MATHEMATICAL ASPECTS OF A GRAPHICAL PROGRAMMING LANGUAGE AND ITS IMPLEMENTATION

by

Bon-boon Chan
B.Sc., University of California, Berkeley, 1973

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENT FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE

in the Department
of
Electrical Engineering

We accept this thesis as conforming to
the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

July, 1976

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the Head of my Department or by his representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Electrical Engineering

The University of British Columbia
2075 Wesbrook Place
Vancouver, Canada
V6T 1W5

Date July 14, 1976

ABSTRACT

This thesis describes the improvements in the mathematical aspects of the high-level programming language for interactive graphics (LIG). Improvement possibilities were investigated in three areas: dependence on the order of transformations, arithmetic expressions appearing in LIG statements, and the implementation to allow the programming of graphical functions. The method of implementation of the proposed improvements is also discussed.

TABLE OF CONTENTS

LIST OF FIGURES

PAGE

ACKNOWLEDGEMENT

The author wishes to thank his supervisor, Dr. G.F.Schrack for his interest, advice and encouragement throughout the period of this research.

Financial support for this research was provided by the National Research Council of Canada under grant number A-0148.

I. INTRODUCTION

1.1 Host Language Concept

Graphics languages have been developed for the production of diagrams and pictures. Examples are B-LINE [6] and GROAN [25]. However, a graphics language which is intended to assist computer-aided activities cannot be a pure graphics language, as computer-aided design requires the usual numerical, Boolean, and character string handling capabilities of a problem-oriented high-level language (e.g. FORTRAN, ALGOL, APL etc.). For this reason, LIG was designed to accept an existing procedural language as a host language, in addition to the graphics statements proper. Examples of language extensions are LIG [17], GPL/1 [22], TUNA [2], and XPLG [23].

Fortran is the host language for the high-level graphics language LIG. Thus, an application program contains both Fortran and LIG statements. Fig. 1.1 shows the procedure involved in the generation of the object code.

The LIG preprocessor translates all LIG statements into Fortran call statements, but passes the pure Fortran statements directly to the output file. The construction of the LIG preprocessor is discussed in Section 3.5 [12] [8].

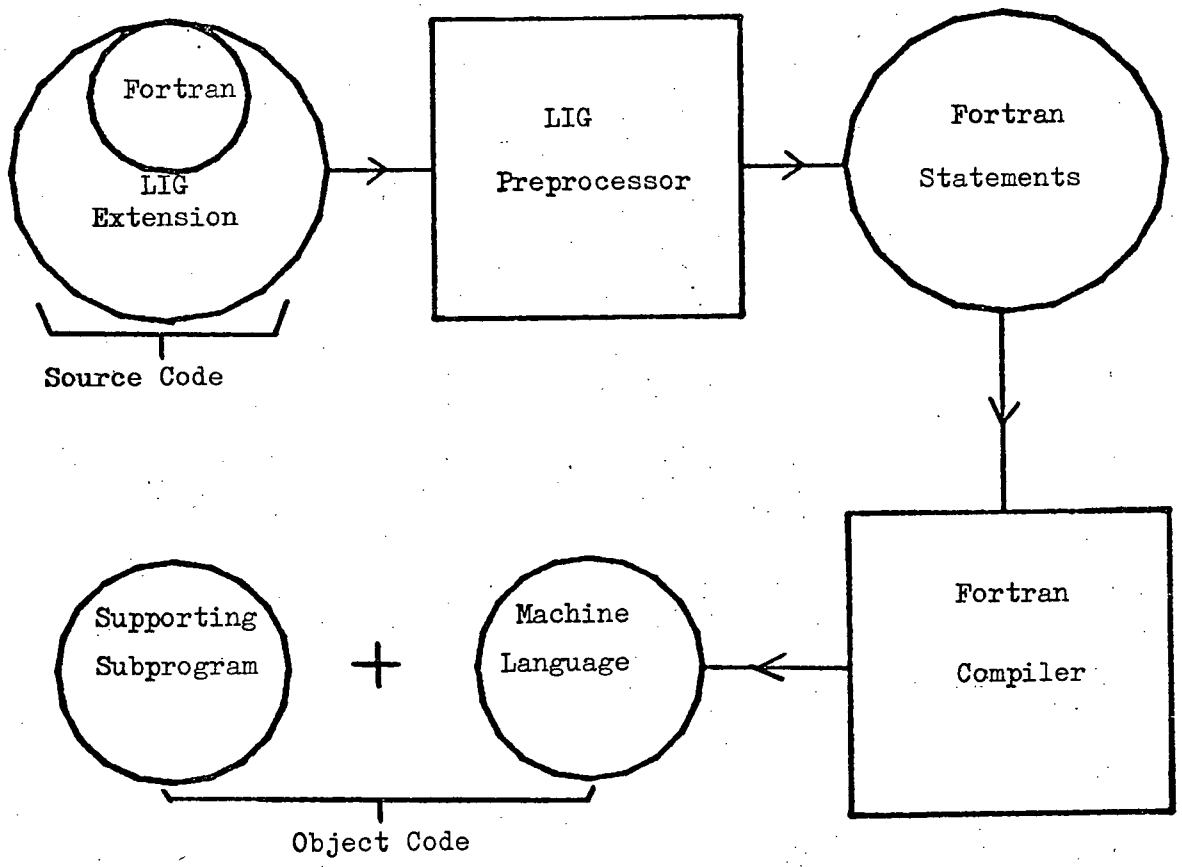


Fig.1.1 Object Code Generation

1.2 Purpose of Research

The mathematical aspects of the LIG graphics language and graphics system are investigated. The areas of study are listed as follows:

(a) the dependence on the order of transformation

(b) LIG statement syntax improvement

(c) implementation of graphical functions

In the previous LIG system, the order of transformation was always evaluated in a fixed order. The transformations in question are translation, scaling and rotation. The three operators associated with these transformations can be arranged into different sequences. Different transformation sequences were investigated to determine if identical pictures were generated from the same graphical object.

In the previous version of LIG, statements could not contain imbedded arithmetic expressions. For example, a subscript of a variable has to be a single number or an identifier. The grammatical rules as well as the LIG preprocessor were changed.

A graphical object is built up from a fixed set of graphical primitives. These graphical primitives are: BLANK, LINE, TRIANGLE, SQUARE , CIRCLE and SCIRCLE. From this set, it is difficult to generate curves described by an algebraic

equation. Thus, the variety of graphical objects generated is somewhat limited.

In order to permit algebraic curves, the notion of a graphical function was introduced. A graphical function is a function which returns a value of type graphical, i.e. a graphical object.

II. ORDER OF GRAPHICAL TRANSFORMATIONS

2.1 The Requirement for Transformations

A graphics system should allow the application programmer to define pictures, to display these predefined pictures on the screen, and to manipulate these pictures. The basic type of manipulation should include the capability to position a picture at any desired location of the screen, to change the dimension of a picture in X- or Y-direction, and to rotate a picture by a specified angle in the clockwise or anti-clockwise direction. In graphics terminology, these types of picture manipulation are called translation, scaling and rotation. They are collectively known as graphical transformations.

2.2 The Formulation of Transformations

Essentially, there are two aspects in the formulation of graphical transformations [15] :

(a) A transformation which is a single mathematical entity, and which can be denoted by a single name or a symbol;

(b) Two or more transformations which can be combined, or concatenated, to yield a single transformation which has the same effect as the

sequential application of the original transformations.

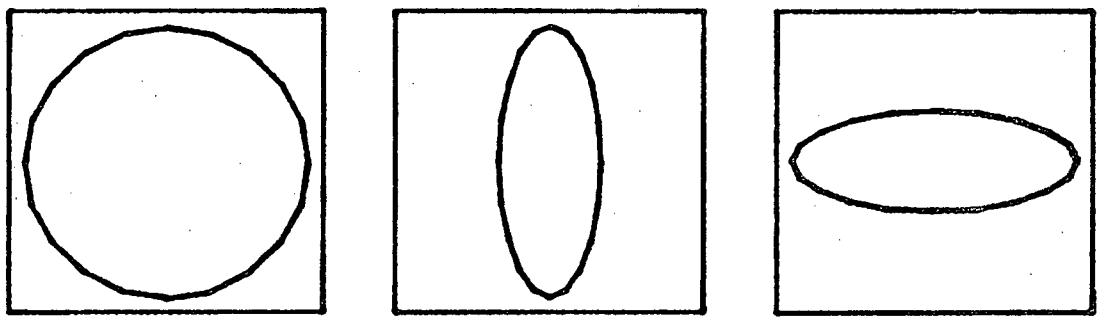
In the high-level interactive graphics language (LIG), translation is denoted by the keyword AT, scaling by SCALE, and rotation by ANGLE. This conforms with the first aspect of formulation. Moreover, when these transformations are concatenated, the result is evaluated in a fixed order. [16][17][20] Independent of the order of appearance of the transformation entity in a LIG statement, the order of evaluation is scaling, rotation and translation. Thus the resulting transformations are identical for all of the following LIG statements:

```
AT X,Y SCALE SX,SY ANGLE A
AT X,Y ANGLE A SCALE SX,SY
SCALE SX,SY AT X,Y ANGLE A
SCALE SX,SY ANGLE A AT X,Y
ANGLE A SCALE SX,SY AT X,Y
ANGLE A AT X,Y SCALE SX,SY
```

As the resulting transformations are not order dependent, the formulation of the transformation in the previous LIG graphics system does not conform with the second aspect of the formulation of transformations. Using a simple example, one will notice that the order in the concatenation of graphical transformations is important. The significance of the sequence of transformations applied to a graphical

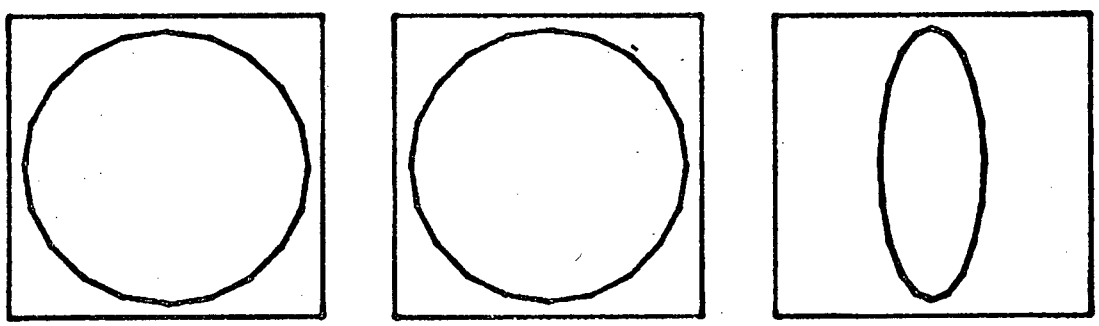
object is shown in Fig. 2.1.

Sequence 1 (scaling + rotation)



original picture after scaling after rotation

Sequence 2 (rotation + scaling)



original picture after rotation after scaling

FIG. 2.1 Significance of the sequence of transformations

Hence, the previous LIG graphics system can be improved by implementing into the system the second aspect in the formulation of transformations.

2.3 The Transformation Equations and Matrices

First, consider the transformations applied to a single point with respect to the origin of the co-ordinate system. The equations can then be applied to the co-ordinates of the set of points which defines a graphical object. Algebraically, the equations are [5] :

a. Translation of a point (X1,Y1) to (X2,Y2) by (Xt,Yt)

$$X2 = X1 + Xt$$

$$Y2 = Y1 + Yt$$

b. Rotation of a point (X1,Y1) with respect to the origin through an angle θ in the anti-clockwise direction

$$X2 = X1 \cos\theta - Y1 \sin\theta$$

$$Y2 = X1 \sin\theta + Y1 \cos\theta$$

c. Scaling the axes with value factors Sx,Sy about the origin

$$X2 = X1 Sx$$

$$Y2 = Y1 Sy$$

When using the above algebraic equations for the

concatenation of transformations, the resulting set of algebraic equations will be difficult to manipulate. Morever, these transformations can be represented in a uniform way by a 3x3 matrix [1]. This is called the homogeneous coordinate representation.

The homogeneous coordinate representation of the transformations of a point about the origin are given as follows:

a. Translation

$$[X2\ Y2\ 1] = [X1\ Y1\ 1]\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Xt & Yt & 1 \end{bmatrix}$$

b. Rotation

$$[X2\ Y2\ 1] = [X1\ Y1\ 1]\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

c. Scaling

$$[X2\ Y2\ 1] = [X1\ Y1\ 1]\begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Then, for any sequence of translation, rotation and scaling, the resulting transformation can be represented as follows:

$$[X2\ Y2\ 1] = [X1\ Y1\ 1]\begin{bmatrix} a & d & 0 \\ b & e & 0 \\ c & f & 1 \end{bmatrix}$$

Due to its uniform characteristics, the homogeneous coordinate representation of the transformation is more convenient.

2.4 Transformations in LIG

In the LIG graphics system, graphical primitives are the basic graphical objects [20]. The variety of graphical primitives available is discussed in Section 4.1. All other graphical objects are to be built from this basic set of graphical primitives. The LIG graphical primitives are defined within a unit square, which is 750 x 750 points. Each graphical primitive is as large as the dimension of the unit square. The graphical object is considered to be located at a reference point which is the center of the unit square. The origin of the co-ordinate system is defined to be in the lower left-hand corner of the screen. Thus for a graphical primitive, the reference point is at (0.5,0.5). Hence, if a graphical primitive is transformed by rotation through an angle θ, the following three steps need to be performed:

(a) Translate from reference point (0.5,0.5) to the origin (0,0)

(b) Rotate about the origin

(c) Translate from the origin back to the reference point (0.5,0.5).

Using matrix notation, the homogeneous co-ordinate representation is as follows:

(a) Translate to the origin

$$[X2\ Y2\ 1] = [X1\ Y1\ 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -0.5 & -0.5 & 1 \end{bmatrix}$$

(b) Rotate about the origin by angle θ in anti-clockwise direction

$$[X3\ Y3\ 1] = [X2\ Y2\ 1] \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(c) Translate back to the reference point

$$[X4\ Y4\ 1] = [X3\ Y3\ 1]\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.5 & 0.5 & 1 \end{bmatrix}$$

Hence, we have

$$[X4\ Y4\ 1] = [X1\ Y1\ 1][M]$$

where

$$[M] = \begin{bmatrix} r & u & 0 \\ s & v & 0 \\ t & w & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -0.5 & -0.5 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.5 & 0.5 & 1 \end{bmatrix}$$

The resulting transformation matrix [M] can be applied to every co-ordinate pair of the graphical primitive which is to be rotated.

2.5 Non-commutativity property

In the process of implementing the second aspect in the formulation of transformations, the non-commutative property of rotation and scaling has to be taken into account, so as to unify the method of implementation. All cases in the concatenation of transformations must be examined.

Using the procedure discussed in Section 2.4, the

different cases are formulated in matrix form, using the following notation:

(X1,Y1) is the co-ordinate pair before transformation

(X2,Y2) is the co-ordinate pair after transformation

(Xr,Yr) is the coordinate pair of the reference point before transformation

(Xt,Yt) is the coordinate pair of the reference point after transformation

(Sx,Sy) is the scale value

θ is the angle of rotation. It has a positive value for rotation in the anti-clockwise direction, or it has a negative value for rotation in the clockwise direction.

$$[T0]=\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -Xr & -Yr & 1 \end{vmatrix}$$, translation from the original reference point

$$[T1]=\begin{bmatrix}0 & 1 & 0\\ 0 & 1 & 0\\ Xr & Yr & 1\end{bmatrix}$$, translation back to the original reference point

$$[T2]=\begin{bmatrix}0 & 1 & 0\\ 0 & 1 & 0\\ Xt & Yt & 1\end{bmatrix}$$, translation to the new reference point

$$[R]=\begin{bmatrix}\cos\theta & \sin\theta & 0\\ -\sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}$$, rotation matrix

$$[S]=\begin{bmatrix}Sx & 0 & 0\\ 0 & Sy & 0\\ 0 & 0 & 1\end{bmatrix}$$, scaling matrix

A list of all possible concatenations of transformations is given as follows:

a. Translation

$$[X2\ Y2\ 1] = [X1\ Y1\ 1][T0][T2]$$

b. Translation + rotation

$$[X2\ Y2\ 1]=[X1\ Y1\ 1][T0][R][T2]$$

c. Translation + scaling

$$[X2\ Y2\ 1] = [X1\ Y1\ 1][T0][S][T2]$$

d. Translation + rotation + scaling

$$[X2\ Y2\ 1] = [X1\ Y1\ 1][T0][R][S][T2]$$

e. Translation + scaling + rotation

$$[X2\ Y2\ 1] = [X1\ Y1\ 1][T0][S][R][T2]$$

f. Rotation

$$[X2\ Y2\ 1] = [X1\ Y1\ 1][T0][R][T1]$$

g. Rotation + translation

$$[X2\ Y2\ 1] = [X1\ Y1\ 1][T0][R][T2]$$

h. Rotation + scaling

$$[X2\ Y2\ 1] = [X1\ Y1\ 1][T0][R][S][T1]$$

i. Rotation + translation + scaling

$$[X2\ Y2\ 1] = [X1\ Y2\ 1][T0][R][S][T2]$$

j. Rotation + scaling + translation

$$[X2\ Y2\ 1] = [X1\ Y1\ 1][T0][R][S][T2]$$

k. Scaling

$$[X2\ Y2\ 1] = [X1\ Y1\ 1][T0][S][T1]$$

l. Scaling + rotation

$$[X2\ Y2\ 1] = [X1\ Y1\ 1][T0][S][R][T1]$$

m. Scaling + translation

$$[X2\ Y2\ 1] = [X1\ Y1\ 1][T0][S][T2]$$

n. Scaling + rotation + translation

$$[X2\ Y2\ 1] = [X1\ Y1\ 1][T0][S][R][T2]$$

o. Scaling + translation + rotation

$$[X2\ Y2\ 1] = [X1\ Y1\ 1][T0][S][R][T2]$$

These fifteen cases can be generalized by two cases. These two cases are listed as follows:

Case 1 $[X2\ Y2\ 1] = [X1\ Y1\ 1][T][R][S][T]$

Case 2 $[X2\ Y2\ 1] = [X1\ Y1\ 1][T][S][R][T]$

Case 1 represents cases for

d. Translation + rotation + scaling

h. Rotation + scaling

i. Rotation + translation + scaling

j. Rotation + scaling + translation

Case 2 represents cases for

e. Translation + scaling + rotation

l. Scaling + rotation

h. Scaling + rotation + translation

o. Scaling + translation + rotation

As for the remaining cases, they can be represented by either Case 1 or Case 2. For example, rotation can be represented either by Case 1 or Case 2 as $Sx = Sy = 1$. The matrix [S] will be

$$[S]=\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}$$ (i.e. the unit matrix)

Therefore the only two cases to be further discussed are

a. Rotation + scaling + translation

b. Scaling + rotation + translation

Setting Xr = Yr = 0.5, and putting Case 1 and Case 2 into the form of algebraic equations, from Case 1,

Set 1

$$X2 = Sx \cos\theta\ (X1 - 0.5) - Sx \sin\theta\ (Y1 - 0.5) + Xt$$

$$Y2 = Sy \sin\theta\ (X1 - 0.5) + Sy \cos\theta\ (Y1 - 0.5) + Yt$$

from Case 2,

Set 2

$$X2 = Sx \cos\theta\ (X1 - 0.5) - Sy \sin\theta\ (Y1 - 0.5) + Xt$$

$$Y2 = Sx \sin\theta\ (X1 - 0.5) + Sy \cos\theta\ (Y1 - 0.5) + Yt$$

The implemention of the second aspect of the formulation of transformations is based on the above two sets of equations.

2.6 To Derive the A and B Matrices

In the previous version of LIG, the set of equations which was used is Set 2. Represented in matrix form,

$$P2 = A \bullet P1 + B$$

$$\text{where } P2 = \begin{bmatrix} X2 \\ Y2 \end{bmatrix}$$

$$P1 = \begin{bmatrix} X1 \\ Y1 \end{bmatrix}$$

$$A = \begin{bmatrix} Sx\cos\theta & -Sy\sin\theta \\ Sx\sin\theta & Sy\cos\theta \end{bmatrix}$$

$$B = \begin{bmatrix} Xt + 0.5\,(Sy\sin\theta - Sx\cos\theta) \\ Yt - 0.5\,(Sx\sin\theta + Sy\cos\theta) \end{bmatrix}$$

The values A and B are stored instead of Xt, Yt, Sx, Sy, and θ. One modification (A2,B2) can be applied to another modification (A1,B1) as follows

$$P3 = A2 \bullet P2 + B2$$

$$= A2 \bullet (A1 \bullet P1 + B1) + B2$$

$$= A2 \bullet A1 \bullet P1 + A2 \bullet B1 + B2$$

$$= C \bullet P1 + D$$

where

$$C = A2 \bullet A1$$

$$D = A2 \bullet B1 + B2$$

$$P3 = \begin{bmatrix} X3 \\ \\ Y3 \end{bmatrix}, \text{ the point after transforming P1 twice}$$

In order to keep this feature and to implement Set 1 and Set 2 into the LIG graphics system, the common aspects and differences between Set 1 and Set 2 was investigated.

For scaling, rotation and translation, the matrices are, from Set 2

$$A1 = \begin{bmatrix} Sx\cos\theta & -Sy\sin\theta \\ Sx\sin\theta & Sy\cos\theta \end{bmatrix}$$

$$B1 = \begin{bmatrix} Tx - 0.5\,(-Sy\sin\theta + Sx\cos\theta) \\ Ty - 0.5\,(Sx\sin\theta + Sy\cos\theta) \end{bmatrix}$$

For rotation, scaling and translation, the matrices are, from Set 1

$$A2 = \begin{bmatrix} Sx\cos\theta & -Sx\sin\theta \\ Sy\sin\theta & Sy\cos\theta \end{bmatrix}$$

$$B2 = \begin{bmatrix} Xt - 0.5\,(-Sx\sin\theta + Sx\cos\theta) \\ Yt - 0.5\,(Sy\sin\theta + Sy\cos\theta) \end{bmatrix}$$

An examination of the pairs (A1,A2) and (B1,B2) shows that they are the same except in two places where Sx and Sy are interchanged.

In the LIG graphics system, the subroutine which sets up the matrices A and B is named TRAMA. To satisfy both transformation orders, a modification is made so that the requested matrices are generated according to the specified order of transformation. The modified TRAMA is listed in Appendix C.

A parameter of the sub-program TRAMA (viz. ITRANS) must supply the order of the transformations of scaling and rotation. This order of AT, SCALE, ANGLE in a LIG statement can only be detected by the LIG preprocessor. The LIG preprocessor is therefore modified accordingly. Fig. 2.2 shows the results of the following LIG statements,

*DISPLAY TRIANGLE AT 0.5,0.5 SCALE 0.3,0.5 ANGLE 1.57079 RAD;
*DISPLAY TRIANGLE AT 0.5,0.5 ANGLE 1.57079 RAD SCALE 0.3,0.5;

demonstrating the effects of an interchange of scaling and rotation. For comparison purposes, the original triangle is also shown in Fig. 2.2.

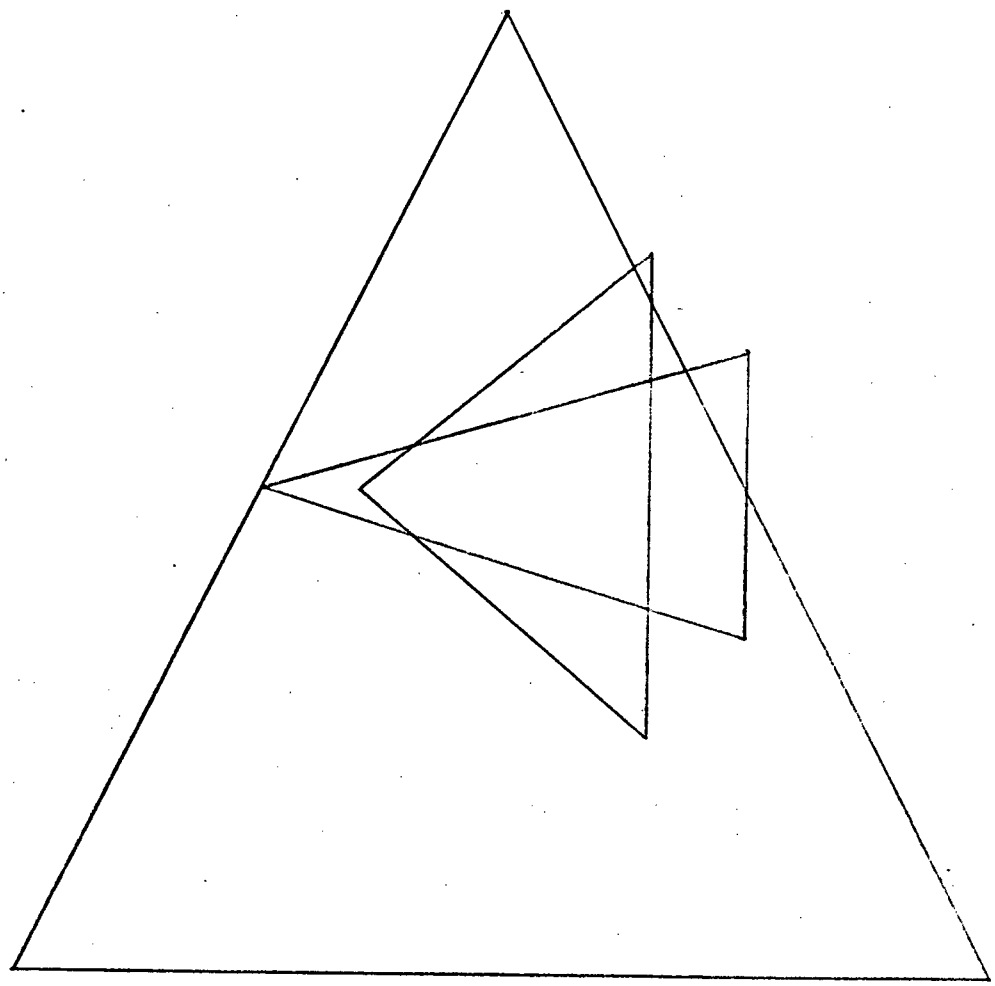


Fig.2.2 Transformation order

III. SYNTACTIC IMPROVEMENT FOR LIG STATEMENTS

3.1 Description of Syntax

Before a language is generated, the syntactic and semantic structure of the language must be specified. There are several ways to describe the syntactic structure of a language. For example, there are:

(a) The Backus Normal Form [8]

(b) Syntactic Chart [9]

(c) Compiler Directing Statement and Sentence [19]

The Compiler Directing Statement and Sentence is the method used to describe the language COBOL (COmmon Business Oriented Language). For comparison purpose, only the Backus Normal Form and the Syntactic Chart are presented, as both of them have been used to describe the language ALGOL [9] [18].

3.2 The Backus Normal Form

The Backus Normal Form is a metalanguage for the definition of the syntax of programming languages. Four metasymbols are employed. Their definitions are the following:

Symbol	Definition
< >	Angular brackets enclosing a construct. A construct is developed from the definitions of

the language. If a construct appears on the left-hand side of a define sign (i.e. ::=), it is defined with the definitions on the right-hand side of the define sign. If a construct appears on the right-hand side, it is used to define the construct on the left-hand side.

::= Define sign means that the construct on the left of the define sign is being defined by the construct or constructs on the right.

| Inclusive or means that the construct on the left of the vertical line is interchangeable with the construct on the right.

As an example, the definition of an integer in ALGOL and its syntactic description in Backus Normal Form is listed as follows:

<integer> :: = <unsigned integer>
| + <unsigned integer>
| - <unsigned integer>

<unsigned integer> ::= <digit>
| <unsigned integer> <digit>

<digit> ::= 0|1|2|3|4|5|6|7|8|9

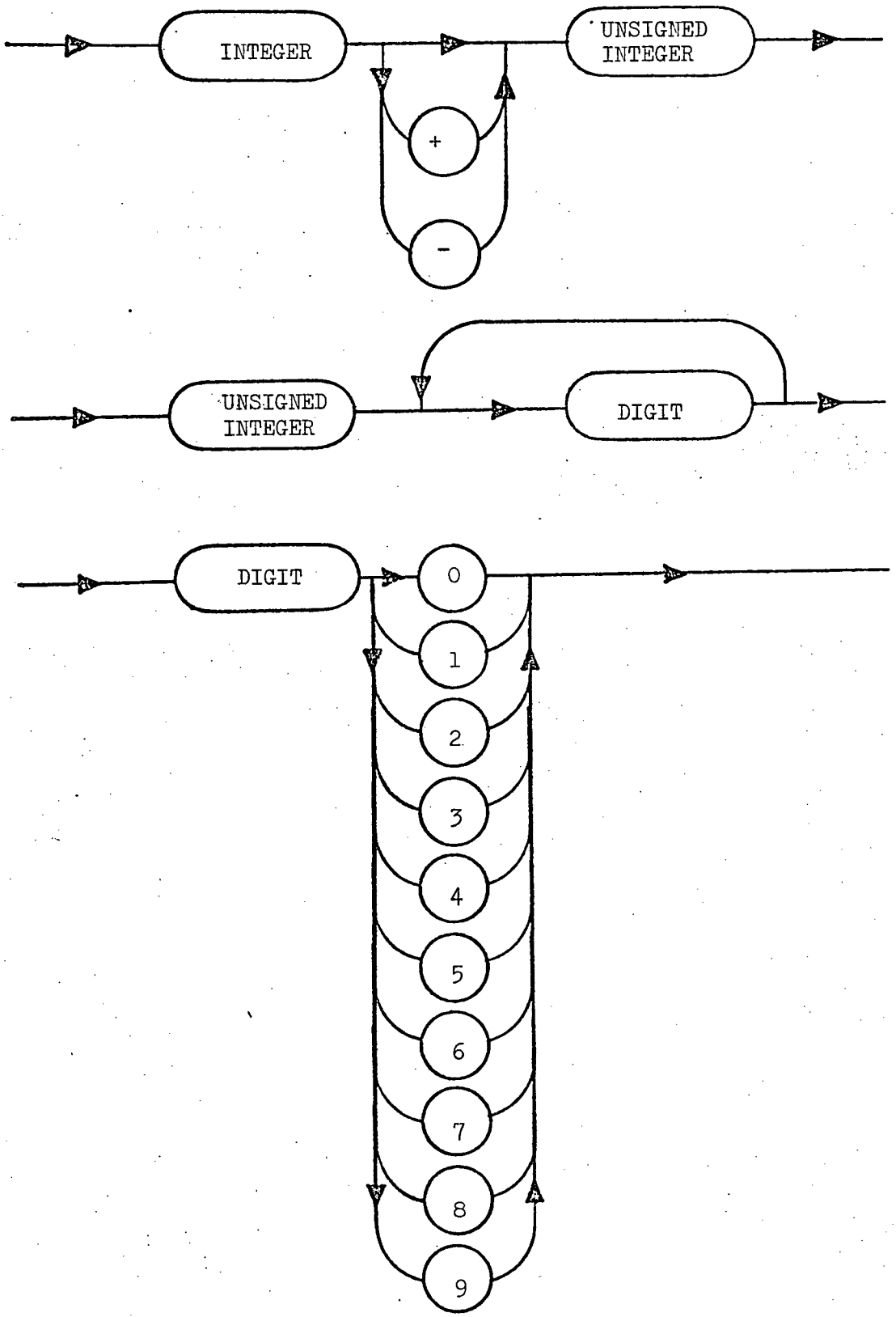


Fig.3.1 Syntactic Chart

The same definition of an integer presented in flowchart form by a Syntactic Chart is shown in Fig. 3.1.

The compiler generator XPL [12] which is used in the generation of the language LIG, requires a description of the syntactic structure of the language LIG in Backus Normal Form.

Specifically, using the Backus Normal Form notation, the graphical assignment of LIG was defined as follows. (Note: each BNF statement is called a production rule.)

```
<graphical assignment> ::=
        <graphical variable> <replace> <expression>
<replace> ::=   :=
<expression> ::= <primary>
               | <expression> + <primary>
               | <expression> - <primary>
<primary> ::= <display variable>
            | <display variable> <modification>
<display variable> ::= <graphical variable>
                     | <graphical primitive>
                     | <literal>
<graphical variable> ::= <identifier>
                       | <identifier> ( <variable> )
<variable> ::= <identifier> | <number>
```

```
<modification> ::= <modifier>
                 | <modification> <modifier>
<modifier> ::= AT <arguments>
             | SCALE <arguments>
             | ANGLE <variable>
             | ANGLE <variable> RAD
             | ANGLE <variable> DEG
             | FROM <arguments>
             | TO <arguments>
             | VSYM <variable>
             | HSYM <variable>
<arguments> ::= <variable> | <arguments> , <variable>
<graphical primitive> ::= BLANK|LINE|SQUARE
                        |TRIANGLE|CIRCLE|SCIRCLE
```

<identifier>, <number> and <literal> are internally defined in the XPL compiler [12]. Hence, to make the graphical assignment of a graphical variable TAT to the display variable TRIANGLE at the co-ordinate point (0.6,1.0), the LIG statement is coded as follows:

```
* TAT := TRIANGLE AT 0.6,1.0 ;
```

The symbols * and ; are defined in the original LIG language. The symbol * is used to notify the LIG preprocessor that this is not a Fortran statement. The LIG preprocessor

will translate the source statement beginning with the symbol *, and it passes all Fortran statement directly to the output file. The symbol ; is used to denote the end of a LIG statement.

3.3 Graphical Variable and Modification Attributes

A two-dimensional picture is composed of lines. There will be visible and invisible lines. Considered as a single unit, a picture is called a graphical object. The value of a graphical variable is a graphical object [21]. A graphical variable can be denoted by an identifier (e.g. LINE1, FLOWER, HOUSE etc). It can also be denoted by a subscripted variable. In Backus Normal Form notation,

<graphical variable> ::= <identifier>
| <identifier> (<variable>)

<variable> ::= <identifier> | <number>

Hence P(1) and BORT are legal graphical variable names. However, P(SQRT(2.3+N)) and Q(1,4) are illegal graphical variable names. The subscript of the first graphical variable is a function and the second graphical variable is not a single subscripted variable.

Compared to many high-level programming languages, the subscript of a variable can be a number, an identifier, a function or even an arithmetic expression. In order to retain

this compact feature, the syntactic structure and its implementation in LIG was improved.

As mentioned in Section 2.1, a graphics system should allow the application programmer to modify and manipulate a graphical object. In LIG statements, the modification operators are AT, SCALE, ANGLE, FROM, TO, VSYM AND HSYM. The value factor associated with these modification operators is called a modification attribute.

With reference to the Backus Normal Form notation in Section 3.2, the following examples are legal LIG modifications in a graphical assignment statements,

```
AT X,Y
SCALE 0.5,0.1
ANGLE 90 DEG
```

However, the following are illegal LIG modification in graphical assignment statements,

```
AT(3+4),(I/N*J)
SCALE SQRT(N),ROOT(6+T)
ANGLE N(1) DEG
```

In the previous LIG syntax, identifiers and numbers were the only two legal modification attributes. Therefore, the LIG syntax was improved, so that the modification attributes can also be a subscripted variable, a Fortran function or an

arithmetic expression. These improvements are reflected in the syntax as given in Appendix A.

3.4 Attribute Assignment and Numerical Assignment

The modification attributes associated with a graphical variable are stored in the data base. There are two LIG statements which can be utilized to modify and examine the values of any simple attribute bonded to a graphical variable. They are the attribute assignment statement and the numerical assignment statement. In Backus Normal Form notation, the grammatical rules are:

<attribute assignment>

::= <attribute function> = <variable>

<numerical assignment>

::= <variable> = <attribute function>

<attribute function>

::= <functionhead> <display variable>)

<functionhead> ::= <functionname> (

<functionname>

::= XLOC|YLOC|XSCALE|YSCALE|ANGLE| SUBSCRIPT

The attribute assignment statement changes the previous attribute of a graphical variable to a new one specified by the Fortran variable on its rignt hand side. The numerical assignment statement extracts the value of the attribute

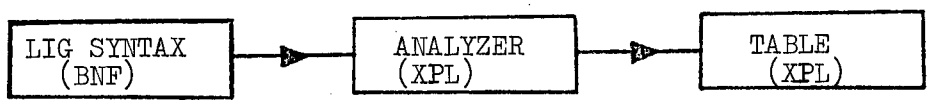
bonded to a graphical variable and assigns the value to a specified FORTRAN variable.

There are cases where an application programmer would like to change the attribute value by assigning the value of a variable, a function or an arithmetic expression. In the case of a numerical assignment statement, he might wish to assign the extracted attribute value to a subscripted Fortran variable.

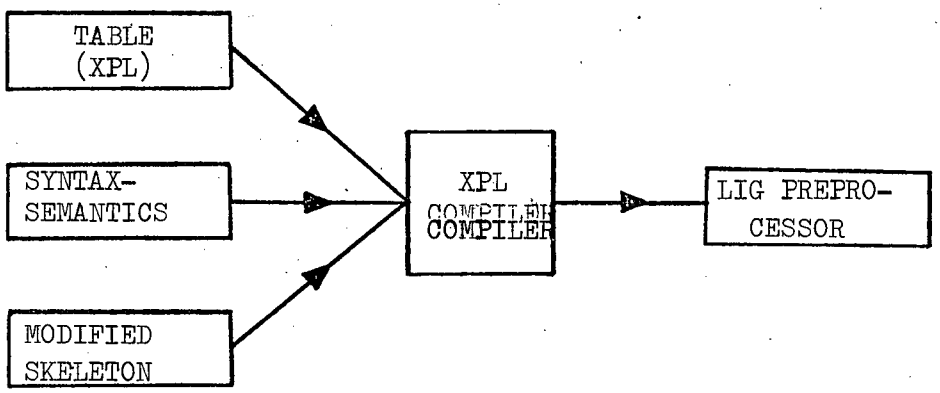
All these have been implemented to improve the syntactic structure of the LIG statements. These improvements are also reflected in the syntax as given in Appendix A.

3.5 LIG Preprocessor and Improvement Result

To change the LIG Preprocessor, the first step is to formulate the correct Backus Normal Form for the LIG syntactic structure. The XPL analyzer takes the LIG syntax in BNF form and determines if it is acceptable for the parser of the preprocessor. If there is no error, a table of XPL statements is generated [12].



The second step is to modify the XPL SKELETON. Then the table generated by the XPL analyzer, the syntax-semantics and the modified XPL SKELETON are combined and input to the XPL compiler as source codes. The resulting object code is the LIG Preprocessor which accepts the revised syntactic structure of LIG statements.



A listing of the test result is shown in the last two pages of this section. For the graphical assignment statement

```
* OBJ1(SUB1) := OBJ2(SUB2) AT X,Y SCALE SX,SY ANGLE A; ,
```

the LIG preprocessor will generate the following Fortran statements:

```
CALL ASSIG('OBJ1    ',SUB1)
CALL PRIMA(1,'OBJ2    ',SUB2,X,Y,SX,SY,A,1)
```

The last parameter of PRIMA will be 2 if scaling is applied before rotation. The preprocessor translates a numerical

assignment statement into a Fortran assignment statement with the function CORF1 on the right hand side. An attribute assignment statement will be translated as a Fortran call statement (i.e. the subprogram CHCHOR is invoked). The first parameter in CHCHOR specifies whether X, Y, SX, SY, ANGLE or SUBSCRIPT is to be examined or modified.

```
      L I G  - PREPROCESSOR   (MAY 27,1976 VERSION)

    TODAY IS: 06-28-76.  TIME = 15:46:55.80.


     1          *NOVA VERSION
     2          *STATE 1:
     3          C --- TO TEST THE LIG SYNTACTIC STRUCTURE
     4          C ----1ST , THE GRAPHICAL ASSIGNMENT STATEMENTS
     5          *OBJ(M(N(O(P(Q(R(S(((D+D)/(M+D))*G)))))))):=CIRCLE AT (S+D(I)+M(D*G+F/K)
     6          */M(I+1)),(SQRT((J+K)/M)) ANGLE FMT SCALE HAHA;
     7          *OBJ(SQRT(I+2*K/D)):=CIRCLE AT (X3+X4*(SQRT(D))),Y3 SCALE(F(I+M)),(M-D)
     8          *ANGLE ((ROOT(F)+SQRT(G))*D);
     9          C ----2ND, THE NUMERICAL ASSIGNMENT STATEMENTS
    10          *NEM1 = XLOC(OBJ2);
    11          *NUMBER(I) = YLOC(Q(1+I));
    12          *INT((SQRT(M**2)+C-D/Q)*D(I)-D)=XSCALE(F(ROOT(F*G)));
    13          *INT(1)=YSCALE(OBJ(1));
    14          *INT(I)= ANGLE (M (6));
    15          *INT(I*J/M)= SUBSCRIPT(OBJJJJJ);
    16          C ----3RD, ATTRIBUTE ASSIGNMENT STATEMENTS
    17          *XLOC(OBJ1) =ARITHEXP;
    18          * YLOC(OBJ   (2)   )    =    A+C    /D;
    19          *XSCALE(OBJ(I))=M**SQRT(D/C+D);
    20          *YSCALE(OBJ(ROOT(M)*D-C(I)))=SQRT(M**N**1)  - ROOT(Q);
    21          *ANGLE (QUEET) = 6;
    22          *SUBSCRIPT(INTEG(6+6)) = 6+1-4**2;
    23                STOP
    24          *END STATE 1;


     1          *EOF EOF EOF
*    ---    MAIN PROGRAM GENERATED    ---

COMPILATION TIME     0:0:0.14.


END OF COMPILATION
NO ERRORS WERE DETECTED.
```

```
      LIG ---- PREPROCESSOR OUTPUT
C --- NOVA VERSION --- (MAY 27,1976)
      OVERLAY OVS01
      SUBROUTINE S01
C      ----- STATE 1 -----
      LOGICAL HITON
      COMMON IDUMMY(5147)
C --- TO TEST THE LIG SYNTACTIC STRUCTURE
C ----1ST , THE GRAPHICAL ASSIGNMENT STATEMENTS
      CALL ASSIG('OBJ      ',(M(N(O(P(Q(R(S(((D+D)/(M+D))*G)))))))))
      CALL PRIMA(1,'CIRCLE  ',0,(S+D(I)+M(D*G+F/K)/M(I+1)),(SQRT((J+K)/M
     *)),HAHA,HAHA,FMT,2)
      CALL ASSIG('OBJ      ',(SQRT(I+2*K/D)))
      CALL PRIMA(1,'CIRCLE  ',0,(X3+X4*(SQRT(D))),Y3,(F(I+M)),(M-D),((RO
     *OT(F)+SQRT(G))*D),1)
C ----2ND, THE NUMERICAL ASSIGNMENT STATEMENTS
      NEM1 = CORF1(1,'OBJ2    ',0)
      NUMBER(I) = CORF1(2,'Q       ',(1+I))
      INT((SQRT(M**2)+C-D/Q)*D(I)-D) = CORF1(3,'F       ',(ROOT(F*G)))
      INT(1) = CORF1(4,'OBJ     ',(1))
      INT(I) = CORF1(5,'M       ',(6))
      INT(I*J/M) = CORF1(6,'OBJJJJJ ',0)
C ----3RD, ATTRIBUTE ASSIGNMENT STATEMENTS
      CALL CHCHOR(1,'OBJ1    ',0,ARITHEXP)
      CALL CHCHOR(2,'OBJ     ',(2),A+C/D)
      CALL CHCHOR(3,'OBJ     ',(I),M**SQRT(D/C+D))
      CALL CHCHOR(4,'OBJ     ',(ROOT(M)*D-C(I)),SQRT(M**N**1)-ROOT(Q))
      CALL CHCHOR(5,'QUEET   ',0,6)
      CALL CHCHOR(6,'INTEG   ',(6+6),FLOAT(6+1-4**2))
       STOP
       END
```

IV. PICTURE GENERATION BY GRAPHICAL FUNCTIONS

4.1 Picture Generation in LIG

In the high-level graphics language LIG, the graphical assignment statement is used to build up a picture, which is called the graphical object [7]. The graphical display statement will place a graphical object on the screen. Graphical objects can be decomposed into basic graphical objects, which are called graphical primitives. These graphical primitives cannot be further decomposed into other graphical elements. The set of graphical primitives consists of BLANK, LINE, TRIANGLE, SQUARE, CIRCLE and SCIRCLE (refer to Fig. 4.1).

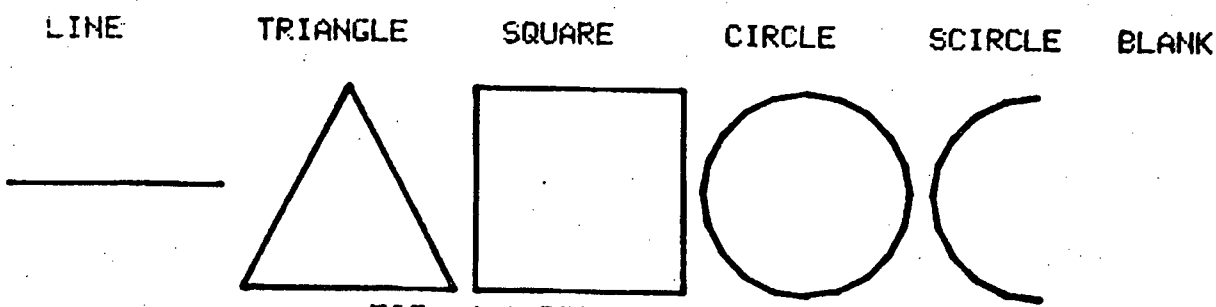


FIG. 4.1 PRIMITIVES

All LIG graphical primitives are defined within a unit square with the reference point at the centre of the unit square. Using these six graphical primitives, other graphical objects are built. The following two examples show the formation of a 2-input AND gate and the picture of a truck.

(a) 2-input AND gate (refer to Fig. 4.2)

```
* GATE :=  LINE FROM 0.1, 0.5 TO 0.3, 0.5
*         + SCIRCLE SCALE 0.6, 0.4 AT 0.6, 0.5
*         + LINE FROM 0.6, 0.7 TO 0.6, 0.3
*         + LINE FROM 0.6, 0.6 TO 0.9, 0.6
*         + LINE FROM 0.6, 0.4 TO 0.9, 0.4;
* DISPLAY GATE;
```

(b) A truck (refer to Fig. 4.3)

```
* TRUCK := SQUARE SCALE 0.3 AT 0.35, 0.65
*          + SQUARE SCALE 0.8, 0.3 AT 0.5, 0.35
*          + CIRCLE SCALE 0.2 AT 0.3, 0.2
*          + CIRCLE SCALE 0.2 AT 0.7, 0.2;
* DISPLAY TRUCK;
```

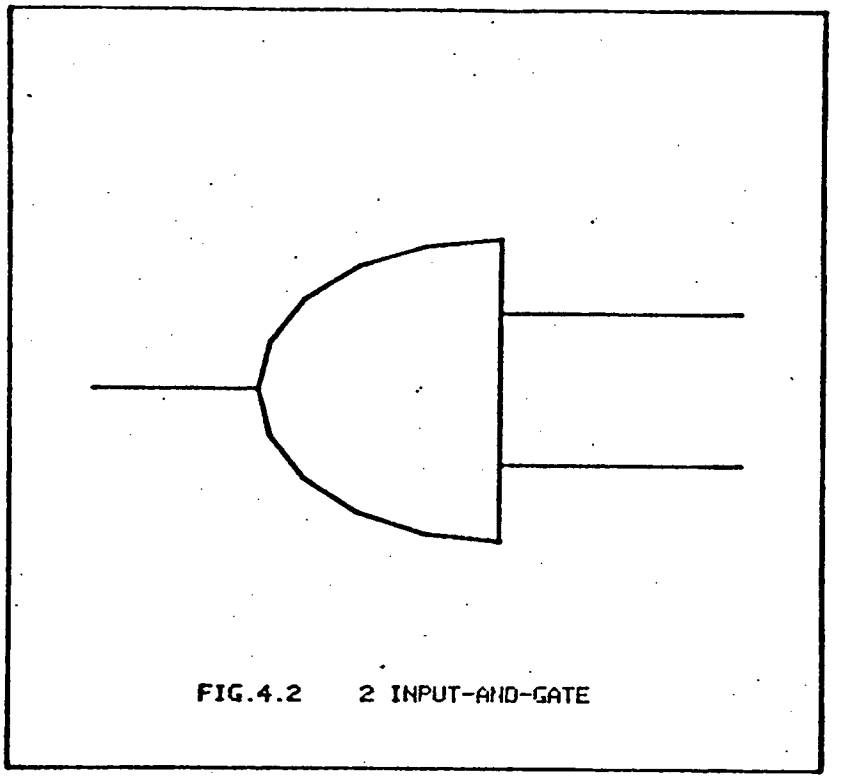


FIG.4.2 2 INPUT-AND-GATE

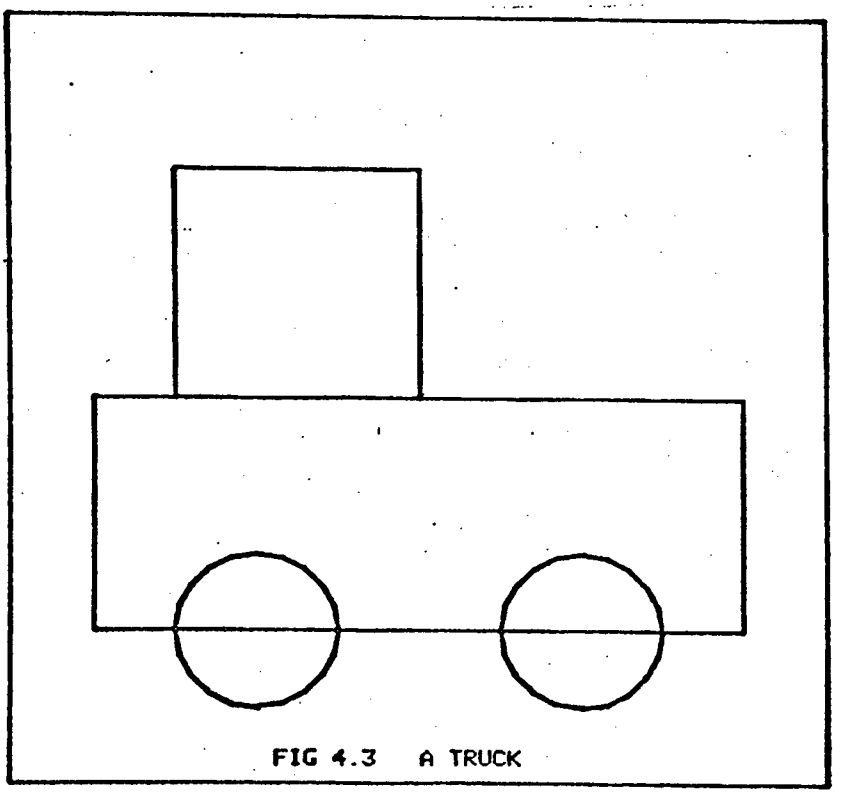
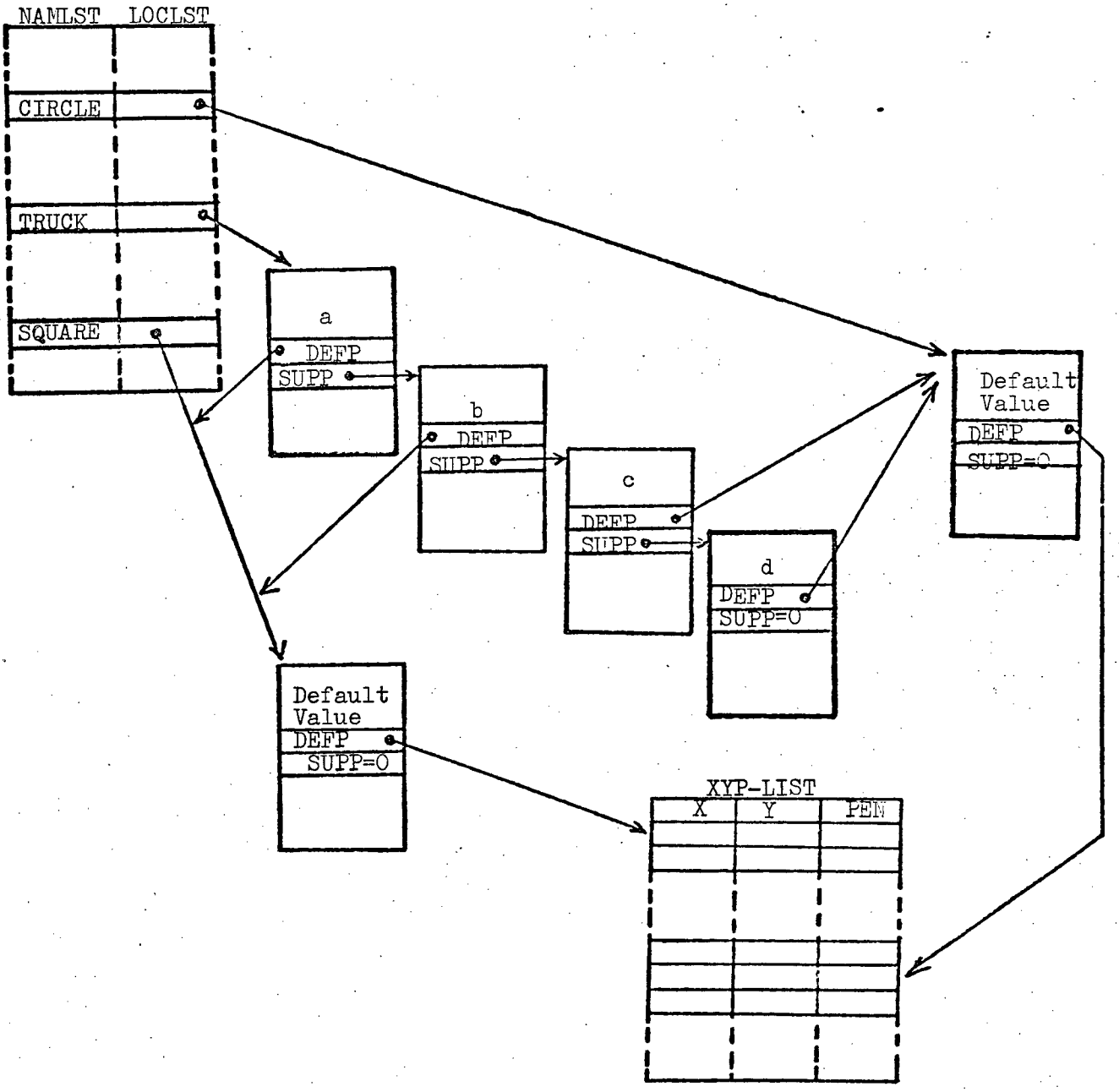


FIG 4.3 A TRUCK

4.2 The LIG Data Base

The LIG data base structure is shown in Fig. 4.4. It is a linked list structure [3] [4] [7] [24]. During system initialization, the name of the graphical primitives are inserted into the NAMLST. A hash algorithm is employed for name insertion [11] [13]. This algorithm was previously used in the previous LIG graphics system. The associated LOCLST contains the pointer to a LIG data block. There are 200 LIG data blocks. Each data block consists of the modification attribute values in compact form, the superposition pointer (SUPP), the definition pointer (DEFP), the index, the index pointer and the use pointer.

For a graphical primitive, the definition pointer is biased by a value of 10000. Its true value is used as a pointer, pointing to the first entry in the XYP list. The XYP list contains the co-ordinate pairs and the Pen-value, a value indicating whether the pen is in writing or non-writing position. To display a picture on the screen, the algorithm discussed in Section 2.6 is used. In order to keep track of all the modification attribute values, a push down stack is implemented.



* TRUCK := SQUARE a + SQUARE b + CIRCLE c + CIRCLE d ;
where a,b,c and d are modification attributes.

Fig.4.4 LIG Data Base Structure

Besides the superposition operator '+', there is also a deletion operator '-'. The deletion operator will delete the appropriate data block from the LIG data base. The deleted data block in the previous LIG version could not be re-used again. For the improved LIG graphic system, a simple garbage collection algorithm is implemented. The released data block is linked back to the data block pool, which is then available for further use.

4.3 Graphical Function

As there are only six basic LIG graphical primitives, the generation of a graphical object is somewhat limited. This is particularly true when an application programmer wants to generate a graphical object composed of irregularly curved lines. An appropriate example is the generation of an arc. Hence, a graphical function feature was implemented in the graphic language LIG [10] [14].

By definition, a function is almost a universal operator, which yields a result, called the value of the function. In the following Fortran statement,

```
A = SUM(C,D)
```

SUM is a function, whose value is the sum of C and D. The resulting value of C + D is associated with the name SUM. Similarly, a graphical function is a function which delivers

a value of type graphical. The sequence of operations defining the function may be dependent on a set of input parameters which usually appear in an argument list. For the above Fortran statement, with different C and D, the resulting A is different. A graphical function generates different graphical objects depending on the parameters. A graphical function is invoked by its appearance in a graphical expression. Thus, ARC(X1, Y1, X2, Y2, X3, Y3) is a graphical function which will generate an arc passing through the points (X1,Y1), (X2,Y2) and (X3, Y3). For the LIG statement,

```
* DISPLAY ARC(0.1, 0.2, 0.4, 0.6, 0.1, 0.3); ,
```

the graphical function ARC is referred to, and so the requested arc is generated and displayed on the screen. In this case, the parameters will not be stored in the data base. For the LIG statements

```
* P := ARC(X1, Y1, X2, Y2, X3, Y3) AT X4, Y4 SCALE M ;
* DISPLAY P ;
```

the value of the parameters and the modification attributes have to be stored into the data base. These data values will be retrieved when the graphical object P is to be displayed on the screen.

4.4 Implementation Scheme

In the former LIG graphics system, when a graphical variable was referred to for display on the screen, its name was checked by the subroutine NMSRCH (name searching subprogram). If it is not a graphical primitive or a predefined graphical object, an error message is generated. For the implementation of the graphical function feature, instead of generating an error message, a name not identified as a graphical variable or primitive is handled as a function call. The program will call the subroutine FSWCH (graphical function switch subprogram) which will further check for the existence of this graphical function. Error messages will be generated if there is no such graphical function. If there is a match, the system or user graphical function is called. That is, the appropriate XYP-list will be generated and passed back to the subroutine DISPL (graphical object displaying subprogram). DISPL will apply the necessary modification to this XYP-list and display the generated picture on the screen.

In the case when a graphical assignment is involved, as in

```
* P := A(M,N) AT X, Y SCALE SX, SY ANGLE A DEG; ,
```

the values of the modification attributes in compact form are stored into a data block. The name of the graphical function

A, and the parameters (M,N) have to be kept. This information is registered in another data block. The definition pointer of this data block, which contains the modification attribute values, is biased by a value of 20,000. Its true value is a pointer, pointing to the data block which stores the graphical function name and the associated parameters. Therefore, when P is to be displayed on the screen, the biased pointer will be used in directing the necessary action to be performed.

4.5 Generation of Special Plane Curves by a Graphical Function: Examples

Using the graphical function feature, an application programmer can generate a graphical object which is impossible in the previous LIG graphic system. Some of the examples are listed as follows:

a) Hypocycloid with four cusps (refer to Fig. 4.5)

This is a curve described by a point P on a circle of radius a/4 as it rolls on the inside of a circle of radius a. The curve can be described by the equations in parametric form:

$$x = a \cos^3 A$$

$$y = a \sin^3 A$$

Using LIG statements, the function was coded as follows

```
*     FUNCTION CUSPS:
C --- TO GENERATE A HYPOCYCLOID WITH FOUR CUSPS
```

```
*       START(1.0,0.5);
        DO 10 K=1,59
        T=T*0.1047199
        T1=COS(T)
        T2=SIN(T)
        T3=0.5+0.5*T1*T1*T1
        T4=0.5-0.5*T2*T2*T2
*       FORWARD(T3,T4);
10      CONTINUE
*       STOP(1.0,0.5);
        RETURN
        END
```

(b) Three-leaved Rose (refer to Fig. 4.6)

The curve is described by the equation r=a cos 3A .

Using LIG statements, the function was coded as follows:

```
*       FUNCTION ROSE3:
C --- TO GENERATE A ROSE OF 3 LEAVES
        ANGLE=0.01282
*       START(1.0,0.5);
        DO 10 K=1,490
        T=K*ANGLE
        TT=A*COS(3*T)
        T1=0.5+TT*COS(T)
        T2=0.5-TT*SIN(T)
*       FORWARD(T1,T2);
```

```
10      CONTINUE
*       STOP(1.0,0.5);
        RETURN
        END
```

(c) Four-leaved Rose (refer to Fig. 4.7)

The curve is described by the equation r=a cos2A.

Using LIG statements, the function was coded as follows:

```
*       FUNCTION ROSE4:
C --- TO GENERATE A ROSE OF 4 LEAVES
        ANGLE=0.00641
*       START(1.0,0.5);
        DO 10 K=1,490
        T=K*ANGLE
        TT=A*COS(4*T)
        T1=0.5+TT*COS(T)
        T2=0.5-TT*SIN(T)
*       FORWARD(T1,T2);
10      CONTINUE
*       STOP(1.0,0.5);
        RETURN
        END
```

In general, for the case (b) and (c) the equation is

$$r = a \cos nA$$

or $r = a \sin nA$

The generated graphical object has n leaves if n is odd, or

2n leaves if n is even and it degenerates into a circle for n = 0.

Using LIG statements, the function was coded as follows:

```
*       FUNCTION ROSE(N):
        ANGLE=0.01282
        L=N/2
        L=L*2
        IF(N.NE.L) ANGLE=0.00641
*       START(1.0,0.5);
        DO 10 K=1,490
        T=K*ANGLE
        TT=A*COS(N*T)
        T1=0.5+TT*COS(T)
        T2=0.5-TT*SIN(T)
*       FORWARD(T1,T2);
10      CONTINUE
*       STOP(1.0,0.5);
        RETURN
        END
```

4.6 Graphical Assignment Statement with a Graphical Function

When a graphical function is invoked in a graphical assignment statement, the name of the graphical function and its associated parameters have to be stored into the LIG data base. In the display process, this information is retrieved

from the data base. Computations are then performed and the requested graphical object, which is generated by the graphical function, is built. After applying the necessary modifications to this generated graphical object, it will be displayed on the screen. For example, the LIG statements

```
* P:= ROSE(0) + ROSE(2) + ROSE(4) ;
* DISPLAY P ;
```

will generate a picture as shown in FIG. 4.8.

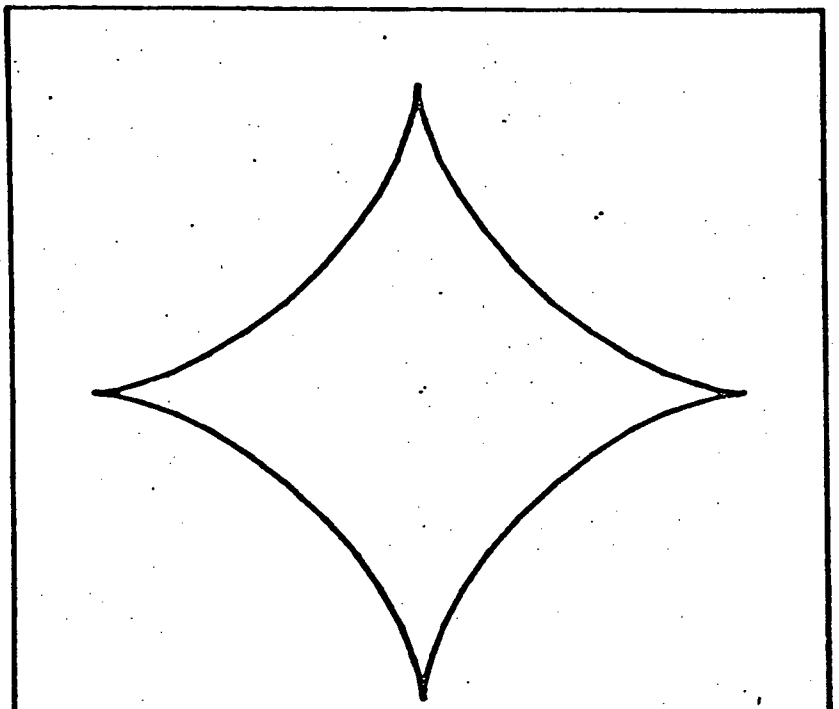


FIG.4.5 HYPOCYCLOID WITH FOUR CUSPS

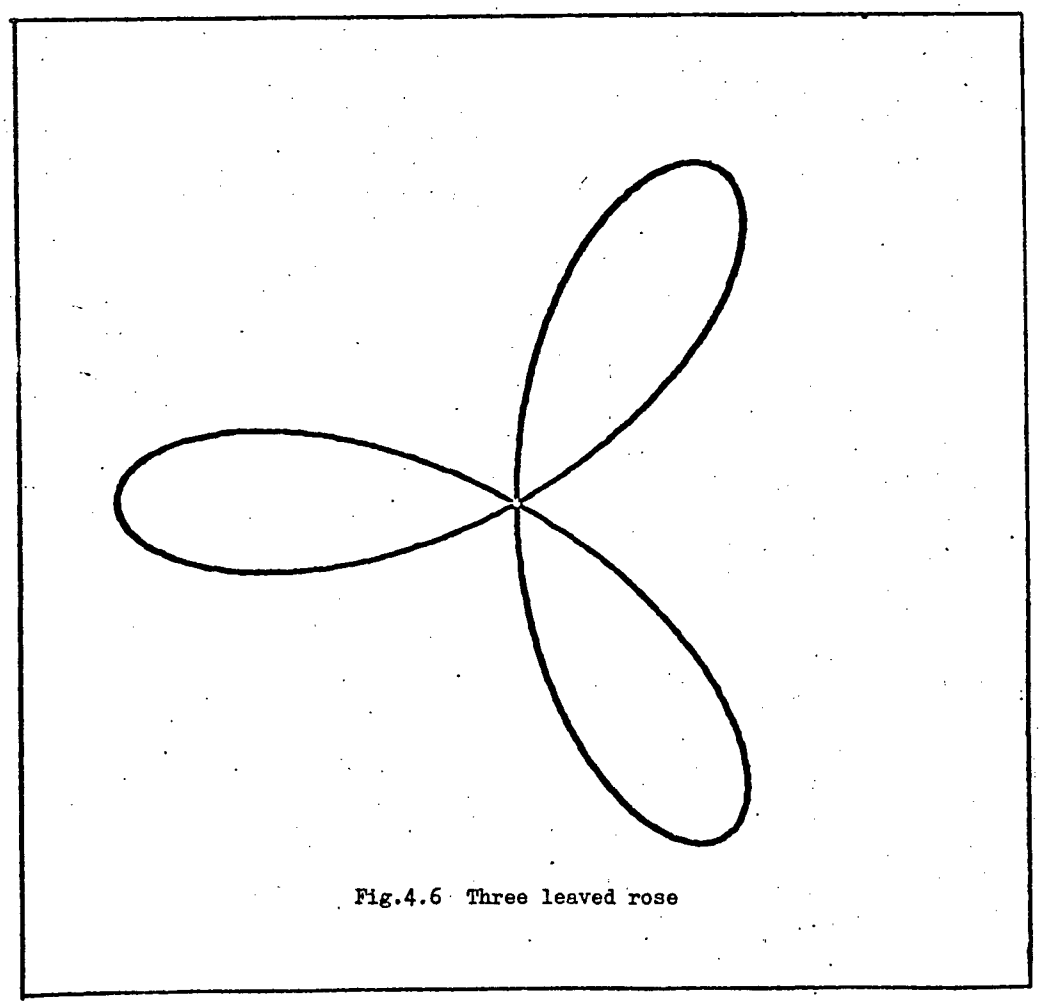


Fig.4.6 Three leaved rose

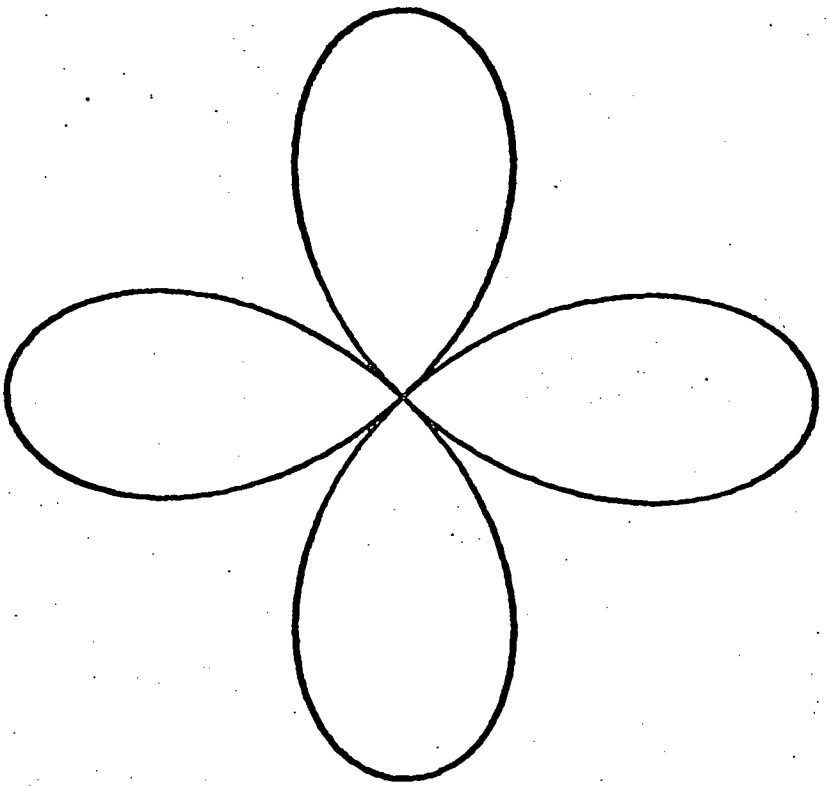


Fig.4.7 Four leaved rose

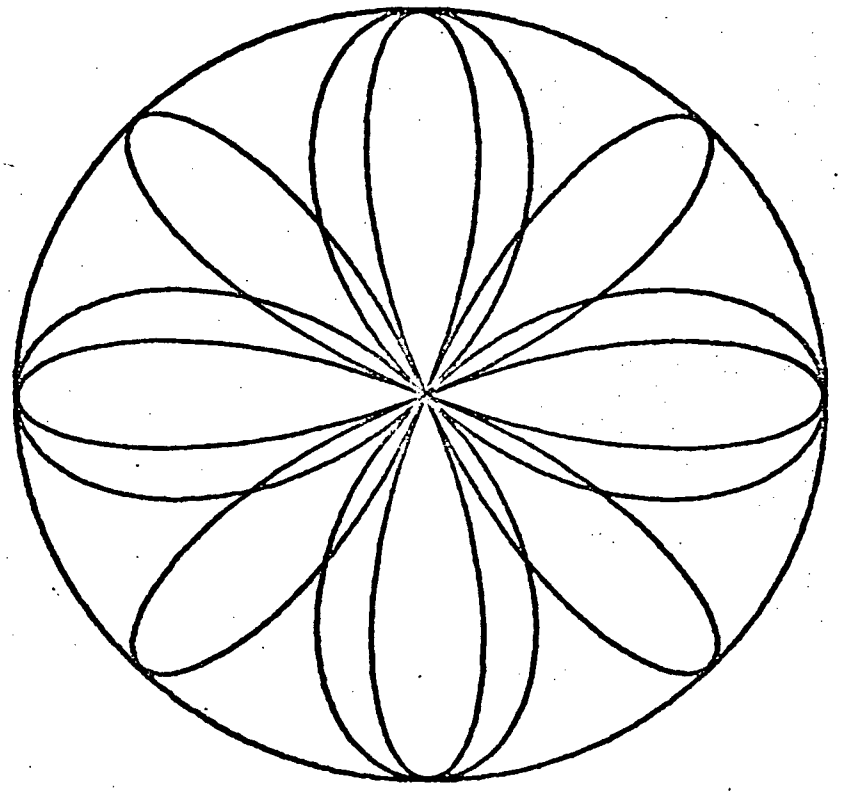


Fig.4.8 Function Generation

V. CONCLUSION

After investigating the effect of applying a different sequence of transformations to the same graphical object, it was shown that the pictures generated were not identical. Thus, the dependence on the order of transformation is significant. The previous LIG graphics system ignores the order of appearance of the transformation entity in a LIG statement. Then for whatever sequence of transformations were applied to a graphical object, the same picture was generated.

As the order of appearance of the transformation entity in a LIG statement can only be detected by the LIG preprocessor, the XPL Skeleton was modified to incorporate the detection of the transformation sequence. Then a modified LIG preprocessor was generated by means of the XPL compiler. The LIG supporting system was modified as well, such that it will generate the appropriate picture.

The grammar of the previous LIG graphics language does not accept arithmetic expressions in any LIG statements. The modification attribute and the subscript could only be a number or an identifier.

In order to accept any imbedded arithmetic expression, the grammar of the LIG graphics language was changed. With

the revised grammar, a new XPL table was generated, and was used to build the revised LIG preprocessor. The LIG statements which were affected by the changed grammatical rules are the graphical assignment statement, the numerical assignment statement and the attribute assignment statement.

A graphical function feature was proposed and incorporated into the LIG graphics system. It was shown that a variety of graphical objects can be generated by the use of this feature. The graphical object generated also depends on the associated parameters. The grammar of the previous LIG graphics language was modified to accept the graphical function feature. The LIG supporting system was also modified. The revised LIG graphics system will distinguish a graphical function from a graphical variable. At the same time, a simple garbage collection algorithm was implemented. Thus, a released data block can be made available for future use. In the previous LIG graphics system, released data blocks were not made available for re-use.

The following list summarizes the improvements, which have been implemented into the LIG graphics language and system:

(a) For concatenated transformations, the resulting transformation has the same effect as sequential applications of the original

transformations.

(b) LIG statements can have imbedded arithmetic expressions.

(c) More variety of graphical objects can be generated by means of graphical functions. Using a graphical function, the graphical object can be changed dynamically by altering the associated paramenter list.

(d) The garbage collector scheme enables the re-use of released data blocks.

(e) The LIG system supporting subprograms were recoded and combined to reduce the number of unnecessary subroutine calls. This reduces the possibility of stack overflow during program execution.

Further improvements which can be made to the LIG graphics language and system are:

(a) Investigate the possibility of using the center of the screen as the origin of the co-ordinate system.

(b) Delete the symbol * and ; from the LIG statement.

(c) Modify the HITON construct, so that

identification will not be made to an non-displayed graphical variable.

(d) Implement a graphical input statement, using a writing tablet as an input device, or using the inking technique.

(e) Include graphical type parameters in a graphical function.

REFERENCES

[1] F.Ayres, Theory and Problems of Matrices, McGraw-Hill, 1967

[2] T.S.Berk, TUNA : A High-Level Graphical Programming Language, Technical Report, Mathematical Sciences Department, Florida International University, February 1973

[3] A.T.Berztiss, Data Structure Theory and Practice, Academic Press, 1971

[4] J.Earley, Toward an Understanding of Data Structures, Communication of ACM, Vol. 14, No. 10, October 1971, p.617-627

[5] L.Fox, An Introduction to Numerical Linear Algebra, Oxford University Press, London 1964

[6] A.J.Frank, B-line, Bell Line Drawing Language, AFIPS 1968 FJCC, p.179-191

[7] J.C.Gray, Compound Data Structure of Computer Aided Design: A Survey, Proceedings ACM National Meeting, 1967, p.355-365

[8] D.Gries, Compiler Construction for Digital Computer, John Wiley and Sons, 1972

[9] H.Kanner, P.Kosinski and C.L.Robinson, The Structure of Yet Another ALGOL Compiler, Programming Systems and Languages, McGraw-Hill, 1967, p.228-252

[10] N.Marovac, A Single Data-display Structure: A New View on Interactive Computer Graphics in CAD, The Computer

Journal, Vol. 16, No. 2, June 1971, p.152-156

[11] W.D.Maurer, T.G.Lewis, Hash Table Methods, Computing Surveys, Vol. 7, No. 1, March 1975, p.5-19

[12] W.M.McKeeman, J.J.Horning and D.B.Wortman, A Compiler Generator, Prentice-Hall, 1970

[13] R.Morris, Scatter Storage Techniques, Communications of ACM, Vol. 11, No. 1, January 1968, p.38-44

[14] W.Newman, Display Procedures, Communications of ACM, Oct. 1971, Vol. 14, No. 10, p.651-660

[15] W.M.Newman, R.F.Sproull, Principles of Interactive Computer Graphics, McGraw-Hill, 1973

[16] B.Pieké, Design and Implementation of a High-Level Language for Interactive Computer Graphics, Department of Electrical Engineering, The University of British Columbia, July 1973

[17] B.Pieké, G.F.Schrack, Implementation of an Interactive Graphics Language, Proc. Third Man-Computer Communications Seminar of the National Research Council, Ottawa, 1973, p.7.1-9

[18] S.Rosen, The ALGOL Programming Language, Programming Systems and Languages, McGraw-Hill 1976, p.48-78

[19] J.E.Sammet, Basic Elements of COBOL 61, Programming Systems and Languages, McGraw-Hill, 1967, p.119-159

[20] G.F.Schrack, LIG, Language for Interactive Graphics: Users' Manual, Department of Electrical Engineering, The

University of British Columbia, 1975

[21] G.F.Schrack, Design, Implementation and Experiences with a High-level Graphics Language for Interactive Computer-Aided Design Purposes, Computer Graphics, Vol. 10, Spring 1976, p.10-17

[22] D.N.Smith, GPL/I - A PL/I Extension for Computer Graphics, AFIPS 1971 SJCC, p.511-528

[23] C.N.Turrill, W.R.Mallgren, XPLG - Experiences in Implementing an Experimental Interactive Graphics Programming System, Comput. and Graphics, Vol. 1, 1975, p.55-63

[24] R.Williams, A Survey of Data Structures for Computer Graphics Systems, Computing Surveys, Vol. 3, No. 1, March 1971, p.1-21

[25] B.L.M.Wyvill, GROAN, University of Bradford, June 1974

APPENDIX A

LIG GRAMMAR IN BNF NOTATION

```
1   <PROGRAM>  ::=  <STATES>

2   <STATES>  ::=  <STATE>
3               |  <STATES> <STATE>

4   <STATE>  ::=  <STATE HEAD> : <STATEMENTLIST> <STATE ENDING> ;
5               |  <FUNCTION SUBPROGRAM HEAD> : <FUNCTION STATEMENTLIST> <FUNCTION SUBPROGRAM ENDING> ;

6   <STATE HEAD>  ::=  STATE <NUMBER>

7   <STATEMENTLIST>  ::=  <STATEMENT>
8                       |  <STATEMENTLIST> <STATEMENT>

9   <STATE ENDING>  ::=  END STATE <VARIABLE>

10  <STATEMENT>  ::=  <GRAPHICAL ASSIGNMENT> ;
11                  |  <ATTRIBUTE ASSIGNMENT> ;
12                  |  <NUMERICAL ASSIGNMENT> ;
13                  |  <DISPLAY STATEMENT> ;
14                  |  <IDENTIFICATION CONSTRUCT> ;
15                  |  <STATE-TRANSFER STATEMENT> ;
16                  |  <CURSOR STATEMENT> ;
17                  |  <ERASE STATEMENT> ;
18                  |  <DRAW STATEMENT> ;
19                  |  <FILE STATEMENT> ;
20                  |  <READ STATEMENT> ;
21                  |  ;

22  <FUNCTION SUBPROGRAM HEAD>  ::=  <FUNCTION HEADING> ( <FUNCTION PARAMETERLIST> )

23  <FUNCTION HEADING>  ::=  FUNCTION <IDENTIFIER>

24  <FUNCTION PARAMETERLIST>  ::=  <FUNCTION PARAMETERLIST> , <IDENTIFIER>
25                               |  <IDENTIFIER2>

26  <FUNCTION SUBPROGRAM ENDING>  ::=  RETURN <IDENTIFIER>

27  <FUNCTION STATEMENTLIST>  ::=  <FUNCTION STATEMENT>
28                               |  <FUNCTION STATEMENTLIST> <FUNCTION STATEMENT>

29  <FUNCTION STATEMENT>  ::=  <LINE STATEMENT> ;

30  <LINE STATEMENT>  ::=  START <XYVALUES>
31                       |  FORWARD <XYVALUES>
32                       |  STOP <XYVALUES>

33  <XYVALUES>  ::=  <ARGUMENT EXPRESSION> , <ARGUMENT EXPRESSION>

34  <GRAPHICAL ASSIGNMENT>  ::=  <GRAPHICAL VARIABLE> <REPLACE> <EXPRESSION>

35  <REPLACE>  ::=  : =
```

```
36  <ATTRIBUTE ASSIGNMENT>  ::=  <ATTRIBUTE FUNCTION> = <ARITHMETIC EXPRESSION>

37  <NUMERICAL ASSIGNMENT>  ::=  <IDENTIFIER> = <ATTRIBUTE FUNCTION>
38                             |  <SUBSCRIPTED VARIABLE> = <ATTRIBUTE FUNCTION>

39  <DISPLAY STATEMENT>  ::=  DISPLAY <DISPLAY>
40                          |  DISPLAY <DISPLAY> ONTO <ARGUMENTS>

41  <DISPLAY>  ::=  <DISPLAY VARIABLE>
42                |  <DISPLAY VARIABLE> <MODIFICATION>
43                |  <DISPLAY VARIABLE> WITHIN <ARGUMENTS>

44  <IDENTIFICATION CONSTRUCT>  ::=  <IDENTIFICATION HEAD> : <STATEMENTLIST> <ENDING>

45  <IDENTIFICATION HEAD>  ::=  FOR HIT ON <IDENTIFIER>

46  <ENDING>  ::=  END

47  <STATE-TRANSFER STATEMENT>  ::=  GOTO STATE <VARIABLE>
48                                 |  GO TO STATE <VARIABLE>

49  <CURSOR STATEMENT>  ::=  CURSOR ON

50  <ERASE STATEMENT>  ::=  ERASE SCREEN

51  <DRAW STATEMENT>  ::=  DRAW <MODIFICATION>

52  <FILE STATEMENT>  ::=  STORE <ARGUMENTS>
53                       |  RESTORE <ARGUMENTS>

54  <READ STATEMENT>  ::=  READ CURSOR <IDENTIFIER4> <,3> <IDENTIFIER>
55                       |  READ KEY <IDENTIFIER>
56                       |  READ SUBSCRIPT <IDENTIFIER>

57  <EXPRESSION>  ::=  <PRIMARY>
58                   |  <EXPRESSION> + <PRIMARY>
59                   |  <EXPRESSION> - <PRIMARY>

60  <PRIMARY>  ::=  <DISPLAY VARIABLE>
61                |  <DISPLAY VARIABLE> <MODIFICATION>

62  <DISPLAY VARIABLE>  ::=  <GRAPHICAL VARIABLE>
63                         |  <GRAPHICAL PRIMITIVE>
64                         |  <LITERAL>

65  <GRAPHICAL VARIABLE>  ::=  <IDENTIFIER>
66                           |  <SUBSCRIPTED VARIABLE>

67  <SUBSCRIPTED VARIABLE>  ::=  <IDENTIFIER3> <(1> <INDEX OR PARAMETERS> )
```

```
68  <INDEX OR PARAMETERS>  ::=  <ARITHMETIC EXPRESSION1> <,2> <INDEX OR PARAMETERS>
69                           |  <ARITHMETIC EXPRESSION>

70  <LITERAL>  ::=  <STRING>
71               |  <CONVERSION FUNCTIONHEAD> <ARGUMENTS> )

72  <CONVERSION FUNCTIONHEAD>  ::=  <CONVERSION FUNCTIONNAME> (

73  <CONVERSION FUNCTIONNAME>  ::=  IVALUE
74                               |  RVALUE
75                               |  TVALUE

76  <VARIABLE>  ::=  <IDENTIFIER>
77                |  <NUMBER>

78  <ATTRIBUTE FUNCTION>  ::=  <FUNCTIONHEAD> <DISPLAY VARIABLE> )

79  <FUNCTIONHEAD>  ::=  <FUNCTIONNAME> (

80  <FUNCTIONNAME>  ::=  XLOC
81                    |  YLOC
82                    |  XSCALE
83                    |  YSCALE
84                    |  ANGLE
85                    |  SUBSCRIPT

86  <MODIFICATION>  ::=  <MODIFIER>
87                    |  <MODIFICATION> <MODIFIER>

88  <MODIFIER>  ::=  AT <ARGUMENTS>
89                |  SCALE <ARGUMENTS>
90                |  ANGLE <ARGUMENTS>
91                |  ANGLE <ARGUMENTS> RAD
92                |  ANGLE <ARGUMENTS> DEG
93                |  FROM <ARGUMENTS>
94                |  TO <ARGUMENTS>
95                |  VSYM <ARGUMENTS>
96                |  HSYM <ARGUMENTS>

97  <ARGUMENTS>  ::=  <ARGUMENT EXPRESSION>
98                 |  <ARGUMENTS> <,1> <ARGUMENT EXPRESSION>

99  <GRAPHICAL PRIMITIVE>  ::=  BLANK
100                           |  LINE
101                           |  SQUARE
102                           |  TRIANGLE
103                           |  CIRCLE
104                           |  SCIRCLE
```

```
105  <ARGUMENT EXPRESSION>  ::=  <(2> <ARITHMETIC EXPRESSION> )
106                             |  <VARIABLE>

107  <ARITHMETIC EXPRESSION>  ::=  <ARITHMETIC EXPRESSION> <+1> <TERM>
108                               |  <ARITHMETIC EXPRESSION> <-1> <TERM>
109                               |  <+2> <TERM>
110                               |  <-2> <TERM>
111                               |  <TERM>

112  <TERM>  ::=  <TERM> * <ARITHMETIC PRIMARY>
113            |  <TERM> / <ARITHMETIC PRIMARY>
114            |  <TERM> <*1> <*2> <ARITHMETIC PRIMARY>
115            |  <ARITHMETIC PRIMARY>

116  <ARITHMETIC PRIMARY>  ::=  <(3> <ARITHMETIC EXPRESSION> )
117                           |  <SUBSCRIPT HEAD> <ARITHMETIC EXPRESSION> )
118                           |  <VARIABLE>

119  <SUBSCRIPT HEAD>  ::=  <IDENTIFIER1> (
120                       |  <SUBSCRIPT HEAD> <ARITHMETIC EXPRESSION> ,

121  <(1>  ::=  (

122  <(2>  ::=  (

123  <(3>  ::=  (

124  <IDENTIFIER1>  ::=  <IDENTIFIER>

125  <IDENTIFIER2>'  ::=  <IDENTIFIER>

126  <IDENTIFIER3>  ::=  <IDENTIFIER>

127  <IDENTIFIER4>  ::=  <IDENTIFIER>

128  <+1>  ::=  +

129  <+2>  ::=  +

130  <-1>  ::=  -

131  <-2>  ::=  -

132  <*1>  ::=  *

133  <*2>  ::=  *

134  <,1>  ::=  ,

135  <,2>  ::=  ,

136  <,3>  ::=  ,

137  <ARITHMETIC EXPRESSION1>  ::=  <ARITHMETIC EXPRESSION>
```

APPENDIX B

SYNATX SEMANTICS AND SKELETON

```
/*            SYNTAX SEMANTICS FOR THE  L I G  LANGUAGE    */

FORTOUT:              /* OUTPUT A FORTRAN STATEMENT */
  PROCEDURE(LINEP);
     DECLARE (LINEP, LINE1) CHARACTER;

     IF LENGTH(LINEP) < 67 THEN
     DO;
        OUTPUT(2) = '      '||LINEP; RETURN;
     END;
     ELSE
     DO;
        LINE1 = SUBSTR(LINEP,0,66);
        LINEP = SUBSTR(LINEP,66);
        OUTPUT(2) = '      '||LINE1;
     END;
     DO FOREVER;
       IF LENGTH(LINEP) < 67 THEN
        DO;
           OUTPUT(2) = '     *'||LINEP; RETURN;
        END;
        ELSE
        DO;
           LINE1 = SUBSTR(LINEP,0,66); LINEP = SUBSTR(LINEP,66);
           OUTPUT(2) = '     *'||LINE1;
        END;
     END;
  END FORTOUT;

REMOVE_DECIMAL:     /*   REMOVE DECIMAL POINT FROM END OF NUMBER   */
  PROCEDURE(S1) CHARACTER;
     DECLARE S1 CHARACTER;

     K = LENGTH(S1)-1;
     IF BYTE(S1,K) = BYTE('.') THEN S1 = SUBSTR(S1,0,K);
     RETURN S1;
  END REMOVE_DECIMAL;

ARG_CHECK:    /*   CHECK FOR CORRECT NUMBER OF ARGUMENTS   */
  PROCEDURE(NUM);
     DECLARE NUM FIXED;

     IF NUM ¬= ARG_COUNT THEN
               CALL ERROR('NUMBER OF ARGUMENTS INCORRECT',0);
     ARG_COUNT = 0;
  END ARG_CHECK;
```

```
RESET:   /*   RESET MODIFIERS TO DEFAULT VALUES   */
  PROCEDURE;

     LOC = '.5,.5';
     SCALE = '1.,1.';
     ANGLE = '0.';
     SUBSCRIPT = '0';
     PRCODE = 0;
         TRANSFORMATION_ORDER = '1';
         FIRST_ENCOUNTER = 1;
        PARAMETER(1)='0';

        PARAMETER(2)='0';
        PARAMETER(3)='0';
        PARAMETER(4)='0';
        PARAMETER(5)='0';
        PARAMETER(6)='0';
        PARAMETERCOUNT=0;
  END RESET;

PAD_8:
  PROCEDURE(S1) CHARACTER;
     DECLARE S1 CHARACTER;

        S1 = SUBSTR(S1||'         ',0,8);
     RETURN S1;
  END PAD_8;

EXOUT:    /*   TO OUTPUT CALLS INVOLVING EXPRESSIONS   */
  PROCEDURE(EXCODE,VARTYPE,PRCODE);
     DECLARE(EXCODE,VARTYPE,PRCODE) FIXED;

     K = EXCODE;
     IF PRCODE = 1 THEN
     DO;
        OUTPUT(2) = '      CALL UNARY('||UNARY_OP||')';
        K = -K;
     END;
     I = LENGTH(VAR(MP))-2;        /*  THE MAXIMUM LENGTH OF AN INDIVIDUAL   */
     IF I>50 THEN                  /*      STRING IS 50 CHARACTERS           */
     DO;
        VAR(MP) = SUBSTR(VAR(MP),0,49);
        I = 50;
     END;
        PARAMETERLINE=PARAMETER(1)||','||PARAMETER(2)||','||PARAMETER(3)||
                      ','||PARAMETER(4)||','||PARAMETER(5)||','||PARAMETER(6);
```

```
      DO CASE VARTYPE;
         ; /* CASE 0: DUMMY */
         LINE = 'CALL PRIMA('||K||','||VARNAME||','
            ||SUBSCRIPT||','||LOC||','||SCALE||','||ANGLE||
            ','||TRANSFORMATION_ORDER||','||PARAMETERLINE||')';
         LINE = 'CALL PRIMS('||K||','||LOC||','||VAR(MP)||','||I||')';
         LINE = 'CALL IVALUE('||K||','||LOC||','||VAR(MP)||')';
         LINE = 'CALL RVALUE('||K||','||LOC||','||VAR(MP)||')';
         LINE = 'CALL TVALUE('||K||','||LOC||','||VAR(MP)||')';
      END;
      CALL FORTOUT(LINE);   CALL RESET;
   END EXOUT;

SYNTHESIZE:
   PROCEDURE(PRODUCTION_NUMBER);
      DECLARE PRODUCTION_NUMBER FIXED;
      DECLARE LABEL_STACK(100) CHARACTER, PREVLOC CHARACTER,
         LP FIXED INITIAL (0);
      DECLARE CARDNO_STACK(100) FIXED;
      DECLARE STMTNO FIXED INITIAL (1000);
      DECLARE (IM_VALUE,NUM_VALUE,VARTYPE,FTYPE) FIXED;

      DO CASE PRODUCTION_NUMBER;
         /*    CASE 0 IS A DUMMY    */
         ;
         /* 1: <PROGRAM> ::= <STATES>    */

         IF MP ¬= 2 THEN
         DO;
            CALL ERROR('EOF AT INVALID POINT',1);
            CALL STACK_DUMP;
         END;
         ELSE
         DO;
            COMPILING = FALSE;
            DO;
               OUTPUT(2) = 'C   ---   MAIN PROGRAM --- ('||VERSION||')';
               IF CONTROL(BYTE('D')) THEN
                                                  /*  370-VERSION  */
               DO;
                  OUTPUT(2) = '      INTEGER*2 MSG,IDUMMY';
                  OUTPUT(2) = '      COMMON MSG,IDUMMY(5147)';
               END;
               ELSE
               DO;                                /*  NOVA-VERSION  */
                  S = 'OVSYS';
                  DO I = 0 TO NO_OF_STATES;
                     S = S||',OVS'||STATES(I);
                  END;
                  OUTPUT(2) = '      EXTERNAL '||S;
                  OUTPUT(2) = '      COMMON MSG,IDUMMY(5147)';
                  OUTPUT(2) = '      CALL OVOPN(5,"MAIN.OL",IERR)';
               END;
```

```
      OUTPUT(2) = '      CALL SYSIN';
      S = '';
      DO I = 0 TO NO_OF_STATES;
         T = SUBSTR('     ',0,5-LENGTH(STATES(I)))||STATES(I);
         S = S||STATES(I)||',';
         IF CONTROL(BYTE('D')) THEN  /*  370-VERSION  */
            OUTPUT(2) = T||' CALL S'||STATES(I);
         ELSE
         DO;                                    /*  NOVA-VERSION  */
            OUTPUT(2) = T||' CALL OVLOD(5,OVS'||STATES(I)||',0,IERR)';
            OUTPUT(2) = '      CALL S'||STATES(I);
         END;
         OUTPUT(2) = '      GOTO 100';
      END;  /* OF LOOP I */
      OUTPUT(2) = '  100 GOTO ('||S||'101),MSG';
      OUTPUT(2) = '  101 STOP 98';
      OUTPUT(2) = '      END';
      OUTPUT = '*   ---   MAIN PROGRAM GENERATED   ---';
   END;
END;
/* 2: <STATES> ::= <STATE>    */
;
/* 3: <STATES> ::= <STATES> <STATE>    */
;
/* 4: <STATE> ::= <STATE HEAD> : <STATEMENTLIST> <STATE ENDING> ;    */
;
/* 5: <STATE> ::= <FUNCTION SUBPROGRAM HEAD> : <FUNCTION STATEMENTLIST> <FUNCTION SUBPROGRAM ENDING> ; */
;
/* 6: <STATE HEAD> ::= STATE <NUMBER>    */
DO;
   IF LENGTH(VAR(SP)) = 1 THEN S='0'||VAR(SP);
   ELSE S = VAR(SP);
   IF ¬CONTROL(BYTE('D')) THEN OUTPUT(2) = '      OVERLAY OVS'||S;
   OUTPUT(2) = '      SUBROUTINE S'||S;
   OUTPUT(2) = 'C     ----- STATE '||VAR(SP)||' -----';
   IF CONTROL(BYTE('D')) THEN OUTPUT(2) = '      INTEGER*2 IDUMMY';
   OUTPUT(2) = '      LOGICAL HITON';
   IF CONTROL(BYTE('D')) THEN
      OUTPUT(2) = '      COMMON IDUMMY(5147)'; /* 370-VERSION  */
   ELSE OUTPUT(2) = '      COMMON IDUMMY(5147)'; /* NOVA-VERSION */
   STATE_NUMBER = VAR(SP);
   NO_OF_STATES = NO_OF_STATES + 1;
   DO I = 0 TO NO_OF_STATES - 1;
      IF STATE_NUMBER = STATES(I) THEN
         CALL ERROR('DUPLICATE STATE DECLARATION',1);
   END; /* OF LOOP I */
   STATES(NO_OF_STATES) = S;
END;
/* 7: <STATEMENTLIST> ::= <STATEMENT>    */
;
/* 8: <STATEMENTLIST> ::= <STATEMENTLIST> <STATEMENT>    */
;
```

```
/* 9: <STATE ENDING> ::= END STATE <VARIABLE>    */
DO;
   VAR(SP) = REMOVE_DECIMAL(VAR(SP));
   OUTPUT(2) = '      END';
   IF LP ¬= 0 THEN
   DO I = 1 TO LP;
      CALL ERROR('UNMATCHED FOR-STATEMENT, LINE:'||CARDNO_STACK(LP),1);
   END;
   IF STATE_NUMBER ¬= VAR(SP) THEN CALL ERROR('STATE NUMBER '||
      STATE_NUMBER||' AND END-STATE NUMBER '||VAR(SP)||
      ' DO NOT MATCH',0);
   LP = 0;
   DOUBLE_SPACE;
   CARD_COUNT = 0;
END;
/*  10: <STATEMENT> ::= <GRAPHICAL ASSIGNMENT> ;    */
;
/* 11: <STATEMENT> ::= <ATTRIBUTE ASSIGNMENT> ;    */
;
/* 12: <STATEMENT> ::= <NUMERICAL ASSIGNMENT> ;    */
;
/* 13: <STATEMENT> ::= <DISPLAY STATEMENT> ;    */
;
/* 14: <STATEMENT> ::= <IDENTIFICATION CONSTRUCT> ;    */
;
/* 15: <STATEMENT> ::= <STATE-TRANSFER STATEMENT> ;    */
;
/* 16: <STATEMENT> ::= <CURSOR STATEMENT> ;    */
;
/* 17: <STATEMENT> ::= <ERASE STATEMENT> ;    */
;
/* 18: <STATEMENT> ::= <DRAW STATEMENT> ;    */
;
/* 19: <STATEMENT> ::= <FILE STATEMENT> ;    */
;
/* 20: <STATEMENT> ::= <READ STATEMENT> ;    */
;
/* 21: <STATEMENT> ::= ;    */
;
/* 22: <FUNCTION SUBPROGRAM HEAD> ::= <FUNCTION HEADING> ( <FUNCTION PARAMETERLIST> )        */
OUTPUT(2)=VAR(MP)||'('||VAR(MPP1+1)||')';
/* 23: <FUNCTION HEADING> ::= FUNCTION <IDENTIFIER>        */
VAR(MP)='      SUBROUTINE '||VAR(SP);
```

```
/* 24: <FUNCTION PARAMETERLIST> ::= <FUNCTION PARAMETERLIST> , <IDENTIFIER>    */
VAR(MP)=VAR(MP)||','||VAR(SP);
/* 25: <FUNCTION PARAMETERLIST> ::= <IDENTIFIER2>    */
;
/* 26: <FUNCTION SUBPROGRAM ENDING> ::= RETURN <IDENTIFIER>     */
DO;
  OUTPUT(2)='      RETURN';
  OUTPUT(2)='      END';
END;
/* 27: <FUNCTION STATEMENTLIST> ::= <FUNCTION STATEMENT>    */
;
/* 28: <FUNCTION STATEMENTLIST> ::= <FUNCTION STATEMENTLIST> <FUNCTION STATEMENT>    */
;
/* 29: <FUNCTION STATEMENT> ::= <LINE STATEMENT>          :    */
;
/* 30: <LINE STATEMENT> ::= START <XYVALUES>    */
OUTPUT(2)='      CALL POINT('||VAR(SP)||',0)';
/* 31: <LINE STATEMENT> ::= FORWORD <XYVALUES>          */
OUTPUT(2)='      CALL POINT('||VAR(SP)||',1)';
/* 32: <LINE STATEMENT> ::= STOP <XYVALUES>             */
OUTPUT(2)='      CALL POINT('||VAR(SP)||',11)';
/* 33: <XYVALUES> ::= <ARGUMENT EXPRESSION> , <ARGUMENT EXPRESSION>    */
VAR(MP)=VAR(MP)||','||VAR(SP);
/* 34: <GRAPHICAL ASSIGNMENT> ::=
                    <GRAPHICAL VARIABLE> <REPLACE> <EXPRESSION>  */
;
/* 35: <REPLACE> ::= : =    */
OUTPUT(2) = '      CALL ASSIG('||VARNAME||','||SUBSCRIPT||')';
/* 36: <ATTRIBUTE ASSIGNMENT> ::= <ATTRIBUTE FUNCTION> = <ARITHMETIC EXPRESSION> */
DO;
   IF FTYPE = 6 THEN
   DO;
      VAR(MP) = '6,'||VAR(MP);
      VAR(SP) = 'FLOAT('||VAR(SP)||')';
   END;
   LINE = 'CALL CHCHOR('||LINE||','||VAR(SP)||')';
   CALL FORTOUT(LINE);
END;
/* 37: <NUMERICAL ASSIGNMENT> ::= <IDENTIFIER> = <ATTRIBUTE FUNCTION> */
OUTPUT(2) = '      '||VAR(MP)||' = CORF1('||LINE||')';
/* 38: <NUMERICAL ASSIGNMENT> ::= <SUBSCRIPTED VARIABLE> = <ATTRIBUTE FUNCTION> */
OUTPUT(2) = '      '||VAR(MP)||' = CORF1('||LINE||')';
/* 39: <DISPLAY STATEMENT> ::= DISPLAY <DISPLAY>    */
DO;
 PARAMETERLINE=PARAMETER(1)||','||PARAMETER(2)||','||PARAMETER(3)||
               ','||PARAMETER(4)||','||PARAMETER(5)||','||PARAMETER(6);
   DO CASE IM_VALUE;
      LINE = 'CALL DISPL('||VARNAME||','||SUBSCRIPT||','||LOC||','
         ||SCALE||','||ANGLE||','||TRANSFORMATION_ORDER||','||PARAMETERLINE||')';
```

```
      DO;
         IF LENGTH(VAR(SP))>50 THEN VAR(SP)=SUBSTR(VAR(SP),0,50)||'''';
         LINE = 'CALL DISPS('||VAR(SP)||','||LOC||','
            ||LENGTH(VAR(SP))-2||')';
      END;
   END;
   CALL FORTOUT(LINE);  CALL RESET;
END;
/* 40: <DISPLAY STATEMENT> ::= DISPLAY <DISPLAY> ONTO <ARGUMENTS>    */
IF IM_VALUE = 0 THEN
DO;
   OUTPUT(2) = '       CALL ONTO('||VAR(SP)||')';
   LINE = 'CALL DISPL('||VARNAME||','||SUBSCRIPT||','||LOC||','
      ||SCALE||','||ANGLE||')';
   CALL FORTOUT(LINE);  CALL RESET;
END;
/* 41: <DISPLAY> ::= <DISPLAY VARIABLE>    */
;
/* 42: <DISPLAY> ::= <DISPLAY VARIABLE> <MODIFICATION>    */
;
/* 43: <DISPLAY> ::= <DISPLAY VARIABLE> WITHIN <ARGUMENTS>    */
DO;
   LINE = 'CALL WITHIN('||VAR(SP)||')'; CALL FORTOUT(LINE);
END;
/* 44: <IDENTIFICATION CONSTRUCT> ::=
               <IDENTIFICATION HEAD> : <STATEMENTLIST> <ENDING> */
DO;
   OUTPUT(2) = '0'||LABEL_STACK(LP)||' CONTINUE';
   LP = LP-1;
END;
/* 45: <IDENTIFICATION HEAD> ::= FOR HIT ON <IDENTIFIER>    */
DO;
   LP = LP+1;
   STMTNO = STMTNO+1;
   LABEL_STACK(LP) = STMTNO;
   CARDNO_STACK(LP) = CARD_COUNT;
   VAR(SP) = PAD_8(VAR(SP));
   OUTPUT(2) = '       IF(.NOT. HITON('''||VAR(SP)||''')) GOTO 0'||
      LABEL_STACK(LP);
END;
/* 46: <ENDING> ::= END    */
;
/* 47: <STATE-TRANSFER STATEMENT> ::= GOTO STATE <VARIABLE>    */
DO;
   T = REMOVE_DECIMAL(VAR(SP));
   OUTPUT(2) = '       CALL CHSTAT('||T||')';
   OUTPUT(2) = '       RETURN';
END;
```

```
/* 48: <STATE-TRANSFER STATEMENT> ::= GO TO STATE <VARIABLE>    */
DO;
   T = REMOVE_DECIMAL(VAR(SP));
   OUTPUT(2) = '      CALL CHSTAT('||T||')';
   OUTPUT(2) = '      RETURN';
END;
/* 49: <CURSOR STATEMENT> ::= CURSOR ON    */
OUTPUT(2) = '      CALL CRSRON';
/* 50: <ERASE STATEMENT> ::= ERASE SCREEN    */
OUTPUT(2) = '      CALL ERASE';
/* 51: <DRAW STATEMENT> ::= DRAW <MODIFICATION>    */
DO;
   OUTPUT(2) = '      CALL DRAW('||LOC||','||SCALE||')';
   CALL RESET;
END;
/* 52: <FILE STATEMENT> ::= STORE <ARGUMENTS>    */
DO;
   CALL ARG_CHECK(2);
   OUTPUT(2) = '      CALL STORE('||VAR(SP)||')';
END;
/* 53: <FILE STATEMENT> ::= RESTORE <ARGUMENTS>    */
DO;
   CALL ARG_CHECK(2);
  OUTPUT(2) = '      CALL RESTOR('||VAR(SP)||')';
END;
/* 54: <READ STATEMENT> ::= READ CURSOR <IDENTIFIER4> <,3> <IDENTIFIER> */
OUTPUT(2) = '      CALL RDCRSR(1,'||VAR(MP+2)||','||VAR(SP)||',KDUM)';
/* 55: <READ STATEMENT> ::= READ KEY <IDENTIFIER>    */
OUTPUT(2) = '      CALL RDCRSR(2,XDUM,YDUM,'||VAR(SP)||')';
/* 56: <READ STATEMENT> ::= READ SUBSCRIPT <IDENTIFIER>    */
OUTPUT(2) = '      CALL RDCRSR(3,XDUM,YDUM,'||VAR(SP)||')';
/* 57: <EXPRESSION> ::= <PRIMARY>    */
DO;
   CALL EXOUT(1,VARTYPE,PRCODE);
END;
/* 58: <EXPRESSION> ::= <EXPRESSION> + <PRIMARY>    */
DO;
   VAR(MP) = VAR(SP);
   CALL EXOUT(2,VARTYPE,PRCODE);
END;
/* 59: <EXPRESSION> ::= <EXPRESSION> - <PRIMARY>    */
DO;
   VAR(MP) = VAR(SP);
   CALL EXOUT(3,VARTYPE,PRCODE);
END;
/* 60: <PRIMARY> ::= <DISPLAY VARIABLE>    */
;
/* 61: <PRIMARY> ::= <DISPLAY VARIABLE> <MODIFICATION>    */
;
/* 62: <DISPLAY VARIABLE> ::= <GRAPHICAL VARIABLE>    */
IM_VALUE = 0;
```

```
/* 63: <DISPLAY VARIABLE> ::= <GRAPHICAL PRIMITIVE>    */
DO;
   VARTYPE = 1; VAR(MP) = PAD_8(VAR(MP)); VARNAME ='''||VAR(MP)||''';
   IM_VALUE = 0;  PARAMETER(1)='0';  SUBSCRIPT = '0';
END;
/* 64: <DISPLAY VARIABLE> ::=  <LITERAL>    */
IM_VALUE = 1;
/* 65: <GRAPHICAL VARIABLE> ::= <IDENTIFIER>    */
DO;
   VARTYPE = 1; VAR(MP) = PAD_8(VAR(MP)); VARNAME ='''||VAR(MP)||''';
   SUBSCRIPT = '0';   PARAMETER(1)='0';
END;
/* 66: <GRAPHICAL VARIABLE> ::= <SUBSCRIPTED VARIABLE>    */
VARTYPE=1;
/* 67: <SUBSCRIPTED VARIABLE> ::= <IDENTIFIER3> <(1> <INDEX OR PARAMETERS> )    */
DO;
 TEMPCH=PAD_8(VAR(MP));
 VARNAME='''||TEMPCH||''';
 VAR(MP)=VAR(MP)||'('||VAR(MPP1+1)||')';
 SUBSCRIPT='('||VAR(MPP1+1)||')';
END;
/* 68: <INDEX OR PARAMETERS> ::= <ARITHMETIC EXPRESSION1> <,2> <INDEX OR PARAMETERS>    */
DO;
PARAMETERCOUNT=PARAMETERCOUNT+1;
PARAMETER(PARAMETERCOUNT)=VAR(MP);
END;
/* 69: <INDEX OR PARAMETERS> ::= <ARITHMETIC EXPRESSION>    */
DO;
PARAMETERCOUNT=PARAMETERCOUNT+1;
PARAMETER(PARAMETERCOUNT)=VAR(MP);
END;
/* 70: <LITERAL> ::= <STRING>    */

DO;
   VARTYPE = 2;
   VAR(MP) = '''||VAR(MP)||''';
END;
/* 71: <LITERAL> ::= <CONVERSION FUNCTIONHEAD> <ARGUMENTS> )    */
DO;
   DO CASE VARTYPE;
      ;
      ;
      ;
      CALL ARG_CHECK(1);
      CALL ARG_CHECK(2);
      CALL ARG_CHECK(2);
   END;
   VAR(MP) = VAR(MPP1);
END;
```

```
/* 72: <CONVERSION FUNCTIONHEAD> ::= <CONVERSION FUNCTIONNAME> (    */
;
/* 73: <CONVERSION FUNCTIONNAME> ::= IVALUE    */
VARTYPE = 3;
/* 74: <CONVERSION FUNCTIONNAME> ::= RVALUE    */
VARTYPE = 4;
/* 75: <CONVERSION FUNCTIONNAME> ::= TVALUE    */
VARTYPE = 5;
/* 76: <VARIABLE> ::= <IDENTIFIER>    */
;
/* 77: <VARIABLE> ::= <NUMBER>    */
      /* CONCATENATE A DECIMAL POINT TO NUMBER */
DO;
   K = 0;
   DO I = 0 TO LENGTH(VAR(SP))-1;
      IF BYTE(VAR(SP),I) = BYTE('.') THEN K = 1;
   END;
   IF K = 0 THEN VAR(SP) = VAR(SP)||'.';
END;
/* 78: <ATTRIBUTE FUNCTION> ::= <FUNCTIONHEAD> <DISPLAY VARIABLE> ) */
LINE = FTYPE||','||VARNAME||','||SUBSCRIPT;
/* 79: <FUNCTIONHEAD> ::= <FUNCTIONNAME> (    */
;
/* 80: <FUNCTIONNAME> ::= XLOC    */
FTYPE = 1;
/* 81: <FUNCTIONNAME> ::= YLOC    */
FTYPE = 2;
/* 82: <FUNCTIONNAME> ::= XSCALE    */
FTYPE = 3;
/* 83: <FUNCTIONNAME> ::= YSCALE    */
FTYPE = 4;
/* 84: <FUNCTIONNAME> ::= ANGLE    */
FTYPE = 5;
/* 85: <FUNCTIONNAME> ::= SUBSCRIPT    */
FTYPE = 6;
/* 86: <MODIFICATION> ::= <MODIFIER>    */
;
/* 87: <MODIFICATION> ::= <MODIFICATION> <MODIFIER>    */
;
/* 88: <MODIFIER> ::= AT <ARGUMENTS>    */
DO;
   CALL ARG_CHECK(2); LOC = VAR(SP);
END;
/* 89: <MODIFIER> ::= SCALE <ARGUMENTS>    */
```

```
DO;
   IF ARG_COUNT = 1 THEN SCALE = VAR(SP)||','||VAR(SP);
   ELSE DO;
      CALL ARG_CHECK(2); SCALE = VAR(SP);
   END;
   ARG_COUNT = 0;
 IF FIRST_ENCOUNTER THEN TRANSFORMATION_ORDER = '1';
 FIRST_ENCOUNTER = 0;
END;
/* 90: <MODIFIER> ::= ANGLE <ARGUMENTS>    */
 DO;
 IF FIRST_ENCOUNTER THEN TRANSFORMATION_ORDER = '2';
 FIRST_ENCOUNTER= 0;
ANGLE = VAR(SP);
 END;
/* 91: <MODIFIER> ::= ANGLE <ARGUMENTS> RAD    */
 DO;
 IF FIRST_ENCOUNTER THEN TRANSFORMATION_ORDER = '2';
 FIRST_ENCOUNTER = 0;
ANGLE = VAR(MPP1);
 END;
/* 92: <MODIFIER> ::= ANGLE <ARGUMENTS> DEG;       THE NUMBER
   1.745329E-2 IS THE CONVERSION FACTOR FROM DEGREES TO RADIANS */
 DO;
 IF FIRST_ENCOUNTER THEN TRANSFORMATION_ORDER = '2';
 FIRST_ENCOUNTER = 0;
ANGLE = VAR(MPP1)||'*1.745329E-2';
 END;
/* 93: <MODIFIER> ::= FROM <ARGUMENTS>    */
DO;
   CALL ARG_CHECK(2);
   PREVLOC = VAR(SP);
END;
/* 94: <MODIFIER> ::= TO <ARGUMENTS>    */
DO;
   CALL ARG_CHECK(2); LOC = PREVLOC;
   SCALE = VAR(SP); PREVLOC = SCALE;
   ANGLE = '-999.';
END;
/* 95: <MODIFIER> ::= VSYM <ARGUMENTS>    */
DO;
   UNARY_OP = '1,'||VAR(SP); PRCODE = 1;
END;
/* 96: <MODIFIER> ::= HSYM <ARGUMENTS>    */
DO;
   UNARY_OP = '2,'||VAR(SP); PRCODE = 1;
END;
/* 97: <ARGUMENTS> ::= <ARGUMENT EXPRESSION>    */
ARG_COUNT = 1;
```

```
/* 98: <ARGUMENTS> ::= <ARGUMENTS> <,1> <ARGUMENT EXPRESSION>    */
DO;
   ARG_COUNT = ARG_COUNT+1; VAR(MP) = VAR(MP)||','||VAR(SP);
END;
/* 99: <GRAPHICAL PRIMITIVE> ::= BLANK    */
;
/* 100: <GRAPHICAL PRIMITIVE> ::= LINE    */
;
/* 101: <GRAPHICAL PRIMITIVE> ::= SQUARE    */
;
/* 102: <GRAPHICAL PRIMITIVE> ::= TRIANGLE    */
;
/* 103: <GRAPHICAL PRIMITIVE> ::= CIRCLE    */
;
/* 104: <GRAPHICAL PRIMITIVE> ::= SCIRCLE    */
;
/* 105: <ARGUMENT EXPRESSION> ::= <(2> <ARITHMETIC EXPRESSION> )    */
VAR(MP)='('||VAR(MPP1)||')';
/* 106: <ARGUMENT EXPRESSION> ::= <VARIABLE>    */
;
/* 107: <ARITHMETIC EXPRESSION> ::= <ARITHMETIC EXPRESSION> <+1> <TERM>    */
VAR(MP)=VAR(MP)||'+'||VAR(SP);
/* 108: <ARITHMETIC EXPRESSION> ::= <ARITHMETIC EXPRESSION> <-1> <TERM>    */
VAR(MP)=VAR(MP)||'-'||VAR(SP);
/* 109: <ARITHMETIC EXPRESSION> ::= <+2> <TERM>    */
VAR(MP)='+'||VAR(SP);
/* 110: <ARITHMETIC EXPRESSION> ::= <-2> <TERM>    */
VAR(MP)='-'||VAR(SP);
/* 111: <ARITHMETIC EXPRESSION> ::= <TERM>    */
;
/* 112: <TERM> ::= <TERM> * <ARITHMETIC PRIMARY>    */
VAR(MP)=VAR(MP)||'*'||VAR(SP);
/* 113: <TERM> ::= <TERM> / <ARITHMETIC PRIMARY>    */
VAR(MP)=VAR(MP)||'/'||VAR(SP);
/* 114: <TERM> ::= <TERM> <*1> <*2>  <ARITHMETIC PRIMARY>    */
VAR(MP)=VAR(MP)||'**'||VAR(SP);
/* 115: <TERM> ::= <ARITHMETIC PRIMARY>    */
;
/* 116: <ARITHMETIC PRIMARY> ::= <(3> <ARITHMETIC EXPRESSION> )    */
VAR(MP)='('||VAR(MPP1)||')';
/*  117: <ARITHMETIC PRIMARY> ::= <SUBSCRIPT HEAD> <ARITHMETIC EXPRESSION> )
                */
VAR(MP)=VAR(MP)||VAR(MPP1)||')';
/* 118: <ARITHMETIC PRIMARY> ::= <VARIABLE>    */
VAR(MP)=REMOVE_DECIMAL(VAR(MP));
/* 119: <SUBSCRIPT HEAD> ::= <IDENTIFIER1>  (    */
VAR(MP)=VAR(MP)||'(';
/* 120: <SUBSCRIPT HEAD> ::= <SUBSCRIPT HEAD> <ARITHMETIC EXPRESSION> ,    */
VAR(MP)=VAR(MP)||VAR(MPP1)||',';
/* 121: <(1> ::= (    */
;
/* 122: <(2> ::= (    */
;
```

```
      /* 123: <(3> ::= (    */
      ;
      /* 124: <IDENTIFIER1> ::= <IDENTIFIER>    */
      ;
      /* 125: <IDENTIFIER2> ::= <IDENTIFIER>    */
      ;
      /* 126: <IDENTIFIER3> ::= <IDENTIFIER>    */
      ;
      /* 127: <IDENTIFIER4> ::= <IDENTIFIER>    */
      ;
      /* 128: <+1> ::= +    */
      ;
      /* 129: <+2> ::= +    */
      ;
      /* 130: <-1> ::= -    */
      ;
      /* 131: <-2> ::= -    */
      ;
      /* 132: <*1> ::= *    */
      ;
      /* 133: <*2> ::= *    */
      ;
      /* 134: <,1> ::= ,    */
      ;
      /* 135: <,2> ::= ,    */
      ;
      /* 136: <,3> ::= ,    */
      ;
      /* 137: <ARITHMETIC EXPRESSION1> ::= <ARITHMETIC EXPRESSION>   */
      ;
   END;
END SYNTHESIZE;
```

APPENDIX C

LISTING OF TRAMA

```
      SUBROUTINE TRAMA(TVAL,ITRANS)
C --- TO BUILD THE RRANSFORMATION MATRICES
C --- ITRANS=1, TO PERFORM SCALING, ROTATIONAND
C                AND TRANSLATION
C --- ITRANS=2, TO PERFORM ROTATION, SCALING
C                AND TRANSLATION
      REAL TVAL(6),SX(2),SY(2)
C --- INITIALIZALLY, TVAL(1)=XLOC
C                    VAL(2)=YLOC
C                    TVAL(3)=XSCALE
C                    TVAL(4)=YSCALE
C                    TVAL(5)=ANGLE IN RADINAS
C --- TO TEST IF XSCALE OR YSCALE IS TOO SMALL
      IF(TVAL(3).LT0.001.OR.TVAL(4).LT.0.001)STOP 15
      SINA=SIN(TVAL(5))
      COSA=COS(TVAL(5))
      XLOC=TVAL(1)
      YLOC=TVAL(2)
      SX(1)=TVAL(3)
      SX(2)=TVAL(4)
      SY(1)=TVAL(4)
      SY(2)=TVAL(3)
      TVAL(1)=COSA*SX(1)
      TVAL(2)=SINA*SX(ITRANS)
      TVAL(3)=-SINA*SY(ITRANS)
      TVAL(4)=COSA*SY(1)
      TVAL(5)=XLOC-0.5*(TVAL(1)+TVAL(3))
      TVAL(6)=YLOC-0.5*(TVAL(2)+TVAL(4))
      RETURN
      END
```