IMPLEMENTING THE CSPC CORE LANGUAGE RECOMMENDATIONS IN AN EXISTING CAL LANGUAGE, NATAL

R.A. Orchard

Electrical Engineering Division
National Research Council of Canada

ABSTRACT

NATAL is a high level programming language designed for the efficient production of computer-assisted learning (CAL) materials. As originally designed, it incorporated extensive facilities for the display of alphanumeric and monochrome graphic information and for response processing. It was decided to extend the graphics capabilities of NATAL to support recent developments in low-cost colour graphics terminals. One major objective of this exercise was to ensure, insofar as possible, consistency with commonly accepted computer graphics practices, as recommended in the GSPC Status Report on the 'core' language.

Since there existed a number of application programs using the original alphanumeric/graphics facilities of NATAL, it was essential that the changes to the existing structure be minimal. A further consideration involved the purpose of NATAL, which is to make course writing as simple as possible for authors (while at the same time providing support for a wide variety of terminals). To this end it was felt necessary to keep the graphics support from becoming too complex. The present report describes the extensions to the NATAL graphics support based on the 'core' language recommendations and the implications of the preceding considerations.

RÉSUMÉ

NATAL est un langage de programmation conçu pour réaliser facilement des moyens d'enseignement assisté par ordinateur (EAO). Dans sa forme initiale, il contient un grand nombre de fonctions pour l'affichage monochrome alphanumérique et infographique, de même que pour le traitement des réponses. Il a été décidé d'étendre les fonctions infographiques de NATAL afin de l'adapter aux récents perfectionnements de terminaux infographiques couleurs de faible prix. Un des principaux objectifs de cette entreprise était de se conformer, autant que possible, aux pratiques infographiques acceptées généralement, telles que recommandées par le Rapport de situation du GSPC sur le langage "central".

En raison de programmes rédigés en tenant compte des fonctions alphanumériques/infographiques initiales de NATAL, il était essentiel de changer le moins possible la structure existante. Une autre considération concernait la raison d'être du NATAL, c'est-à-dire rendre la rédaction des cours la plus simple possible pour les auteurs, tout en permettant leur utilisation par une large gamme de terminaux. Pour ce faire, on a jugé bon de ne pas trop augmenter la complexité des fonctions infographiques. On trouve dans le présent rapport une description des développements de la fonction infographique NATAL réalisés en se conformant aux recommandations sur la langage "central", de même qu'une description de la mise en oeuvre des considérations précédentes.

1. NATAL

NATAL (acronym for NATional Author Language) is a high-level programming language which was developed specifically to meet the requirements of course authors for preparing Computer Aided Learning (CAL) materials. One of the primary objectives in the development of NATAL was to produce a standard CAL language that could be implemented on a variety of computer equipment [1]. This allows CAL materials to be easily transferred from one centre to another, eliminating much duplication of effort and cost.

A most important requirement of any CAL language is that it provides course authors with a rich environment in which to present information to students and to accept and process their responses. This includes capabilities for the display of textual information; the display of graphical information (both the vector graphic and font graphic variety); the control of special display devices such as random access slide projectors or videodiscs; the control of random access audio devices; the acceptance of input from a keyboard; and the acceptance of input from other input devices such as touch sensitive panels or digitizing tablets. While providing these extensive capabilities, the language should remain easy to learn and use. Authors should not be burdened with the need to understand any details concerning the operation of these specific devices. The language should contain a simple interface between the user and the control of input and output. NATAL provides such an environment for course authors and facilitates the efficient production of CAL materials.

The initial implementation of NATAL provided the means for creating displays on graphics terminals. It became apparent, however, that these capabilities were incomplete, especially with regard to support for recent developments in low-cost colour graphics displays. There were several factors to be considered. In view of the progress made by the SIGGRAPH Graphics Standards Planning Committee (GSPC) towards the definition of standards for graphics packages, it was decided to review the recommendations in the GSPC report as a guideline for revising the NATAL graphics support. At the same time, however, it was important to remember that NATAL was an existing language with an existing graphics component included. Applications had been developed and had made use of this support. Therefore, the effects that any changes to NATAL would have on these programs had to be considered and were to be kept minimal. Also, NATAL was designed to provide facilities for course authors that are simple to use so that the graphics support was not to become too complex. NATAL provides a tool for the development of CAL materials and was never intended to provide a graphics development facility. The differences between NATAL's objectives and those of the Core System are quite fundamental and this led to some compromise on the complete implementation of the GSPC recommendations.

The present report describes the current NATAL graphics facilities and examines the implications of the preceding considerations that led to this support.

2. CORE SYSTEM

The main aim of the Core System is the promotion of program portability [2]. In very ideal situations, programs that are produced at one installation could be taken to other installations that support the Core System standard and with no changes to the programs they could be executed, producing identical results. In practice this is extremely difficult to achieve. The implementations of the Core System would have to be done using a common language, programmers would have to restrict themselves to using standard subsets of this language, and the implementations would need to use identical techniques and naming conventions. Varying degrees of program transportability may actually be realized short of this ideal. Many of the changes required to move a program from one system to another might be accomplished through an automatic translation procedure (simple editing such as changing the names of the graphics routines or reordering parameter lists could easily be done). Other programs may require changes to their structure before transporting can be achieved. This may be due to different hardware capabilities at the installations that require different programming techniques to produce similar results. Some programs could even require such extensive modification that a complete rewriting of the application would prove more cost effective. Foley suggests that a program can be called portable if the cost of moving the program is less than the cost of rewriting the program [3] and refers to this as realistic or "pragmatic" portability. The Graphics Standards Plan-

ning Committee, in order to assist with these problems of portability, have identified different implementation levels for the Core System for both input and output. By specifying the level of support provided and including all of the routines recommended by the GSPC report, portability will be more easily attained.

There are, however, a couple of underlying requirements of the Core System specification that must be remembered if portability is to be achieved. The intended target of the support packages for the Core System is a high-level language like FORTRAN or PL/1. It is expected that subroutine packages would be developed and added as libraries of these languages. The implication of this point is that providing the functional support of the Core System as specified in the GSPC report is not sufficient to address the objective of program portability. The vehicle used to provide this support plays an important role. Also it should be noted that the Core System is primarily directed towards the support of medium-performance interactive vector graphics terminal equipment.

An objective related to program portability is that of programmer portability; the ability of a programmer to move to various installations and need only limited retraining to perform his intended duties. Given the difficulties in achieving complete program portability, perhaps the widespread acceptance of the standard graphics techniques and terminology will be just as significant in the success of the Core System. There will be a common ground that will encourage the exchange of information and allow programmers to understand each other more fully.

3. COMPROMISING

3.1 The UNIT

NATAL, as originally designed, included many features necessary for the effective control of textual display, a basic capability for display of vector graphics, features for displaying font graphics, as well as the ability to easily control a host of other output and input devices. These features are essential in any advanced CAL system. The control of the various types of input and output has been isolated in a separate component of the NATAL language called the UNIT (see Figure 1). Within this module, all trans-

THE NATAL UNIT

```
GEOGRAPHY:  ENTRY PROCEDURE;                          - beginning of the course
            CALL *GREAT_LAKES;                        - execute the UNIT named *GREAT_LAKES
END; /* GEOGRAPHY */

*GREAT_LAKES:  UNIT;                                  - beginning of the UNIT
        DISPLAY &E &L1 The great Lakes are            - erase the page (&E), move down 1
                Lake Erie, Lake Ontario, Lake           line (&L1), display some text, move
                Huron, Lake Michigan, and Lake          down 2 lines (&L2), and ask
                Superior. &L2 Which of these            the question.
                lakes does the city of Toronto
                border on? &;
        RESPONSE POSN=(8,1);                          - ask for the answer to the question
        EDIT SHUP;                                    - shift all lower case characters in the
                                                        response to upper case
        RIGHT CC('&ONTARIO&') *                       - set RIGHT to true if answer includes
              -CC('&(ERIE,HURON,SUPERIOR,MICHIGAN)&');  Ontario but not one of the others
        REINF RIGHT &L That is correct&;              - if correct, tell him and exit.
        RETRY UNREC &L No, try again &E(8,1);         - 1st time not correct (unrecognized)
                                                        let him try again
        REINF UNREC &E(10,1) The correct answer is    - next time not correct display
                             Lake Ontario.&;            correct answer and exit
END; /* GREAT_LAKES */
```

Figure 1

actions between the program and the student take place. This allows course authors to structure their materials into manageable pieces that perform identifiable tasks -- present some information, ask a question, receive a response, analyze and categorize the response. This makes NATAL quite different in structure from other high-level languages such as FORTRAN. For this reason it was immediately apparent that support of the Core System in NATAL could not promote the development of programs that would be easily transported to implementations in other languages. The Core System was not meant to be implemented in a language like NATAL and still provide easy portability to languages such as FORTRAN. However, by providing the functional capabilities of the Core System, commonly accepted graphics practices are followed and the benefits of following such practices are realized.

3.2 Text Display

In CAL applications, the display of textual information is of major importance. On the other hand, in strictly graphics applications, such as would be expected with Core System implementations, textual display plays the somewhat minor role of annotating pictures. NATAL required that text presentation be handled in a manner that was natural and simple for authors, so that they could easily express what they wanted to do. Also, it was necessary to allow the control of textual display to be done in a way that would relieve an author from having to know the exact terminal model that would be used for display (e.g. an author should not have to write different versions of the same course to account for terminals with 16 rows and 72 columns and those with 24 rows and 80 columns if the nature of the material being displayed doesn't make specific requirements of the size of the display). This is achieved in NATAL through the DISPLAY statement of the UNIT. Included as part of the DISPLAY statement is a display sublanguage that allows for flexible control and formatting of text. Authors can allow NATAL to process text automatically, filling rows on the terminal with as many words as will fit. Or they can control text formatting through the use of many simple commands that are interspersed in the text:

centre text in middle of line (&M command);

request that text be right justified on each line (&J command);

erase the entire display area or portions of it (&E command);

control the colour of text (&HC command);

etc.

Figure 2 is an example using some of these commands. Text locations are conveniently referred to by the row and column position on the display. This is a natural way for authors to think about positioning text when formatting output for screens.

DISPLAYING TEXT

```
DISPLAY

  &E &M Rain #1 &M &L2 The rain in
  Spain falls mainly on the plain. This
  text will be automatically formatted
  by NATAL with lines being
  broken on word boundaries.&L2 &M Rain
  #2 &M &L2 &J The rain in Spain falls
  mainly on the plain. This text will be
  formatted also but justification is
  turned on so that the right margin is
  aligned.&;
```

The above DISPLAY statement produces the screen image shown below for terminals with 40 characters per row.

```
                Rain #1

The rain in Spain falls mainly on the
plain. This text will be automatically
formatted by NATAL with lines being
broken on word boundaries.

                Rain #2

The rain in Spain falls mainly on the
plain. This text will be formatted also
but justification is turned on so that
the right margin is aligned.
```

Figure 2

A NATAL author may also define a rectangular viewport or 'box' on the display into which text is to be written. The NATAL runtime system will automatically format the text to fit the defined box. This allows the screen to be subdivided for use as the author sees fit (see Figure 3). For example, one

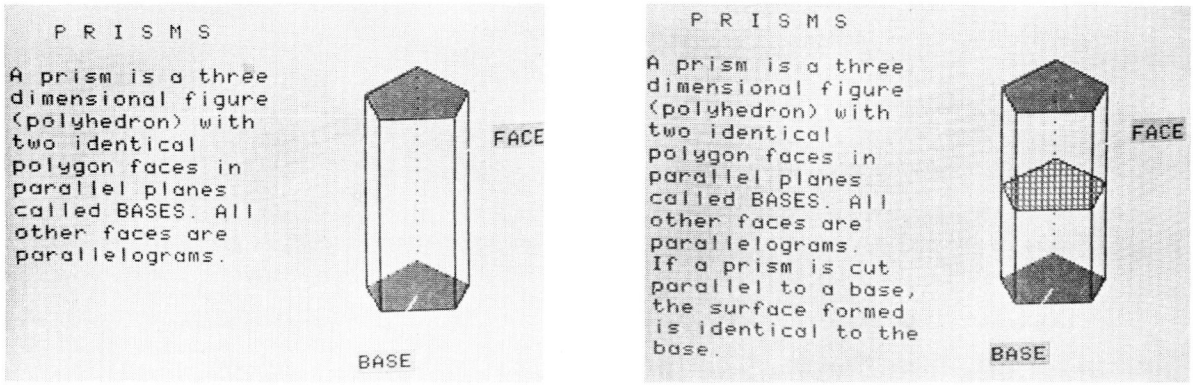


Figure 3

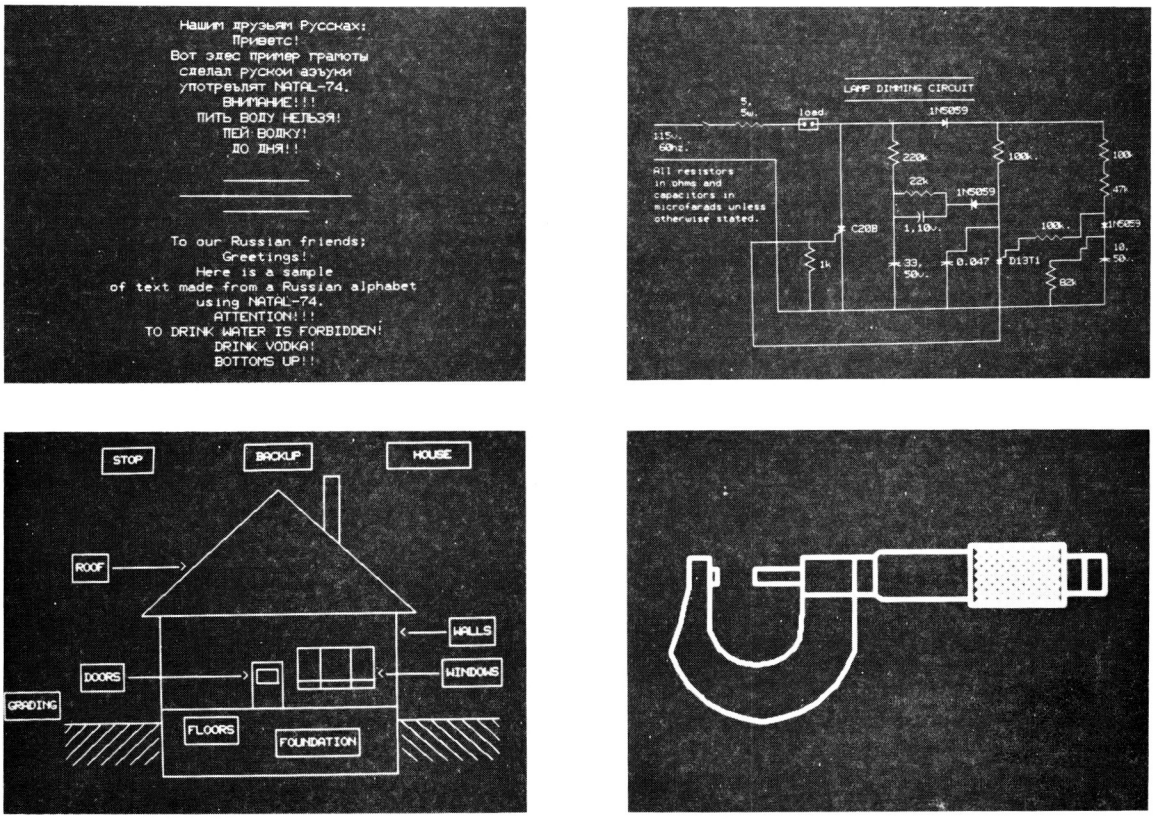


Figure 4

part of the screen might contain some static information while other parts of the display could contain dynamically changing information that relates to this static part. This capability proves very useful in practice.

The display sublanguage also provides the means for selecting other than a standard character set for display. These alternate character sets could be predefined as an integral feature of the terminal allowing an ability to switch between an English and a French set for example. Some terminals allow user created character sets to be defined and loaded. This allows authors to prepare character sets to meet the needs of their courses. A special character set that displays mathematical and electrical symbols could be developed to assist in teaching electronics. Language training programs might require the display of the Cyrillic alphabet to assist in teaching Russian. This ability to create special symbols is called 'font' graphics. Characters can be used individually or combined as a mosaic in which the pieces fit together to form a picture. Figure 4 illustrates some examples of such font graphics.

The preceding few paragraphs presented a brief description of some of the text formatting and control features of NATAL. It was intended to show that a variety of facilities are required in CAL systems if they are to be effective. A strictly graphics system has no such requirements. Text is generally controlled by specifying starting positions in a graphics coordinate space. This would be less natural for authors and NATAL has addressed the problem to meet CAL requirements. (Note however, that text may be positioned with respect to the graphics space when required.)

3.3 Graphics Output

Graphics functions are executed by using one of the commands of the display sublanguage.

```
&G( f1, f2, ... , fn )
```

The fi are a list of graphic functions that are executed in the order that they appear within the brackets. This is the mechanism through which NATAL implements many of the Core System recommendations. Current support is two-dimensional, basic output, and synchronous input -- within the limitations imposed by the nature of the NATAL language. Appendix 1 shows the list of functions embedded in the language as system graphics functions. Figure 5 presents a simple example using some of these functions.

Since NATAL has been implemented to interpret an intermediate level code produced by the compiler and these functions are included as part of the run-time system, it was necessary to keep the number of graphic functions to a minimum. Some functions that are part of the Core System recommendation were not included for various reasons: the function was already handled in a quite different and usually more natural way (such as the control of text or selection of character fonts); the function may have added a level of complexity to NATAL; the function could be added later as a library function written in NATAL (those functions less likely to be used); or the function would serve no useful purpose by being included (e.g. CREATE_TEMP_SEG, CLOSE_TEMP_SEG).

NATAL also provides a mechanism which allows authors to define their own graphic functions. This is the user defined GRAPHIC FUNCTION. Within these functions other user defined graphic functions or system graphic functions can be called to produce the display required by the author. Figure 6 shows a definition of a rectangle. This routine could be added to a library of graphic functions and made available to all NATAL authors.

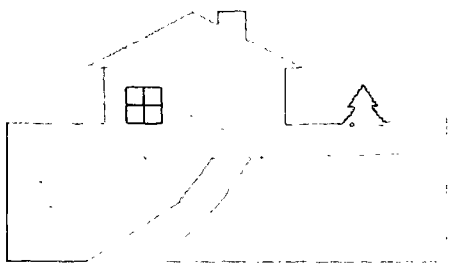
3.4 Input

Response handling had already been included in NATAL in a way quite incompatible to the techniques suggested for synchronous input by the GSPC report. The RESPONSE statement of the UNIT provides for most of the functional capability required, as well as some features not addressed by the Core System but necessary in a CAL environment. Input from keyboards, pointer devices (touch tablets, joysticks, digitizer tablets), and other devices is accommodated. Also, various controls on the inputs can be maintained: the number of characters or graphic points accepted during a response can be limited; echo or no echo of inputs is controllable; the elapsed time allowed for an input can be specified; the position on the display where keyboard input will be echoed can be indicated; and graphic input can be returned as a row,column position or in an x,y graphic

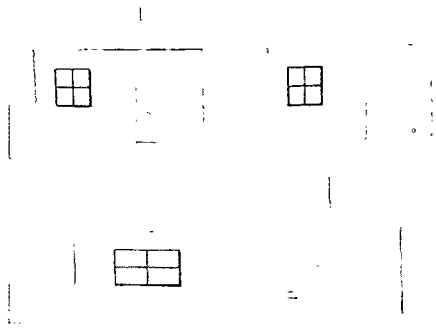
GRAPHICS

```
*HOUSE: UNIT;
DISPLAY
 &G( SET_WINDOW(0,0,1.2,1.0), SET_FILL(2), SET_COLOUR(240),
     MOVE(0,0), RPOLY((0,1.2,0),(.45,0,-.45)),
     MOVE(0,.45), SET_COLOUR(300), RPOLY((0,1.2,0),(.55,0,-.55)),
     SET_COLOUR(180), CIRCLE(1.0,.85,.1),
     TREE(.92,.35), TREE(-.05,.2),
     SET_FILL(2), SET_COLOUR(0), MOVE(.25,.6), RPOLY((.3,.3),(.15,-.15)),
     SET_COLOUR(120), MOVE(.3,.35), RPOLY((0,.5,0),(.25,0,-.25)),
     MOVE(.62,.68), RPOLY((0,.07,0),(.07,0,-.09)),
     SET_COLOUR(0,100), MOVE(.36,.45), RPOLY((0,.1,0),(.1,0,-.1)),SET_COLOUR(0,0),
     MOVE(.36,.45),RLINE((0,.1,0,-.1,0,.1,0,-.05,0),(.1,0,-.1,0,.05,0,-.05,0,.1)),
     MOVE(.6,.35),SET_COLOUR(0),RPOLY((0,.1,0),(.15,0,-.15)),
     SET_FILL(0),SET_COLOUR(180),CIRCLE(.63,.42,.01),SET_FILL(2),
     MOVE(.2,.02), RPOLY((.3,.1,.1,-.09,-.2),(.2,.13,0,-.13,-.2)),MOVE(.5,.5)
   )&;
END;
```

The above UNIT can be used to produce the displays shown below. (Note that the function TREE is a call to a user defined GRAPHIC FUNCTION.)



(a)
full screen



(b)
varying windows and viewports

Figure 5

space format. Consider the following example of a response statement:

```
RESPONSE POSN=(5,1),TIME=10,NCHAR=5;
```

This statement prompts for a response from the keyboard (default input device) and causes the characters typed on the keyboard to be echoed starting at column 1 of row 5. It would allow up to 10 seconds for the response to be completed before generating an out of time condition and it would allow a maximum of 5 characters to be entered as input. One can readily see the advantages of such control on responses for CAL applications.

4. CONCLUSIONS

The current NATAL system is successful in providing authors with a convenient tool for creating and controlling the displays that they require. By using the Core System recommendations as a guide, a graphics facility has emerged that allows programmers famili-

ar with graphics concepts to quickly adapt themselves to NATAL. Recent experience with the TELIDON terminal has shown that authors can easily take advantage of all of the features of this new generation of low-cost colour graphics terminals.

USER DEFINED GRAPHIC FUNCTIONS

```
/* This graphic function is used to display
   a rectangle that is centered about the
   current cursor position with width w
   and height h, and is rotated about the
   centre through deg degrees.
*/
RECTANGLE:  GRAPHIC FUNCTION ( W,H,DEG );
/* Find vertices of the rectangle relative
   to the centre.
*/
XV <- ( -W/2, -W/2, +W/2, +W/2 );
YV <- ( -H/2, +H/2, +H/2, -H/2 );
/* Perform the rotation */

DO I = 1 TO 4;

   X <- XV.(I);  Y <- YV.(I);
   XV.(I) <- X*COS(DEG) - Y*SIN(DEG);
   YV.(I) <- X*SIN(DEG) + Y*COS(DEG);

END;
/* Find positions of vertices relative to
   each other.
*/
DO I = 4 TO 2 BY -1;

   XV.(I) <- XV.(I) - XV.(I-1);
   YV.(I) <- YV.(I) - YV.(I-1);

END;
/* Now display the rotated rectangle */
PLOT  RMOVE( XV.(1), YV.(1) ),
      RPOLY( SUBVEC(XV,2,3),SUBVEC(YV,2,3) );
END; /* RECTANGLE */
```

Figure 6

REFERENCES

1. NATAL-74 - Concept to Reality. J.W. Brahan, W.H. Henneker, A.M. Hlady. Proceedings of the Third Canadian Symposium on Instructional Technology. February, 1980.

2. Status Report of the Graphics Standards Planning Committee. Computer Graphics. August, 1979. Vol 13 #3, pp 1-270.

3. A Standard Computer Graphics Subroutine Package. J.D. Foley. Computers and Structures. April, 1979. Vol 10 #1-2, pp 141-147.

BIBLIOGRAPHY

1. Computer-Aided Learning - A Cooperative Research Project of the National Research Council of Canada. J.W. Brahan. AMTEC (Association for Media & Technology in Education in Canada) Conference. June, 1976.

2. The Workstation Concept of GKS and the Resulting Conceptual Differences to the GSPC Core System. J. Encarnacao, G. Enderle, K. Kansy, G. Nees, E.G. Schlechtendahl, J. Weiss and P. WiBkirchen. Computer Graphics. Siggraph 1980 Conference Proceedings, Seventh Annual Conference on Computer Graphics & Interactive Techniques, Seattle, Washington, USA. July, 1980. Vol 14 #3, pp 226-230.

3. Computing Surveys. December, 1978. Vol 10 #4, pp 363-464.

SYSTEM GRAPHICS FUNCTIONS

MOVE(x,y)	- move the cursor to the absolute position (x,y)
RMOVE(dx,dy)	- move the cursor relative to the current position
LINE(x,y)	- draw a line or series of lines with the initial position as the current cursor position and the remaining line endpoints determined by x and y. If x and y are scalars then one line segment is displayed with absolute endpoint (x,y). If x and y are vectors then display a series of lines with (xi,yi) determining the absolute endpoints of the connected line segments.
RLINE(dx,dy)	- as for LINE except that the dx,dy pairs will be relative to the previous cursor position.
MARK(x,y)	- plot marker symbols at the absolute locations determined by the x and y arguments which may be scalars or vectors as for the LINE function.
RMARK(dx,dy)	- as for MARK except that dx,dy pairs are relative displacements from the previous cursor position.
POLY(xv,yv)	- display a polygon with initial (and final) position at the current cursor location and other vertices determined by xv,yv pairs which are absolute locations. xv and yv must be vectors of length at least 2.
RPOLY(dxv,dyv)	- as for POLY except that arguments dxv and dyv determine relative positions of the vertices.
CHAR(text)	- display the text starting at the current cursor position. Note that the text may contain display sublanguage commands embedded in it.
CIRCLE(xc,yc,radius)	- display a circle with centre at xc,yc of given radius.
ARC(dx,dy,deg)	- draw an arc with centre located relative to current cursor position (by dx and dy) and making a traverse of the indicated number of degrees.
SET_LINE(l)	- set the line type attribute to that indicated by l.
SET_MARK(c)	- set the marker symbol attribute to the indicated character.
SET_COLOUR(h,l,s)	- set the colour attribute for lines that follow to the given hue, lightness, and saturation specified.
SET_BLINK(sw)	- set the blink status attribute on or off.
SET_FILL(n,dx,dy,xdeg,ydeg)	- set the filling attribute. Polygons, arcs, and circles may be filled with solid fill or patterned fill using horizontal and vertical lines that may be spaced as required or rotated up to 45 degrees.
SET_ERASE(sw)	- determines whether elements of the picture (lines, polygons, etc.) will be displayed normally or will cause an erase of the portion of the screen that they would draw on (i.e. draw in the background colour).
SET_WINDOW(xl,yb,w,h)	- set the window of the user coordinate (world) space.