A drawing system based on a formal picture concept

A drawing system based on a formal picture concept.

by Rens Kessener, Herman Willemsen
(Eindhoven University of Technology, Computing Centre).

Summary.

A definition of the concept "PICTURE" is used to implement a drawing system on a Burroughs B6700 of the Eindhoven University of Technology. This concept contains a set of elements which may be PICTURES as well. A number of procedures has been constructed based on the definition of the PICTURE concept.
This procedure package covers many application areas and is independent of the drawing device.
The constructed drawing may be mapped on any device suitable to draw (e.g. a storage tube, a plotter, a printer).

Introduction.

A drawing (or a FYSICAL PICTURE) is considered to exist of the definition of a PICTURE. This PICTURE is mapped on a device that is suitable to draw. Therefore a concept "PICTURE" is necessary and a number of procedures building the PICTURE in order to draw it (i.e. a procedure package).
The aims which must be fulfilled by the concept are:

- the concept must be structured;
- the concept must be easy to manipulate and easy to understand.

For the procedure package it is required that:

- the package covers all disciplines that appear at a university (e.g. the Eindhoven University of Technology);
- the package is independent of the device;
- the use of the package involves operator interference as little as possible;
- it is possible to generate several drawings simultaneously i.e. within a program as well as in a multiprogramming environment.

To reach the aim of simultaneous generation of drawings, it is necessary to create a sort of backup mechanism. In addition, to be device independent the information must be such that it can be interpreted by a drawing machine for a specific device.

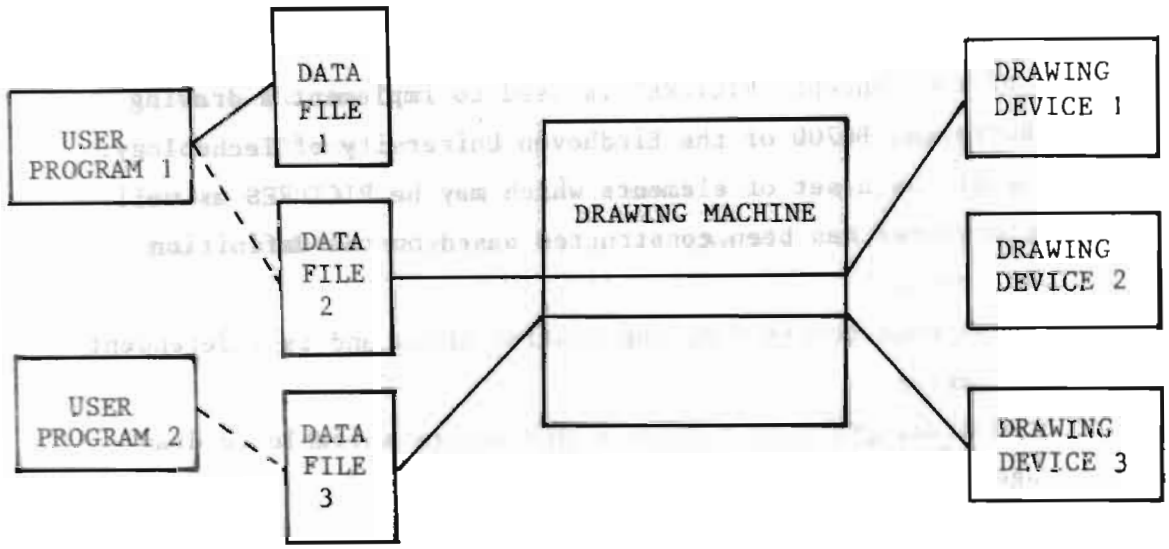

Figure 1: Schema of drawing system

The user generated PICTURE info may be interpreted by a drawing machine for a specific device.

In the succeeding sections we will explain the PICTURE concept.
We will discuss some differences between the concept and the actual implementation.
Finally we will discuss the procedure package and the software drawing machine.

The PICTURE concept.

In order to formulate the PICTURE concept we draw distinction between PICTURES and IMAGES.

An IMAGE consists of a set of elements. Each element is determined by its type and its position. The position is expressed in units of R_2 (two dimensional) or R_3 (three dimensional). By mapping an IMAGE into another IMAGE the former becomes a PICTURE.
We will explain this map in more detail later.

More formally we may define:

< image > ::= < open bracket > < set of elements > < close bracket >

< set of elements > ::= < empty > | < element > < set of elements >

We already saw that an IMAGE can be mapped into another IMAGE and then we call it a PICTURE.

So a PICTURE can be an element of an IMAGE (its type is IMAGE, its position is indicated by the map).

We distinguish only one other element type: PRIMITIVE.

A PRIMITIVE may have the following kind:

- dot
- point
- straight line piece
- curved line piece
- text
- axis

Now let us explain some terms more explicitly:

MAP.

A map consists of:

1. scaling
   i.e. $\underline{x} \longrightarrow \underline{y}$ such that $\underline{y} = \lambda\underline{x}$ (λ = scaling factor)
2. translation
   i.e. $\underline{x} \longrightarrow \underline{y}$ such that $\underline{y} = \underline{t} + \underline{x}$ ($\underline{t}$ is translation vector)
3. rotation
   i.e. $\underline{x} \longrightarrow \underline{y}$ such that $\underline{y} = R\underline{x}$ (R is rotation matrix).

The map is performed by means of a matrix multiplication. The way this matrix is constructed is very dependent of the implementors point of view. (See [3], [4])

WINDOW.

A window is defined by specifying of a rectangle in the current space. All elements, or parts of elements that appear outside that window will never be visualized.

PRIMITIVES.

Dot — A dot is indicated by its coordinates.
Its representation is a simple dot.

Point — A point is indicated by its coordinates.
Its representation is a special character (⧗, *, + etc.).

Straight line piece

A straight line piece is indicated by the coordinates of its extreme points.
Its representation may be fully drawn, dotted, dashed or dot-dashed.

Curved line piece

A curved line piece is indicated by the coordinates of its extreme points and the derivatives in these points. This holds for 2D. For 3D it probably must be indicated by a parameter form. For 2D a part of a third degree polynomial is drawn through the indicated points in which the derivatives agree with the indicated values.
Its representation may be fully drawn, dotted, dashed or dot-dashed.

Text — A text is indicated by its extreme points and its content.
Its representation may be italicized.

Axis — An axis is indicated by its extreme points.
Optionally numbering and comment may be specified.

Remarks:

Text — The text was our main problem and actually it still is. Because by mapping an IMAGE into another IMAGE the ultimate visible text may be considerably deformed.
As we consider a text to be comment which should be readable we defined a character to be represented in a square when it is drawn in the real world.

Axis — For axis a comparable remark as for text holds.
We consider an axis to be a piece of comment which should be readable in its final representation.

Curved line piece

The choice of a third degree polynomial approximation is arbitrary. Any other approximation should suit as well.

FINAL FORMAL DEFINITION.

```
< image >              ::= < open bracket > < set of elements > < close bracket >
< set of elements >    ::= < empty >|< element > < set of elements >
< element >            ::= < primitive >|< picture >
< picture >            ::= < map > < window > < image >
< window >             ::= < empty >|< row of coordinates >
< map >                ::= < translation > < scaling > < rotation >
< primitive >          ::= < dot >|< point >|< straight line piece >|
                           < curved line piece >|< text >|< axis >
< dot >                ::= < row of coordinates >
< point >              ::= < row of coordinates > < point type >
< straight line piece >
                       ::= < row of coordinates > < row of coordinates >
                           < representation >
< curved line piece >
                       ::= < row of coordinates > < slope value >
                           < row of coordinates > < slope value > < representation >
< representation >     ::= < fully drawn >|< dotted >|< dashed >|< dot-dashed >
< text >               ::= < row of coordinates > < row of coordinates >
                           < italicity > < string >
< axis >               ::= < row of coordinates > < row of coordinates >
                           < number of intervals >|< row of coordinates >
                           < row of coordinates > < number of intervals >
                           < numbering information >|
                           < row of coordinates > < row of coordinates >
                           < number of intervals > < numbering information >
                           < string >
```

The final appearance of the items, that are not elaborated, is implementor dependent.

MAPPING THE IMACE ONTO A VISIBLE MEDIUM.

So far the IMAGE is still a virtual thing. By mapping the IMAGE onto the real world (say a piece of paper or a screen) a final map must be performed and the IMAGE can be made visible, i.e. the FYSICAL PICTURE. As the real world, onto which the IMAGE is mapped, is limited, this final map has to be slightly different than the MAP described before. That is why we call this final map the WINDOWMAP.

WINDOWMAP.

A windowmap consists of:

1. a declaration of an area of the real world that will be visible;
2. a map of the image onto the real world.

ad 1. This declaration may be considered as the piece of paper on which the IMAGE will be visualized.
This holds for two dimensions.
For three dimensions there must be a definition of the point of view and the plane on which the IMAGE will be projected as well.

ad 2. The map must describe the relation between the points in the real world and the points of the IMAGE.

The windowmap is performed by indicating a rectangle in the "visible" world and indicating a rectangle in the images world. The content of the rectangle in the IMAGE will be mapped onto the rectangle in the real world.

It is obvious that for 3D a block in the images world must be defined, a viewing point and/or a plane onto which the block content will be mapped.

Conclusion.

We didn't mention any implementation yet, nor any data structure in which the image is represented. This is very dependent of the application area. For example the data structure and procedure package for Graphical Displays will be considerably different from those for plotters.
But it must be quite easy to develop a drawing machine that is able to interpret any data structure for any device.

Implementation.

Until now we implemented this concept for plotters, based on a sequential data structure and only for two dimensions.

The implementation consists of two parts:

1. a package of procedures by which a user may generate his FYSICAL PICTURE;
2. a drawing machine which interprets the user-generated FYSICAL PICTURE and plots it on a device.

The FYSICAL PICTURE must be generated from the outer level down to the inner level of IMAGES.

This means the first item of the sequential data structure must be the outer (window) map. The last item must be a close bracket of the outer IMAGE. In between the items must be elements.

The unit that is presented to the drawing machine must be a data file which contains a FYSICAL PICTURE. This means that for such a data file the device kind must be determined already.

Also it is possible to generate an IMAGE. But this cannot be presented to the drawing machine.

It only may appear as an element of another IMAGE.

Devices for which the implementation is suitable now:

CALCOMP C565 drumplotter, 11 inch, 300 steps/sec., 0.1 mm/step.

CALCOMP C1136 drumplotter, 30 inch, 2600 steps/sec., 0.05 mm/step.

Lineprinter, 30 cm., 5 cm/step.

The backup mechanisme.

Because there is no need to retreive picture elements in a plotting system we decided to use a simple sequential data structure with fixed recordlength.
Each record contains a primitive.
Primitives containing a textstring may cover several records.
Assignment to a drawing device is performed by assigning a specific title to the data file.
For example assignment to C565 is performed by a title which is prefixed by "PLOT11", to C1136 by a title prefixed by "PLOT30".
Because it is not possible to assign windowmap information to data file attributes, the first record must contain that windowmap.
When the file has been generated and locked, it will be drawn. The drawing machine will look for data files which obviously contain a fysical picture (identifiable by its prefix) and subsequently the drawing machine will interpret the data file and map it onto the drawing device.

User procedures.

All user procedures are written in BEATHE (Burroughs Extended Algol for EINDHOVEN UNIVERSITY OF TECHNOLOGY).
All procedure headings are described as well as a short description of the function performed by it.
The user must declare one or more files in which he will generate the picture(s).
The picture file is one of the parameters of most of the procedures, so the user is able to generate simultaneously as much pictures as he wants to.

Only the most frequently used and/or necessary procedures are described.

Map and brackets.

```
'PROCEDURE' WINDOWMAP(FILE, WINDOWXORIGIN, WINDOWYORIGIN, WINDOWNIGIROX,
                      WINDOWNIGIROY, XORIGIN, YORIGIN, NIGIROX, NIGIROY);
'VALUE' WINDOWXORIGIN, WINDOWYORIGIN, WINDOWNIGIROX, WINDOWNIGIROY,
        XORIGIN, YORIGIN, NIGIROX, NIGIROY;
'REAL' WINDOWXORIGIN, WINDOWYORIGIN, WINDOWNIGIROX, WINDOWNIGIROY,
       XORIGIN, YORIGIN, NIGIROX, NIGIROY;
'FILE' FILE;
```

As already mentioned the windowmap is performed by indicating a rectangle in the real world and a rectangle in the images world.
The latter will be mapped onto the former.
The indication is performed by two vertexes of the rectangle diagonally to each other.
The matrix of the map is calculated such that:

- (XORIGIN, YORIGIN) of the image is mapped to (WINDOWXORIGIN, WINDOWYORIGIN);
- (NIGIROX, NIGIROY) of the image is mapped to (WINDOWNIGIROX, WINDOWNIGIROY);
- (XORIGIN, NIGIROY) of the image is mapped to (WINDOWXORIGIN, WINDOWNIGIROY).

```
'PROCEDURE' MAPIMAGE(FILE, X1ORIGINAL, Y1ORIGINAL, X2ORIGINAL, Y2ORIGINAL,
                     X1IMAGE, Y1IMAGE, X2IMAGE, Y2IMAGE, XROTATION, YROTATION,
                     ALPHA);
'VALUE' X1ORIGINAL, Y1ORIGINAL, X2ORIGINAL, Y2ORIGINAL, X1IMAGE, Y1IMAGE,
        X2IMAGE, Y2IMAGE, XROTATION, YROTATION, ALPHA;
'REAL' X1ORIGINAL, Y1ORIGINAL, X2ORIGINAL, Y2ORIGINAL, X1IMAGE, Y1IMAGE,
       X2IMAGE, Y2IMAGE, XROTATION, YROTATION, ALPHA;
'FILE' FILE;
```

As we think the most convenient way to imagine a map is thinking about two rectangles which at least must be mapped onto each other and potentially rotated after that.
The image that is to be mapped we call the original. The image into which the original is mapped, is indicated by image.(X1ORIGINAL, Y1ORIGINAL) and (X2ORIGINAL, Y2ORIGINAL) denote two vertexes of the rectangle in the original, diagonally to each other.
(X1IMAGE, Y1IMAGE) and (X2IMAGE, Y2IMAGE) denote in the same way a rectangle in the image into which (X1ORIGINAL, Y1ORIGINAL), (X2ORIGINAL, Y2ORIGINAL) will be mapped.
To be able to calculate uniquely the mapping matrix it is assumed that (X1ORIGINAL, Y2ORIGINAL) is mapped to (X1IMAGE, Y2IMAGE).
After that the image of the original is rotated around a certain point (XROTATION, YROTATION) in the current image over ALPHA degrees.

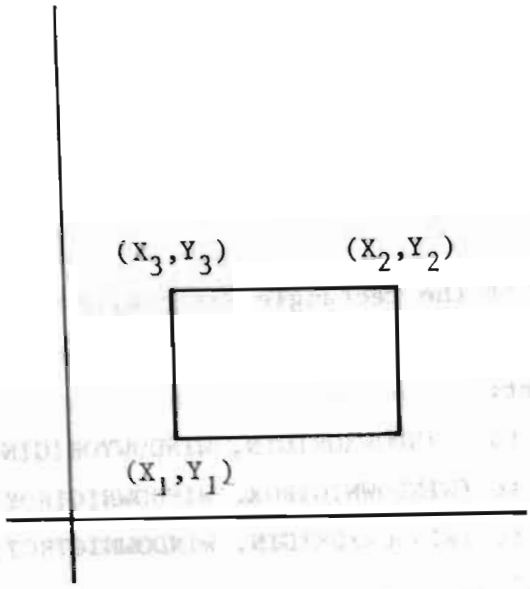

fig. 2a Image to be mapped indicated by original.

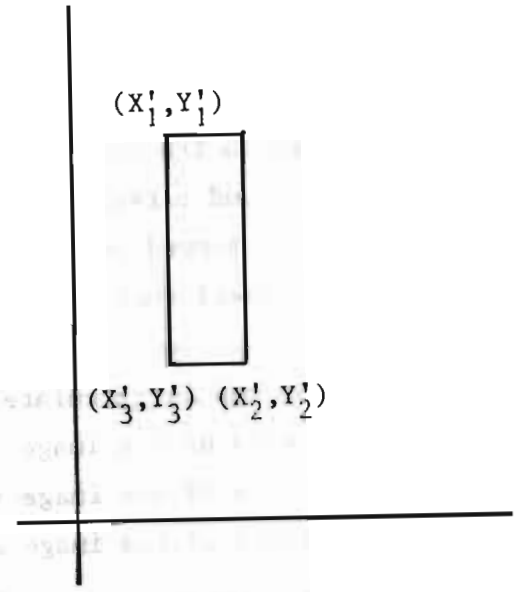

fig. 2b The image of the originals rectangle in the current image before rotation.

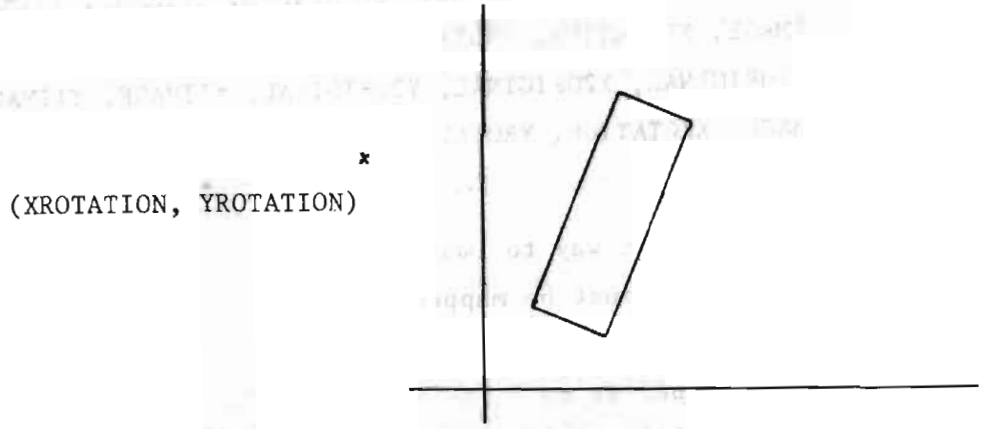

fig. 2c The image of the originals rectangle in the current image after rotation.

'PROCEDURE' CLOSEBRACKET(FILE);
'FILE' FILE;
This procedure writes the close bracket of the last opened image into the file.

'PROCEDURE' WINDOW(FILE, WINDOWXMIN, WINDOWYMIN, WINDOWXMAX, WINDOWYMAX);
'VALUE' WINDOWXMIN, WINDOWYMIN, WINDOWXMAX, WINDOWYMAX;
'REAL' WINDOWXMIN, WINDOWYMIN, WINDOWXMAX, WINDOWYMAX;
'FILE' FILE;
This procedure defines a window in the current space.
If it appears, it must appear immediatly after a call of MAPIMAGE.

```
'PROCEDURE' INSERTANIMAGE(FILE, IMAGESFILETITLE, X1ORIGINAL, Y1ORIGINAL,
                          X2ORIGINAL, Y2ORIGINAL, X1IMACE, Y1IMAGE,
                          X2IMAGE, Y2IMAGE, XROTATION, YROTATION, ALPHA);
'VALUE' X1ORIGINAL, Y1ORIGINAL, X2ORIGINAL, Y2ORIGINAL, X1IMACE, Y1IMAGE,
        X2IMAGE, Y2IMAGE, XROTATION, YROTATION, ALPHA;
'REAL' X1ORIGINAL, Y1ORIGINAL, X2ORIGINAL, Y2ORIGINAL, X1IMAGE, Y1IMAGE,
       X2IMAGE, Y2IMAGE, XROTATION, YROTATION, ALPHA;
'FILE' FILE;
'STRING' IMAGESFILETITLE;
```

This procedure only is handy to insert an already existing image, stored in the file titled by IMAGESFILETITLE into an image.
The file must contain a complete image, the close bracket inclusive.
The variables that contain "original" denote a rectangle in the image to be mapped.
The variables that contain "image" or "rotation" denote points in the image into which is mapped.
See also the procedure MAPIMAGE.

Primitives.

```
'PROCEDURE' DOT(FILE, X, Y);
'VALUE' X, Y;
'FILE' FILE;
'REAL' X, Y;
```

The point (X, Y) is dotted.

```
'PROCEDURE' POINT(FILE, X, Y, POINTTYPE, POINTHEIGHT);
'VALUE' X, Y, POINTTYPE, POINTHEIGHT;
'FILE' FILE;
'REAL' X, Y, POINTHEIGHT;
'INTEGER' POINTTYPE;
```

The point (X, Y) is marked with a special symbol. There are 16 different marking symbols, numbered from 0 up to 15 inclusive. The value of POINTTYPE determines which symbol is placed on (X, Y) on the output device. POINTHEIGHT indicates the height of the special symbol expressed in the current data units in Y direction.

```
'PROCEDURE' TEXT(FILE, XFROM, YFROM, XTO, YTO, ITALICITY, STRING);
'VALUE' XFROM, YFROM, XTO, YTO, ITALICITY;
'FILE' FILE;
'REAL' XFROM, YFROM, XTO, YTO, ITALICITY;
'STRING' STRING;
```

This procedure defines a text as an image element. The position of the text is fixed by its starting-point and end-point, that is: the lower left point of the first character and the lower right point of the last character of the text. Each character is written in a parallelogram ABCD. The base AB lies in the direction of writing, the height AE of the character is (on paper) equal to the length of the base of one character. The italicity is given by ITALICITY, the angle α, in degrees, measured clockwise, between the normal on the base (AE) and AD (see figure 3).

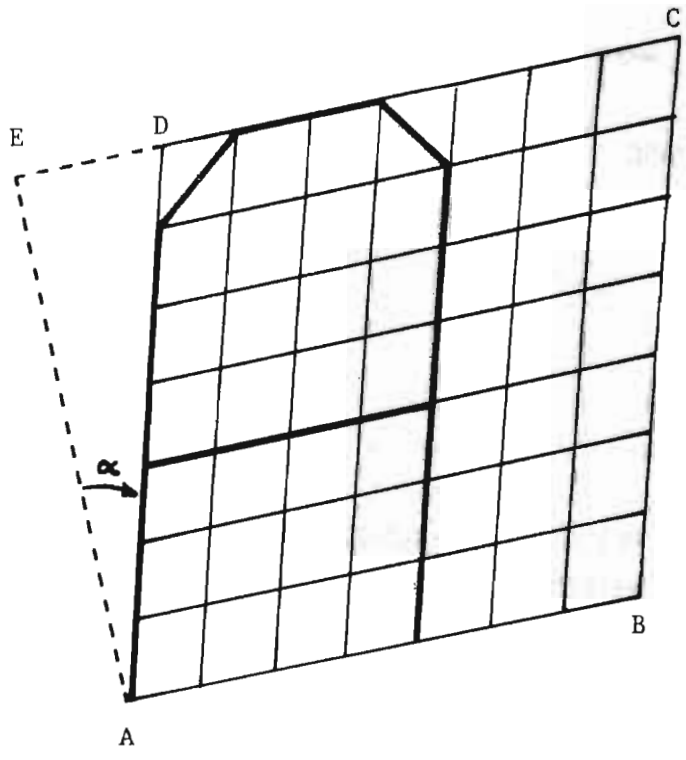

Figure 3. This figure shows how a character will be drawn.

```
'PROCEDURE' STRAIGHTLINEPIECE(FILE, XFROM, YFROM, XTO, YTO, LINEMODE);
'VALUE' XFROM, YFROM, XTO, YTO, LINEMODE;
'FILE' FILE;
'REAL' XFROM, YFROM, XTO, YTO;
'INTEGER' LINEMODE;
```

This procedure defines a straight line piece between (XFROM, YFROM) and (XTO, YTO). The representation of the straight line piece is given by the value of LINEMODE. Possible representations are: traced, dashed, dotted and dashed-dotted.

```
'PROCEDURE' CURVEDLINEPIECE(FILE, XFROM, YFROM, SLOPEFROM, XTO, YTO,
                            SLOPETO, LINEMODE);
'VALUE' XFROM, YFROM, SLOPEFROM, XTO, YTO, SLOPETO; LINEMODE;
'FILE' FILE;
'REAL' XFROM, YFROM, SLOPEFROM, XTO, YTO, SLOPETO;
'INTEGER' LINEMODE;
```

This procedure defines a curved line piece between (XFROM, YFROM) and (XTO, YTO) with the boundary conditions

$$\frac{dy}{dx}_{(XFROM,\ YFROM)} = SLOPEFROM \qquad \text{and}$$

$$\frac{dy}{dx}_{(XTO,\ YTO)} = SLOPETO$$

The parameter LINEMODE has the meaning as described in STRAIGHTLINEPIECE. Possible representations are: traced, dashed, dotted and dashed-dotted.

```
'PROCEDURE' AXIS COMPLETE(FILE, XFROM, YFROM, XTO, YTO, NUMBER OF INTERVALS,
                          MARKTYPE, VALUEFROM, VALUETO, PARALLELNUMBERING,
                          NUMBERING TO AXIS, SYMBOLHEIGHT, STRING);
'VALUE' XFROM, YFROM, XTO, YTO, NUMBERING OF INTERVALS, MARKTYPE, VALUEFROM,
       VALUETO, PARALLELNUMBERING, NUMBERING TO AXIS, SYMBOLHEIGHT;
'FILE' FILE;
'REAL' XFROM, YFROM, XTO, YTO, VALUEFROM, VALUETO, SYMBOLHEIGHT;
'INTEGER' NUMBER OF INTERVALS, MARKTYPE;
'BOOLEAN' PARALLELNUMBERING, NUMBERING TO AXIS;
'STRING' STRING;
```

This procedure defines the axis, complete with tickmarks, numbering and a comment string. The axis from (XFROM, YFROM) to (XTO, YTO) has a specified number of intervals. At each intersection point tickmarks are situated perpendicular to the axis. There are three possible kinds of situating these tickmarks, dependent of the value of MARKTYPE:

MARKTYPE = 0 : tickmarks on the upperside of the axis;
MARKTYPE = 1 : tickmarks on the lowerside of the axis;
MARKTYPE = 2 : tickmarks through the axis.

If MARKTYPE = 0 or MARKTYPE = 2 then numbering of the axis is performed on the lowerside of the axis else on the upperside. VALUEFROM contains the value belonging to (XFROM, YFROM) and VALUETO contains the value belonging to (XTO, YTO). The drawing machine determines which tickmarks can be numbered. The direction of writing these values is given by the parameters PARALLELNUMBERING and NUMBERING TO AXIS.

The possibilities are shown in the table below:

PARALLELNUMBERING \ NUMBERING TO AXIS	'TRUE'	'FALSE'
'TRUE'	the drawn number is parallel to the axis the base of the digits is turned to the axis	parallel to the axis, the top of the digits is turned to the axis
'FALSE'	perpendicular to the axis, the direction of writing is to the axis	perpendicular to the axis, the direction of writing is from the axis

If necessary a scaling factor (with representation: * @ < exponent >) is drawn, by which every number has to be multiplied. Finally a string can be added to the axis as comment. The direction of writing of the contingent scaling factor and the string is parallel to the axis.
The base of the characters is turned to or turned from the axis dependent of NUMBERING TO AXIS.
SYMBOLHEIGHT indicates the height of the characters, used to comment the axis and is expressed in current data units.

Procedures based on straight line pieces and points.

```
'PROCEDURE' POLYGON(FILE, I, I1, I2, XI, YI, LINEMODE);
'VALUE' I1, I2, LINEMODE;
'FILE' FILE;
'INTEGER' I, I1, I2, LINEMODE;
'REAL' XI, YI;
```

The polygon is defined by the sequence of points (XI, YI) for $I1 \leq I \leq I2$, i.e. the points (XI, YI) will be connected by straight line pieces.

```
'PROCEDURE' QUEUE OF POINTS(FILE, I, I1, I2, XI, YI, POINTTYPE, POINTHEIGHT);
'VALUE' I1, I2, POINTTYPE, POINTHEIGHT;
'FILE' FILE;
'INTEGER' I, I1, I2, POINTTYPE;
'REAL' XI, YI, POINTHEIGHT;
```

Every point in the sequence (XI, YI) for $I1 \leq I \leq I2$ is marked with a special marking symbol (see procedure POINT).

Basic procedures to draw a curve.

It is possible to define a curve by initializing the curve by two points and a contingent slope in the first point, a certain number of points and a termination of the curve, with the final slope at the last point given. It is essential that, while generating the curve, information about the last generated curved line piece is preserved. Therefore an array STATUS is used with bounds[0:5]. Initialization of the curve performs initialization of this array.
The total number of points has to be greater than or equal to 3.

```
'PROCEDURE' INITCURVE POINTS DERIV(X1, Y1, X2, Y2, DERIV, STATUS);
'VALUE' X1, Y1, X2, Y2, DERIV;
'REAL' X1, Y1, X2, Y2, DERIV;
'REAL' 'ARRAY' STATUS[0];
```

The curve is initialized by the two points (X1, Y1) and (X2, Y2) and the initial value of the slope $\frac{dy}{dx}$ in the point (X1, Y1), given in DERIV.

The two points must be different.

```
'PROCEDURE' INITCURVE POINTS(XI, YI, X2, Y2, STATUS);
'VALUE' X1, Y1, X2, Y2;
'REAL' X1, Y1, X2, Y2;
'REAL' 'ARRAY' STATUS[0];
```

The curve is initialized by the two points (XI, YI) and (X2, Y2). These points must be different.

```
'PROCEDURE' NEXTPOINTONCURVE(FILE, XNEXT, YNEXT, LINEMODE, STATUS);
'VALUE' XNEXT, YNEXT, LINEMODE;
'FILE' FILE;
'REAL' XNEXT, YNEXT;
'INTEGER' LINEMODE;
'REAL' 'ARRAY' STATUS[0];
```

This procedure adds the point (XNEXT, YNEXT) to the current curve. If we call (XNEXT, YNEXT) point, p_i, ($i \geq 3$) then the effect of the procedure is as follows:

A curved line piece between p_{i-2} and p_{i-1} is determined, such that:

1. the connection with the previous curved line piece is continuous in the first derivative;
2. if i = 3, then the initial slope is equal to the given slope by INITCURVE POINTS DERIV or the initial slope is equal to the slope in point 1 of the circle, passing through p1, p2 and p3 (initialization with INITCURVE POINTS);
3. the slope in p_{i-1} is equal to the slope in p_{i-1} on the circle passing through p_{i-2}, p_{i-1} and p_i.

The representation of the drawed curve is determined by the parameter LINEMODE (see CURVEDLINEPIECE).

```
'PROCEDURE' FINISH CURVE DERIV(FILE, DERIV, LINEMODE, STATUS);
'VALUE' DERIV, LINEMODE;
'FILE' FILE;
'REAL' DERIV;
'INTEGER' LINEMODE;
'REAL' 'ARRAY' STATUS[0];
```

The last curved line piece is generated. P_{i-1}, the slope in p_{i-1} and p_i are already known (in STATUS).

The slope $\frac{dy}{dx}$ in the last point is given by DERIV.

```
'PROCEDURE' FINISH CURVE(FILE, LINEMODE, STATUS);
'VALUE' LINEMODE;
'FILE' FILE;
'INTEGER' LINEMODE;
'REAL' 'ARRAY' STATUS[0];
```

The last curved line piece is generated. P_{i-1}, the slope in P_{i-1} and P_i are already known (in STATUS).
The final slope is the one on the circle, fixed by P_{i-1}, P_i and the slope in P_{i-1}.

Procedures for drawing complete curves.

```
'PROCEDURE' MARKEDCURVE1(FILE, I, I1, I2, XI, YI, MARKINGI, POINTTYPE,
                         POINTHEIGHT, LINEMODE);
'VALUE' I1, I2, POINTTYPE, POINTHEIGHT, LINEMODE;
'FILE' FILE;
'INTEGER' I, I1, I2, POINTTYPE, LINEMODE;
'REAL' XI, YI, POINTHEIGHT;
'BOOLEAN' MARKINGI;
```

A complete curve without initial slope and final slope is generated, passing through the points (XI, YI) for I1 ≤ I ≤ I2. It is necessary that there are at least 3 different points (XI, YI).
The value of MARKINGI determines whether the point (XI, YI) is marked with a special marking symbol, represented by POINTTYPE and POINTHEIGHT (see procedure POINT). The representation of the curve depends on the value of LINEMODE (see procedure CURVEDLINEPIECE).

```
'PROCEDURE' MARKEDCURVE2(FILE, I, I1, I2, XI, YI, STARTDERIV, MARKINGI,
                         POINTTYPE, POINTHEIGHT, LINEMODE);
'VALUE' I1, I2, STARTDERIV, POINTTYPE, POINTHEIGHT, LINEMODE;
'FILE' FILE;
'INTEGER' I, I1, I2, POINTTYPE, LINEMODE;
'REAL' XI, YI, STARTDERIV, POINTHEIGHT;
'BOOLEAN' MARKINGI;
```

The same as MARKEDCURVE1 except: the initial slope is given by STARTDERIV.

```
'PROCEDURE' MARKEDCURVE3(FILE, I, I1, I2, XI, YI, FINDERIV, MARKINGI,
                         POINTTYPE, POINTHEIGHT, LINEMODE);
'VALUE' I1, I2, FINDERIV, POINTTYPE, POINTHEIGHT, LINEMODE;
'FILE' FILE;
'INTEGER' I, I1, I2, POINTTYPE, LINEMODE;
'REAL' XI, YI, FINDERIV, POINTHEIGHT;
'BOOLEAN' MARKINGI;
```

The same as MARKEDCURVE1 except: the final slope is given by FINDERIV.

```
'PROCEDURE' MARKEDCURVE4(FILE, I, I1, I2, XI, YI, STARTDERIV, FINDERIV,
                         MARKINGI, POINTTYPE, POINTHEIGHT, LINEMODE);
'VALUE' I1, I2, FINDERIV, POINTTYPE, POINTHEIGHT, LINEMODE;
'FILE' FILE;
'INTEGER' I1, I2, POINTTYPE, LINEMODE;
'REAL' XI, YI, STARTDERIV, FINDERIV, POINTHEIGHT;
'BOOLEAN' MARKINGI;
```

The same as MARKEDCURVE1 except: the initial slope is given by STARTDERIV and the final slope is FINDERIV.

```
'PROCEDURE' CURVE1(FILE, I, I1, I2, XI, YI, LINEMODE);
'VALUE' I1, I2, LINEMODE;
'FILE' FILE;
'INTEGER' I, I1, I2, LINEMODE;
'REAL' XI, YI;
```

The same as MARKEDCURVE1 except marking.

```
'PROCEDURE' CURVE2(FILE, I, I1, I2, XI, YI, STARTDERIV, LINEMODE);
'VALUE' I1, I2, STARTDERIV, LINEMODE;
'FILE' FILE;
'INTEGER' I, I1, I2, LINEMODE;
'REAL' XI, YI, STARTDERIV;
```

The same as MARKEDCURVE2 except marking.

```
'PROCEDURE' CURVE3(FILE, I, I1, I2, XI, YI, FINDERIV, LINEMODE);
'VALUE' I1, I2, FINDERIV, LINEMODE;
'FILE' FILE;
'INTEGER' I, I1, I2, LINEMODE;
'REAL' XI, YI, FINDERIV;
```

The same as MARKEDCURVE3 except marking.

```
'PROCEDURE' CURVE4(FILE, I, I1, I2, XI, YI, STARTDERIV, FINDERIV, LINEMODE);
'VALUE' I1, I2, STARTDERIV, FINDERIV, LINEMODE;
'FILE' FILE;
'INTEGER' I, I1, I2, LINEMODE;
'REAL' XI, YI, STARTDERIV, FINDERIV;
```

The same as MARKEDCURVE4 except marking.

Procedures based on TEXT.

```
'PROCEDURE' NUMBER(FILE, XFROM, YFROM, XTO, YTO, ITALICITY, FORMATSTRING,
                   VALUE);
'VALUE' XFROM, YFROM, XTO, YTO, ITALICITY, VALUE;
'FILE' FILE;
'REAL' XFROM, YFROM, XTO, YTO, ITALICITY;
'STRING' FORMATSTRING;
'DOUBLE' VALUE;
```

With this procedure, the user is able to draw formatted numbers.
The string FORMATSTRING contains the information for formatting the number VALUE.
The value of FORMATSTRING must be conform to the syntax of < format string >.

```
< format string >          ::= < free-field format > |
                               < fieldprecision letter > < field width > .
                               < decimal places > |
                               < field letter > < field width >
< free-field format >      ::= < slash >
< fieldprecision letter >  ::= F|R|D|E
< field letter >           ::= A|C|H|I|J|K
< field width >            ::= < unsigned integer >
< decimal places >         ::= < unsigned integer >
```

```
'PROCEDURE' BOOLEAN VALUE(FILE, XFROM, YFROM, XTO, YTO, ITALICITY,
                          WIDTH, VALUE);
'VALUE' XFROM, YFROM, XTO, YTO, ITALICITY, WIDTH, VALUE;
'FILE' FILE;
'REAL' XFROM, YFROM, XTO, YTO, ITALICITY;
'INTEGER' WIDTH;
'BOOLEAN' VALUE;
```

The value of the boolean VALUE is drawn with WIDTH characters. In fact the fieldprecision letter L is used with field width = WIDTH.

Procedures based on AXIS COMPLETE.

```
'PROCEDURE' AXIS(FILE, XFROM, YFROM, XTO, YTO, NUMBER OF INTERVALS, MARKTYPE,
                 VALUEFROM, VALUETO, PARALLELNUMBERING, NUMBERING TO AXIS,
                 SYMBOLHEIGHT;
'VALUE' XFROM, YFROM, XTO, YTO, NUMBER OF INTERVALS, MARKTYPE, VALUEFROM,
        VALUETO, PARALLELNUMBERING, NUMBERING TO AXIS, SYMBOLHEIGHT;
'FILE' FILE;
'REAL' XFROM, YFROM, XTO, YTO, VALUEFROM, VALUETO, SYMBOLHEIGHT;
'INTEGER' NUMBER OF INTERVALS, MARKTYPE;
'BOOLEAN' PARALLELNUMBERING, NUMBERING TO AXIS;
```

The procedure defines an axis with tickmarks, numbering and a contingent scaling factor. Its effect is equal to that of the procedure AXIS COMPLETE, with exception of the comment string. The meaning of the parameters is the same as the corresponding ones in AXIS COMPLETE.

```
'PROCEDURE' BASIC AXIS(FILE, XFROM, YFROM, XTO, YTO, NUMBER OF INTERVALS,
                       MARKTYPE, SYMBOLHEIGHT);
'VALUE' XFROM, YFROM, XTO, YTO, NUMBER OF INTERVALS, MARKTYPE, SYMBOLHEIGHT;
'FILE' FILE;
'REAL' XFROM, YFROM, XTO, YTO, SYMBOLHEIGHT;
'INTEGER' NUMBER OF INTERVALS;
```

The procedure defines an axis with tickmarks. The meaning of the parameters is the same as the corresponding ones in AXIS COMPLETE.

Other procedures.

```
'INTEGER' 'PROCEDURE' SCALE(TI, I, I1, I2, NUMBER OF INTERVALS, MODE,
                            MIN, MAX);
'VALUE' I1, I2, NUMBER OF INTERVALS, MODE;
'REAL' TI, MIN, MAX;
'INTEGER' I, I1, I2, NUMBER OF INTERVALS, MODE;
```

If we call the set (1, 1.25, 2, 2.5, 4, 5, 8) a set of "basic round numbers", r_j, j = 1, 2, ..., 7, then we will call any number $x = r_j * 10^k$, for all integers k, a "round number", and any integer multiple of a round number is called a "nice number".

For optimal subdivision of axes it is required:

1. the value of the length of an interval must be a round number;
2. each subdivision point must be indicated by a nice number;
3. the data must fit into the allotted space;
4. the data must occupy as much as possible of the allotted space.

The procedure calculates the values of MIN and MAX such that both MIN and MAX are nice numbers as defined above, satisfying:

$$MIN \leq TI \leq MAX \quad \text{for} \quad I1 \leq I \leq I2$$

MODE may have the values 0, 1 or 2 with the following meaning:

MODE = 0 the interval (MIN, MAX) is subdivided into precisely NUMBER OF INTERVALS subintervals;

MODE = 1 the actual number of subintervals is between .625 * NUMBER OF INTERVALS and NUMBER OF INTERVALS;

MODE = 2 the actual number of subintervals is between .625 * NUMBER OF INTERVALS and 1.6 * NUMBER OF INTERVALS.

The value delivered by SCALE is the number of actual subintervals. (See [1], [2])

The drawing machine.

The drawing machine is built out of two parts:

1. a control part; this part looks for data files suitable for plotting and hands then over to the drawing part;
2. a drawing part which does the actual drawing.

The control part.

The total drawing machine is a MCS, because it must be able to reach an online plotting device (Calcomp C565), so written in DCALGOL.
All data files that are presented to the drawing machine are identified by a prefix in the file title.
Prefixes:

PLOT11 ⟶ Calcomp 565
PLOT30 ⟶ Calcomp 1136
PLOTPR ⟶ PRINTER

```
WHILE NOT DISASTER
DO
BEGIN
IF THEREISAPLOT11DATAFILE
THEN BEGIN PUTFILEINTOQUEUEFORC565DEVICES;
           IF NUMBEROFACTIVEC565 < NUMBEROFC565PLOTTERS
           THEN STARTANOTHERC565PLOTTER
     END;
IF THEREISAPLOT30DATAFILE
THEN BEGIN PUTFILEINTOQUEUEFORC1136DEVICE;
           IF NUMBEROFACTIVEC1136PLOTTERS < NUMBEROFC1136PLOTTERS
           THEN STARTANOTHERC1136PLOTTER
     END;
IF THEREISAPLOTPRDATAFILE
THEN BEGIN PUTFILEINTOQUEUEFORPRINTERPLOTTER;
           IF NOT PRINTERPLOTTERBUSY
           THEN STARTPRINTERPLOTTER
     END;
IF THEREISSOMEDATACOMFUNCTION
THEN PERFORMDATACOMFUNCTION;
END
```

This is a simplified version of the mainloop of the MCS.

For each plotter device a process is fired up whenever there is a data file to plot.
If another data file is found, its title is put into a queue to be interpreted by the plot process.
The plot process dies if and only if there is no other data file to plot.
This approach has been chosen, because there might be several identical devices (for example we have two C565 plotters).
The processes attached to these devices share the same queue in order to prevent the plotting of a data file twice.
In order to prevent that a data file which title was already put into the queue will be identified again, its title prefix is changed (for example PLOT11 is changed to PLIT11).

The drawing part.

The simplified version of the drawing part (procedure AUTOPLOT) is represented by the following program text.
For each device the same procedure is used.

```
WHILE SOMETHINGINMYDEVICEQUEUE
DO
BEGIN TAKENEXTFILETITLE;
      IF FILEISPRESENT
      THEN IF PLOTIT(FILETITLE)          % THIS IS THE ACTUAL PLOTTING
           THEN REMOVEFILE(FILETITLE)
           ELSE                          % SOMETHING WENT WRONG
               PUTFILETITLEBACKINTOQUEUE
      ELSE                               % PITY, FILE DISAPPEARED
END
```

The actual plotting is performed by PLOTIT.
The value of PLOTIT may be false which indicates that something went wrong while plotting (for example the DCP died).

Simplified text of PLOTIT.

```
INITIALISE;
PLOTIDENTIFICATION;
WHILE FILENOTEMPTY AND NOT ABORTED
DO
BEGIN GETNEXTITEM;
   IF ITEMTYPE NEQ WINDOWMAP OR ITEMTYPE NEQ CLOSEBRACKET
   THEN ABORT(NOFYSICALDRAWING)
   ELSE IF ITEMTYPE = WINDOWMAP
        THEN BEGIN CREATEWINDOW;
                   BUILDMAPMATRIX;
                   DRAWER(MAPMATRIX)          % MAPS IMAGE TO PAPER
             END
        ELSE
END;
PLOTIT := NOT ABORTED;
```

Simplified text of DRAWER.

```
% ALL INDICATED POINTS WILL BE TRANSFORMED
% ACCORDING TO THE MATRIX

WHILE FILENOTEMPTY AND NOT ABORTED AND NOT ENDOFIMAGE
DO
BEGIN GETNEXTITEM;
   IF NOT VALIDITEM
   THEN ABORT(INVALIDITEM)
   ELSE
   BEGIN
      CASE ITEMTYPE OF
      BEGIN
         ENDOFIMAGE := TRUE;                  % CLOSE BRACKET
         BEGIN                                % IMAGE ELEMENT
            CASE ELEMENTTYPE OF
```

```
            BEGIN
               DRAWDOT;
               DRAWPOINT;
               DRAWSTRAIGHTLINE;
               DRAWCURVEDLINE;
               DRAWTEXT;
               DRAWAXIS;
               BEGIN                    % PICTURE
                  BUILDMAPMATRIX;
                  CREATEMATRIX(MAPMATRIX, MATRIX, NEWMATRIX);
                  DRAWER(NEWMATRIX)
               END
            END
         END
      END
   END
END;
```

This description of the drawing machine is a very brief one, but we hope it gives an idea about its structure. The actual drawing machine obviously looks quite different, although the global structure is maintained. The distinction between the several devices is made on the lowest possible level.

References.

J.A.Th.M. van Berckel, B.J. Mailloux

[1] MR73 - Some Algol plotting procedures.
Stichting Mathematisch Centrum Amsterdam
January 1970.

[2] Prof.dr. F.E.J. Kruseman Aretz
Het plottersysteem in MILLI op EL X8.
Philips research laboratories, computernote nr. 1972/2.

[3] Dr.ing. José Encarnaçao
Untersuchungen zum Problem der rechnergesteurten räumlichen Darstellung auf ebene Bildschirmen.
Dissertation Technischen Universität Berlin.
Juli 1970.

[4] William M. Newman, Robert F. Sproull
Principles of interactive computer graphics.
McGraw-Hill 1973.

4 april 1974.