A STRUCTURED APPROACH TO COMPUTER GRAPHICS

N. Thalmann* and D. Thalmann†
** Section Systèmes d'information, Université Laval*
† Département d'Informatique et de Recherche Opérationnelle, Université de Montréal

ABSTRACT

Graphical programming requires the creation of basic graphical types and a method to construct abstract graphical data types. It is necessary to give the user the means of defining and using specific graphical types. That is the reason why we have designed and implemented a PASCAL extension based on abstract graphical types. The goal of this paper is to show how this extension allows the user to develop applications with a top-down methodology. We show by means of complete examples how it is possible to construct complex figures in the same way as the development of large programs by stepwise refinement. This graphical extension has been designed for computer aided instruction and design and will be applied in various areas, for which we consider that the man-machine communication is important.

UNE APPROCHE STRUCTURÉE DU TRAITEMENT GRAPHIQUE PAR ORDINATEUR

RÉSUMÉ

La programmation d'applications graphiques exige la création de types graphiques de base et une méthode de construction de types graphiques abstraits. Il faut fournir à l'utilisateur les moyens de définir et d'utiliser des types graphiques spécifiques. C'est la raison pour laquelle nous avons conçu et implanté une extension graphique de PASCAL basée sur les types graphiques abstraits. Le but de cet article est de montrer comment cette extension permet de développer des applications avec une méthodologie descendante. Nous montrons au moyen d'exemples complets comment il est possible de construire des figures complexes de la même manière que l'on développe de grands programmes par raffinements graduels. Cette extension graphique a été créée pour l'enseignement et la conception assistés par ordinateur et sera utilisée dans des domaines variés, où nous considérons que la communication homme-machine est importante.

A STRUCTURED APPROACH TO COMPUTER GRAPHICS*

Nadia Thalmann
Section Systèmes d'Information
Université Laval
Québec, Canada

Daniel Thalmann
Département d'Informatique et de Recherche Opérationnelle
Université de Montréal
Montréal, Canada

1. INTRODUCTION

Since the first ideas on structured programming were established by Dijkstra [1], they have been used time and again and have led to the concepts of systematic programming [2] and program proof [3,4]. We can enumerate the major ideas of disciplined programming when we quote three fundamental points which were respectively developed by Dijkstra [5], Hoare [6] and Wirth [7].

1) It is proved possible to completely describe a program by means of a decomposition into subactions, which can be sequential or controlled by selection or repetition clauses.

2) A program manipulates data; data organization and structuring must be carefully defined. In a high level programming language, the concept of a data type is very important.

3) A program can be gradually developed in a sequence of refinement steps. In each step, program instructions are decomposed into more detailed instructions.

The actual language which seems to the most adequate to structured programming techniques is the PASCAL [8,9] language which has been designed for this purpose [10].

If the development of disciplined programming is necessary, it is also very desirable for the human being to visualize his results; that is the reason why he has tried to develop another aspect of computer science: graphical processing.

With the evolution of technology, very powerful graphical displays have appeared and some graphical languages have been designed [11,12]. The default of these languages is that there are no graphical types and that good graphic programming requires the creation of

*This research was supported by National Research Council of Canada, Grants A3006 and E4262 and LRSA of Laval University.

basic graphical types as well as a method to construct more complicated graphical types. This point is very fundamental, especially when the graphical language is used for design automation, because this design can be made with a top-down approach [13].

The goal of this paper is to show the use of graphical types in a well-structured language in order to construct complicated figures with a top-down methodology.

After having introduced the basic concepts of our graphical PASCAL extension, we will present a complete example to show the construction of a digital network by stepwise refinement.

2. GRAPHICAL TYPES

Why graphical types are required?

A variable of square type must be easily distinct from a variable of circle type. When we write a function to calculate the cubic root of a real number, we find that it is necessary to check if the actual parameter is not a character. The same problem must be solved in graphical processing; writing a procedure to find the intersection point of the diagonals of a quadrilateral has no significance if the actual parameter is a circle. It is necessary to give the user the means of defining and using specific graphical types.

The vector type

The vector type has been introduced and can be used in the same way as the real type. A vector can be given by its two coordinates:

≪ E1,E2 ≫ where E1 and E2 are real expressions.

A vector arithmetic has been developed using vector addition and a scalar product. A vector can be read and written, and can be the result of a function.

e.g.

```
PROGRAM SIMPLE(INPUT,OUTPUT);
VAR V1,V2,V3: VECTOR; R:REAL;
BEGIN V1:= <<-5.2,4.3>>;
V2:= 2*V1; READ(V1);
V3:= V1+V2; R:= V1*V2;
WRITE(V3,R)
END.
```

The figure type

Most objects which are familiar to us can be constructed with the help of very simple figures, characterized by a few parameters. For example, a circle is determined by its radius and its center, a segment by its two vectors, a triangle by three.

Because of these considerations, it was necessary to use variables of graphical types, characterized by a few parameters. That is the reason why we have introduced a new structured type in PASCAL, the figure type, which is an abstract type. The syntax of such a type is shown in Fig. 1.

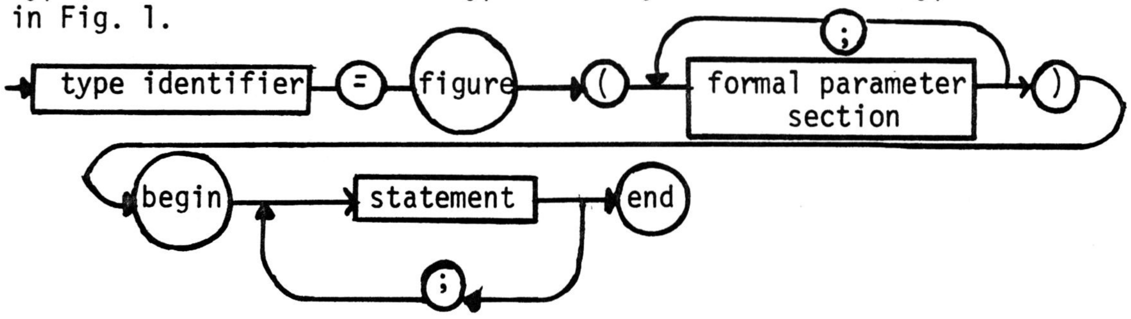


Fig. 1 Syntactic diagram of the figure type

To define a new figure type, it is necessary to take the following steps:

1° Find the figure characterics, which give the figure parameters

2° Find the algorithm, which allows one to construct the figure with the help of the parameters.

To facilitate this second step, we have introduced, in our extension, the statement connect (X1, X2, ..., Xn) which links the vectors X1, X2, ... Xn.

Let's show two examples:

a) The triangle

1° Its characteristics are its 3 vertices a,b,c

2° The figure can be build by connecting its 3 vertices.

The following program shows how to define the triangle type:

```
PROGRAM TRY(INPUT,OUTPUT);

TYPE TRIANGLE = FIGURE(A,B,C:VECTOR);
                BEGIN CONNECT(A,B,C)
                END;

VAR T:TRIANGLE; X,Y,Z:VECTOR;
BEGIN READ(X,Y,Z);
CREATE T(X,Y,Z); DRAW T
END.
```

The create statement creates dynamically in memory a new figure; at this time, the real characteristics of the figures are determined. The draw statement permits a created figure to be drawn. The erase statement erases the figure, while the delete statement releases it in the memory.

b) <u>The circle</u>

1^o The circle characteristics are the center c and the radius r.

2^o It is possible to construct a circle by using a sequence of points with the coordinates $<r\cos\phi, r\sin\phi>$ where ϕ has a range between o and 2π.

We can give the following program:

```
PROGRAM CURVE(INPUT,OUTPUT);

TYPE CIRCLE= FIGURE(C:VECTOR; R:REAL);
             VAR STEP,THETA:REAL; X1,X2:VECTOR;
             BEGIN THETA:=0.0; X1:= <<R,0>>+C;
             STEP:=0.1;
             WHILE THETA<2*PI DO
                   BEGIN THETA:=THETA+STEP;
                   X2:= R*<<COS(THETA),SIN(THETA)>>+C;
                   CONNECT(X1,X2); X1:=X2
                   END;

VAR C:CIRCLE; V:VECTOR; R:REAL;
BEGIN READ(V,R); CREATE C(V,R);
DRAW C
END.
```

In fact, figures such as the circle or the triangle are so common that we have decided to define them as standard types, in the same way as the following ones:

1. The segment defined by its 2 vectors

   segment = <u>figure</u> (X1, X2 : vector).

2. The line defined by 2 vectors

   line = <u>figure</u> (X1, X2 : vector)

3. The square defined by its center and one vertex s.

   square = figure (c,s : vector)

We have also defined a universal figure type, named <u>fig</u>.

The following standard functions and procedures are available:

<u>functions</u>

angle (d1, d2)	gives the angle of the 2 lines d1 and d2.

inter (d1, d2)	gives the intersection point of the 2 lines d1 and d2.
center (f)	gives the center of gravity of the figure f.
dist (v1, v2)	gives the distance between the 2 points v1 and v2.
projx (v), projy (v)	give the coordinates of the point v.

<u>procedures</u> (For the first four procedures, the figure f2 is obtained by a symmetry operation on figure f1).

symmetry (f1, 1, f2)	symmetry about the 1-axis
translation (f1, t, f2)	translation by vector t
rotation (f1, c, alpha, f2)	rotation about c by the angle alpha
homothety (f1, c, r, f2)	homothety of center c and ratio r.
union (f1, f2, f3)	construction of figure f3 through the union of the figures f1 and f2.
readgraph (p1, p2, ..., pn)	allows to enter figures and/or vectors by graphical input.

In the following program, the user enters a figure f and a vector v, the figure is reduced by a homothety of center v and ratio 0.3. Then n copies of the figure are stored in an array after a rotation about v by a variable angle. The n figures are then drawn.

```
PROGRAM IMAGE(INPUT,OUTPUT);
VAR F:ARRAY [1..10] OF FIG; G:FIG;
    I:1..10;
    V:VECTOR;
BEGIN READGRAPH(G,V);
HOMOTHETY(G,V,0.3,G);
FOR I:=1 TO 10 DO ROTATION(G,V,PI*I/10,F[I]);
FOR I:=1 TO 10 DO DRAW F[I]
END.
```

We can see, in this example, interactive techniques and how it is easy to use figure types as other PASCAL types.

3. A COMPLETE EXAMPLE OF A FIGURE BUILT BY STEPWISE REFINEMENT

Our intention is to draw the digital circuit corresponding to the logical gate and. As we know it, such a circuit is divided into three parts: two diodes d1 and d2 and one resistor r. The circuit has 2 inputs a and b and one output c (see Fig. 2). Moreover, an auxiliary point d is necessary for the connection to voltage power.

```
CIRCUIT= FIGURE(A,B,C:VECTOR; VAR D:VECTOR);
         VAR D1,D2:DIODE; R:RESISTOR; W1,W2:VECTOR;
         BEGIN W1:=A+0.8*(C-A); W2:=B+W1-A;
         D:=2*W1-W2;
         CREATE D1(A,W1); CREATE D2(B,W2); CREATE R(D,W1);
         CONNECT(W1,C); CONNECT(W1,W2);
         INCLUDE D1,D2,R
         END;
```

We have made the following geometrical assumptions:

1) w1 is at about 80% of the distance $\overline{ac}$

2) a, b, w1 and w2 are the 4 vertices of a rectangle.

3) d is the symmetrical of w2 about w1.

The create statement creates 2 diodes and one resistor between the 4 points a, b, c, d. The figure is completely created by including these three elements (include statement) and the 2 segments w_1c and w_1w_2 . Now, we have to construct a diode (Fig.3) between the 2 points a and b.

The diode type can be defined as:

```
DIODE= FIGURE(A,B:VECTOR);
       VAR X1,X2,P1,P2,P:VECTOR; ALPHA:REAL; T:TRIANGLE;
       BEGIN X1:=A+0.3*(B-A); X2:=A+0.7*(B-A);
       ALPHA:=ANGLEPROJ(X1,X2);
       P:=(DIST(X1,X2)/2)*<<COS(ALPHA),SIN(ALPHA)>>;
       P2:=X2+P; P1:=X1+P;
       CONNECT(A,X1,P1,2*X1-P1);
       CREATE T(X1,P2,2*X2-P2);
       INCLUDE T; CONNECT(X2,B)
       END;
```

The function angleproj (a,b) gives the angle between the vertical axis and the line $\overline{ab}$.

At this point, the resistor (Fig.4) is constructed by stepwise refinement; two segments $\overline{ax_1}$ and $\overline{x_2b}$ are used with a length of 20% of the distance $\overline{ab}$ and a sequence of ten zigzags which have a width of 1/3 of the distance $\overline{x_1x_2}$.

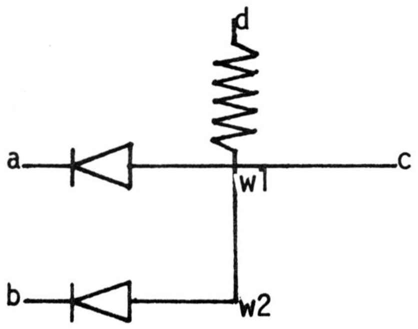
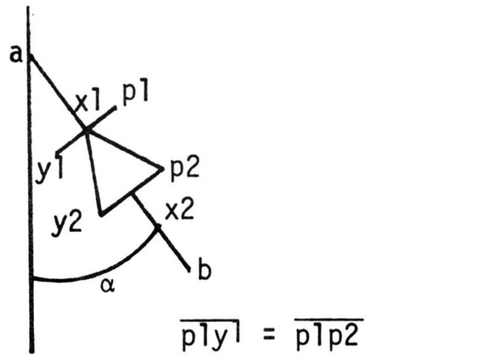


Fig. 2 The AND circuit



$\overline{p1y1} = \overline{p1p2}$

$\overline{ax1} = \overline{bx2} = 0.3\ \overline{ab}$

Fig. 3 The diode

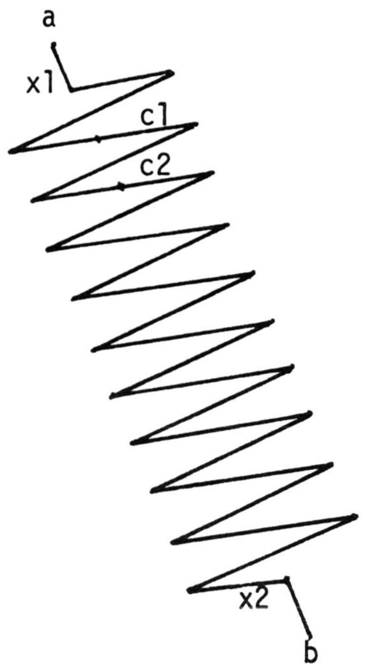


Fig. 1 The resistor

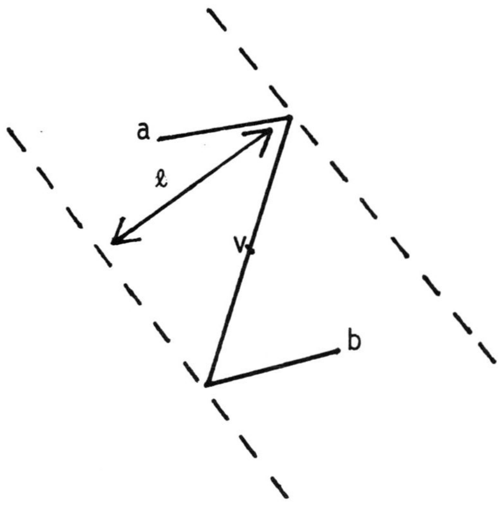


Fig. 5 A zigzag

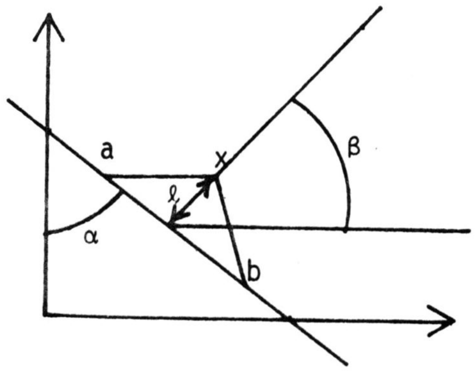


Fig. 6 A "zig"

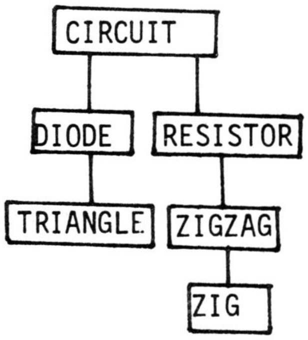


Fig. 7 Hierarchy of types

```
RESISTOR= FIGURE(A,B:VECTOR);
          VAR X1,X2,C1,C2:VECTOR; I:1..10; Z:ZIGZAG; D:REAL;
          BEGIN X1:=A+0.2*(B-A); X2:=A+0.8*(B-A);
          D:=DIST(X1,X2)/3;
          CONNECT(A,X1); C1:=X1;
          FOR I:=1 TO 10 DO BEGIN C2:=X1+(I/10)*(X2-X1);
                                  CREATE Z(C1,C2,D); INCLUDE Z; C1:=C2
                                  END;
          CONNECT(X2,B)
          END;
```

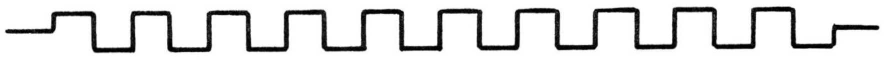
Each zigzag (Fig.5) can be divided into two distinct parts: the first one is of zig type and the other one is obtained by an homothety operator using a value of ratio of -1.

```
ZIGZAG= FIGURE(A,B:VECTOR; L:REAL);
        VAR Z:ZIG; V:VECTOR;
        BEGIN V:=(A+B)/2;
        CREATE Z(A,V,L/2); INCLUDE Z;
        HOMOTHETY(Z,V,-1.0,Z); INCLUDE Z
        END;
```

The zig type can be defined very easily (Fig. 6)

```
ZIG=FIGURE(A,B:VECTOR; L:REAL);
     VAR BETA:REAL; X:VECTOR;
     BEGIN BETA:=ANGLEPROJ(A,B);
     X:=(A+B)/2+L*<<COS(BETA),SIN(BETA)>>;
     CONNECT(A,X,B)
     END;
```

In short, the goal of this example is to show the use of the stepwise refinement technique to construct graphical objects with a structured method. It is obvious that such an approach is technically not optimized; it is also the case with the conventional top-down approach which can lead to a very large number of procedures that produces some amount of inefficiency. However, with this methodology, it is very simple to modify the characteristics of the drawing. For example, if we want to represent resistors like:



It is only necessary to redefine the zig type and the entire hierarchy of types, as shown in Fig.7 is invariable. A complete listing and an execution are given in appendix.

4. CONCLUSION

This graphical extension of PASCAL has been implemented on CDC Cyber by a portable preprocessor. This preprocessor consists of about 4,000 Pascal source lines and the output is a standard Pascal program.

The figures are created in memory either by the create statement, by a symmetry or union operation or by an assignment. Storage is done by a set of Pascal procedures based on pointers. The delete statement releases memory entirely since an efficient garbage collector has been developed. The include statement corresponds to the operation wherein a pointer is updated; it does not create a copy in memory. The connect statement is processed as chain building.

The graphical devices used are a display HP2648A and a VERSATEC printer. The extension has been designed for computer aided instruction and design and will be applied in various areas, for which we consider that the man-machine communication is important.

5. ACKNOWLEDGEMENT

The authors are grateful to C. Pellegrini who has worked for the preprocessor.

6. REFERENCES

[1] Dijkstra, E.W. "Programming Considered as a Human Activity", Proc. IFIP Congr., 1965, pp. 213-217.

[2] Wirth, N. "Systematic Programming: An Introduction", Prentice-Hall, 1973.

[3] Naur, P. "Proof of Algorithms by General Snapshots", BIT, 6 (1966), pp. 310-316.

[4] Hantler, S.L. and King, J.C. "An Introduction to Proving the Correctness of Programs", Computing Surveys 8 (1976) 3.

[5] Dijkstra, E.W. "Notes on Structured Programming", N.Y., Academic Press, 1972.

[6] Hoare, C.A.R. "Notes on Data Structuring", N.Y., Academic Press, 1972.

[7] Wirth, N. "Program Development by Stepwise Refinement", Comm. ACM, 14 (1971) 4, pp. 221-227.

[8] Jensen, K. and Wirth, N. "Pascal User Manual and Report", Berlin Springer-Verlag, 1975.

[9] Hoare, C.A.R. and Wirth, N. "An Axiomatic Definition of the Programming Language PASCAL", Acta Informatica, 2(1973), pp. 335-355.

[10] Thalmann, N. and Thalmann, D. "The Use of Pascal as a Teaching Tool in Introductory, Intermediate and Advanced Computer Science Courses", Proc. SIGCSE/CSA Techn. Symp., Detroit, 1978, SIGCSE Bulletin, 10 (1978) 1, pp. 277-281.

[11] Shapiro, L.G. "ESP3: A High-Level Graphics Language", Proc. Siggraph'75, Computer Graphics, 9 (1975) 1, pp. 70-77.

[12] O'Brien, C.D. et Brown, H.G. "IMAGE: a Language for the Interactive Manipulation of a Graphics Environment", Proc. Siggraph'75, Computer Graphics, 9 (1975) 1, pp. 53-60.

[13] Stevens, W.P.; Myers, G.J. and Constantine, L.L. "Structured Design", IBM Systems Journal 13 (1974) 2, pp. 115-139.

Appendix

```
PROGRAM DESIGN(INPUT,OUTPUT);

TYPE CIRCUIT= FIGURE(A,B,C:VECTOR; VAR D:VECTOR);

TYPE
   FUNCGRAPH ANGLEPROJ(A,B:VECTOR) : REAL;
   CONST PI=3.14159;
   BEGIN IF PROJX(B)=PROJX(A) THEN ANGLEPROJ:=0
              ELSE ANGLEPROJ:=PI/2-ARCTAN((PROJY(B)-PROJY(A))/(PROJX(B)-PROJX(A)))
   END;


   RESISTOR= FIGURE(A,B:VECTOR);

     TYPE ZIGZAG= FIGURE(A,B:VECTOR; L:REAL);

          TYPE ZIG=FIGURE(A,B:VECTOR; L:REAL);
                   VAR BETA:REAL; X:VECTOR;
                   BEGIN BETA:=ANGLEPROJ(A,B);
                   X:=(A+B)/2+L*<<COS(BETA),SIN(BETA)>>;
                   CONNECT(A,X,B)
                   END;

                  VAR Z:ZIG; V:VECTOR;
                  BEGIN V:=(A+B)/2;
                  CREATE Z(A,V,L/2); INCLUDE Z;
                  HOMOTHETY(Z,V,-1.0,Z); INCLUDE Z
                  END;

             VAR X1,X2,C1,C2:VECTOR; I:1..10; Z:ZIGZAG; D:REAL;
             BEGIN X1:=A+0.2*(B-A); X2:=A+0.8*(B-A);
             D:=DIST(X1,X2)/3;
             CONNECT(A,X1); C1:=X1;
             FOR I:=1 TO 10 DO BEGIN C2:=X1+(I/10)*(X2-X1);
                                 CREATE Z(C1,C2,D); INCLUDE Z; C1:=C2
                                 END;
             CONNECT(X2,B)
             END;


   DIODE  =   FIGURE(A,B:VECTOR);
              VAR X1,X2,P1,P2,P:VECTOR; ALPHA:REAL; T:TRIANGLE;
              BEGIN X1:=A+0.3*(B-A); X2:=A+0.7*(B-A);
              ALPHA:=ANGLEPROJ(X1,X2);
              P:=(DIST(X1,X2)/2)*<<COS(ALPHA),SIN(ALPHA)>>;
              P2:=X2+P; P1:=X1+P;
              CONNECT(A,X1,P1,2*X1-P1);
              CREATE T(X1,P2,2*X2-P2);
              INCLUDE T; CONNECT(X2,B)
              END;

            VAR D1,D2:DIODE; R:RESISTOR; W1,W2:VECTOR;
            BEGIN W1:=A+0.8*(C-A); W2:=B+W1-A;
            D:=2*W1-W2;
            CREATE D1(A,W1); CREATE D2(B,W2); CREATE R(D,W1);
            CONNECT(W1,C); CONNECT(W1,W2);
            INCLUDE D1,D2,R
            END;


VAR N:CIRCUIT; D:VECTOR;

BEGIN CREATE N(<<1.0,1.0>>,<<1.0,-3.0>>,<<6.0,1.0>>,D);
DRAW N;
WRITELN(D)
END.
```