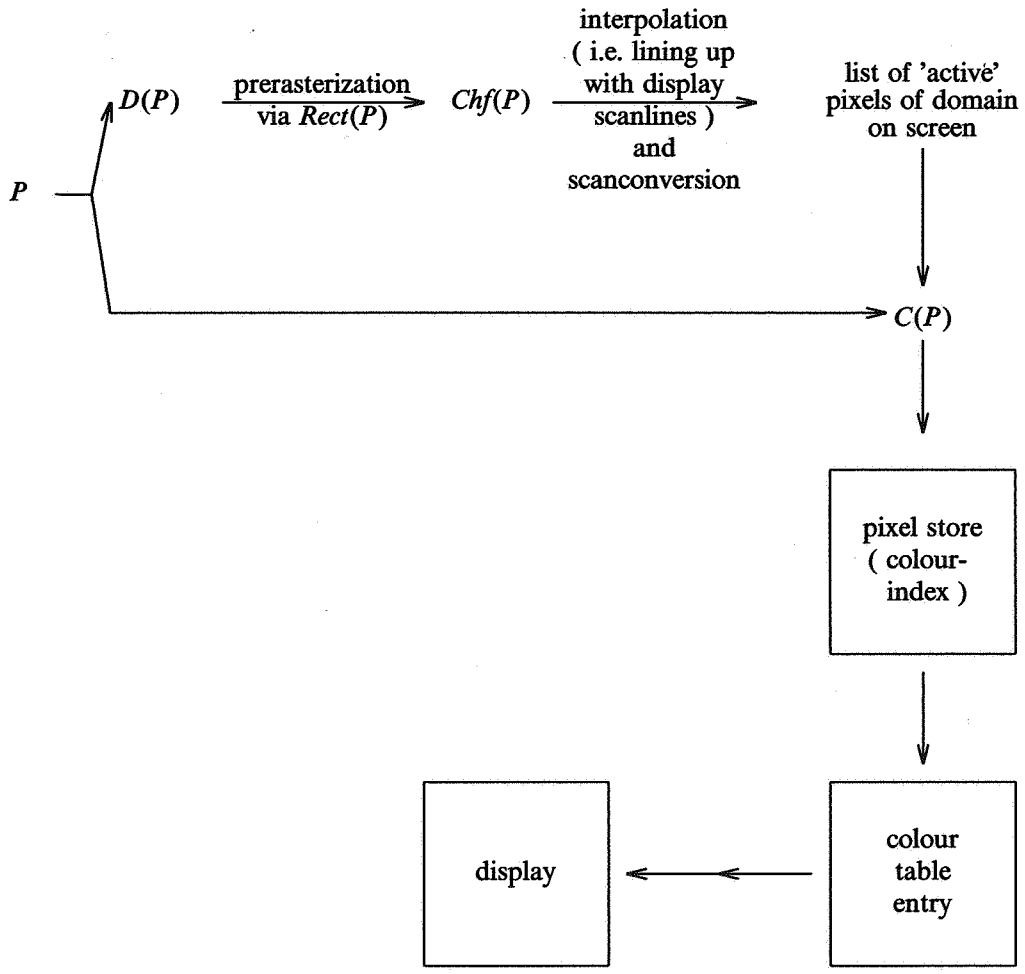
2. Primitive Pattern representation

In ten Hagen, de Ruiter and Trienekens [1] a system for area-oriented picture elements, called **patterns** is proposed. In this section we will summarise this proposal giving some emphasis on the aspects relevant for representation. The proposal defines the pattern functionality and its associated semantics for application programmers. As a result the notation in that paper is different from the one choosen here. In this paper implementation issues are discussed for which mathematical notation is more appropriate.
The representation issues and display list solution will be discussed for 2D primitives. The primitives in [1] are 3D primitives. In section 3 it will be pointed out how 2D and 3D primitives are related and what consequences this has for the implementation.

A pattern primitive P is a pair (D,C) where:
D is a domain function, which specifies a 2D or 3D planar region, and
C is a colourfunction, which assigns a colour to each point of the domain.
e.g.: P = (D,C).
Conceptually what will take place when P is visualized is that D is mapped onto the raster of the device. For each rasterpoint the colour of the corresponding point of D is calculated by C. This colour is assigned to the rasterpoint (pixel).
The system has a number of elementary domain functions, such as: POLYGON, PARALELLOGRAM, and a number of elementary colour functions, such as: SOLID_COLOUR, CELLS and INTERPOLATE. Any combination of domain and colour function is a picture primitive. Domains can have holes or consist of unconnected regions. However, domains are finite.
Pattern primitives can be operands in pattern expressions. The operators for these expressions are set theoretic operators, such as UNION, INTERSECTION, DIFFERENCE etc.. They are called pattern combining operators. These operators are affecting the interior of patterns. The result of, say, a union is a new domain which is the union of the interiors of the domains of the operands. The resulting colour function of the union will be for overlapping parts of the regions a new colour function which is somehow a mixture of the two original functions, or for non-overlapping regions it is the original function of the corresponding pattern. How the colour function will mix colours is determined by a parameter attributed to the union function. This parameter is called the **mix-attribute** for pattern combining operators.
The representation needs to be concerned primarily with pattern representations **after** combining. This means that it is important to know how a representation is constructed for a given pattern expression. It is, however, not required that the representation accomodates fast (real-time) combining. The combining operator is for convenient pattern definition. Pattern manipulations for realizing picture changes in real time do not include combining.
A second category of operators are monadic operators for geometric and colour transformations of patterns. In as far as these are used in the pattern definition phase they have to fulfill similar

requirements: support convenient pattern definition, be able to represent patterns after transformation which need not to be applied in real-time. These requirements are derived from the basic design principle for the area oriented primitives, namely that there will be a definition phase which is distinct from the presentation and manipulation phase.
During the manipulation phase complex patterns originating from a possibly complex pattern expression will be treated as a unit which does not change. For these complex patterns it is necessary to have a representation which allows all kinds of manipulations to support interaction such as: transformations (again!), blinking, removal, identification and grouping.
In the next paragraphs, the prerasterized representation will be described from two points of view.
First, how to construct it from a pattern expression and second, how it can support interaction. The latter will only be done from the point of view of transformations. The other aspects will be dealt with in a future paper.
A representation function must store both domain- and colour function. These will be treated as separate components. The correspondence will be, that at all times, the colour function can supply the colour value of a point in the domain. This property must be maintained for complex patterns and under transformations.
Note that a geometric transformation can affect both domain and colour function. A colour transformation will only affect the colour function.
Eventually a rasterization is envisaged to take place as follows: The domain is mapped onto the raster. As a result all rasterpoints (pixels) belonging to the mapped domain can be associated with a point of the original domain. For each of these points the colourfunction will supply the value.

In scheme:



In order for the colourfunction to be a separate component in the representation it is necessary that the colourfunction is defined for all points in the domain. Moreover, the colour function must be independant of the domain. If, for instance, as a result of intersection the shape of the domain changes drastically, then the colour function still must be defined. This requirement can cause severe performance problems which must be solved by choosing an adequate representation. For instance, under union a colour function may look quite different depending on whether a point is in an overlapping part or not. Colour functions should preferably not have to reconstruct parts of the domains during their evaluation.

The prerasterization method that will be described anticipates that the primitives and the raster they will be mapped onto, will be oriented the same. The orientation is invariant under translation and scaling. This suggests that these operations can be carried out efficiently. Rotation requires a re-

orientation and hence calculation of a completely new representation. However, the original representation will still be of some help when calculating the rotated one.

2.1. Domain representation

The first facility provided by the representation functions for domains is to define the smallest upright rectangle that contains the domain. This is the initial step for rasterization later on.
All subsequent manipulations with the domain must keep the smallest rectangle associated with the domain updated.
This rectangle has on each side at least one intersection with the pattern.(See figure 1). Hence, one way to represent it could be storing the minimum and maximum pattern domain values in the axes directions in a 'lower-left' and an 'upper-right' rectangle point.

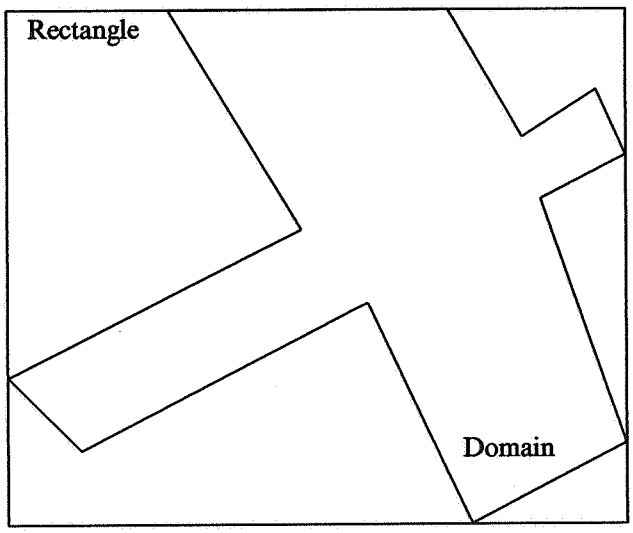


FIGURE 1
Smallest upright rectangle containing the domain

A further step in the direction of rasterization which is provided as part of the representation function is a characteristic function[1] over this smallest rectangle.
The characteristic function is raster independent or (approximately) continuous. When final rasterization will take place, the characteristic function only needs to be evaluated on the rasterpoints. In particular, the colourfunction needs to be evaluated for those points.
The characteristic function can be stored using run-length-encoding (RLE) techniques. By applying linear interpolation techniques the RLE for an arbitrary scanline can be reconstructed from its nearest neighbours in the coding, provided that the precision along the scanline (real numbers) and the distance between the nearest neighbours is within the precision required for the approximation. Figure 2 gives an example of encoding, where coding lines (not all depicted) are choosen to have equal ∇y separation.

1. This characteristic function indicates whether or not an element is a member of the pattern. Its outcome only results in two values: 1 (or true) if that element is part of the pattern and 0 (or false) if it falls outside the pattern.

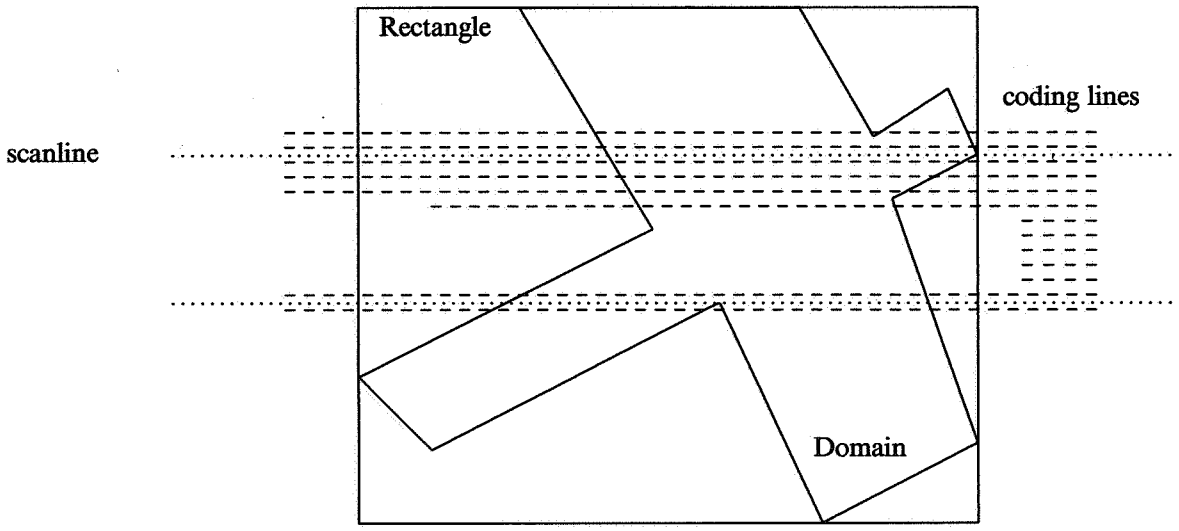


FIGURE 2
Some arbitrary scanlines between their nearest neighbour coding lines

2.2. Colour representation

Colour functions can work with true colours or with colour indices. In the latter case a colour table exists which defines a true colour for each valid index. A colourfunction has at all times access to the colour table. Hence, whenever a true colour is required rather than the index, it can be retrieved. This property will be used in cases of colour mixing, which is done with true colours.
Intermediate true colour values need not be mapped back into the (nearest) corresponding colour index, at least not until real rasterization takes place. This makes it possible to avoid propagation of rounding off errors of intermediate colour values.
Colourfunctions can be generators or pseudo-rasters. Generators are procedures yielding a colour for each point in a domain, pseudo-rasters are two-dimensional arrays of colour cells covering a domain. It is possible to rewrite a generator as a pseudo-raster, but the reverse is not generally the case. Examples of generators are SOLID and RANDOM (See doc [1]).
Combining patterns during patternexpression evaluation may require rewriting of generators or pseudo-rasters (as new pseudo-rasters) in order to have one colourfunction after mixing in cases of overlap.

3. Pattern expression representations

Pattern expressions are defined in more detail in [1]. They are built from operands (primitive patterns or pattern(sub)expressions) and operators.
We will first discuss the representation requirements for the various operands assuming that the operands are primitive patterns. Subsequently the description of the representation functions will be extended to cover sub.expressions as well.
It will turn out that two basic representation strategies can be followed. At this time no decision will be taken about which strategy has preference. However, properties that allow a comparison of the two will be highlighted.

A pattern can be written as

$$P = (D_p, C_p)$$

with

$D(P) = D_p$, the domain of P

and

$C(P) = C_p$, the colourfunction of P.

In the following we will point out that the

quadruple $R (D_p, C_p, Rect_p, Chf_p)$

where D, C are the primitive domain function and the colour function
(C is not necessarily primitive), and Rect and Chf are
the enclosing rectangle and characteristic function respectively
providing redundant information

is the general form of representing a primitive pattern, which suits our purposes.

Depending on domain or colourfunctions these four elements are explicitly present or can be generated when needed. Hence, the system will interface to primitive pattern objects through four elementary functions:

Domain of P	$(D(P))$
Colourfunction of P	$(C(P))$
Rectangle of P	$(Rect(P))$
Characteristic function of P	$(Chf(P))$

3.1. Pattern intersection

Write $P_1 = (D_1, C_1)$ and $P_2 = (D_2, C_2)$.

A pattern expression of the form

$P_1 \cap_{mix} P_2$,

can yield a resulting pattern with the following properties:

By definition ($=_{def}$) : **Domain function**

$$D(P_1 \cap_{mix} P_2) =_{def} D(P_1) \cap D(P_2) \tag{1}$$

By definition : **Colour function**

$$C(P_1 \cap_{mix} P_2) =_{def} C(P_1)_{/D(P_1 \cap P_2)} \underline{mix} \; C(P_2)_{/D(P_1 \cap P_2)} \tag{2}$$

where
$\underline{mix}$ or mix is used for: 'specified colourmixing function'
and
$_{/D(P_1 \cap P_2)}$ for: 'restricted to' the domain of $P_1 \cap P_2$.

From now on we will use the symbol " subscript / " in the special meaning of *restricted to* the then following domain.

Equation (2) can be further simplified to :

$$C(P_1 \cap_{mix} P_2) = (C(P_1) \underline{mix} \; C(P_2))_{/ D(P_1 \cap P_2)} \tag{3}$$

By definition : **general pattern function**

$$P_1 \cap_{mix} P_2 =_{def} (D(P_1 \cap_{mix} P_2) , C(P_1 \cap_{mix} P_2)) \tag{4}$$

Substituting equations (1) and (3) in (4)
and using the property

$$(D, C_{/D}) = (D,C)$$

gives the **resulting pattern intersection equation** :

$$P_1 \cap_{mix} P_2 = (D(P_1) \cap D(P_2), C(P_1) \underline{mix} \; C(P_2)) \tag{5}$$

Based on these definitions and properties we can now formulate how both **enclosing rectangle** and **characteristic function** can be calculated for intersections.

Let Rect(P) be the function which yields the smallest enclosing rectangle of the domain D (of pattern P).
Let Chf(P) be the characteristic function of the domain D (of pattern P), restricted to the rectangle of P.
Then:

$$D(Chf(P)) = Rect(P) \tag{6}$$

For intersection of two primitive patterns the following properties are true:

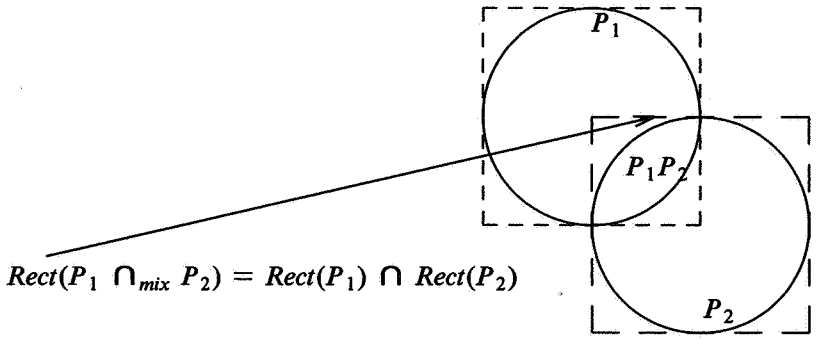
$$Rect(P_1 \cap_{mix} P_2) = Rect(\, D(P_1) \cap D(P_2) \,) \tag{7}$$

with

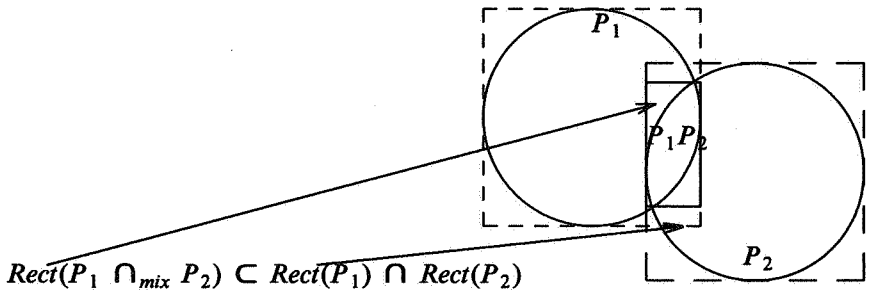
$$Rect(P_1 \cap_{mix} P_2) \subseteq Rect(P_1) \cap Rect(P_2) \tag{8}$$

Note that the smallest enclosing intersection rectangle can be empty even if the enclosing rectangles intersect. See also figure 3.

a: Rectangle around the resultant overlapping domain and the overlap of the rectangles around the two original domains are identical



b: Rectangle around the resultant overlapping domain is smaller then the overlap of the rectangles around the two original domains



c: No overlap of the two original patterns exists although there is an overlap of the rectangles around the two original domains

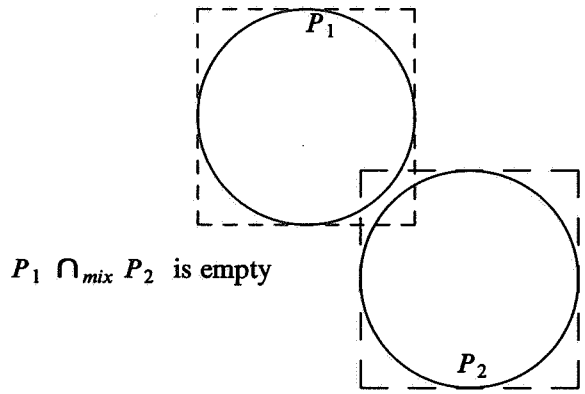


FIGURE 3
Examples of intersection of two patterns

$$Chf(P_1 \cap_{mix} P_2) = Chf(P_1)_{/Rect(P_1 \cap P_2)} \underline{and}\ Chf(P_2)_{/Rect(P_1 \cap P_2)} \tag{9}$$

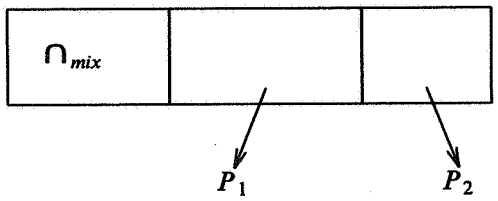
This equation can be rewritten as

$$Chf(P_1 \cap_{mix} P_2) = (\ Chf(P_1) \underline{and}\ Chf(P_2)\ )_{/Rect(P_1 \cap P_2)} \tag{10}$$

The *and* function will be used as follows:

$$\begin{aligned} chf_1 \underline{and}\ chf_2 =_{def} \{ & \\ & \forall x \in D(chf_1) \cap D(chf_2) \mid \\ & \underline{if}\ Chf_1(x) = Chf_2(x) = 1\ \underline{then}\ 1 \\ & \qquad \underline{else}\ 0 \\ \} & \end{aligned} \tag{11}$$

A schematic representation of $R(\ P_1 \cap_{mix} P_2\ )$ is now as follows:



; $Rect(P_1 \cap P_2)$; $Chf(P_1 \cap P_2)$

which can be further simplified to the **pattern intersection representation**

$$\begin{aligned} R(\ P_1 \cap_{mix} P_2\ ) = (& \\ & D(P_1) \cap D(P_2), \\ & C(P_1)\ \underline{mix}\ C(P_2), \\ & Rect(\ D(P_1) \cap D(P_2)\ ), \\ & Chf(P_1)\ \underline{and}\ Chf(P_2) \\ ) & \end{aligned} \tag{12}$$

3.2. Union of patterns

A pattern expression of the form

$P_1 \cup_{mix} P_2$

can yield a pattern with the following properties:

Domain function

$$D(P_1 \cup_{mix} P_2) =_{def} D(P_1) \cup D(P_2) \tag{1}$$

This can be rewritten in an **alternative way** as :

$$\begin{aligned} D(P_1 \cup_{mix} P_2) = {} & D(P_1) \neg (D(P_1) \cap D(P_2)) \cup \\ & D(P_2) \neg (D(P_1) \cap D(P_2)) \cup \\ & D(P_1) \cap D(P_2) \end{aligned} \tag{2}$$

The latter is a decomposition of the resulting domain in three disjunct subdomains, each with a different simple colourfunction.

The symbol $\neg$ will be used for : ' *excluded* the following domain '.

In writing a shorthand for

$D(P_1) \cap D(P_2)$

as

$D(P_1P_2)$

the equation can be simplified to

$$\begin{aligned} D(P_1 \cup_{mix} P_2) = {} & D(P_1) \neg D(P_1P_2) \cup \\ & D(P_2) \neg D(P_1P_2) \cup \\ & D(P_1P_2) \end{aligned} \tag{2'}$$

Using properties and definitions mentioned in this and the previous section the **resultant colourfunction** is :

$$\begin{aligned} C(P_1 \cup_{mix} P_2) = {} & C(P_1)_{/D(P_1) \neg D(P_1P_2)} ; \\ & C(P_2)_{/D(P_2) \neg D(P_1P_2)} ; \\ & (C(P_1) \underline{mix} C(P_2))_{/D(P_1P_2)} \end{aligned} \tag{3}$$

Conclusion:

As a result a union of patterns can be written in two alternative ways:

either as one primitive pattern having a fairly complex (three component) colourfunction;

or as a sequence of three primitive patterns having a simple colourfunction.

We will first give the derivations of both alternative pattern representations.
Following this we will compare the two on the basis of their rectangle- and characteristic functions properties.

general pattern function

$$P_1 \cup_{mix} P_2 =_{def} (D(P_1 \cup_{mix} P_2) , C(P_1 \cup_{mix} P_2)) \tag{4}$$

When substituting the first alternative domain description (equation (1)) and colourfunction (3) in this pattern equation (4) the

first pattern union equation yields :

$$P_1 \cup_{mix} P_2 = (\quad D(P_1) \cup D(P_2) , \quad \begin{array}{l} C(P_1)_{/D(P_1) \neg D(P_1P_2)}; \\ C(P_2)_{/D(P_2) \neg D(P_1P_2)}; \\ (C(P_1) \underline{mix} C(P_2))_{/D(P_1P_2)} \end{array} \quad) \tag{5}$$

However, when substituting the second alternative domain description (equation (2) and colourfunction (3), together with using the property

$$(D, C_{/D} = (D, C) ,$$

in the pattern equation (4) the alternative

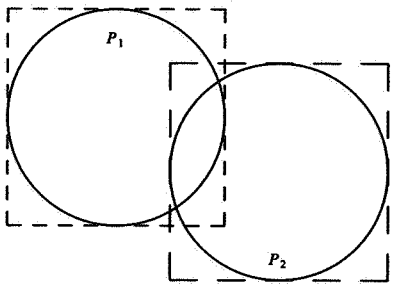
second pattern union equation yields :

$$P_1 \cup_{mix} P_2 = \{ \begin{array}{l} (D(P_1) \neg D(P_1P_2) , C(P_1)) , \\ (D(P_2) \neg D(P_1P_2) , C(P_2)) , \\ (D(P_1P_2) , C(P_1) \underline{mix} C(P_2)) \end{array} \} \tag{6}$$

The two alternative representations are depicted for one intersection in figure 4.

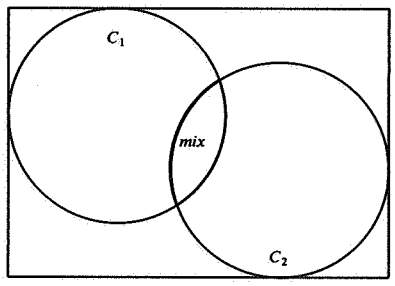
From now on we will use the terms case (1) and case (2) to point to
alternative 1, respectively 2, of the pattern representation.

Two patterns P_1 and P_2 with their resp. enclosing rectangles



Case (1):

One pattern with 'growing' rectangle around one (larger) domain with one (three-component) colourfunction



Case (2):

Three disjunct patterns I, II, III, each with 'shrinking' rectangle and domain and one 'simple colourfunction per pattern

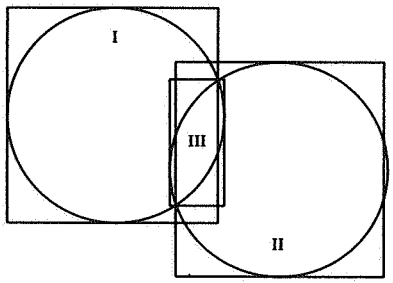


FIGURE 4
Example of a union of two patterns

The rectangles associated with the resulting patterns have the following properties:

Case (1);

$$Rect(P_1 \cup_{mix} P_2) = Rect(\, D(P_1) \cup D(P_2) \,) \tag{7}$$

Case (2);

$$\begin{aligned} Rect(P_1 \cup_{mix} P_2) = {} & Rect(\, D(P_1) \neg D(P_1P_2) \,), \\ & Rect(\, D(P_2) \neg D(P_1P_2) \,), \\ & Rect(\, D(P_1P_2) \,) \end{aligned} \tag{8}$$

with

$$\begin{aligned} Rect(\, D(P_1) \cup D(P_2) \,) \supseteq (\; & Rect(\, D(P_1) \neg D(P_1P_2) \,) \cup \\ & Rect(\, D(P_2) \neg D(P_1P_2) \,) \cup \\ & Rect(\, D(P_1P_2) \,) \\ ) \end{aligned} \tag{9}$$

The second alternatives set of rectangles can be considerably smaller in area than the first alternatives rectangle (see figure 4).

For case (1) the characteristic function is also fairly complex:

$$Chf(P_1 \cup_{mix} P_2) = Chf(P_1) \underline{or} \; Chf(P_2) \tag{10}$$

The *or* function will be used as follows:
(Keep in mind that $D(\, Chf(P) \,) = Rect(P)$)

$$
\begin{aligned}
Chf(P_1)\ \underline{or}\ Chf(P_2) =_{def} \{ & \\
& \forall x \in Rect(\, D(\, Chf(P_1)\,) \cup D(\, Chf(P_2)\,)\,) \mid \\
& \quad \underline{if}\ x \in (\, D(\, Chf(P_1)\,) \cap D(\, Chf(P_2)\,)\,) \\
& \quad\quad \underline{then}\ Chf(P_1) \vee^1 Chf(P_2) \\
& \quad \underline{else}\ \underline{if}\ x \in D(\, Chf(P_1)\,)\ \underline{then}\ Chf(P_1) \\
& \quad \underline{else}\ \underline{if}\ x \in D(\, Chf(P_2)\,)\ \underline{then}\ D(\, Chf(P_2)\,) \\
& \quad\quad \underline{else}\ 0 \\
\} &
\end{aligned}
\tag{11}
$$

In case (2) the characteristic function is

$$
\begin{aligned}
Chf(P_1 \cup_{mix} P_2) = \quad & Chf(P_1)_{/Rect(\, D(P_1) \setminus D(P_1P_2)\,)}\ ; \\
& Chf(P_2)_{/Rect(\, D(P_2) \setminus D(P_1P_2)\,)}\ ; \\
& (\, Chf(P_1)\ \underline{and}\ Chf(P_2)\,)_{/Rect(\, D(P_1P_2)\,)}
\end{aligned}
\tag{12}
$$

As a result the **pattern union representation** can be written as:

Case (1)

$$
\begin{aligned}
& R(\, P_1 \cup_{mix} P_2\,) = \\
& \quad (\, D(P_1) \cup D(P_2)\ , \\
& \qquad C(P_1)_{/D(P_1) \setminus D(P_1P_2)}\ ;\ C(P_2)_{/D(P_2) \setminus D(P_1P_2)}\ ;\ (\, C(P_1)\ \underline{mix}\ C(P_2)\,)_{/D(P_1P_2)}\ , \\
& \qquad Rect(\, D(P_1) \cup D(P_2)\,)\ , \\
& \qquad Chf(P_1)\ \underline{or}\ Chf(P_2) \\
& \quad)
\end{aligned}
\tag{13}
$$

[1] $Chf(P_1)(x) \vee Chf(P_2)(x) =_{def}$
$\underline{if}\ Chf(P_1)(x) = Chf(P_2)(x) = 0\ \underline{then}\ 0\ \underline{else}\ 1\ .$

Case (2)

$$
\begin{aligned}
&R(P_1 \cup_{mix} P_2) = \\
&\{ \\
&\quad (D(P_1) \neg D(P_1P_2),\ C(P_1),\ Rect(D(P_1) \neg D(P_1P_2)),\ Chf(P_1)), \\
&\quad (D(P_2) \neg D(P_1P_2),\ C(P_2),\ Rect(D(P_2) \neg D(P_1P_2)),\ Chf(P_2)), \\
&\quad (D(P_1P_2),\ C(P_1)\ \underline{mix}\ C(P_2),\ Rect(D(P_1P_2)),\ Chf(P_1)\ \underline{and}\ Chf(P_2)) \\
&\} \\
&\qquad = \{R(P_1 \neg P_2),\ R(P_2 \neg P_1),\ R(P_1 \cap_{mix} P_2)\}
\end{aligned}
\tag{14}
$$

Note that the colourfunction only needs to be given for the pattern domain.
Similarly, that the characteristic function only needs to be defined for the rectangle surrounding the domain.
For instance, the first two characteristic functions in (14) are identical to the Chf for P_1 and P_2 respectively but they have smaller domains. This may lead to fewer storage locations for their representation.
Based on the observation that case (2) leads to smaller rectangles and simpler colourfunctions we give preference to case (2).
However, this introduces the need for considering lists of primitive patterns as operands rather than single ones. We only need to consider lists containing disjunct patterns!
All operations will have to maintain the disjunctness property for lists of patterns.

3.3. Pattern reduction

The pattern reduction operator is used to maintain the disjunction property.
Let P,Q be patterns such that $P \supset Q$.
Then

$$
\begin{aligned}
R(P \underline{\mathrm{r}}\, Q) \quad &=_{def} (\\
&\quad D(P) \neg D(Q), \\
&\quad C(P)_{/D(P) \neg D(Q)}, \\
&\quad Rect(D(P) \neg D(Q)), \\
&\quad Chf(P)_{/D(P) \neg D(Q)} \\
&)
\end{aligned}
\tag{1}
$$

(pronounce P reduced by Q)

This can be rewritten as:

$$R(P \underline{\mathrm{r}}\, Q) = (D(P) \neg D(Q),\ C(P),\ Rect(D(P) \neg D(Q)),\ Chf(P)) \tag{2}$$

Hence, the reduction operator removes the domain of Q from the domain of P, called the reduced domain of P. The colour function, rectangle function and characteristic function now apply only to the reduced domain.
Using the reduction operator we can *reformulate* the union operation for patterns for case (2) as given in the previous section.

$$P_1 \cup_{mix} P_2 = \{ (P_1P_2)_{\underline{mix}},\ P_1 \underline{\mathrm{r}}\, P_1P_2,\ P_2 \underline{\mathrm{r}}\, P_1P_2 \} \tag{3}$$

where P_1P_2 is a shorthand notation for $P_1 \cap P_2$.

This makes immediately clear what happens:
The result is written in three disjunct patterns, the first one contains the domain for which colour-function mixing has to be done. This domain is taken away from the two original domains.

Now the union of a list of disjunct patterns with a new one can proceed by calculating the union of the new pattern with each of the patterns from the list, thereby reducing the new pattern by the intersection of each pattern from the list:

$$
\begin{aligned}
\cup_{mix}(P_1, P_2, P_3) = \{ & \\
& (P_1P_2P_3)_{mix} , \\
& \{ (P_1P_2)_{mix}, (P_1P_3)_{mix}, (P_2P_3)_{mix} \} \underline{r}\, P_1P_2P_3 , \\
& P_1 \underline{r}\, (P_1P_2, P_1P_3) , \\
& P_2 \underline{r}\, (P_1P_2, P_2P_3) , \\
& P_3 \underline{r}\, (P_1P_3, P_2P_3) \\
\} &
\end{aligned}
\tag{4}
$$

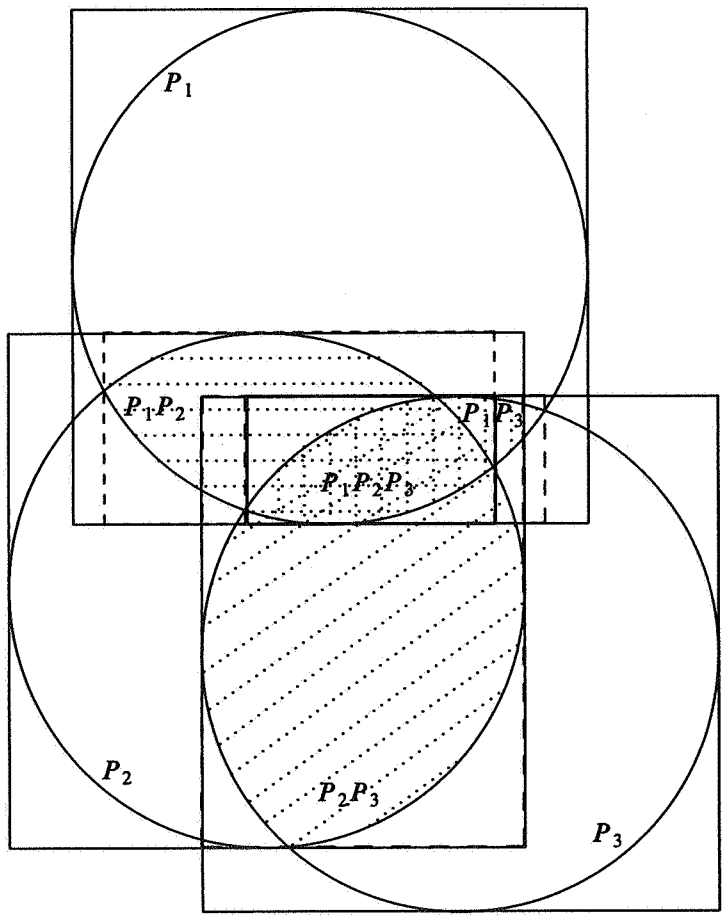
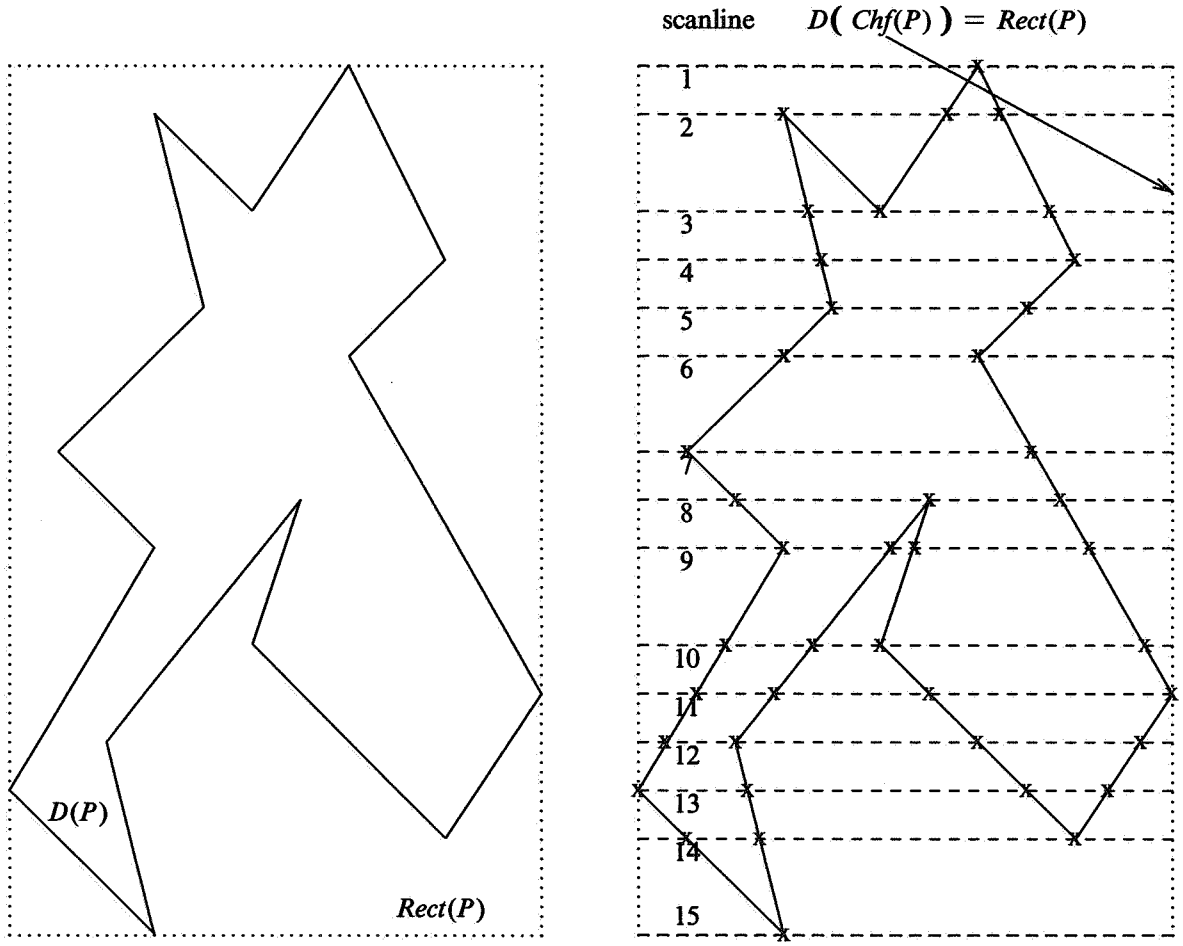


FIGURE 5
Union of three patterns, resulting in seven 'new' disjuncted patterns

4. Run length encoding for the characteristic function

The representation of the domains in the prerasterized form is similar to run length encoding methods. The coding here however, must be raster independent.
Differences are that the scan lines in our encoding can be on arbitrary y-values (but always in x-direction) and the number of scanlines used is minimal.
For instance, a POLYGON primitive with n points (specifying the boundary) will have at most n scanlines represented in the encoding characteristic function. Hence, the coding skips over all scanlines which can be reconstructed unambiguously. In this way transformations can be applied and domains whose "rasters" are not lined up can be matched. The chf provides a uniform representation for domains which can be taken instead of the domain itself when needed.
An example is depicted in figure 6.



Polygon of 15 points
and its rectangle

Chf of the polygon:
15 scanlines, plus scanpoints x

FIGURE 6

In order to be able to reconstruct the pattern lines from the chf while interpolating or during display rasterization it is necessary to add more details to the coordinates of the rasterpoints, namely the 'type' of the intersecting line at each rasterpoint.

The 'type' is indicated by means of lettersymbols.
The lettersymbols and their meanings are the following:
c: intersection of a continuously polygon line, i.e. with same slope between its two nearest neighbour scanlines.
C: intersection of a continuously polygon line, i.e. with different slopes from the previous scanline to the current one, and from the current one to the next scanline.
B: 'bottom' intersection point, i.e. from here two connecting lines only exists with the previous scanline.
T: 'top' intersection point, i.e. from here two connecting lines only exists with the next scanline.
H1 - H2: 'horizontal' intersection points. H1 can have a line connected to either the previous or next scanline. The same is true for H2. ".LP Note that for the types C, B, T, H1, H2 the intersection of the polygon and the scanline is a specified boundary point!

The different intersection types are represented in figure 7.

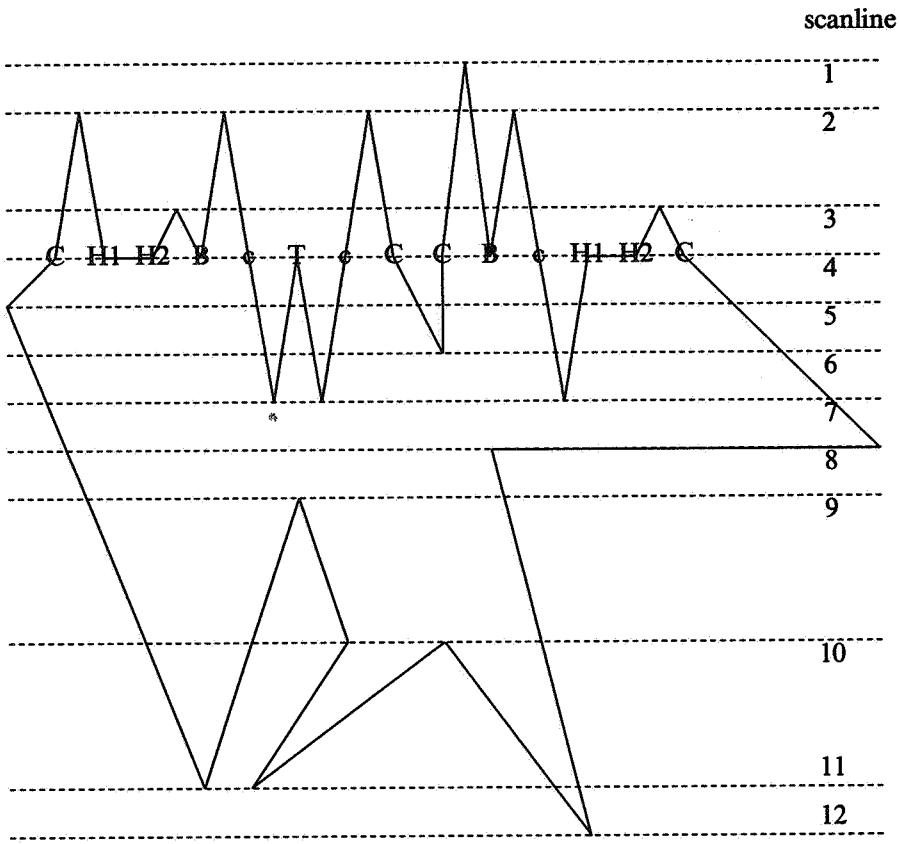


FIGURE 7
POLYGON of 31 points, only 12 scanlines.
Characteristic function needs extra 'type' information.
Different types of intersecting polygon lines are depicted in scanline 4.

By way of example the representation of POLYGONal domains will be specified. The treatment of

most other domains is similar.
First a rectangle is given by four real values:
minx, miny, ,maxx, maxy.
e.g. in C(GKS)-notation:

```
typedef float    Real;

typedef  struct {
         Real      minx, miny, maxx, maxy;
         } Rectan;
```

Next a sequence of scanlines is given as follows:

```
typedef struct {
        Int        nrofsl;    /* number of scanlines */
        Scanline   *crle_sl;  /* list of scanlines */
        } Crle;               /* continuous run length encoding */
```

with
typedef int Int;

```
typedef struct SCANLINE {
        Real                deltay;       /* relative to miny */
        Int                 nrofsp;       /* number of scanpoints */
        Scanpoint           *scanl_sp;
        struct SCANLINE     *scanl_next;
        } Scanline;
```

```
typedef  struct SCANPOINT {
         Sptype                Sp_type;      /* type of scanpoint */
         Real                  deltax;       /* relative to minx */
         struct SCANPOINT      *scanp_next;
         } Scanpoint;
```

```
typedef enum { c, C, B, T, H1, H2 }    Sptype;
```

Given this representation it now matters how easily a number of elementary operations can be carried out.
The first group is linear transformations, which can be dealt with as follows:

- translation only affects Rect
- scaling affects all scanlines of Chf, including the points defining the scanlines themselves, besides Rect; but only as multiplication factor(s)
- rotation requires a complete new calculation of Chf and Rect, however, the current already scan-converted representation can be dealt with more easily than the general case.

The second group is made up by the elementary actions used in intersection calculations, such as

generating common scanlines and merging scanlines. The same elementary actions are used in conversion to discrete rasters.
An important strategic question is, to what extent pathological cases must be identified and possibly removed. Many applications would for instance become much more efficient if domains were guaranteed to be connected. Maintenance of this property under intersection is one of the cases to be studied.

5. Relation between 2D and 3D area's

The 3D primitives of [1] are all planar primitives. This means that the area's can be described as 2D area's relative to a local coordinate system in the plane. If a mapping from 3D to 2D is taken following the GKS-3D model (see [3]) there will be an intermediate representation where the x-y values under projection will not change. If the local coordinate system is choosen such that it coincides with x and y directions, they can be treated (but for z-values) as 2D area's.
In a future paper it will be described how it can be arranged that the pattern expression will be generated in this representation. This implies that the x-y part of the geometry can be described as for the 2D case. The z-value wil only affect the colour- and mix- functions.
As a result we only have to consider the 2D case for domains.
Projections, hidden surface effects and the like can be represented as part of the colourfunction expressions.

6. References

[1]P.J.W. ten Hagen, M.M. de Ruiter and C.G. Trienekens (1985). *Raster Graphics Facilities (RGF) in Programming Languages,* Preliminary report CWI, department of Interactive Systems.

[2](1984). *Information Processing - Graphical Kernel System (GKS) - Functional Description,* ISO DIS 7942 (draft).

[3](1985). *Information Processing Systems - Computer Graphics - Graphical Kernel System for Three Dimensions (GKS-3D) - Functional Description,* ISO/TC97/SC21/WG-2 N277 Rev (draft).

Contents