Picaso: a computer language for art & design.
Available from Middlesex University’s Research Repository.

Copyright:

Middlesex University Research Repository makes the University’s research available electronically.

Copyright and moral rights to this thesis/research project are retained by the author and/or other copyright owners. The work is supplied on the understanding that any use for commercial gain is strictly forbidden. A copy may be downloaded for personal, non-commercial, research or study without prior permission and without charge. Any use of the thesis/research project for private study or research must be properly acknowledged with reference to the work’s full bibliographic details.

This thesis/research project may not be reproduced in any format or medium, or extensive quotations taken from it, or its content changed in any way, without first obtaining permission in writing from the copyright holder(s).

If you believe that any material held in the repository infringes copyright law, please contact the Repository Team at Middlesex University via the following email address:
eprints@mdx.ac.uk

The item will be removed from the repository while any claim is being investigated.
PICASO
A
COMPUTER LANGUAGE
FOR
ART & DESIGN
by
J A Vince
1975
BRUNEL UNIVERSITY
ABSTRACT

PICASO is a computer language specifically designed to enable the artist/designer to use the digital computer as a graphic tool.

It is a unique development in that it provides the artist for the first time, an integrated range of sophisticated graphic software in a format meaningful to the non-numerate user.

This thesis examines the problem area of art and design, and reviews relevant computer software that is currently available. It continues to define the software requirements of the artist and designer, and illustrates how these are met by PICASO.
6  PICASO EXAMPLES
   6.1 EXAMPLE OF THE 2-D LIBRARY
   6.2 EXAMPLE OF THE 3-D LIBRARY
   6.3 SHAPE MANIPULATION
   6.4 SURFACES
   6.5 SPECIAL EFFECTS
   6.6 VARIOUS EXAMPLES

7  APPLICATION AREAS FOR PICASO
   7.1 ART & DESIGN
   7.2 MATHEMATICS
   7.3 ENGINEERING

8  CONCLUSIONS

9  APPENDICES
   9.1 SUMMARY OF PICASO SYSTEM COMMANDS
   9.2 SUMMARY OF PICASO INPUT/OUTPUT COMMANDS
   9.3 SUMMARY OF PICASO SHAPES
   9.4 SUMMARY OF PICASO OBJECTS
   9.5 SUMMARY OF PICASO STRUCTURE MANIPULATING COMMANDS
   9.6 SUMMARY OF PICASO PLOTTING COMMANDS
   9.7 SUMMARY OF PICASO STRUCTURE ANALYSIS COMMANDS
   9.8 SUMMARY OF PICASO FUNCTIONS
   9.9 SUMMARY OF SPECIAL EFFECTS
   9.10 SUMMARY OF ARRAY HANDLING COMMANDS
9.11 SUMMARY OF PICASO SURFACE COMMANDS
9.12 PICASO DATA STRUCTURE
9.13 WINDOWING ALGORITHM
9.14 ISOMET ALGORITHM

10 REFERENCES

11 PICASO USER MANUAL
1 INTRODUCTION

There has never been a deliberate attempt to create and develop computer art, its existence and evolution has always been unstable and uncontrolled which has accounted for considerable waste of effort and time in rediscovering well established graphic concepts. Consequently, artists working with computers have a serious communication handicap in that there is not available a universal language to interface the artist to the computer, which requires them to master unsuitable programming languages to achieve relatively primitive graphic effects.

PICASO is a modular structured language developed by the author for the artist or designer, enabling him to manipulate real and abstract graphic concepts without the burden of advanced programming. PICASO has also been designed to work with a small digital computer with the minimum of graphic hardware, namely a graph plotter, and yet produce sophisticated graphic output.

The creation of PICASO is the result of several years research by the author into computer graphics to discover and establish the important problem areas encountered by the novice, and its development has been influenced considerably by the opinions of artists working with computers.
1 INTRODUCTION Continued.

This thesis describes the problems encountered by computer artists and how PICASO offers a real and original contribution to the development of computer graphics in the areas of art and design.
STATEMENT OF THE PROBLEM

2.1 COMPUTER ART

Computer art is an art form that has emerged over the past two decades involving the digital computer and graphic peripherals to express graphic ideas. The art form is unique in that it permits the artist to explore techniques that were avoided due to the time and skills required, and also provides a vast source of graphic effects that are unique to the computer.

The earliest forms of computer art consisted of generating pictures on a line printer to create images from patterns of characters and over-printing them to create shading. Obviously, this had limited applications but was a first step to realising the potential of the computer as a graphic tool.

The development of the digital plotter and the graphic display provided two important tools for the artist, as these permitted the creation of simple line drawings. However, the control of these ideas demanded a programming expertise that is not common among artists. This severe user restriction has inhibited the growth of computer art as a universally available art form.
2.2 THE GROWTH OF COMPUTER ART

As early as 1960, the Boeing Company were using the computer to draw three-dimensional human figures to assist designers in the ergonomics of cockpit design, and since this pioneer work computer graphics has continued to play a major role in computer aided design. Computer art however, emerged in the wake of computer graphics exploiting software primarily designed for engineers to the advantage of the artist.

The lack of specific software at this time severely restricted the growth of computer art, as it has been estimated that in 1971 no more than 1000 researchers were working on computer art in the entire world. Even today in 1975 there is still a real problem with the availability of software, whilst one institution might have access to advanced graphic hardware and the latest software, another might not be beyond the stage of reproducing simple geometric patterns on a graph plotter. So the problem with growth is not just a question of rate, but that it has not been geographically uniform.
2.3 THE PRESENT PROBLEM

Investigations have shown that the majority of computers in educational establishments with graphic equipment are not exploiting the full capability of this equipment, due to the limited software available. This situation is so serious that immediate steps have been taken to organise at a national level the acquisition of graphic software, and the redistribution to educational users. Therefore the present problems are the lack of general graphic software at a global level and specifically software for the artist/designer.

It is incorrect to believe that there is a total absence of graphic software, obviously it must and does exist, but it tends to fall into various categories.

Machine dependent software

It is not always possible to design software to be independent of hardware, for example a language like SPROGS has been specifically designed to operate a SD4020 microfilm recorder, but there is software available that is machine dependent from the aspects of core requirement and compiler availability. Often one can attempt to implement algorithms developed on large machines, only to discover that the programmer made no attempt to conserve core space, with the result that the user with a small or medium size computer is unable to use the program. There exists also other software that requires associated programming languages such as PL1 and APL, languages that are still not universally
available.

**Specialist software**

The nature of computer graphics is such that it is not, and probably will not be supported in the near future by a universal language; this has resulted in the use of existing languages and the development of specialist graphic languages to cope with individual problem areas. (6)

MULTIPATCH a language for interactive design and COMPAX a language for scene analysis are totally different in structure, syntax and application, but they are both useful graphic languages. This specialisation of languages presents a problem to the user investigating several problem areas simultaneously, but is of little assistance to the artist/designer who is unable to define the extent of his problem area.

**Expensive software**

All software is expensive, and Licklider's article in 1969 'A Picture is Worth a Thousand Words - and It Costs' still holds true today. (9) The DISSPLA package is probably one of the most sophisticated graphic software systems available. It is machine and device independent, implemented in FORTRAN, works in two and three dimensions with or without hidden-line removal, but as one can appreciate it has a correspondingly high outright purchase price. The educational user might require DISSPLA's features, but could not afford them.
Here then is the dilemma. On one hand the artist needs graphic software, and on the other, software is available but is either too expensive, unproven, unsuitable, too specialised or machine dependent.

In spite of these limitations, computer art has managed to survive and grow into a world-wide activity, and has had considerable influence on art and design.
3 STATE OF THE ART

3.1 COMPUTER GRAPHICS IN EDUCATION

With the advent of mini-computer graphic systems, computer graphics is now within the budget of many educational institutions. It has already influenced many areas of education, and it is difficult to isolate one subject that has remained untouched.

The author's personal work alone has involved him in civil engineering, electrical engineering, control-engineering, music, mathematics, fine art, and graphic design. Unfortunately this influence has been local rather than global, and has treated privileged areas of activity where there exists adequate expertise, but many under privileged areas where there is a desire to enjoy the same facilities, but no expertise available.

3.2 COMPUTER GRAPHICS IN ART & DESIGN

Computer graphics finds its way into art and design through courses like DipAD in Fine Art, Graphic Design and Interior Design. The course content depends entirely upon the graphics system implemented, and consequently varies considerably. This is an unfortunate state of affairs, but little can be done until the software aspect is formalized.

In the author's experience, computers can produce a wide variety of responses from art students, ranging
from a total rejection to complete absorption in computer techniques, neither of which is totally healthy. An initial reaction is to regard the computer as totally non-creative, which is true in some respects, but difficult to prove until the student has had a chance to experience using the machine.

At first, the whole concept of using computers in the area of art appears foreign, but students who study the subject with an open mind discover that there is a real place for the computer.

A good graphics system can teach art students many things about their own subject. It can help formalize many abstract concepts concerning space, symmetry, perspective etc., and even provide a completely new insight into shapes and objects and their topological properties.

There will always be students who remain unconvinced about the use of computer techniques in art, and perhaps the image of the computer together with unsuitable software, has a lot to answer for in this context. Perhaps the next generation of software will assist in providing greater cohesion in this entire subject area.
4 SOFTWARE REQUIREMENTS FOR ART & DESIGN

4.1 LANGUAGE TYPE

The most popular language for implementing graphic programs is FORTRAN, as it provides features for expressing mathematical ideas that can directly represent graphic concepts. However, for an artist the use of FORTRAN creates two major problems; the first is learning FORTRAN, and the second is the acquisition of programming techniques such as data structure handling, algorithms, and possibly vector analysis, coordinate geometry, topology and matrix theory. The author has actually taught FORTRAN to artists and discovered from first-hand experience the conceptual problems this language poses to the artistic intellect.

The artist obviously requires a language that handles directly graphic concepts, and therefore calls for a special language. Although many languages have been developed there is not one that has managed to acquire the status of a language like FORTRAN or ALGOL. Some of these existing languages are described in Section 4.3.

4.2 LANGUAGE PHILOSOPHY

The majority of problem-oriented languages enable a user to identify system elements, groups, procedures and operations with names familiar and relevant to the user. Therefore any graphics language must be capable of handling ideas such as space, form,
movement, texture and colour etc, in a way that is natural for the artist. He must be able to express his thoughts in a notation that will not hide the meaning of his ideas. The following features are considered essential requirements for such a language.

4.2.1 SPACE
The language must permit the user to handle two and three-dimensional spaces so that 3-D objects may be manipulated together with 2-D shapes with equal ease.

4.2.2 FORM
The data structure handling shapes and objects should enable essential features of form to be controlled such as: vertices, surfaces, visibility, size, displacement etc..

4.2.3 VIEWPOINT
A two-dimensional space presents no unusual problems regarding a viewpoint, however, three-dimensional systems must provide natural perspective transformations for all viewpoints.

4.2.4 MOVEMENT
Both the observer and object must be capable of movement within space.

4.2.5 SCENE ANALYSIS
A graphic scene must be available for analysis to
permit development by the user before a final drawing is created.

4.2.6 **LANGUAGE SYNTAX**
The hardest aspect of learning any computer language, is understanding the syntax, therefore the non-numerate user must not be faced with problems such as precedence relationships, evaluation procedures and algorithm design.

4.2.7 **LANGUAGE SEMANTICS**
The language semantics must be clear and precise, with meaningful graphic names associated with programming elements with the minimum amount of symbolic notation.

4.2.8 **IMPLEMENTATION**
Assuming that the previous graphic requirements can be met, the remaining features relate to the method of implementation.

If the language is to enjoy a wide range of users, it must be implemented in a language that is readily available, and also be independent of hardware configurations.

The life of the language will depend upon its effectiveness in solving problems, and as the boundaries of art and design are virtually undefined, its life will ultimately depend on how far it can be extended.
4.3 EXISTING SOFTWARE

In May 1963, L.G. Roberts published a paper illustrating a solution to the hidden-line problem. During the following twelve years many researchers developed other elegant solutions to the same problem, notably, Warnock, Loutrel, and Matsushita.

It is strange that what appears on the surface to be a trivial problem, has attracted so much devoted attention. Even today this one aspect of computer graphics still attracts the inquisitive mind, to discover new methods of resolving the problem.

This complete problem area is concerned with similar conceptual problems, and the deep underlying complexity has created virtual world-wide activity in computer graphics, from the formulation of graphic languages, to the production of computer aided motion pictures.

Solutions to these problems tend to be of a mathematical or logical nature, and demand a skill in computer programming to implement them. Obviously the non-numerate person would be unable to cope in this situation, and is why the majority of artists are unable to develop their own software.

Knowlton describes programmers and artists as "creative, imaginative, intelligent, energetic, industrious,"
competitive, and driven", but programmers were "logical, inhibited, methodical...", whilst artists were "alogical impulsive, and intuitive". This gross difference in behaviour, in the author's opinion, is why the artist is unable to cope with the algorithmic nature of scientific programming languages.

Knowlton's collaboration with Mrs. Schwartz although resulted in several creative productions, had similar problems. The programmer (Knowlton) was unable to influence the artistic content of the project, and the artist (Schwartz) was unable to contribute to the programming side of the work. This division of effort is completely unrealistic from both sides, for ideally there should be no need for a programmer. The artist must be in a position to control and guide the entire project from concept through implementation to the final goal.

In comparison to technical graphic software systems, there are very few systems supporting the art and design area. However the relevant systems are discussed below.

4.3.1 TARPS (KNOWLTON 1971)

TARPS (Two-dimensional Alphanumeric Raster Picture System) was written by Knowlton at the University of California. He describes this language as a set of macros written in terms of BELFIX which describe
operations on a large 92x126 array of alphanumeric characters. The films he produced with Vanderbeek were rather limited in their scope, as they were only concerned with the manipulation of patterns of characters. Knowlton's own criticism was "I was disappointed in the language - it seemed too restrictive".

4.3.2 EXPLOR (KNOWLTON 1970)

EXPLOR (EXplicit Patterns, Local Operation and Randomness). This was also written at the University of California and was developed in a collaboration with Lillian Schwartz, resulting in the films "UFO's and Pixillation". Again, Knowlton was not completely satisfied with the language because of the lack of feedback into the language design process by the users.

It should be realised that TARPS and EXPLOR were not attempts to completely interface the artist to the computer, one can regard them as exploratory systems in the area of animation.

EXPLOR is currently being implemented at Imperial College London.

4.3.3 PDL (G.WYVILL 1972)

PDL (Pictorial Description Language) was written by WYVILL at the University of Bradford, to produce computer drawings with a small budget. This might only
consist of a mini-computer, teletype and a digital plotter.

It is aimed at the artist/designer and includes important language features such as programmer defined words for shapes, simple syntax and an efficient data base system. The main restriction with PDL is its limitation to work in two dimensions. A completely new design philosophy is required to include the handling of 3-D structures.

4.3.4 SPROGS (R.E.THOMAS 1974)

SPROGS (SD 4020 PDP15 Rapid Output Graphics System) was developed by Thomas at the Atlas Computer Laboratory. It is a language consisting of a set of FORTRAN subroutines that interface the user to the SD 4020 microfilm recorder, and VT04 refreshed display.

It is an extremely powerful language to handle an equally powerful computer system. In some respects this is a disadvantage as its user is restricted to the sophisticated computer hardware necessary to drive the recorder.

The language does not appear to contain a large library of shapes, and those that do exist are two-dimensional. The creation of 3-D objects is left to the user. SPROGS also requires a reasonable ability in FORTRAN programming, and thus restricts its use in the area of art and design.
4.3.5 **CAMP & CAMPER** (FRANCIS, HOPGOOD & RALPHS 1973)

CAMP (Computer Aided Motion Pictures) was developed at the Atlas Computer Laboratory, from an original idea by S. Anderson to produce an efficient picture language for producing computer animated films. CAMPER is basically a 3-D extension of the 2-D CAMP package.

The authors of the system claim that no knowledge of computer programming languages is required to use it. The user writes a program by creating a sequence of CAMP statements that have a standard format. In this way the beginner need not be concerned with memorizing different statement structures.

The prime objective of CAMP is to interface the user to a microfilm recorder without the user being aware of the underlying complexity that actually exists. So often when a problematical hardware feature is disguised by software, the problem is not always removed, but transformed. CAMP seems to overcome this aspect.

4.3.6 **GINO** (WOODFORD et al 1965)

GINO (Graphical INput and Output) was developed by the University of Cambridge Computer Aided Design Group, and was designed and implemented as a general purpose graphics package.

The GINO system is accessed via a high-level language
normally FORTRAN, which enables the user to develop, manipulate and eventually display a graphic scene. Probably the greatest advantage of this system is its ability to be interfaced with any system of hardware.

It includes all the features one expects from a sophisticated commercial package and consequently demands a high-level programming expertise from the user. GINO has not been designed to cater for the user working in art and design, but could play an important role in implementing such a system.

4.3.7 ART1 (NASH & WILLIAMS 1970)

ART1 was developed at the University of New Mexico to permit students to produce graphic output using a line-printer or teletype. Obviously the limitations are considerable, but it presents a simple and practical method of introducing computing concepts to art students.

At present Teeside Polytechnic are using the package in their DipAD and are hoping to develop an interactive version.

4.3.8 PLAD (SAUNDERS, R 1972)

PLAD (Programming Language for Art and Design) was written by Roger Saunders as a B.Sc. project at Brighton Polytechnic. It is based upon the language ART1, but provides a formal problem-oriented language
to express primitive graphic ideas that is not available in ART1. The output is in the form of patterns of characters that may be overprinted to achieve shading, and therefore has limited applications. Although this is an excellent tool for the artist, it still does not provide a language that can be used with a wide variety of graphic problems.
PICASO

5.1 DESIGN PHILOSOPHY

PICASO has been designed to satisfy a need for software in the area of art and design. The design specification for the language was as follows:

1) allow the user to manipulate graphic concepts in a meaningful way,
2) maintain language syntax and semantics at a non-technical level,
3) ensure an extensible language,
4) machine independent,
5) permit interfacing with various graphic peripherals,
6) permit implementation on small memory computers, and
7) be easily implemented by other users.

The design of any program normally requires some sort of compromise in the design specification or mode of implementation to ensure a successful completion to the project. If any compromise occurred in the design of PICASO it was rarely detrimental to the above specification, but did cause great concern when algorithms were developed to produce surfaces with hidden-line removal, and still allow them to be implemented on small machines.

The small-machine environment makes great demands upon a programmer's creativity, especially in discovering algorithmic techniques to implement
ideas that at first sight demand a vast memory. Whenever there was a choice between object run-time and memory, the former was sacrificed, but to help offset this, the coding stage was greatly influenced by statement execution times with the emphasis on efficiency.

The project demanded extensive research to establish existing software and the retrieval of papers concerned with important concepts such as hidden-line removal, windowing, perspective etc. Although this was an essential exercise, the author was unable to implement these ideas without sacrificing the design specification. As a result of this, considerable time and effort was devoted to research into algorithms to solve the same problems within the small machine environment. The outcome of this work established a highly integrated structure that enabled PICASO to be designed in modules. From the outset, the complete modularity of the system permitted it to be thoroughly tested with the introduction of a new module which has given PICASO a high level of integrity.

A detailed evaluation of how the system was implemented follows.
5.2 SYSTEM STRUCTURE

PICASO is derived from the words:

PIpicture Computer Algorithms Subroutine Orientated,

and consists of an integrated system of procedures that are sensitive to a common data structure to handle two and three-dimensional graphic structures.

The system is implemented in FORTRAN IV to ensure that it is machine independent and easily implemented by other users.

Figure 5.2.1 shows a block diagram illustrating the underlying concepts employed in PICASO. The user is permitted to work in two and three-dimensional space in the same program and manipulate shapes and objects with equal ease. The shape and object libraries supply a comprehensive range of structures including polygons, ellipses, cubes, cones, spheres etc., and external structures may be input via any peripheral capable of communicating coordinate data.

Algorithms may then be used to manipulate structures within the conceptual and projection spaces before finally being realised on the projection space which represents the computer graphic media, such as a digital plotter of display screen.

The realisation of 2-D shapes is a simple process of interpreting the shape's spatial frame of reference as the projection frame of reference, but 3-D objects are realised by locating an observer within the object's conceptual space, and viewing through a picture plane which represents the projection space. The mode of
viewing may be true three-point perspective or wide-angle perspective, but could be extended to include any type of mathematical projection.

To permit the user to optimise a scene, algorithms are available to reference the conceptual and projection spaces and supply spatial and graphic information back to the user who is then able to control the growth and development of his work.
Figure 5.2.1
5.3 LANGUAGE SYNTAX AND SEMANTICS

PICASO is an integrated set of FORTRAN subroutines and functions that may be called or referenced by a main program. The modularity is such, that a beginner is required to know extremely little about the syntax of FORTRAN apart from the CALL statement, however, there are three programming concepts that must be understood, namely: numbers (INTEGER and REAL), variables and sequential processing, but these are relatively easy to comprehend.

A typical program will only consist of CALL statements, apart from STOP and END, with the underlying complexity of the system completely transparent to the user, but the language semantics are made self-evident by the choice of procedure names. The following program might be attempted by someone who has only had approximately two to three hours tuition in PICASO, but allows him to immediately appreciate and handle three-dimensional objects and realise perspective views.

```
CALL START
CALL ORIGIN()
CALL EYE()
CALL CUBE(BOX, arguments are)
CALL ROW3D(BOX, omitted for clarity.)
CALL FINISH()
STOP
END
```
The objective of the program is to initialise plotting, (CALL START) establish a new origin on the paper, (CALL ORIGIN) and locate an observer in the 3-D conceptual space (CALL EYE). A cube is called from the object library (CALL CUBE) and given the name BOX, and a regular row of cubes is drawn in perspective as seen by the observer (CALL ROW3D). Finally, plotting is terminated, the program stopped (CALL FINISH and STOP).

The ability to identify structures by real-world names is an essential requirement of any graphics language, and what the PICASO user is actually doing when he identifies an object by a name, is reference a FORTRAN vector that stores the vertex data of the structure.

Once the user has written several programs consisting entirely of CALL statements, he is in a position to use arithmetic statements and the DO statement which provide programming techniques to solve quite complex problems. Independent of the language chosen, the user would have to master and understand its syntax and semantics, and it is believed that these aspects of PICASO have been maintained at a level that is acceptable to the non-numerate user.
5.4 PICASO SPATIAL CONVENTIONS

As described in section 5.2, PICASO employs a conceptual and projection space that are referenced by conventional methods. The axial systems are shown in Figure 5.4.1.

For the two-dimensional mode the conceptual and projection space origins are coincident, whereas the three-dimensional mode enables the origin of the projection space picture plane, to be located anywhere on the line connecting the observer's eye, and the point under observation in the 3-D conceptual space. Figure 5.4.2 illustrates this transformation process diagramatically.

A common feature of algorithms creating perspective transformations, is their excessive execution time due to the use of trigonometric functions, consequently, PICASO employs a labelled-common block to hold partially evaluated data concerning direction cosines, the eye
location and the point under observation; this avoids
the repetitive evaluation of items that are constant
for a given observer's location. The common block
also communicates the mode of projection which can
either be true or wide-angle perspective, but as
these transformations are produced by one module,
any future requirement can be catered for by
substitution of this module.

Figure 5.4.2
5.5 PICASO STRUCTURES

A PICASO structure may either be a two-dimensional shape or a three-dimensional object, and is stored in a FORTRAN vector as shown in Appendix XII.

A structure may consist of one or more continuous sequences of vertices, that are eventually connected by straight lines when realised on the projection space. These contours may be open or closed depending on whether the first vertex is referenced again as a terminal vertex. A 2-D contour is nothing more than a line existing on a surface, namely the picture plane, however, a 3-D contour may be interpreted as a surface existing in conceptual space and be used to assemble transparent or opaque objects.

The vertex sequence has great significance, as it is used to provide hidden-line removal, masking, windowing and shading, but this is explained later in section 5.7.

PICASO permits the user to access structures in two modes, either from the internal library, or from an external source via an input peripheral. The contents of the 2 and 3-D libraries is shown in appendices III and VI.

The author believes that a structure library is an extremely important aspect of any graphics language, as it can determine whether the user will succeed in mastering the system, or surrender through frustration.
in being unable to reproduce any form of graphic output.

At present, the subroutine SHAPE and OBJECT are used to access structures from punched cards, future routines are planned to access vertex data from other sources of coordinate data.

The majority of shapes are generated by functions, but the system includes others such as HORSE, FACE and BUTFLY (butterfly) etc., that are actually stored in coordinate form; this is included because the beginner is then able to handle familiar shapes and thus acquire an immediate confidence in the language. Obviously this aspect of PICASO is completely extensible and will depend entirely upon the needs of the user.

The family of function derived shapes consists of standard mathematical forms and provides the user with anything from a line to a hypotrochoid.

All structures are assigned a programmer defined REAL FORTRAN name, and are manipulated and drawn by reference to this name.
5.6 STRUCTURE MANIPULATION

An essential feature of computer graphics is structure manipulation, and PICASO includes a powerful system of commands enabling the user to control and manipulate precisely any PICASO structure. These commands are listed in appendix V.

The basic operations to control size, displacement and rotation are obvious requirements, but the artist is always interested in transformations that probably have no direct application in standard graphic systems, but are useful from an artistic point of view. Consequently, the author designed many unusual algorithms to complement the standard list of procedures; some of these are now explained.

5.6.1 CYCLE

Any PICASO contour consists of vertices connected by straight lines. What CYCLE does, is to move the position of a vertex to a new position on the line connecting it to its neighbour. If this process is repeated continuously, some effective designs result. Three examples are shown in Figure 5.6.1.

5.6.2 ASMDSH

PICASO's graphic power derives from the ability to handle complex geometric structures by name, and manipulate them as an entity. At an early stage in this project, the author considered the graphic
analogue of the FORTRAN arithmetic statement. For example, the following statement:

\[ I = J + K \times L \]
could be written in graphic concepts as:

\[ \text{hand} = \square + \triangle \times \bigcirc \]

where a triangle is multiplied by a circle and added to a square, producing some resultant shape. ASMDSH (Add, Subtract, Multiply and Divide SHapes) is the result of considerable research into this problem, and performs four arithmetic operations upon any PICASO structure by manipulating the numeric values of the vertex coordinates. When structures contain unequal numbers of vertices, linear interpolation establishes intermediate positions. ASMDSH provides a powerful method of generating families of shapes that would be extremely difficult to define mathematically, due to the nature of their derivation.

Figure 5.6.2 shows the effects of adding, subtracting, multiplying and dividing a circle and a triangle.

5.6.3 FORM3D

Provides a quick method of creating distorted 3-D surfaces from 2-D shapes. The user supplies a PICASO shape, together with a open 2-D contour which 'forms' the z-coordinate thus forming the third dimension. Figure 5.6.3 shows the result of an elephant formed by a bend.
CYCLE

Figure 5.6.1
5.6.4 TRACE

TRACE generates a new shape that results from tracing a point a specified distance in or outside some PICASO shape. The result depends entirely upon the convex and concave features of the original shape, and often surprises the user. Figure 5.6.4 illustrates the effect of tracing around a horse and tracing inside a butterfly.

5.6.5 TRANSH

Shape transformation has often been exploited in computer art, and PICASO is supplied with the subroutine TRANSH to transform any 2-D contour into another, or any 3-D surface into another. When a sequence of transformations are superimposed upon one another, a distinctive and effective picture results. Figure 5.6.5 shows a square-law transformation of a face into a square.
Figure 5.6.5
5.6.6 WARP2D & WARP3D

These two subroutines enable the user to establish different laws to control the conceptual space, thus creating 'space-warp' effects. For example, it might be required to apply a logarithmic law to the z-direction, a hyperbolic law to the y-direction, and a square law to the x-direction, and then view a system of objects through a wide-angle lens. The less esoteric operation of folding about the x-axis could be achieved by making the x-coordinate function equal to: $F(x) = \text{ABS}(x)$; this effect is shown in Figure 5.6.6 with a rhinoseros.
WARP2D

Figure 5.6.6
5.7 PICA S0 DRAWING COMMANDS

The PICA S0 drawing commands transform data from the conceptual space to the projection space. At the individual structure level this includes DRAW and DRAW3D which manipulate a single shape or object respectively; but at the group structure level, shapes and objects may be referenced by rows and two types of regular grids.

One practical problem with the projection space is being capable of containing the contents of the transformed conceptual space. To safeguard the user from drawing beyond the boundaries available, a powerful windowing facility is included which accepts any multiple contour PICA S0 shape to window the projection space.

If the window shape contains contours with their vertices specified in a clockwise sense, then these behave as transparent windows, but counter-clockwise contours mask the drawing. The example shown in Figure 5.7.1 is of a scene with and without a window. The technique employed in this process is explained in Appendix XIII.

When an observer is located in 3-D conceptual space, care must be taken to ensure a location is chosen that is physically meaningful, as this can result in a confused distorted projection. For example, if the observer's eye was inside an object, vertices behind the eye would not be seen and must be clipped.
Figure 5.7.2
from the final projection. DRAW3D which realises projections of objects, includes this clipping feature. The minimum distance before clipping occurs, is normally 1.0 inch, but this is adjustable to enable cross-sections of objects to be produced. Figure 5.7.2 shows views of a cone as the observer moves through its interior.

Hidden-line removal only occurs at the individual structure level, and is achieved by testing the sense of the vertices when a surface is projected onto the picture plane. If the vertex sense is found to be counter-clockwise, then it is being viewed from behind, and may be removed by the user. Figure 5.7.3 shows a cube and cone with and without hidden-lines removed. This feature requires initially that all surfaces are declared in a clockwise sequence, which is a PICASO convention.

Two simple methods are available for producing texture, and they are HATCH and STIPLE. HATCH will draw a series of parallel lines within a PICASO shape at any angle or separation; STIPLE will create a random pattern of dots inside any PICASO shape. Figure 5.7.4 shows examples of HATCH and STIPLE.

Although it is assumed that PICASO structures consist of vertices joined by straight lines, it is very useful to be able to join them together with a smooth curve.
Figure 5.7.4

STIPLE

HATCH

Figure 5.7.4
An existing algorithm was modified to accept the data structure of PICASO, which uses a cubic spline to join together vertices. This routine SMOOTH, works for open or closed contours and an example is shown in Figure 5.7.5.

A complete list of plotting commands is shown in Appendix VI.
5.8 PICASO STRUCTURE ANALYSIS COMMANDS

The analytical commands represent an important feature of PICASO in that they permit graphic problem solving in a non-interactive mode. It is not essential that all graphic programs have to engage the user in a real-time dialogue with the program, many types of problems may be solved by advance analysis which develops useful skills in problem solving, and can provide a deeper understanding to the problem area. Appendix VII contains the list of commands at present available in PICASO, but is expected to grow in content as other graphic effects are demanded.

The concept of inside and outside is quite important to many aspects of computer graphics, therefore, a special function was developed for PICASO, this is INSIDE. INSIDE is a logical function that returns a value of .TRUE. if a given point is found to be 'inside' a specified PICASO shape. The shape may be multiple-contour, and the vertex sequence of these contours is highly significant to the operation of INSIDE.

When a PICASO contour has vertices in a clockwise sense, INSIDE will be .TRUE. for all points 'inside' the contour, but for a counter-clockwise contour INSIDE will be .FALSE.. By altering the sense of contours with REVERS, the user is able to create
complex shapes that include holes. Figure 5.8.1 shows two systems of contours and the interpretation of inside and outside.

The two functions CLOCK and CLOK3D may be used to detect the vertex sense of 2 and 3-D contours respectively, and CLOK3D is actually used by DRAW3D to produce hidden-line removal. This type of function is very useful in software writing as it permits statements such as:

\[
\text{IF(CLOCK(A,1)) CALL REVERS(A,1)}
\]

which represents a powerful graphic operation.

Further shape and object analysis is performed by the functions:
NDIMEN which returns the dimension of the structure,

NLINES which returns the number of contours in a structure,

NPOINT which returns the number of vertices in a contour,

and the three general type functions:

\[
\begin{aligned}
YCOORD \\
ZCOORD
\end{aligned}
\]

which return the respective coordinate of a vertex.

These permit the user to retrieve important characteristics of structures in conceptual space.
5.9 PICASO SURFACES

Computer drawn surfaces have always played a useful means of communicating large quantities of data effectively, they are also aesthetically pleasing to the 'artistic eye', and have already been used in computer art.(17)

PICASO includes a variety of simple ways of creating three-dimensional surfaces that may be exploited to create very effective graphic output.

There are basically three ways of simulating surfaces on a plotter or visual display. The first consists of reducing the surface to a number of parallel lines, the second divides the surface into rectangular tiles, and the third, shades the surface either with dots for the plotter, or alters the screen intensity for a graphic display. PICASO incorporates all three techniques, however the shading algorithms are still in the process of development.

The subroutine SURFAC uses the first technique of assembling a surface from a number of lines, and includes an effective hidden-line algorithm. SURFAC produces a front-elevation view of a surface that is generated by spinning a PICASO open contour about some vertical axis. The user supplies information concerning the contour, position of
rotation, the angle of elevation etc, and SURFAC provides a realistic surface. Two such surfaces are illustrated in Figure 5.9.1.

The subroutine ISOMET generates a surface composed of rectangular opaque tiles viewed in isometric projection. Again the user supplies similar information as supplied to SURFAC, and the drawings in Figure 5.9.2 result.

The style of surface created by SURFAC and ISOMET is limited as they only permit symmetrical surfaces. Consequently, two extra routines were developed to enable irregular line and tiled surfaces to be drawn, those are HIDE and ASOMET.

HIDE draws out one line supplying hidden-line removal based upon the previous lines drawn, and the user develops a surface by calling HIDE with new data. ASOMET accepts a matrix of numbers, that represent vertical measurements on a surface at regular positions. From this, an isometric view is produced. An example is shown in Figure 5.9.3.

The isometric view is popular in computer graphics as it allows a simple algorithm to remove hidden-lines; a perspective view made from any point in space requires considerable execution time and memory. However, a technique employed in SURF3D, breaks up a surface into rectangular tiles and then tests to see if it is seen. Although there are areas where the
surface remains transparent – it is still very effective. Figure 5.9.4 shows two views of the same surface, with and without hidden-lines.

Contour maps are extensively used in geographic problems, but were included in PICASO to complete the range of surface techniques. MAP2D produces a plan view of a contour map whilst MAP3D realises a 3-D perspective view, with each contour in its true position in space. Data are supplied in the form of a matrix together with a range of heights to search for. Figure 5.9.5 shows how the same set of data is interpreted by the two programs.

Surfaces that form solid objects may also be created with equal ease by the subroutine SILUET. This develops a tiled opaque or transparent surface by rotating an open 2-D PICASO shape about a vertical axis, the resulting envelope creates the surface. Figure 5.9.6 illustrates various examples produced by SILUET.
Figure 5.9.3

ASOMET
5.10 PICASO SPECIAL EFFECTS

PICASO special effects are created by a series of subroutines that produce a final graphic effect, without the requirement of user programming. These are useful from an educational aspect when teaching the beginner, as programs consisting of no more than six statements can produce surprising graphic output. Perhaps the best example of this is the routine CONECT, which produces a popular example of computer art. CONECT accepts a PICASO shape and joins every vertex to every other vertex by a straight line. Two examples of CONECT are shown in Figure 5.10.1. A three-dimensional version CONEC3 performs the same operation on PICASO objects.

GROW, GROW2D, and GROW3D are a family of subroutines to simulate recursive growth patterns. In the 2-D mode (GROW), a shape is drawn, and the same shape is drawn at every vertex of the original shape; this process may be continued to any depth, but excessive execution time dictates a final limit. Figure 5.10.2 shows a pentagon 'grown' to a depth of two.

MODS11 is derived from the words 'modulate shape'. This performs a substitution process on a PICASO shape by replacing a line joining two vertices by an open 2-D PICASO contour, creating a simple modulation effect. The process may be repeated to a
depth that is dictated by available memory space. Figure 5.10.3 shows the result of modulating a hexagon by a parabola.

SNOW is another type of recursive program which turns every line joining two vertices into a small 'snow-flake'. The process may be repeated to any depth, but requires enormous amounts of processor time when the shape is large and the depth exceeds four. Figure 5.10.4 shows an original five-pointed star and a snowflake version grown to a depth of three.

These subroutines are only written as an aid to the beginner, but it is clear that they will grow in number as future demands are made upon the system.

A complete list of subroutines is given in Appendix IX.
Figure 5.10.2
SNOW

Figure 5, 10, 4
5.11 PICASO FUNCTIONS

Generally, students learning to use PICASO tend to experiment with the structure handling routines before working with functions. Functions are only normally used when conventional methods break-down, or an effect relies entirely upon their use.

The majority of the functions are related to shape generation, and duplicate the shapes contained in the PICASO shape library. The functions always begin with an X or Y, for example XPARAB & YPARAB identify the coordinates of a particular point on a parabolic curve. When three dimensional trajectories are required, they may be created from combinations of these generators. For example, a 3-D spiral could be formed from a 2-D Archimedean spiral (XASPIR & YASPIR) and a line (XYZLIN), the actual orientation in space would depend on how the functions were assigned to the axes of space.

To maintain user mathematics at a bare minimum, the functions are supplied with general characteristics of the shape together with the vertex required relative to the total number of vertices on the shape, e.g. the third vertex out of 20. This procedure is found to work very well in practice.

Other functions include TAKE and ITAKE which return
a pseudo-random REAL or INTEGER number respectively from a specified range. These functions are useful when a random element is needed to control the size and position of structures in space. An example is shown in Figure 5.11.1 which illustrates a wide-angle view of cubes with random size and position in space.

A complete list of functions is shown in Appendix VIII.
TAKE & FISHI

Figure 5.11.1
5.12 PICASO ARRAY HANDLING COMMANDS

The majority of FORTRAN compilers only permit 2-D arrays. This was regarded as a severe limitation to a graphics language, that worked in a three dimensional mode. Consequently, an entire range of subroutines and functions was designed to access and initialise one, two and three-dimensional REAL, INTEGER and LOGICAL arrays; these are listed in Appendix X.
6 EXAMPLES OF PICASO PROGRAMS

This section is concerned with the practical aspects of PICASO, but as it is impossible to explore every facet of the language various small programs have been written that illustrate typical graphic output.

6.1 EXAMPLE OF THE 2-D LIBRARY

An Archimedean spiral has been chosen to illustrate the multitude of effects that can be realised with just one basic curve.

The format for the spiral is:

\[
\text{CALL ASPIRA(ARRAY, RADIUS, CYCLES, N)}
\]

where:

- **ARRAY** is the name given to the spiral
- **RADIUS** is the final radius of the spiral
- **CYCLES** is the number of convolutions
- **N** is the number of points on the spiral.

An example could be:

\[
\text{CALL ASPIRA(SPIRAL, 2.0, 4.0, 201)}
\]

This statement generates an Archimedean spiral SPIRAL, consisting of 201 points, with four convolutions and a final radius of two inches.

If the user wanted to actually plot out this curve, the following program would be required:

\[
\begin{align*}
\text{DIMENSION SPIRAL(405)} \\
\text{CALL START} \\
\text{CALL ORIGIN(5.0, 5.0)}
\end{align*}
\]
CALL ASPIRA(SPIRAL, 2.0, 4.0, 201)
CALL DRAW(SPIRAL, 1.0, 0.0, 0.0)
CALL FINISH(10.0)
STOP
END

and the output would be as shown in Figure 6.1.1.

Figure 6.1.1

A slight modification to this program produces the examples shown in Figure 6.1.2. The actual change is in the number of convolutions.
Figure 6.1.2

Nc = 50

ASPIRA

Nc = 60
6.2 Example of the 3-D Library

This example illustrates how a wide-angle view could be made of a 20 by 20 grid of cubes. The stages of the program design are:

1) Initialise plotting.
2) Select an origin on the plotter paper.
3) Generate one cube.
4) Select a point of observation.
5) Draw a 3-D grid.
6) Terminate plotting.

The output shown in Figure 6.2.1 was produced by the following program:

```fortran
DIMENSION C(98)
CALL START
CALL ORIGIN(5.0,5.0)
CALL CUBE(C,1.0)
CALL FISHI(5.0,5.0,-20.0,5.0,5.0,0.0,0.0,4.0)
CALL GRID3D(C,2.0,20,1.5,20,1.5,0.0,0.0,0.0,-1)
CALL FINISH(10.0)
STOP
END
```
6.3 EXAMPLE OF SHAPE MANIPULATION

An interesting manipulative command is PULL, which distorts shapes by pulling or compressing them in a specified direction.

This example draws out four views of a horse that is being compressed vertically. The program design stages are:

1) Initialise plotting.
2) Select an origin on the plotter paper.
3) Generate a horse.
4) Repeat four times.
   4.1) Pull horse
   4.2) Draw horse
   4.3) Change origin
5) Terminate plotting.

The output shown in Figure 6.3.1 was produced by the following program:

```plaintext
DIMENSION H(237), P(237)
CALL START
CALL ORIGIN(5.0, 2.0)
CALL HORSE(H)
DO 1 I=1, 4
   CALL PULL(H, 0.0, 0.0, 90.0, FLOAT(I)/4.0, P)
   CALL DRAW(P, 1.0, 0.0, 0.0)
   CALL ORIGIN(0.0, 1.0)
1 CONTINUE
CALL FINISH(10.0)
STOP
END
```
6.4 EXAMPLE OF SURFACES

This example shows how shapes may be generated from others, and their eventual use in sweeping out a surface.

The requirements of this program are to create a surface of a waveform which consists of a linearly damped cosine wave.

The design stages of the program are:

1) Initialise plotting.
2) Select an origin on the plotting paper.
3) Generate a cosine wave.
4) Generate a line.
5) Create a damped cosine wave with ASMDSH.
6) Draw the surface with SURFAC.
7) Terminate plotting.

The surface in Figure 6.4.1 was produced by the following program:

```fortran
DIMENSION A(203), B(7), W(200), OV(200), UN(200)
CALL START
CALL ORIGIN(5.0,5.0)
CALL SINE(A,1.0,4.0,90.0,1260.0,100,0.0,1.0)
CALL LINE(1,1.0,1.0,1.0,0.0,2)
CALL ASMDSH(A,1,B,1,A,3)
CALL SURFAC(A,6.0,200,6.0,50,30.0,3.0,3.0,10.0)
W,OV,UN,0.0,0.0)
CALL FINISH(10.0)
STOP
END
```
Figure 6.4.1
6.5 EXAMPLE OF SPECIAL EFFECTS

In this example, CONECT is used to join together vertices on a 20 pointed star.

The program design stages are:

1) Initialise plotting.
2) Select an origin on the plotter paper.
3) Generate a 20 pointed star.
4) Call CONECT.
5) Terminate plotting.

The following program produced the output shown in Figure 6.5.1.

```
DIMENSION A(50)
CALL START
CALL ORIGIN(5.0,5.0)
CALL STAR(A,20,2.0,0.5)
CALL CONECT(A,1.0,0.0,0.0)
CALL FINISH(10.0)
STOP
END
```
Figure 6.5.1
6.6 VARIOUS EXAMPLES

EXAMPLE I

Reference to Figure 6.6.1 will show a 5 by 20 grid of cubes that have been 'cycled' 10 times. The secret of the solution is to identify that the complete drawing consists of two different rows of cubes interwoven together.

The first row has the cubes displaced by:

\[ X = \text{cube width} \]
\[ Y = 0.0 \]
\[ Z = \text{cube width} \]

and the second row has the cubes offset by:

\[ X = \text{cube width} \]
\[ Y = 1.0 \]
\[ Z = \text{cube width} \]

The program design stages are:

1) Initialise plotting.
2) Select an origin on the plotter paper.
3) Generate a cube.
4) Select an observation point.
5) Repeat the following 10 times:
   5.1) Repeat the following 10 times:
      5.1.1) Draw row of cubes.
      5.1.2) Draw row of offset cubes.
      5.1.3) Change offset variables.
   5.2) Cycle cube.
6) Terminate plotting.

The output illustrated in Figure 6.6.1 was produced by the following program:
A similar drawing is shown in Figure 6.6.2, but the scene is seen through a wide-angle lens.
Figure 6.6.2
EXAMPLE II

The butterfly shown in Figure 6.6.3 was created by repeatedly tracing around the contour of a butterfly at distances increasing according to a square law.

The program design stages are:

1) Initialise plotting.
2) Select an origin on the plotter paper.
3) Generate a butterfly.
4) Repeat 20 times:
   4.1) Draw butterfly.
   4.2) Increase tracing distance.
5) Terminate plotting.

The following program produced the drawing shown in Figure 6.6.2.

```
DIMENSION B(209), C(209)
CALL START
CALL ORIGIN(5.0, 5.0)
CALL BUTFLY(B)
DO 1 I=1, 20
   CALL TRACE(B, FLOAT((I-1)x(I-1))/400.0, C)
   CALL DRAW(C, 1.0, 0.0, 0.0, 0.0)
1 CONTINUE
CALL FINISH(10.0)
STOP
END
```

A similar drawing shown in Figure 6.6.4 was produced by the same program with more tracing.
7. APPLICATION AREAS FOR PICASO

The terms of reference for this project were to develop a computer graphics system, that could be easily used by people working in the areas of art and design. The result has been the graphics system PICASO.

Throughout the evolution of PICASO, the author has been aware of its potential as a general purpose graphics system. In fact it is difficult to specify a technical area that could not in some way use the system to some advantage. This section continues to examine three different application areas where PICASO is already having some influence.
7.1 ART & DESIGN

PICASO has been primarily designed to enable the artist/designer to use the graphic capabilities of the digital computer. The contents of the language has been formalised to anticipate the requirements of this type of user and the program examples in Section 6 illustrate the effectiveness and simplicity of the language.

The author has been teaching PICASO with a high degree of success to the following courses:

DipAD Fine Art (FT) Middlesex Polytechnic.
DipAD Graphic Design (FT) Middlesex Polytechnic
Computer Graphics in Middlesex Polytechnic.
Art & Design.

The system has also been implemented at the Slade School of Art on an IBM 360 system, and is being used by post-graduate students in Fine Art. It is also to be implemented at the Royal College of Art in the near future.

There are numerous application areas for computer graphics in art and design, some of them being:

1) Pattern generation,
2) Fabric design,
3) Animation,
4) Technical drawing aid for 2 and 3-D work,
5) Sculpture,
6) Cover design,
7) Fine art.

In all of these areas there is a place for computer graphics to assist the design process, not as a replacement for artistic creativity, but as a stimulus to explore and invent.
7.2 MATHEMATICS

Much of mathematics is concerned with the analysis of systems existing in space, and even when there is not a direct association with space, the spatial analogue can present a very realistic interpretation of the system.

As PICASO was developing, the author became aware of the potential of the system in teaching geometric concepts in mathematics. So often in mathematics, a concept is misunderstood because the student cannot visualise what is really happening in space; but with PICASO, the student has a means of simulating the mathematical model graphically. For example, there are two subroutines MATOP2 and MATOP3 which perform matrix multiplication upon two and three-dimensional structures. These allow the student to experiment and investigate directly, the relationships between matrices and the movement of objects in space.

The solution of equations is often taught from a graphical aspect, and PICASO has features that could be employed in this area.

Functions of the form:

\[ \text{Constant} = f(x, y) \]

could be represented as a two-dimensional array and viewed as a three-dimensional surface. This type of picture reveals so much data that a truer insight is often gained by this approach.
Other areas of mathematics that could be affected include:

1) Function generation,
2) Rotations and reflections
3) Mappings,
4) Surfaces,
5) Vectors.
7.3 ENGINEERING

Engineering can cover so many different areas that it might be useful to mention just two application areas with which the author has been associated.

The first is in civil engineering. Students studying for the BSc in Civil Engineering at Middlesex Polytechnic are taught PICASO as a general purpose graphics language. It is used in a wide variety of ways, from the analysis of experimental data, to the realisation of three-dimensional views of bridges and buildings. As this type of student already has an understanding of FORTRAN, the teaching of PICASO is just a natural extension to the language, and presents no problem to the student.

The second application is in control engineering, where PICASO is being employed as a method of identifying conditions of stability in hypothetical non-linear control systems. The technique employed, is to interpret the stability equation as a three-dimensional surface, draw this with PICASO and then identify the topological characteristics that reflect the stability of the system.

It is believed that this combination of activities is unique in this field, and that it could develop into a general method of systems analysis. Students studying control systems at Middlesex Polytechnic are to be taught PICASO in order that they might use it in this mode of analysis.
8 CONCLUSIONS

Although PICASO has only been operational for a few months, it is already implemented in one University and three Polytechnics, with the prospect of being adopted by every Polytechnic in Great Britain. It is also to be made available to every interested educational institution.

This immediate response reflects the success of PICASO, for no matter how sophisticated a graphics language might be, unless it can provide quality drawings for the non-technical user, it has failed. The art-work supplied with this thesis illustrates the wide range of graphic effects that can be achieved with PICASO. They have been produced by the author, who is not an artist, not for any artistic end, but simply to express the effectiveness of PICASO in various graphic modes.

PICASO has already been taught by the author to many types of students working in art and design, engineering and mathematics to establish any weaknesses in the language. Although minor adjustments have been made, the underlying structure has withstood this exacting and final method of testing computer packages.

PICASO has for the first time enabled the author to successfully teach computer graphics to the non-numerate student. Previous attempts had failed
due to technical problems raised by students, such as:

"What is a radian?"
"What is the distance between these two points?"
"Do these lines intersect?"
"How can I shade this shape?"
"How can I test this point to see if it is inside this shape?"

These problems and many more, destroyed the normal teaching continuity, and eventually displaced the objective of the course to such a distance, that this goal was never achieved.

The teaching methodology with PICASO is completely different. A course begins with a brief talk about computers and how they can be made to draw. The student is then introduced to essential concepts of the language such as numbers, variables, space and then shown how to code. He is then given the PICASO User Manual shown in Section 11, and then taken through some simple example programs. After this introduction, he writes his first program and this is normally after one hours tuition. Subsequent programs require very little effort as they have a consistent format.

Some time can be spent on two-dimensional work, but
the student often wishes to explore three-dimensions and surfaces; this demands an explanation of the 3-D spatial conventions and the significance of the picture plane. The student then continues to use PICASO in various projects where he develops his ideas in greater depth.

PICASO is an extensible language and already includes many additions proposed by students on previous courses. The ease with which this can be performed is essential to the continued use of PICASO, and the author is determined that PICASO will continue to enjoy its early success by supporting the software until some independent scheme can be arranged to maintain it.

PICASO has been designed to be used. The author now hopes that it will be used to design.
9.1 APPENDIX I

Summary of PICASO system commands.

EYE Establishes the position of the observer in the 3-D conceptual space.

FINISH Terminates plotting.

FISHI Establishes the position of a wide-angle observer in the 3-D conceptual space.

ORIGIN Establishes a new origin on the 2-D projection space.

PLANE Alters the distance between the picture plane and the observer.

START Initializes plotting.

WINDOW Creates a virtual window on the 2-D projection plane.
9.2 APPENDIX II

Summary of PICASO input/output commands.

ARRAY1 Reads from punched cards a REAL one-dimensional array.

ARRAY2 Reads from punched cards a REAL two-dimensional array.

ARRAY3 Reads from punched cards a REAL three-dimensional array.

IRRAY1 Reads from punched cards an INTEGER one-dimensional array.

IRRAY2 Reads from punched cards an INTEGER two-dimensional array.

IRRAY3 Reads from punched cards an INTEGER three-dimensional array.

LRRAY1 Reads from punched cards a LOGICAL one-dimensional array.

LRRAY2 Reads from punched cards a LOGICAL two-dimensional array.

LRRAY3 Reads from punched cards a LOGICAL three-dimensional array.

OBJECT Reads from punched cards a PICASO 3-D object.

PRINT Prints to a line-printer a PICASO structure.

PUNCH Punches onto punched cards a PICASO structure.

SHAPE Reads from punched cards a PICASO 2-D shape.
9.3 APPENDIX III

Summary of PICASO shapes.

AGNESI  Fermat's 'Witch of Agnesi' curve.
ARC    Circular arc.
ASPIRA Archimedean spiral.
ASTROI Astroid.
BUTFLY Butterfly.
CARDI  Cardioid.
CUBPAR Cubic parabola.
DLINE  Dashed line.
DELOTI Deltoid.
DOT    Dot.
ELLIPS Ellipse.
EPITRO Epitrochoid.
FACE   Human face.
GRILL  Regular grill of lines.
HORSE  Horse.
HYPERB Hyperbola.
HYPOTR Hypotrochoid.
LEFANT Elephant.
LINE   Line.
LISSAJ Lissajous figure.
LSPIRA Logarithmic spiral.
PARAB  Parabola.
POLYGN Regular polygon.
RECTNG Rectangle.
RHINO  Rhinosceros.
APPENDIX III Continued.

RHODON  Rhodon.
SHARK    Shark.
SINE     Sinusoid.
SQUARE   Square.
STAR     Regular star.
Summary of PICASO objects.

BOX Box.
CONE Cone.
CYLIND Cylinder.
DOT3D Dot.
GRIL3D Regular grill of lines.
LINE3D Line.
SPHERE Sphere.
9.5 APPENDIX V

Summary of PICASO structure manipulating commands.

ASMDSH  Adds, subtracts, multiplies and divides two PICASO STRUCTURES.

COPYSH  Makes a copy of a PICASO structure.

CYCLE   Moves vertices along lines to their nearest neighbour.

EXTSH   Extracts a contour or surface from a PICASO structure.

FIT     Adjusts a 2-D PICASO contour to fit between two points.

FORM3D  Creates a 3-D PICASO surface from a 2-D shape moulded to some contour.

JOIN    Augments a PICASO structure by another.

MATOP2  Performs a 2 by 2 matrix multiplication on a PICASO shape.

MATOP3  Performs a 3 by 3 matrix multiplication on a PICASO object.

MIRROR  Produces a mirror image of a PICASO shape.

MIX2D   Exchanges X and Y coordinates in a PICASO shape.

MIX3D   Exchanges X, Y and Z coordinates in a PICASO object.

NORMAL Adjusts a vertex to a specific position and makes corresponding adjustments to the rest for a PICASO shape.

NORM3D  Adjusts a vertex to a position and makes corresponding adjustments to the rest for PICASO object.
9.5 APPENDIX V Continued.

PERSHP Generates a PICASO shape from a perspective view of a PICASO object.

PULL Distorts a PICASO shape in a specified direction.

REMOVE Removes a contour or surface from a PICASO STRUCTURE.

REVERS Reverses the vertex sequence in a PICASO structure.

RIPPLE Alters the starting vertex for a PICASO CONTOUR OR SURFACE.

ROTATE Rotates a PICASO shape about a fixed point.

SHIFT Shifts a PICASO shape in a given direction.

SHIFT3 Shifts a PICASO object in a given direction.

SIZE Alters the size of a PICASO shape.

SIZE3 Alters the size of a PICASO object.

STICK Sticks two open PICASO contours together.

THICK Creates a 3-D extrusion of a PICASO shape.

THIN Creates a 2-D projection of a PICASO object.

TRACE Traces around a PICASO shape.

TRANSH Transforms one PICASO structure into another.

TRANSPO Transposes a PICASO shape into a 3-D PICASO SURFACE.

TURN3D Rotates a PICASO object about an axis.

WARP2D Subjects a PICASO shape to new laws of space.

WARP3D Subjects a PICASO object to new laws of space.
9.6 **APPENDIX VI**

Summary of PICASO plotting commands.

- **DRAW** Draws out a PICASO shape.
- **DRAW2D** Draws out a PICASO shape in perspective.
- **DRAW3D** Draws out a PICASO object.
- **DSHAPE** Draws out a dashed PICASO shape.
- **DSHLNE** Draws out a dashed line.
- **FILL** Draws out a tesselation of PICASO shapes.
- **FILL2D** Draws out a tesselation of PICASO shapes in perspective.
- **FILL3D** Draws out a tesselation of PICASO objects.
- **GRID** Draws out a regular grid of PICASO shapes.
- **GRID2D** Draws out a regular grid of PICASO shapes in perspective.
- **GRID3D** Draws out a regular grid of PICASO objects.
- **HATCH** Hatches in a PICASO shape with parallel lines.
- **MASK** Draws out a PICASO shape masked by another.
- **ROW** Draws out a row of PICASO shapes.
- **ROW2D** Draws out a row of PICASO shapes in perspective.
- **ROW3D** Draws out a row of PICASO objects.
- **PLIT** Moves the plotter pen to a new position.
- **SMOOTH** Draws out a PICASO shape with its vertices connected by a cubic spline curve.
- **STIPLE** Draws a specified number of random dots inside a PICASO shape.
9.7 **APPENDIX VII**

Summary of PICASO structure analysis commands.

**CLOCK** Determines whether a PICASO contour is clockwise or counter-clockwise.

**CLOK3D** Determines whether a PICASO surface is clockwise or counter-clockwise.

**CLOSE** Closes a PICASO contour.

**CLOSED** Determines whether a PICASO contour or surface is closed or open.

**CROSS** Determines whether two lines intersect and the nature of the intersection.

**HULL** Determines the rectangular hull of a PICASO shape.

**INSIDE** Determines whether a given point is in or outside a PICASO shape.

**NDIMEN** Determines the core requirement of a PICASO structure.

**NLINES** Determines the number of contours or surfaces in a PICASO structure.

**NPOINT** Determines the number of vertices in a PICASO contour or surface.

**NSUB** Determines the data structure position for the start of a PICASO contour or surface.

**NXSUB** Determines the data structure position for the X-coordinate of a given vertex.

**NYSUB** Determines the data structure position for the Y-coordinate of a given structure.
9.7 APPENDIX VII Continued.

NZSUB  Determines the data structure position for the Z-coordinate of a given vertex.
SENSE  Determines the spatial relationship between a point and a line.
SLOPE  Determines the slope of a line joining two vertices in a PICASO shape.
### 9.8 **APPENDIX VIII**

**Summary of PICASO functions.**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITAKE</td>
<td>Generates a random INTEGER number between two boundaries.</td>
</tr>
<tr>
<td>PERSP</td>
<td>Calculates the picture plane position of a point in 3-D space.</td>
</tr>
<tr>
<td>TAKE</td>
<td>Generates a random REAL number between two boundaries.</td>
</tr>
<tr>
<td>XARC</td>
<td>Returns the X-coordinate of a point on a circular arc.</td>
</tr>
<tr>
<td>XCOORD</td>
<td>Returns the X-coordinate of a given vertex.</td>
</tr>
<tr>
<td>XELLIP</td>
<td>Returns the X-coordinate of a point on an ellipse.</td>
</tr>
<tr>
<td>XEPIT</td>
<td>Returns the X-coordinate of a point on an epitrochoid.</td>
</tr>
<tr>
<td>XHYPER</td>
<td>Returns the X-coordinate of a point on a hyperbola.</td>
</tr>
<tr>
<td>XHYPOT</td>
<td>Returns the X-coordinate of a point on a hypotrochoid.</td>
</tr>
<tr>
<td>XLISS</td>
<td>Returns the X-coordinate of a point on a Lissajous figure.</td>
</tr>
<tr>
<td>XLSPIR</td>
<td>Returns the X-coordinate of a point on a logarithmic spiral.</td>
</tr>
<tr>
<td>XMIRRO</td>
<td>Returns the X-coordinate of the reflection of a point.</td>
</tr>
<tr>
<td>XPARAB</td>
<td>Returns the X-coordinate of a point on a parabola.</td>
</tr>
</tbody>
</table>
9.8 APPENDIX VIII Continued.

XPERSP Returns the X-coordinate of the picture plane position of a point in 3-D space.

XPOLY Returns the X-coordinate of a point on a regular polygon.

XROT Returns the X-coordinate of a point rotated about a given fixed point.

XSIN returns the X-coordinate of a point on a sinusoid.

XYVLIN Returns the X, Y or Z-coordinate of a point on a line.

YARC Returns the Y-coordinate of a point on a circular arc.

YCOORD Returns the Y-coordinate of a given vertex.

YELLIP Returns the Y-coordinate of a point on an ellipse.

YEPIT Returns the Y-coordinate of a point on an epitrochoid.

YHYPER Returns the Y-coordinate of a point on a hyperbola.

YHYPOT Returns the Y-coordinate of a point on a hypotrochoid.

YLISS Returns the Y-coordinate of a point on a Lissajous figure.

YLSPIR Returns the Y-coordinate of a point on a logarithmic spiral.

YMIRRO Returns the Y-coordinate of the point reflected about a line.
9.8 APPENDIX VIII Continued.

YPARAB  Returns the Y-coordinate of a point on a parabola.

YPERSP  Returns the Y-coordinate of the picture plane position of a point in 3-D space.

YPOLY   Returns the Y-coordinate of a point on a regular polygon.

YROT    Returns the Y-coordinate of a point rotated about a given fixed point.

YSIN    Returns the Y-coordinate of a point on a sinusoid.

ZCOORD  Returns the Z-coordinate of a given vertex.
9.9 APPENDIX IX

Summary of special effects.

CONECT Connects every vertex on a PICASO shape to every other vertex, with lines.

CONEC3 Connects every vertex on a PICASO object to every other vertex with lines.

GROW Draws a PICASO shape located at every vertex of itself to any depth.

GROW2D Draws a PICASO shape located at every vertex of itself to any depth, in perspective.

GROW3D Draws a PICASO object located at every vertex of itself to any depth.

MODSH Modulates the boundary of a PICASO shape by an open PICASO contour.

SNOW Transforms a PICASO shape into a crystalline form to any depth.
9.10 APPENDIX X

Summary of array handling commands.

FIND1  Searches a one-dimensional REAL array
for the occurrence of a value.

FIND2  Searches a two-dimensional REAL array
for the occurrence of a value.

FIND3  Searches a three-dimensional REAL array
for the occurrence of a value.

GET1   Returns a value from a one-dimensional
REAL array.

GET2   Returns a value from a two-dimensional
REAL array.

GET3   Returns a value from a three-dimensional
REAL array.

IFIND1 Searches a one-dimensional INTEGER array
for the occurrence of a value.

IFIND2 Searches a two-dimensional INTEGER array
for the occurrence of a value.

IFIND3 Searches a three-dimensional INTEGER array
for the occurrence of a value.

IGET1  Returns a value from a one-dimensional
INTEGER array.

IGET2  Returns a value from a two-dimensional
INTEGER array.

IGET3  Returns a value from a three-dimensional
INTEGER array.

IPUT1  Assigns a value to a one-dimensional
INTEGER array.
9.10 APPENDIX X Continued.

IPUT2 Assigns a value to a two-dimensional INTEGER array.

IPUT3 Assigns a value to a three-dimensional INTEGER array.

LPUT1 Assigns a value to a one-dimensional LOGICAL array.

LPUT2 Assigns a value to a two-dimensional LOGICAL array.

LPUT3 Assigns a value to a three-dimensional LOGICAL array.

PUT1 Assigns a value to a one-dimensional REAL array.

PUT2 Assigns a value to a two-dimensional REAL array.

PUT3 Assigns a value to a three-dimensional REAL array.
9.11 APPENDIX XI

Summary of PICASO surface commands.

HIDE Draws out a contour masked by the previous drawn contour.

ISOMET Produces an isometric projection of a surface generated by a rotated PICASO contour.

MAP2D Produces a 2-D contour map of vertical data stored in an array.

MAP3D Produces a 3-D contour map of vertical data stored in an array.

SILUET Produces a 3-D surface by spinning a PICASO contour.

SURFAC Produces a front-elevation view of a surface generated by rotating a PICASO contour.

SURF3D Realises a mesh type perspective surface from an array containing vertical data.
9.12 APPENDIX XII

PICASO Data structure

```
2.0

X&Y Coordinate
Vertex count
Contour count
Dimension

3.0

X,Y&Z Coordinate
Vertex count
Surface count
Dimension
```
9.13 APPENDIX XIII

Windowing algorithm.

The function SENSE determines whether a point is to the left, coincident or to the right of a given line segment. Using this function it is possible to detect if two lines intersect and the nature of the intersection.

As PICASO uses a convention that clockwise contours create transparent windows and counter-clockwise opaque, the function CROSS can return the mode of intersection when a line segment is tested against the windowing shape. A line intersecting a clockwise contour is -ve, and a counter-clockwise is +ve. Thus when all the intersections have been detected, they are sorted using a binary tree sort, and the plotter pen lowered for each -ve intersection and raised for each +ve value as the line is drawn.

Figure 9.13.1 illustrates a complex window and the intersections a line segment makes.
Figure 9.13.1
The conventional procedure of using two vectors is employed to maintain the current upper and lower horizons, but a space domain technique achieves hidden-line removal.

The picture plane is divided into the six domains shown in Figure 9.14.1, and they are defined as follows:

1) the domain above the upper-horizon
2) the domain below the lower-horizon
3) the domain between the two horizons
4) the domain where the two horizons meet
5) the domain on the upper-horizon
6) the domain on the lower-horizon.
Any line-segment to be drawn must either stay in one domain or cross to another, therefore a state table can describe the action to take under any condition. For example, a move from domain one to two requires the following actions:

a) calculate the intersection with the upper-horizon and draw this line segment,
b) calculate the intersection with the lower-horizon and ignore this segment,
c) draw the remaining line segment and update the lower horizon.

To ensure that the surface is completely opaque, it is created and drawn in small rectangular tiles as shown in Figure 5.9.3.
10 REFERENCES

1 Reichardt, J.
The Computer in Art, Page 8
Studio Vista,
London

2 Association of Computer Units in Colleges
of Higher Education,
CHESS Working Party,
North Staffs. Polytechnic,
Blackheath Lane,
Stafford

3 Thomas, R.E.
Atlas Computer Laboratory,
Didcot,
Berkshire OX11 OQY

4 George, J.E.
GEMS A Graphical Meta System
Department of Computer Science,
Colorado State University,
Colorado, USA

5 Bork, A.M.
APL for the SIGMA
XEROX User Group Proceedings (Dec 7 81972)
XEROX
London

6 Armit, A.P.
MULTIPATCH, A Language for interactive Design
Computer Aided Design Autumn 1971 Page 10

7 Narasimham, R. & Reddy, V.S.N.
Some Experiments in Scene Generation using COMPAX
Tata Institute of Fundamental Research,
IFIP Working Conference on Graphical Languages,
North-Holland Press

8 Licklider, J.C.R.
A Picture is worth a thousand Words... and it costs.
SJCC 1969
AFIPS Press,
Montvale,
New-Jersey, 617

9 DISSPLA (Display Integrated Software System and
Plotting Language)
Integrated Software Systems Corporation,
San Diego,
California, USA
10 Roberts, L.G.  
Machine Perception of Three Dimensional Solids  
Technical Report 315 (may 1963)  
MIT Lincoln Laboratory,  
Cambridge, USA

11 Warnock, J.E.  
A Hidden Surface Algorithm for Computer  
Generated Half-tone Pictures  
Computer Science Department,  
University of Utah,  
Salt Lake City, USA

12 Loutrel, P.P.  
A Solution to the Hidden line Problem for  
Computer Drawn Polyhedra,  
IEEE Transactions on Computers,  
Vol C-19 No 3 March 1970 Page 205

13 Matsushita, Y.  
Hidden line Elimination from a rotating object,  
CACM Vol 15 No 4 Page 245 April 1972

14 Knowlton, K.  
Graphic Languages,  
Proceedings of the Working Conference on  
Graphic Languages (1972)  
North-Holland Press

15 Knowlton, K. & Schwartz  
Films "PIXILLATIONS & UFO'S" (1970 1971)  
University of California,  
Department of Computer Science,  
Santa-Cruz, USA

16 Bush, D.J. & Bell, M.  
SPLINT, ACubic Spline Package (1974)  
Middlesex Polytechnic,  
Computer Centre,  
Queensway,  
Enfield,  
Middlesex

17 Csuri, C.  
The Computer in Art, Page 40  
Jasia Reichardt,  
Studio Vista,  
London
EYE
FINISH
FISHI
ORIGIN
START
WINDOW
FACTOR
PICTASO SYSTEM

PLOTTING COMMANDS

DRAW
DRAW2D
DRAW3D
DSHAPE
DSHLNE
FILL
FILL2D
GRID
GRID2D
HATCH
HIDE
ISOMET
MASK
PLIT
ROW
ROW2D
SHADE
SILUET
SMOOTH
STIPLE
SURFAC
AGNESI
ARC
ASPIRA
ASTROI
CARDI
CUBPAR
DELTOI
DOT
ELLIPS
EPITRO
GRILL
HYPERB
HYPOTR
LINE
LINE3D
LISSAJ
LSPIRA
PARAB
POLYGN
RECTNG
RHODON
SINE
SQUARE
STAR

(1) 26.03.75
BOX
CUBE
PYRAMID
COPYSH
EXPLOD
EXTSH
FIT
FORM3D
JOIN
MIRROR
MIX2D
MIX3D
NORMAL
PERSHP
PULL
REMOVE
REVERS
RIPPLE
ROTATE
SHIFT
SHIFT3
SIZE
STICK
TRACE
TRANSHE
TRANSP
TURN3D
CLOCK
CLOK3D
CLOSED
CROSS
HULL
INSIDE
NDIMEN
NLINES
NPOINT
NSUB
NXSUB
NYSUB
NZSUB
SENSE
SLOPE

(1)28.03.75
Picaso System Functions

ITAKE
PERSP
POSCON
TAKE
XARC
XCOORD
XELLIP
XEPIT
XHYPER
XHYPOT
XLISS
XLSPIR
XMIRRO
XPARAB
XPERSP
XPOLY
XROT
XSIN
XYZLIN
YARC
YCOORD
YEPIT
YELLIP
YHYPER
YHYPOT
YLISS
YLSPIR
YMIRRO
YPARAB
YPERSPS
YPOLY
YROT
YSIN
ZCOORD

(1)28.03.75
CONECT
CYCLE
GROW
GROW2D
MODSH
SNOW
NAME AGNESI (Witch of Agnesi)

FUNCTION Stores Fermat's Witch of Agnesi curve in an array

ARGUMENTS (ARRAY, XSPAN, YSPAN, N)

ARRAY (Real array) stores the 2-d Picaso Agnesi curve
XSPAN (Real inches) is the width of the curve
YSPAN (Real inches) is the height of the curve
N (Integer) is the number of steps in the curve

EXAMPLE CALL AGNESI(A, 5.0, 5.0, 20)
This statement stores in A the Witch of Agnesi curve 5.0 inches wide and 5.0 inches high, in 20 steps

NOTES 1. No plotting takes place

(1) 26.03.75
NAME ARC

FUNCTION Stores a circular Arc in array

ARGUMENTS (ARRAY, X, Y, RADIUS, THETA, PHI, N)
  ARRAY (Real array) stores the Arc
  X, Y (Real inches) is the centre of revolution
  RADIUS (Real inches is the radius of curvature
  THETA (Real degrees) is the starting angle of the Arc
  PHI (Real degrees) is the angle of rotation in a clockwise sense
  N (Integer) is the number of steps to perform the rotation

EXAMPLE CALL ARC(S, 0.0, 0.0, 5.0, 90.0, 90.0, 20)
  This statement stores an Arc in S with the centre of revolution at the origin and radius 5.0 inches. The starting angle is 90.0° and the rotated angle is 90.0°.

NOTES 1. No plotting takes place

(1) 26.03.75
ARRAY1 (One Dimension Array)

FUNCTION Reads in punched card data and stores it in a One Dimension Array

ARGUMENTS (ARRAY, COL)

ARRAY (Real array) is the name of the array
COL (Integer) is the number of columns in the array

SAMPLE

CALL ARRAY1(B, 20)

This statement reads in sufficient cards to store 20 numbers in B.

1. The format of the cards is (10F8.0) therefore a decimal point should be punched in the data. A number must be punched within its allocated 8 columns.
2. For the above example 2 cards are required but partially punched cards are accepted as the final card.
NAME    ARRAY2 (Two Dimension Array)

FUNCTION Reads in punched card data and stores it in a Two Dimension Array

ARGUMENTS (ARRAY, COL, ROW)

ARRAY (Real array) is the name of the array
COL (Integer) is the number of columns in the array
ROW (Integer) is the number of rows in the array

EXAMPLE CALL ARRAY2(B, 18, 2)
This statement reads in sufficient cards to store 36 numbers in B.

NOTES
1. The format of the cards is (10F8.0) therefore a decimal point should be punched in the data.
   A number must be punched within its allocated 8 columns.
2. For the above example 4 cards are required.

(1)26.03.75
NAME ASPIRA (Archimedean Spiral)

FUNCTION Stores an Archimedean Spiral in array

ARGUMENTS (ARRAY,RADIUS,CYCLES,N)

ARRAY (Real array) stores the spiral
RADIUS (Real inches) is the final radius of the spiral
CYCLES (Real) is the number of rotations in the spiral
N (Integer) is the number of points in the curve

EXAMPLE CALL ASPIRA(S, 4.0, 2.0, 60)
This statement stores an archimedean spiral in the array S with 2.0 cycles and a final radius of 4.0 inches.

NOTES 1. No plotting takes place
NAME ASTROI (Astroid Roemer 1674)

FUNCTION Stores an astroid in array

ARGUMENTS (ARRAY, WIDTH, N)
ARRAY (Real array) stores the astroid
WIDTH (Real inches) is the maximum radius of the astroid
N (Integer) is the number of points in the curve

EXAMPLE CALL ASTROI(A, 2.0, 60)
This statement stores an astroid of 2.0 inches radius in A.

NOTES 1. No plotting takes place

(1) 26.03.75
PICASO SYSTEM

NAME            BOX

FUNCTION Stores a 3-d *Picaso* box in array

ARGUMENTS (ARRAY, LENGTH, HEIGHT, WIDTH)

ARRAY (Real array) stores the 3-d *Picaso* box
LENGTH (Real inches) is the X-length of the box
HEIGHT (Real inches) is the Y-height of the box
WIDTH (Real inches) is the Z-depth of the box

EXAMPLE CALL BOX(B, 2.0, 1.0, 1.0)
This statement stores a box in B 2.0 inches long 1.0 inch high and 1.0 inch deep

NOTES
1. No plotting takes place
2. Array must be dimensioned (98)
NAME CARDI (Cardioid)

FUNCTION Stores a cardioid curve in an array

ARGUMENTS (ARRAY, XSPAN, N)
- ARRAY (Real array) stores the 2-d Picaso cardioid curve
- XSPAN (Real inches) is the width of the cardioid from the origin to the right-hand end intersection with the X-axis
- N (Integer) the number of steps in the curve

EXAMPLE CALL CARDI(C, 4.0, 50)
This statement stores in array C a cardioid curve 4.0 inches wide in 50 steps

NOTES 1. No plotting takes place

(1) 26.03.75
NAME         CLOCK (Clockwise)

FUNCTION    Determines whether a 2-d Picaso contour is clockwise or anti-clockwise

ARGUMENTS   (ARRAY, N)
            ARRAY    (Real array) stores a 2-d Picaso shape
            N       (Integer) is the number of the contour

EXAMPLE     I=CLOCK(BOX, 1)
            This statement sets I.TRUE. if the first contour of box is clockwise else I is .FALSE.

NOTES        1. Clock is a logical function

(1) 26.03.75
NAME  
CLOK3D (Clockwise 3-d)

FUNCTION  
Determines whether a 3-d Picasso contour is clockwise or anti-clockwise

ARGUMENTS (ARRAY, N, SIZE, X, Y, Z)

ARRAY  
(Real array) stores a 3-d Picasso object

N  
(Integer) references the contour

SIZE  
(Real) is the degree of enlargement or reduction

X, Y, Z  
(Real) is the amount of shift applied to the object

EXAMPLE  
I = CLOK3D (BOX, 1, 1.0, 0.0, 0.0, 0.0, 0.0)
This statement sets I .TRUE. if the 1st contour of box is clockwise else I is set .FALSE..

NOTES  
1. Clok3d is a logical function
NAME  CLOSED

FUNCTION  Determines whether a 2-d Picaso contour is open or closed

ARGUMENTS  \((ARRAY, NC)\)

\(ARRAY\)  (Real array) stores a 2-d Picaso shape

\(NC\)  (Integer) is the number of the contour

EXAMPLE  \(J = \text{CLOSED(BOX1)}\)

\(J\) is set .TRUE. if the first contour of box is closed, else \(J\) is .FALSE.

(1) 26.03.75
NAME CONECT (Connect)

FUNCTION Draws out a 2-d Picaso shape and connects every vertex with every other vertex with lines

ARGUMENTS (ARRAY, SIZE, XS, YS)
- ARRAY (Real array) stores a 2-d Picaso shape
- SIZE (Real) is the degree of enlargement or reduction of the drawn shape relative to the original stored in array
- XS, YS (Real inches) is the degree of shift applied to the shape

EXAMPLE CALL CONECT(HEX, 1.0, 0.0, 5.0)
This statement draws out the shape shown below, assuming that $\text{hex}$ stores a hexagon

NOTES 1. The Picaso shape may be open or closed

(1)26.03.75
NAME COPYSH (Copy Shape)

FUNCTION Makes a copy of a Picaso shape

ARGUMENTS (ARRAY1, ARRAY2)

ARRAY1 (Real array) stores a Picaso shape or object
ARRAY2 (Real array) stores a copy of the shape or object stored in ARRAY1.

EXAMPLE CALL COPYSH(A, B)
This statement copies the shape or object stored in A into B.

NOTES
1. No plotting takes place

(1)26.03.75
NAME  CROSS

FUNCTION  Determines whether two lines cross, and the nature and type of intersection

ARGUMENTS  \((X_1,Y_1,X_2,Y_2,X_3,Y_3,X_4,Y_4,X_I,Y_I)\)

\(X_1,Y_1\) (Real) is the first point on the reference line
\(X_2,Y_2\) (Real) is the last point on the reference line
\(X_3,Y_3\) (Real) is the first point on the test line
\(X_4,Y_4\) (Real) is the last point on the test line
\(X_I,Y_I\) (Real) is the point of intersection if the lines cross or touch

EXAMPLE  \(I=CROSS(0.0,0.0,1.0,1.0,X_A,Y_A,X_B,Y_B,X_I,Y_I)\)

This statement determines if the given two lines intersect

NOTES  1. The following illustrate the type of intersections

[Diagrams of different types of intersections]

2. Cross is an integer type function

(1) 26.03.75
NAME CUBE

FUNCTION Stores a regular cube in a *Picaso* array

ARGUMENTS (ARRAY, SIDE)
- Array (Real array) receives the cube
- Side (Real inches) is the size of the cube

EXAMPLE CALL CUBE(C, 1.0)
- This statement stores a *cube* size 1.0" in C
  - Array C must be dimensioned C(98)

NOTES
1. All surfaces are clockwise
2. The lower left-hand corner is on the origin

![Diagram of a cube in a 3D coordinate system]

(1) 26.03.75
NAME CUBPAR (Cubic Parabola)

FUNCTION Stores a cubic parabola in array

ARGUMENTS (ARRAY, XSPAN, YSPAN, N)
- ARRAY (Real array) stores the cubic parabola
- XSPAN (Real inches) is the X-width of the curve from the origin
- YSPAN (Real inches) is the height of the curve
- N (Integer) is the number of points in the curve

EXAMPLE CALL CUBPAR(C., 2.0, 2.0, 20)
This statement stores a cubical parabola in C 4.0 inches wide 2.0 inches high

NOTES 1. No plotting takes place

(1)26.03.75
NAME  CYCLE (Re-cycle coordinates)

FUNCTION Takes every point defining a 2-dimensional Picaso shape and moves it clockwise or anti-clockwise towards its nearest neighbour

ARGUMENTS (ARRAY,K,F,ARRAY1)

ARRAY  (Real array) stores a 2-dimensional Picaso shape
K      (Integer) VE = Clockwise
       -VE = Anti-clockwise
F      (Real) is the degree of rotation minimum value = 0.0 max = 1.0
ARRAY1 (Real array) stores the cycled shape

EXAMPLE  CALL CYCLE(SQUARE,1,0.5,SQUARE)
          This statement takes the shape stored in square and rotates the points clockwise by 50%. The cycled shape is re-stored in square.

NOTES  1. No plotting takes place

(1)26.03.75
NAME DELTOI (Deltoid)

FUNCTION Stores a deltoid curve in array

ARGUMENTS (ARRAY, RADIUS, N)
- ARRAY (Real array) stores the deltoid curve
- RADIUS (Real inches) is the maximum radius of the deltoid
- N (Integer) is the number of points in the curve

EXAMPLE CALL DELTOI(D, 1.0, 50)
This statement stores a Deltoid in D of radius 1.0 using 50 points

NOTES 1. No plotting takes place
NAME DOT

FUNCTION Stores the coordinates of a point

ARGUMENTS (ARRAY, X, Y)

ARRAY (Real array) stores the dot
X, Y (Real inches) is the dot location

EXAMPLE CALL DOT(D, 5.0, 5.0)

This statement stores the position of a dot in D.

NOTES 1. No plotting takes place
NAME    DRAW

FUNCTION Plots a 2-dimensional *Picaso* shape

ARGUMENTS \( (ARRAY, SIZE, X, Y) \)

- **ARRAY** (Real array) stores a 2-d *Picaso* shape
- **SIZE** (Real) is the degree of enlargement or reduction of the drawn shape relative to the original stored in array
- **X, Y** (Real inches) is the amount of shift applied to the drawn shape

EXAMPLE  \( \text{CALL DRAW(BOX, 2.0, 5.0, 5.0)} \)

This statement draws out the shape stored in *box* twice its original size displaced 5.0 inches along the X-axis and 5.0 inches along the Y-axis.

\[\text{\includegraphics[width=0.5\textwidth]{box.png}}\]

NOTES  1. All size changes are made relative to the origin, for example the point \((1.0, 1.0)\) with a size value of 2.0 is transformed to \((2.0, 2.0)\).
NAME 
DRAW2D (Draw a 2-d shape in 3-d)

FUNCTION 
Draws out a 3-d version of a 2-d Picaso shape by transposing coordinates

ARGUMENTS  
(ARRAY, SIZE, X, Y, N1, N2, N3, V)

ARRAY  
(Real array) stores a 2-d Picaso shape

SIZE  
(Real) is the degree of enlargement or reduction

X, Y  
(Real) is the degree of shift applied to the 2-d shape

N1, N2, N3  
(Integer) determine the transposition

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>IN 3-D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>X</td>
<td>Y</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>X</td>
<td>V</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>Y</td>
<td>X</td>
<td>V</td>
<td>IN 2-D</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>V</td>
<td>X</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>Y</td>
<td>V</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>V</td>
<td>Y</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

V  
(Real) is the value for the third dimension

EXAMPLE 
CALL DRAW2D(A, 1.0, 0.0, 0.0, 3, 1, 2, 2.0)

This statement produces the following transformation

NOTES
1. The position of the eye is set by 'eye'

(1) 26.03.75
NAME  DRAW3D (Draw a 3-d Object)

FUNCTION Draws out a 3-d view of a 3-d *Picaso* object

ARGUMENTS  \((ARRAY, SIZE, X, Y, Z, L)\)
- **ARRAY** (Real array) stores a 3-d *Picaso* object
- **SIZE** (Real) is the degree of enlargement or reduction
- **X, Y, Z** (Real) is the degree of shift applied to the object
- **L** (Integer) if L is +VE all lines are drawn. If L is -VE some hidden-line removal occurs

EXAMPLE  CALL DRAW3D(BOX, 1.0, 0.0, 0.0, 0.0, 0.0, -1)

This statement draws out the object stored in *box* with hidden-line removal

NOTES  The position of the eye is set up by 'eye' or 'fishi'

\(126.03.75\)
NAME

DSHAPE (Dash Shape)

FUNCTION

Draws out the 2-dimensional *Picaso* shape with broken lines

ARGUMENTS *(ARRAY, SIZE, X, Y, DASH)*

- **ARRAY** (Real array) stores a 2-d *Picaso* shape
- **SIZE** (Real) is the degree of enlargement or reduction of the drawn shape relative to the original stored in array
- **X, Y** (Real inches) is the amount of shift applied to the drawn shape
- **DASH** (Real inches) is the length of the dash

EXAMPLE

CALL DSHAPE(BOX, 1.0, 0.0, 0.0, 0.0, 0.1)

This statement draws out the shape stored in box at its original size with zero displacement. The broken line is 0.1 inch.
NAME DSHLNE (Dashed Line)

FUNCTION Draws a broken line between two points

ARGUMENTS \((X_1, Y_1, X_2, Y_2, DASH)\)
\(X_1, Y_1\) (Real inches) coordinate of 1st point
\(X_2, Y_2\) (Real inches) coordinate of 2nd point
\(DASH\) (Real inches) is the length of the broken line

EXAMPLE CALL DSHLNE \((0.0, 0.0, 4.0, 5.0, 0.25)\)
This statement draws a broken line in lengths ¼ inch long between the origin and the point \((4.0, 5.0)\).
NAME ELLIPS (Ellipse)

FUNCTION Stores an ellipse in array

ARGUMENT (ARRAY, MAJOR, MINOR, N)
  ARRAY (Real array) stores the ellipse
  MAJOR (Real inches) is the major radius of the ellipse
  MINOR (Real inches) is the minor radius of the ellipse
  N (Integer) is the number of points on the curve

EXAMPLE CALL ELLIPS(E, 4.0, 2.0, 60)
  This statement stores an Ellipse in E with diameters 8.0 inches by 4.0 inches

NOTES 1. No plotting takes place

(1) 26.03.75
NAME EPITRO (Epitrochoid)

FUNCTION Stores a epitrochoid in array

ARGUMENTS (ARRAY,RADFIX,RADMOV,DIST,N)
- ARRAY (Real array) stores the 2-d epitrochoid
- RADFIX (Real inches) is the radius of the fixed circle
- RADMOV (Real inches) is the radius of the moving circle
- DIST (Real inches) is the distance of the traced point that exists on the moving circle to its centre
- N (Integer) is the number of points on the curve

EXAMPLE CALL EPITRO(E,2.0,1.0,1.0,60)
This statement represents a 1.0" radius circle moving around a fixed 2.0" radius circle. The point traced is 1.0" from the centre.

(1)26.03.75
NAME  
EXPLOD (Explode)

FUNCTION  
Adjusts the size of a shape relative to a specified point

ARGUMENTS  
(ARRAY, XP, YP, F, ARRAY2)

- ARRAY  (Real array) stores the two-dimensional Picaso shape
- XP, YP  (Real inches) is the centre of the explosion
- F  (Real) is the degree of size change
- ARRAY2  (Real array) stores the exploded shape

EXAMPLE  
CALL EXPLOD(BOX, 0.0, 0.0, 2.0, BOX)

This statement doubles the size of the shape stored in box about the origin

![Diagram of a shape before and after explosion]

NOTES  
1. No plotting takes place

(1) 26.03.75
NAME EXTSH (Extract shape)

FUNCTION Extracts a contour from a 2-dimensional Picaso shape

ARGUMENTS (ARRAY, NC, CONTOR)

ARRAY (Real array) stores the 2-d Picaso shape
NC (Integer) references the contour to be extracted
CONTOR (Real array) receives the extracted contour

EXAMPLE CALL EXTSH(BOX, 1, C)
This statement extracts the first contour of box and stores it in C.

NOTES 1. No plotting takes place
NAME EYE

FUNCTION Establishes the position of the eye and the point under observation

ARGUMENTS \((XE, YE, ZE, X, Y, Z)\)

\(XE, YE, ZE\) (Real inches) are the coordinates of the eye in 3-d space

\(X, Y, Z\) (Real inches) are the coordinates of the point in space the eye is looking towards

EXAMPLE CALL EYE\((0.0, 0.0, 0.0, 0.0, 0.0, 5.0, 10.0)\)

This statement places the eye at the origin and points it towards the point specified

NOTES

1. Eye must be called before calling any 3-d subroutine
2. The picture plane is set 10.0 inches from the eye
3. Eye may be called many times
4. The point the eye is looking at is centered on the origin

(1)26.03.75
NAME FILL

FUNCTION Draws a regular grid of 2-d Picasso shapes allowing for alternate horizontal and vertical shift to be applied

ARGUMENTS (ARRAY, SIZE, NX, XSP, XSH, NY, YSP, YSH, X, Y)

ARRAY (Real array) stores a 2-d Picasso shape
SIZE (Real) is the degree of enlargement or reduction of the drawn shape relative to the original stored in array
NX (Integer) is the number of shapes in the 'X' direction
XSP (Real inches) is the distance between points on two shapes in the 'X' direction
XSH (Real inches) is the amount of shift applied to even numbered rows
NY (Integer) is the number of shapes in the 'Y' direction
YSP (Real inches) is the distance between points on two shapes in the 'Y' direction
YSH (Real inches) is the amount of shift applied to even numbered columns
X, Y (Real inches) is the amount of shift applied to the entire grid

EXAMPLE CALL FILL(HEX, 1.0, 3, 1.50, 0.0, 2, 1.732, 0.5, 0.0, 0.0)

This statement produces a tesselated space of Hexagons (assuming Hex stored a Hexagon)
NAME       FILL2D (Draw a 2-d grid in 3-d)

FUNCTION  Draws out a three-dimensional version of a 2-d
           Picasso shape

ARGUMENTS (ARRAY, SIZE, X, Y, NX, XSP, XSH, NY, YSP, YSH, N1, N2, N3, V)

 ARRAY  (Real array) stores a 2-d Picasso shape
 SIZE   (Real) is the size factor
 X, Y   (Real inches) is the degree of shift applied
to the 2-d shape
 NX     (Integer) is the number of shapes in the 'x'
direction
 XSP    (Real inches) is the distance between points
        on two shapes in the 'x' direction
 XSH    (Real inches) is the amount of shift applied
to even numbered rows
 NY     (Integer) is the number of shapes in the 'y'
direction
 YSP    (Real inches) is the distance between points
        on two shapes in the 'y' direction
 YSH    (Real inches) is the amount of shift applied
to even numbered columns
 N1, N2, N3   (Integer) determine the transformation

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>IN 3-D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>X</td>
<td>Y</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>X</td>
<td>V</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>Y</td>
<td>X</td>
<td>V</td>
<td>IN 2-D</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>V</td>
<td>X</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>Y</td>
<td>V</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>V</td>
<td>Y</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

V  (Real) is the value for the third dimension

(1)26.03.75
EXAMPLE  

CALL FILL2D(HEX, 1.0, 0.0, 0.0, 0.3, 1.5, 0.0, 2, 1.732, 0.5, 1, 2, 3, 2.0)

This statement produces the following transformation:

NOTES  

1. The position of the eye must be specified by a call on eye or fishi

(1) 26.03.75
NAME    FINISH

FUNCTION Enables plotting to be completed and moves the pen beyond the drawing to prevent it being overdrawn by the next program

ARGUMENTS (X) 
  X  (Real inches) is the distance moved by the pen after completing the drawing

EXAMPLE  CALL FINISH(10.0)
         Completes plotting and moves the pen 10.0" along the X-axis relative to the origin

NOTES   1. The distance moved is relative to the origin

(1)26.03.75
NAME FISHI (Fish-eye)

FUNCTION Establishes the position of the eye and the radius of a wide-angle lens and the point under observation

ARGUMENTS \((XE, YE, ZE, X, Y, Z, RADIUS)\)

- \(XE, YE, ZE\) (Real inches) are the coordinates of the eye in 3-d space
- \(X, Y, Z\) (Real inches) are the coordinates of the point in space the eye is looking towards
- \(RADIUS\) (Real inches) is the radius of the lens

EXAMPLE CALL FISHI(0.0,0.0,0.0,0.0,0.0,5.0,10.0,1.0)

This statement places the eye and a wide-angle lens of radius 1.0" at the origin and points it towards the point specified

NOTES

1. Fishi must be called to establish the wide-angle lens
2. The picture plane is set 10.0" from the eye
3. Fishi may be called many times
4. The point the eye is looking towards is centered on the origin

(1)26.03.75
NAME  GRID

FUNCTION  Plots out a regular grid of 2-dimensional shapes

ARGUMENTS  (ARRAY, SIZE, NX, XSP, NY, YSP, X, Y)

ARRAY  (Real array) stores a 2-d shape
SIZE  (Real) is the degree of enlargement or reduction of the drawn shape relative to the original stored in array
NX  (Integer) is the number of shapes to be drawn in the X-direction
XSP  (Real inches) is the horizontal displacement between two shapes
NY  (Integer) is the number of shapes to be drawn in the Y-direction
YSP  (Real inches) is the vertical displacement between two shapes
X, Y  (Real inches) is the amount of shift applied to the grid of shapes

EXAMPLE  CALL GRID(SQUARE, 1.0, 3, 1.2, 2, 1.2, 0.0, 0.0)

Draws out the shape stored in SQUARE 6 times at the original size in a grid format 3 by 2.
The entire grid is displaced zero inches from the origin.

NOTES  1. Observe that XSP and YSP are not distances between two shapes, but are distances between the same point on displaced shapes
NAME FIT

FUNCTION Adjusts a 2-d Picaso open contour such that the end points are located at specific positions in space

ARGUMENTS (ARRAY, X1, Y1, X2, Y2, F, ARRAYA)
- ARRAY (Real array) stores the 2-d Picaso contour
- X1, Y1 (Real inches) are the coordinates of the first point
- X2, Y2 (Real inches) are the coordinates of the second point
- F (Real) is a multiplication factor effecting the amplitude of the contour
- ARRAYA (Real array) stores the adjusted contour

EXAMPLE CALL FIT(A, 0.0, 0.0, 1.0, 1.0, 1.0, A)

This statement takes the curve stored in A and adjusts the coordinates such the 1st and last coincide with the origin and 1.0, 1.0.

NOTES 1. No plotting takes place

(1) 26.03.75
GRID2D (Draw a 2-d grid in 3-d)

Draws out a three-dimensional version of a 2-d *Picaso* grid

**ARGUMENTS**

(ARRAY, SIZE, X, Y, NX, XSP, NY, YSP, N1, N2, N3, V)

- **ARRAY** (Real array) stores a 2-d *Picaso* shape
- **SIZE** (Real) alters the size of the final drawing by 'size' times
- **X, Y** (Real inches) is the degree of shift applied to the 2-d grid
- **NX** (Integer) is the number of shapes in the 'x' direction
- **XSP** (Real inches) is the distance between points on two shapes in the 'x' direction
- **NY** (Integer) is the number of shapes in the 'y' direction
- **YSP** (Real inches) is the distance between points on two shapes in the 'y' direction
- **N1, N2, N3** (Integer) determine the transformation

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>IN 3-D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>X</td>
<td>Y</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>X</td>
<td>V</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>Y</td>
<td>X</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>V</td>
<td>X</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>Y</td>
<td>V</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>V</td>
<td>Y</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

**V** (Real) is the value for the third dimension

**EXAMPLE**

CALL GRID2D(SQ, 1.0, 0.0, 0.0, 4, 1.0, 3, 1.0, 1.3, 2, 1.0)

This statement produces the following transformation

(1) 26.03.75
NOTES 1. The position of the eye must be specified by a call on eye or fishi.
NAME GROW (Recursive growth)

FUNCTION Draws out a recursive growth pattern using vertices of any shape as the centre of growth for more shapes

ARGUMENTS (ARRAY, SIZE, X, Y, N, STACK)

ARRAY (Real array) stores a 2-d Picaso shape
SIZE (Real) alters the size of the final drawing by 'size' times
X, Y (Real inches) is the degree of shift applied to the entire structure
N (Integer) is the depth of growth
STACK (Real array) is an array supplied to grow to enable the recursive growth to be computed

EXAMPLE CALL GROW(HEX, 0.0, 0.0, 3, STACK)
This statement produces the following output assuming that hex contains a regular hexagon

NOTES 1. The array stack must be dimension 3*DEPTH*NV
where NV is the number of vertices in the shape

(1)26.03.75
NAME  GROW2D (Grow a 2-d shape in 3-d)

FUNCTION  Draws out a recursive growth pattern using vertices of any shape as the centre of growth for more shapes in 3-d.

ARGUMENTS  (ARRAY, SIZE, X, Y, N, N1, N2, N3, V, STACK)

ARRAY  (Real array) stores a 2-d Picaso shape
SIZE  (Real) alters the size of the final shape
X, Y  (Real is the degree of shift applied to the 2-d shape
N  (Integer) is the depth of growth
N1, N2, (Integer) determine the transposition (see N3 grow)
V  (Real) is the value for the third dimension
STACK  (Real array) is a work array required by Grow2d and is dimensioned:
STACK(3*N*NV) where NV is the number of vertices in array

EXAMPLE  CALL GROW2D(A, 1.0, 0.0, 0.0, 3, 1, 3, 2, 1.0, STACK)

This statement allows the shape in 'A' to grow to a depth of 3 one inch above the X, Z plane

NOTES  1. The position of the eye must be specified by calling eye or fishi
NAME HATCH

FUNCTION Hatches a 2-d *Picasso* shape with parallel lines

ARGUMENTS (ARRAY, THETA, SPACE)

ARRAY (Real array) stores a 2-d *Picasso* shape

THETA (Real degrees) is the angle of the lines

SPACE (Real inches) is the distance between lines

EXAMPLE CALL HATCH(A, 45.0, 0.1)

This statement draws parallel lines at 45°
0.1" apart over the shape stored in

![Diagram of hatched shape]

NOTES 1. Clockwise contours create a shape, holes are created by anti-clockwise contours

(1) 26.03.75
**NAME**  
HIDE

**FUNCTION**  
Draws out the contents of an array depending upon the contents of two arrays storing upper and lower contours

**ARGUMENTS**  
(Array, Over, Under, N, Nlines, Width, Dist, Xs, Ys)

- **Array** (Real array) stores a 2-d Picaso contour
- **Over** (Real array) is used by hide to maintain the upper contour
- **Under** (Real array) is used by hide to maintain the lower contour
- **N** (Integer) is the number of elements in the array (also over & under)
- **Nlines** (Integer) is the number of the line being drawn
- **Width** (Real inches) is the length of the line
- **Dist** (Real inches) is the distance between lines
- **Xs, Ys** (Real inches) is the degree of shift applied to the line

**EXAMPLE**  
CALL HIDE(Array, Over, Under, 3, 20, 5.0, 0.1, 0.0, 0.0)

This statement could produce the output shown below

![Diagram of lines drawn by the Hide function](image)

**NOTES**  
1. Array, over & under must be dimensioned

(1) 28.03.75
NAME HULL (Rectangular Hull)

FUNCTION Determines the minimum and maximum X and Y values of a 2-dimensional Picaso shape

ARGUMENTS (ARRAY, XMIN, XMAX, YMIN, YMAX)

ARRAY (Real array) stores 2-d Picaso shape
XMIN (Real inches) minimum X coordinate
XMAX (Real inches) maximum Y coordinate
YMIN (Real inches) minimum Y coordinate
YMAX (Real inches) maximum Y coordinate

EXAMPLE CALL HULL(BOX, XMIN, XMAX, YMIN, YMAX)
This statement returns values of XMIN, XMAX, YMIN & YMAX for the shape stored in Box

NOTES 1. No plotting takes place

(1)26.03.75
NAME HYPERB (Hyperbola)

FUNCTION Stores a Hyperbola in array

ARGUMENTS (ARRAY, SPAN, DIST, N)

ARRAY (Real array) stores the hyperbola
SPAN (Real inches) is the width and height of the curve
DIST (Real inches) is the distance of the curve from the X or Y axis at the limits of the curve
N (Integer) is the number of points in the curve

EXAMPLE CALL HYPERB(H, 2.0, 0.5, 10)
This statement stores a hyperbola in H 2.0 by 2.0 inches, 0.5 inches from the origin axes

NOTES 1. No plotting takes place

(1)26.03.75
NAME  HYPOTR (Hypotrochoid)

FUNCTION Stores a Hypotrochoid curve as a 2-d Picaso shape

ARGUMENTS (ARRAY, RADFIX, RADMOV, DIST, N)

ARRAY  (Real array) stores the 2-d hypotrochoid
RADFIX  (Real inches) is the radius of the fixed circle
RADMOV  (Real inches) is the radius of the moving circle
DIST  (Real inches) is the distance of the traced point that exists on the moving circle to its centre
N  (Integer) is the number of points on the curve

EXAMPLE  CALL HYPOTR(H, 2.0, 1.0, 1.0, 60)

This statement represents a 1.0" radius circle moving inside a fixed 2.0" radius circle. The point traced is 1.0" from the centre.

NOTES  1. No plotting takes place

(1) 26.03.75
NAME INSIDE

FUNCTION Determines whether the point X, Y is in or outside a shape

ARGUMENTS (ARRA Y, X, Y)
ARRAY (Real array) stores a 2-d Picaso shape
X, Y (Real inches) references the point to be tested

EXAMPLE J = INSIDE (BOX, 0.0, 1.0)
This statement sets J = .TRUE. If the point (0.0, 1.0) is inside the shape stored in box else J = .FALSE.

NOTES 1. A clockwise contour creates a surface whilst an anti-clockwise contour creates a hole

2. Inside is a logical function

(1) 26.03.75
NAME IRRAY1 (Integer one dimension array)

FUNCTION Reads in punched card data and stores it in a one-dimension array

ARGUMENTS (ARRAY,COL)
   ARRAY (Integer array) is the name of the array
   COL (Integer) is the number of columns in the array

EXAMPLE CALL IRRAY1(K,21)
   This statement reads in sufficient cards to store 21 numbers in K.

NOTES
1. The format of the cards is (10I8) the numbers must be right-justified
2. For the above example 3 cards are required:

(1)26.03.75
NAME IRRAY2 (Integer two dimension array)

FUNCTION Reads in punched card data and stores it in a two
dimension array

ARGUMENTS (ARRAY, COL, ROW)
  ARRAY (Integer array) is the name of the array
  COL (Integer) is the number of columns in the array
  ROW (Integer) is the number of rows in the array

EXAMPLE CALL IRRAY2(K, 21, 2)
  This statement reads in sufficient cards to
  store 42 numbers in K.

NOTES
  1. The format of the cards is (10I8) the numbers must
     be right-justified

  2. For the above example 6 cards are required:

(1) 26.03.75
NAME

ISOMET (Isometric surface)

FUNCTION

Draws an isometric projection of a tiled surface created by rotating a 2-d contour about an axis.

ARGUMENTS

(ARRAY, LX, NX, LY, NY, ANGLE, XC, YC, WARRAY, SOLID, OVER, UNDER, XS, YS, V)

ARRAY

(Real array) stores a 2-d open Picasa contour

LX

(Real inches) is the width of the surface

NX

(Integer) is the number of points across the surface

LY

(Real inches) is the depth of the surface

NY

(Integer) is the number of points in the depth of the surface

ANGLE

(Real degrees) is the angle the surface makes with the X-Z plane

XC, YC

(Real inches) are the coordinates of the axis about which rotation occurs

WARRAY

(Real array) is a work array required by isomet and is dimensioned (2, NX+NY)

SOLID

(Integer array) is a work array required by isomet and is dimensioned (NX+NY)

OVER

(Real array) is a work array required to store the upper horizon and is dimensioned (NX+NY)

UNDER

(Real array) is a work array required by isomet to store the lower horizon and is dimensioned (NX+NY)

XS, YS

(Real inches) are the coordinates of the nearest corner relative to the origin

V

(Real inches) is the height of the surface beyond the range of the contour

(1)26.03.75
EXAMPLE  

CALL ISOMET(A, 6.0, 20, 6.0, 20, 30.0, 3.0, 3.0, W, S, O, U, 0.0, 0.0, 0.0)

This statement produces an isometric projection of a surface produced by rotating the contour stored in A about the point 3.0, 3.0.
NAME             ITAKE (Integer take)

FUNCTION        Selects an integer random number from a specified range

ARGUMENTS       $(I, J)$

  $I$ (Integer) defines low-order range
  $J$ (Integer) defines high-order range

EXAMPLE         $K = \text{ITAKE}(-3, 2)$

  This statement selects a random number between the range -3 to 2 inclusive
NAME       JOIN

FUNCTION    Joins together two Picaso structures

ARGUMENTS   (ARRAYA,ARRAYB)
            ARRAYA (Real array) stores a Picaso structure
            ARRAYB (Real array) stores a Picaso structure and
            receives the shape stored in arrayA

EXAMPLE     CALL JOIN(A,B)
            This statement copies across the structure
            stored in 'A' and stores it behind 'B'.

NOTES       1. Join may be used for 2 or 3 dimensioned structures

(1)26.03.75
NAME       LINE

FUNCTION  Stores a line in array

ARGUMENTS  (ARRAY, X1, Y1, X2, Y2, N)

ARRAY  (Real array) stores the line
X1, Y1  (Real inches) are the coordinates of one end of the line
X2, Y2  (Real inches) are the coordinates of the other end of the line
N  (Integer) is the number of points on the line

EXAMPLE  CALL LINE(A, 0.0, 0.0, 1.0, 1.0, 5)
          This statement stores a line in A between the origin and 1.0,1.0 in 5 points

NOTES 1. No plotting takes place

(1)26.03.75
NAME    LINE3D (Line in 3-d)

FUNCTION Stores a three-dimensional line as a Picaso structure

ARGUMENTS (ARRAY, X1, Y1, Z1, X2, Y2, Z2)

ARRAY (Real array) stores the 3-d Picaso line
X1, Y1, Z1 (Real inches) references the first point
X2, Y2, Z2 (Real inches) references the second point

EXAMPLE    CALL LINE3D(A, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0)

This statement stores a line starting at the origin and finishing at X=Y=Z=1.0"

NOTES 1. No plotting takes place

(1) 26.03.75
NAME LISSAJ (Lissajous Curves)

FUNCTION Stores a lissajous figure in an array

ARGUMENTS (ARRAY, XSPAN, YSPAN, F, D, N)

- ARRAY (Real array) stores the lissajous curve
- XSPAN (Real inches) is the width of the curve
- YSPAN (Real inches) is the height of the curve
- F (Real) is a factor affecting the number of loops in the 'X' direction
- D (Real radians) is a displacement value
- N (Integer) is the number of points in the curve

EXAMPLE CALL LISSAJ(A, 2.0, 1.0, 1.0, 0.0, 100)

NOTES 1. No plotting takes place

(1) 26.03.75
NAME LSPIRA (Logarithmic Spiral)

FUNCTION Stores a logarithmic spiral in array

ARGUMENTS (ARRAY, RADIUS, CYCLES, N)
  ARRAY (Real array) stores the spiral
  RADIUS (Real inches) is the final radius after the specified number of revolutions
  CYCLES (Real) is the number of revolutions
  N (Integer) is the number of points in the curve

EXAMPLE CALL LSPIRA(S, 3.0, 2.0, 60)
  This statement stores a logarithmic spiral in S with 2.0 revolutions, the final radius is 3.0 inches

NOTES 1. No plotting takes place

(1)26.03.75
NAME 

MASK

FUNCTION 

Draws a shape masked by another

ARGUMENTS \( (ARRAY, X, Y, WARRAY) \)

\( ARRAY \) (Real array) stores a 2-d Picaso shape to be masked

\( X, Y \) (Real inches) is the amount of shift applied to the drawn shape

\( WARRAY \) (Real array) stores a 2-d Picaso shape that masks array

EXAMPLE 

CALL \( \text{MASK}(A, 0.0, 0.0, B) \)

This statement draws out shape \( A \) masked by shape \( B \).

![Diagram of shapes A and B]

NOTES 

1. The masking shape is not drawn

2. Clockwise masking contours produce holes anti-clockwise contours mask

(1) 26.03.75
NAME  MIRROR (Mirror Image)

FUNCTION  Produces a reflection of a 2-dimensional *Picaso* shape about a specified line

ARGUMENTS  \((ARRAY, X_1, Y_1, X_2, Y_2, ARRAY1)\)
- \(ARRAY\) (Real array) stores a 2-d *Picaso* shape
- \(X_1, Y_1\) (Real inches) is one point on the mirror line
- \(X_2, Y_2\) (Real inches) is a second point on the mirror line
- \(ARRAY1\) (Real array) stores the reflected shape

EXAMPLE  \(\text{CALL MIRROR(TRIANG, } 5.0, 0.0, 5.0, 10.0, \text{REFLEC)}\)

This statement derives the reflection of the shape stored in \(\text{Triang}\) about the line passing through the points \((5.0, 0.0), (5.0, 10.0)\). The reflection is stored in \(\text{Reflec}\).

NOTES  1. No plotting takes place

\((1)26.03.75\)
NAME
MIX2D (Mix coordinates in a 2-d shape)

FUNCTION
Interchanges the coordinates of a 2-d Picaso shape

ARGUMENTS
(ARRAY, N1, N2, ARRAYA)

ARRAY
(Real array) stores a 2-d Picaso shape

N1, N2
(Integer) reference the X & Y coordinates

positions

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>X</th>
<th>Y</th>
<th>OLD COORDINATES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>X</td>
<td>Y</td>
<td>NEW COORDINATES</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Y</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
</tbody>
</table>

ARRAYA
(Real array) stores the 'mixed' shape

EXAMPLE
CALL MIX2D(A, 2, 1, B)
This statement interchanges the X and Y coordinates stored in the shape A and stores the result in B.

NOTES
1. No plotting takes place
2. The vertex sequence is reversed
3. N1 & N2 may be +VE or -VE to enable the sign of coordinates to be changed.

(1) 26.03.75
NAME

MIX3D (Mix coordinates in a 3-d object)

FUNCTION

Interchanges the coordinates of a 3-d Picaso object

ARGUMENTS (ARRAY, N1, N2, N3, ARRAYA)

ARRAY (Real array) stores a 3-d Picaso object

N1, N2, N3 (Integer) reference the X, Y & Z coordinate positions

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>OLD COORDINATES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>X</td>
<td>Z</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>Y</td>
<td>X</td>
<td>Z</td>
<td>NEW COORDINATES</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>Y</td>
<td>Z</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>Z</td>
<td>X</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>Z</td>
<td>Y</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

ARRAYA (Real array) stores the 'mixed' object

EXAMPLE

CALL MIX3D(A, 2, 1, 3, B)

This statement interchanges the X and Y coordinates and leaves the Z coordinate untouched, storing the result in B.

NOTES

1. No plotting takes place
2. The vertex sequences are reversed
3. N1, N2 & N3 may be +VE or -VE to enable the sign of coordinates to be changed.

(1) 26.03.75
NAME MODSH (Modulate Shape)

FUNCTION Modulates the lines describing a 2-dimensional Picasso shape with a 2-dimensional Picasso contour

ARGUMENTS (ARRAY, CONTOR, F, ARRAY1)
ARRAY (Real array) stores a 2-d Picasso shape
CONTOR (Real array) stores a 2-d Picasso contour
F (Real) is the amplitude factor for contour
ARRAY1 (Real array) will store the modulated shape

EXAMPLE CALL MODSH(BOX, CON, 1.0, BOX1)
This statement takes the shape stored in box and joins the points with the contour stored in con. The contour in con is unmodified as F=1.0 the modulated shape is stored in box1.

NOTES
1. No plotting takes place
2. Array1 must not be array
NAME     NDIMEN (Dimension)

FUNCTION  A function to calculate the number of elements required to contain a given Picaso structure.

ARGUMENTS (ARRAY)
ARRAY    (Real array) stores a Picaso structure

TYPE      Ndimen is an integer type function

EXAMPLE   I=NDIMEN(BOX)
This statement returns the number of positions used by the shape stored in box.
NAME  NLINES (Number of lines)

FUNCTION  This function determines the number of lines contained in a 2-dimensional Picaso shape.

ARGUMENTS (ARRAY)

ARRAY  (Real array) stores a Picaso structure

FUNCTION  Nlines is an integer function

EXAMPLE  I=NLINES(BOX)

This statement determines the number of lines stored in box.
NAME
NORMAL (Normalize)

FUNCTION
Adjusts the coordinates of a 2-dimensional Picaso shape such that a specific point is located at a given coordinate. The rest of the points are normalized about this point.

ARGUMENTS (ARRAY, NL, NP, X, Y)

ARRAY  (Real array) stores a 2-d Picaso shape
NL     (Integer) references the line containing the point
NP     (Integer) references the point on the line
X, Y   (Real inches) is the coordinate of the point

EXAMPLE
CALL NORMAL(BOX, 1, 1, 0.0, 0.0)
This statement adjusts the coordinates of box such that the 1st point on the 1st line is set to (0.0, 0.0)

NOTES
1. No plotting takes place

(1) 26.03.75
NAME
NPOINT (Number of points)

FUNCTION
Returns the number of points in a contour on a *Picaso* structure.

ARGUMENTS (ARRAY, NC)
  ARRAY  (Real array) stores a 2 or 3-d *Picaso* structure
  NC     (Integer) references the contour

EXAMPLE
  \[ N = \text{NPOINT}(\text{BOX1}) \]
  This statement establishes how many points there are on the *first* contour of *box*.

(1) 26.03.75
NAME NSUB (Subscript number)

FUNCTION Returns the subscript of the point count descriptor in the l'th contour of a Picaso structure

ARGUMENTS (ARRAY,L)

ARRAY (Real array) stores a 2 or 3-d Picaso structure

L (Integer) references the contour in the structure

EXAMPLE N=NSUB(BOX,2)

This statement returns the subscript of the 2nd point count descriptor.
NAME NXSUB (Number of X subscript)

FUNCTION Returns the number of the X subscript of a point in a Picasso structure

ARGUMENTS (ARRAY, NC, NP)
   ARRAY (Real array) stores a Picasso structure
   NC    (Integer) references the contour
   NP    (Integer) references the vertex

EXAMPLE N = NXSUB(A, 1, 1)
    This statement sets N to the xsubscript of the first vertex on the first contour

(1) 28.03.75
NAME NYSUB (Number of Y subscript)

FUNCTION Returns the number of the Y subscript of a point in a Picaso structure

ARGUMENTS (ARRAY, NC, NP)
- ARRAY (Real array) stores a Picaso structure
- NC (Integer) references the contour
- NP (Integer) references the vertex

EXAMPLE \( N = \text{NYSUB}(A, 1, 1) \)
This statement sets \( N \) to the \( Y \) subscript of the first vertex on the first contour

(1)26.03.75
NAME NZSUB (Number of Z subscript)

FUNCTION Returns the number of the Z subscript of a point in a Picaso structure

ARGUMENTS (ARRAY, NC, NP)
- ARRAY (Real array) stores a Picaso structure
- NC (Integer) references the contour
- NP (Integer) references the vertex

EXAMPLE \( N = \text{NZSUB}(A, 1, 1) \)
This statement sets \( N \) to the Z subscript of the first vertex on the first contour

(1) 26.03.75
NAME          OBJECT

FUNCTION      Reads in a three-dimensional *Picaso* structure from cards and stores it in an array

ARGUMENTS (ARRAY)

ARRAY   (Real array)

EXAMPLE      CALL OBJECT(BOX)

This statements reads in a shape and stores it in *box*.

NOTES
1. Each point requires 3 coordinates
   X cols 4-13
   Y cols 14-23
   Z cols 24-33
2. Each surface is terminated by
   EOL cols 1-2
3. Each object is terminated by
   EOS cols 1-3
4. Each surface should be specified in a clock-wise sense.
NAME ORIGIN

FUNCTION Changes the position of the origin

ARGUMENTS \((x, y)\)

\(x, y\) (Real inches) specifies the position of the new origin relative to the existing origin.

EXAMPLE CALL ORIGIN(5.0, 5.0)

This changes the Origin to a point 5.0 inches from the x-axis and 5.0 inches from the y-axis.

NOTES

1. The origin cannot be located off the paper.
2. During the origin change the pen is up.
NAME PARAB (Parabola)

FUNCTION Stores a parabolic curve in an array

ARGUMENTS (ARRAY, XSPAN, YSPAN, N)

- ARRAY (Real array) stores the parabola
- XSPAN (Real inches) is the total width of the parabola
- YSPAN (Real inches) is the height of the parabola
- N (Integer) is the number of points in the curve

EXAMPLE CALL PARAB(A, 4.0, 4.0, 20)

This statement stores a parabola in A, 4.0 by 4.0 inches

NOTES 1. No plotting takes place

(1)26.03.75
**NAME**  PERSP (Perspective)

**FUNCTION**  Transforms a point in 3-d space onto a 2-d picture plane to obtain perspective or wide-angle effects.

**ARGUMENTS**  \((X, Y, Z, XW, YW)\)

- \(X, Y, Z\) (Real inches) are the coordinates of the point in 3-d space
- \(XW, YW\) (Real inches) are the coordinates of the transformed point on the window plane

**EXAMPLE**  CALL PERSP(5.0, 2.0, 1.0, X, Y)

This statement sets \(X, Y\) to the perspective view of \((5.0, 2.0, 1.0)\)

**NOTES**
1. Normal perspective is obtained if eye has been previously called. Wide-angle effects are obtained if fishi has been called
2. No plotting takes place
NAME        PLIT

FUNCTION    Is the basic plotting command in the Picaso system

ARGUMENTS  (X,Y,PEN)

X,Y        (Real inches) is the point to which the pen moves
PEN        (Integer) is the pen control parameter
    Pen=2=Pen Down
    Pen=3=Pen Up
    Pen=-VE=Change Origin

EXAMPLE     CALL PLIT(1.0,1.0,2)

    This statement moves the pen from its present condition to the point 1.0,1.0 with the pen down

(1)26.03.75
NAME  POLYGON (Polygon)

FUNCTION Stores a regular polygon as a 2-dimensional *Picasso* shape

ARGUMENTS *(ARRAY, RADIUS, N)*

*ARRAY* (Real array) stores the polygon
*RADIUS* (Real inches) is the radius of the polygon
*N* (Integer) is the number of sides

EXAMPLE  CALL POLYGON(S, 1.0, 100)

This statement stores a 100-sided Polygon of radius 1.0 inch in S, with the origin at the centre

NOTES
1. No plotting takes place
2. The first point is always:
   \[ X = \text{Radius}, \ Y = 0.0 \]
3. Coordinates are in clockwise sequence
4. The array must be dimensioned to at least \((2*N) + 3\).
NAME POSCON (Position on a contour)

FUNCTION Returns the height of the contour at a position specified by two coordinates

ARGUMENTS \( (ARRAY, NX, NY, XINC, YINC, XC, YC, V) \)
- \( ARRAY \) (Real array) stores a 2-d open contour
- \( NX \) (Integer) is the number of X increments
- \( NY \) (Integer) is the number of Y increments
- \( XINC \) (Real inches) is the size of the X increment
- \( YINC \) (Real inches) is the size of the Y increment
- \( XC, YC \) (Real inches) are the coordinates of the axis about which rotation occurs
- \( V \) (Real inches) is the default value when the extent of the contour is exceeded

EXAMPLE \( H = \text{POSCON}(A, 20, 10, 0.1, 0.2, 3.0, 3.0, 0.0) \)
This statement sets \( H \) to a value that is equivalent to the height of the contour stored in \( A \) at the specified point
NAME     PRINT

FUNCTION Prints out the coordinate values stored in a shape

ARGUMENTS (ARRAY)

ARRAY (Real array) stores a 2 or 3-d Picaso structure

EXAMPLE CALL PRINT(BOX)

This statement produces a listing of the coordinates of box.

(1) 26.03.75
NAME PULL

FUNCTION Stretches or shears a shape about the point X, Y

ARGUMENTS (ARRAY, X, Y, THETA, F, ARRAYB)
- ARRAY (Real array) stores the shape to be stretched
- X, Y (Real inches) is the point that suffers zero stretch
- THETA (Real degrees) is the direction of stretch
- F (Real) is the degree of stretch
- ARRAYB (Real array) stores the stretched shape

EXAMPLE CALL PULL(A, 0.0, 0.0, 0.0, 0.0, 2.0, A)
This statement stretches shape A about the origin in the X-direction by a factor of 2.0.

NOTES 1. No plotting takes place

(1) 26.03.75
NAME REMOVE

FUNCTION Removes a contour from a *Picaso* shape

ARGUMENTS (ARRAY,N)

- **ARRAY** (Real array) stores a *Picaso* structure
- **N** (Integer) references the contour to be removed

EXAMPLE CALL REMOVE(BOX,2)

This statement removes the second contour from *box* and shifts remaining contours along

NOTES 1. Applies to 2 and 3-d shapes
NAME REVERS (Reverse)

FUNCTION Reverses the sequence of vertices in a 2-d Picaso contour

ARGUMENTS (ARRAY, N)
ARRAY (Real array) stores a 2-d Picaso shape
N (Integer) references the contour to be reversed

EXAMPLE CALL REVERS(BOX, 1)
This statement reverses the sequence of the vertices in the 1st contour of box.

(1) 26.03.75
NAME PYRAM (Pyramid)

FUNCTION Stores a 3-d pyramid in an array

ARGUMENTS (ARRAY, BASE, HEIGHT)
 ARRAY (Real array) stores the pyramid
 BASE (Real) is the width of the pyramid
 HEIGHT (Real) is the height of the pyramid

EXAMPLE CALL PYRAM(P, 1.0, 2.0)
This statement stores a two-inch high pyramid in P with a one-inch base
NAME        RECTNG (Rectangle)

FUNCTION  Stores a rectangle as a 2-dimensional *Picaso* shape

ARGUMENTS  *(ARRAY, SIDEA, SIDEB)*

  *ARRAY*  (Real array) stores the rectangle
  *SIDEA*  (Real inches) is the width
  *SIDEB*  (Real inches) is the height

EXAMPLE     CALL RECTNG(R, 2.0, 1.0)

  This statement stores a rectangle in *R* with
  a height of 1.0 inch and width of 2.0
  inches. The lower left-hand corner is
  located at the origin

NOTES  
1. No plotting takes place
2. The first point is the origin
3. Coordinates are in clockwise sequence
4. The array must be dimensioned to at least 13.

(1)26.03.75
NAME RIPPLE

FUNCTION Chooses a new vertex to become the first vertex of a closed contour

ARGUMENTS (ARRAY, NC, I)

ARRAY (Real array) stores a 2-d Picaso shape
NC (Integer) references the contour to be rippled
I (Integer) is the direction of ripple
I=+VE Ripple forward
I=-VE Ripple backward

EXAMPLE CALL RIPPLE(Q, 1, 1)

This statement rotates the coordinates on the 1st contour of Q one step forward

NOTES
1. No plotting takes place
2. The magnitude of I is immaterial, ripple can only shift one step.

(1) 26.03.75
NAME  RHODON (Rhodonea Curve)

FUNCTION Stores a rhodonea curve as a 2-d Picaso shape

ARGUMENTS \((ARRAY, RADIUS, NP, N)\)
- \(ARRAY\) (Real array) stores the 2-d rhodonea
- \(RADIUS\) (Real inches) is the radius of the curve
- \(NP\) (Integer) is the number of petals in the curve
- \(N\) (Integer) is the number of points in the curve

EXAMPLE  \(\text{CALL RHODON}(R, 2.0, 4, 100)\)
This statement stores a 4-petaled rhodonea in \(R\).

NOTES  1. No plotting takes place

(1)26.03.75
NAME  ROTATE

FUNCTION Rotates a 2-d Picaso shape about a specified point

ARGUMENTS (ARRAY, XP, YP, THETA, ARRAYA)

ARRAY (Real array) stores a 2-d Picaso shape
XP, YP (Real inches) is the centre of rotation
THETA (Real degrees) is the amount and direction of rotation
  +VE = Anti-clockwise
  -VE = Clockwise
ARRAYA (Real array) receives the rotated shape

EXAMPLE CALL ROTATE(S, 0.0, 0.0, 40.0, S)
This statement rotates the shape stored in S 40.0 degrees about the origin

NOTES 1. No plotting takes place

(1) 26.03.75
NAME ROW

FUNCTION Plots out a row of 2-dimensional *Picaso* shapes

ARGUMENTS \((ARRAY, SIZE, N, XSP, X, Y)\)

- **ARRAY** (Real array) stores a 2-d *Picaso* shape
- **SIZE** (Real) is the degree of enlargement or reduction of the drawn shape relative to the original stored in array
- **N** (Integer) is the number of shapes to be drawn
- **XSP** (Real inches) is the horizontal displacement between two shapes
- **YSP** (Real inches) is the vertical displacement between two shapes
- **X, Y** (Real inches) is the amount of shift applied to the row of shapes

EXAMPLE CALL ROW(BOX, 1.0, 3, 1.0, 1.0, 0.0, 0.0, 0.0)

Draws out 3 shapes stored in box with their original size, displaced 1.0 inch horizontally and vertically, the entire row is shifted zero inches from the origin.

![Diagram of shapes]

NOTES 1. Observe that XSP and YSP are not distances between two shapes, they are the distances between the same point on the displaced shapes.

\(1126.03.75\)
NAME ROW2D (Draw a 2-d row in 3-d)

FUNCTION Draws out a three-dimensional version of a 2-d Picaso row of shapes

ARGUMENTS (ARRAY, SIZE, N, XSP, YSP, X, Y, N1, N2, N3, V)

ARRAY (Real array) stores a 2-d Picaso shape

SIZE (Real) alters the size of the final shape by 'size' items

N (Integer) is the number of shapes in the row

XSP (Real inches) is the distance between points on two shapes in the 'X' direction

YSP (Real inches) is the distance between points on two shapes in the 'Y' direction

X, Y (Real inches) is the degree of shift applied to the 2-d row

N1, N2, N3 (Integer) determine the transformation

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>X</td>
<td>Y</td>
<td>V</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>X</td>
<td>V</td>
<td>Y</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>Y</td>
<td>X</td>
<td>V</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>V</td>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>Y</td>
<td>V</td>
<td>X</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>V</td>
<td>Y</td>
<td>X</td>
</tr>
</tbody>
</table>

IN 2-D

V (Real inches) is the value of the third dimension

(1) 26.03.75
EXAMPLE

CALL ROW2D(SQ, 1.0, 3, 1.5, 1.5, 0.0, 0.0, 1, 2, 3, 1.0)

This statement produces the following transformation:
NAME

SENSE

FUNCTION
Determines whether a point is to the left, in-line or to the right of a reference line

ARGUMENTS

\[(X_P, Y_P, X_1, Y_1, X_2, Y_2)\]

- \(X_P, Y_P\) (Real inches) is the position of the test point
- \(X_1, Y_1\) (Real inches) are the coordinates of the first point on the line
- \(X_2, Y_2\) (Real inches) are the coordinates of the second point on the line

SENSE is an Integer Function

- = -1 if \(X_P, Y_P\), is to the right
- = 0 if \(X_P, Y_P\) is in-line
- = +1 if \(X_P, Y_P\) is to the left

\[\begin{align*}
\text{+ve} & \quad \bullet X_P, Y_P \\
\emptyset & \quad \bullet X_1, Y_1 \quad \bullet X_2, Y_2 \\
\text{-ve} & \quad \bullet X_P, Y_P
\end{align*}\]
NAME SHADE

FUNCTION Shades a 2-d Picaso shape with parallel lines

ARGUMENTS (ARRAY, THETA, SPACE)
- ARRAY (Real array) stores a 2-d Picaso shape
- THETA (Real degrees) is the slope of the lines
- SPACE (Real inches) is the distance between the shading lines

EXAMPLE CALL SHADE(A, 45.0, 0.1)
This statement shades the shape stored in A with line 0.1" apart at 45°.

NOTES 1. A clockwise contour is assumed to be solid whilst in anti-clockwise contour creates a hole.

(1)26.03.75
NAME SHAPE

FUNCTION Reads a 2-dimensional Picaso shape from punched cards and stores it in a real array.

ARGUMENT (ARRAY)

ARRAY (Real array) accepts the 2-d Picaso shape

EXAMPLE CALL SHAPE(STARTS)

This statement reads in the next Picaso shape and stores it in the array Stars.

NOTES 1. The array must be declared on a dimension statement and must be large enough to store the entire shape.
2. No plotting takes place.
NAME       SHIFT

FUNCTION  Alters the location of a 2-dimensional Picaso shape

ARGUMENTS  (ARRAY, X, Y)
            ARRAY  (Real array) stores a 2-d Picaso shape
            X, Y    (Real inches) is the degree of shift

EXAMPLE    CALL SHIFT(BOX, 3.0, 1.0)
            This statement shifts the coordinates stored
            in BOX, 3.0 inches along the X-axis and 1.0
            inch along the Y-axis

NOTES      1. No plotting takes place
NAME SHIFT3 (Shift 3-d)

FUNCTION Alters the location of a 3-d Picaso object

ARGUMENTS (ARRAY, X, Y, Z)

ARRAY (Real array) stores a 3-d Picaso object
X, Y, Z (Real inches) is the degree of shift

EXAMPLE CALL SHIFT3(BOX, 1.0, 1.0, 1.0)
This statement adds 1.0 inch to every coordinate of box.

NOTES 1. No plotting takes place

(1) 26.03.75
NAME        SILUET (Silhouette)

FUNCTION    Draws out a tiled version of a 2-d contour rotated about an axis

ARGUMENTS   (ARRAY, SIZE, X, Y, Z, ANGS, ANGF, N, L)
            ARRAY    (Real array) stores a 2-d Picaso open contour
            SIZE     (Real) alters the size of the drawing
            X, Y, Z  (Real) is the amount of shift applied to the object
            ANGS     (Real degrees) is the starting angle of rotation
            ANGF     (Real degrees) is the final angle of rotation
            N        (Integer) is the number of rotational steps
            L        (Integer) if L is -VE a degree of hidden-line removal is included. If L is +VE all lines are drawn

EXAMPLE     CALL SILUET(A, 1.0, 0.0, 0.0, 0.0, 0.0, 360.0, 20, 1)
            This statement draws out a complete rotation of the contour stored in A.

NOTES       1. The position of the eye must be specified by calling eye or fishi

(1)26.03.75
NAME
SINE (Sine wave)

FUNCTION
Stores a sine wave in an array

ARGUMENTS  
\((ARRAY, AMP, LENGTH, ANGS, ANGF, N, XP, YP)\)
- \(ARRAY\) (Real array) stores the 2-d \(Picaso\) sine wave
- \(AMP\) (Real inches) the amplitude of the sine wave
- \(LENGTH\) (Real inches) the length of the waveform
- \(ANGS\) (Real degrees) the starting angle of the waveform
- \(ANGF\) (Real degrees) the final angle of the waveform
- \(N\) (Integer) number of steps in waveform
- \(XP, YP\) (Real inches) the coordinate of the first point

EXAMPLE

\[
\text{CALL SINE(S,1.0,2.0,0.0,180.0,20.0,0.0,0.0)}
\]

This statement stores a waveform consisting of a half-sine wave in \(S\).

NOTES
1. No plotting takes place

(1)26.03.75
NAME  SIZE

FUNCTION  Changes the size of a 2-dimensional Picaso shape

ARGUMENTS  (ARRAY, F, X, Y)

ARRAY  (Real array) stores a 2-d Picaso shape
F  (Real) is the size change factor
X, Y  (Real inches) is the centre of the size change

EXAMPLE  CALL SIZE(BOX, 3.0, 1.0, 1.0)
This statement modifies the shape coordinates stored in box by a factor of 3.0 relative to the point (1.0, 1.0). For example, the point (2.0, 2.0) becomes (4.0, 4.0)

NOTES  1. No plotting takes place

(1) 26.03.75
NAME SMOOTH

FUNCTION Draws a smooth curve (Cubic Spline) through a given set of data points stored as a 2-d Picaso shape

ARGUMENTS (ARRAY, SIZE, X, Y, WARRAY)
- ARRAY (Real array) stores a 2-d Picaso shape
- SIZE (Real) alters the size of the drawn shape by a factor of 'size'.
- X, Y (Real inches) is the degree of shift applied to the shape
- WARRAY (Real array) is a work array required by smooth. It has dimensions (5, N) where N is equal to the number of vertices in any contour of the shape

EXAMPLE CALL SMOOTH(A, 1.0, 0.0, 0.0, W)
This statement draws a curve through the shape stored in A.
NAME      SLOPE

FUNCTION Calculates the slope in degrees of a 2-d Picaso shape.

ARGUMENTS (ARRAY, NC, NL)
  ARRAY   (Real array) stores a 2-d Picaso shape
  NC      (Integer) references the contour
  NL      (Integer) references the line

EXAMPLE  A=SLOPE(BOX, 1, 2)
          This statement calculates the angle of the
          second line on the first contour of box.

NOTES
1. Slope is a real function
2. If a contour has N vertices, then there are N-1 lines

(1)26.03.75
NAME       SNOW (Snowflake Curve)

FUNCTION  Develops a snowflake curve from a 2-d Picaso shape

ARGUMENTS (ARRAY, XO, YO, F, DEPTH)
ARRAY    (Real array) stores the 2-d Picaso shape to be transformed
XO, YO    (Real inches) is the degree of shift to be applied to the final shape
F         (Real) is the degree of enlargement or reduction of the drawn shape relative to the original stored in array
DEPTH    (Real) is the degree of crystalisation

EXAMPLE  CALL SNOW(F, 0.0, 0.0, 1.0, 3)
This statement crystalises the shape F to a depth of 3 with zero shift and no size change

NOTES 1. Plotting does take place
2. Depth should not exceed 4
3. Clockwise shapes grow, anti-clockwise reduce

(1)26.03.75
NAME   SQUARE

FUNCTION Stores a square as a 2-dimensional *Picaso* shape

ARGUMENTS *(ARRAY, SIDE)*

ARRAY   (Real array) stores the square
SIDE    (Real inches) is the size of the square

EXAMPLE  CALL SQUARE(S, 1.0)
          This statement stores a square with sides 1.0
          inch long, and the lower left-hand corner at
          the origin.

NOTES   1. No plotting takes place
        2. The first point is the origin
        3. Coordinates are in clockwise sequence
        4. The array must be dimensioned to at least 13.

(1) 26.03.75
NAME STAR

FUNCTION Stores a regular star as a 2-d Picaso shape

ARGUMENTS (ARRAY, NP, RADOUT, RADIN)

ARRAY (Real array) stores the 2-d Picaso star
NP (Integer) is the number of points to the star
RADOUT (Real inches) is the radius of the outer point
RADIN (Real inches) is the radius of the inner point

EXAMPLE CALL STAR(S, 4, 2.0, 1.0)

This statement stores a 4-pointed star in S.

NOTES
1. No plotting takes place
2. If there are N points to a star then there will be 2*N+1 vertices. This implies that the array must be dimensioned at least 4*N+5.
NAME               START

FUNCTION          Initialises all plotting processes

ARGUMENTS         None

EXAMPLE           CALL START
                   This statement enables all plotting commands
to function

NOTES             1. It is only necessary to call start before plotting
NAME STICK

FUNCTION Takes two 2-dimensional Picaso shapes and adjusts the coordinates of the second shape such that its first point is coincident with the last point of the first shape

ARGUMENTS (ARRAYA, ARRAYB, ARRAYC)
- ARRAYA (Real array) stores a 2-d Picaso shape
- ARRAYB (Real array) stores a 2-d Picaso shape
- ARRAYC (Real array) receives the two shapes connected together

EXAMPLE CALL STICK(A, B, C)
This statement connects shape B to shape A and stores the result in C.

NOTES
1. No plotting takes place
2. ArrayA may be a complex shape but only the first line is used
3. ArrayB must only contain one line
4. ArrayC will store arrayA followed by arrayB
5. ArrayC must be large enough to contain arrayA and arrayB.

(1)26.03.75
NAME STIPLE (Stipple)

FUNCTION Draws a random pattern of dots within a given shape

ARGUMENTS (ARRAY,N)

ARRAY (Real array) stores a 2-d Picaso shape

N (Integer) is the number of dots to draw

EXAMPLE CALL STIPLE(B,100)

This statement draws 100 dots inside the shape B.

NOTES 1. The shape in 'B' is not drawn

(1)26.03.75
NAME Surfac (Surface)

FUNCTION Creates a three-dimensional surface by rotating a
two-dimensional contour about an axis

ARGUMENTS (ARRA Y, LX, NX, LY, NY, ANGLE, XC, YC, WARRAY, OVER, UNDER, XS, YS)

ARRAY (Real array) stores a 2-d Picaso open contour

LX (Real inches) is the width of the surface

NX (Integer) is the number of points across the

surface

LY (Real inches) is the depth of the surface

NY (Integer) is the number of lines creating the

surface

ANGLE (Real degrees) is the angle the surface makes

with the X-Z plane

XC, YC (Real inches) are the coordinates of the axis

about which rotation occurs

WARRAY (Real array) is a work array required by

surfac and must be dimensioned (NX)

OVER (Real array) is a work array required by surfac
to maintain the upper horizon and must be
dimensioned (NX)

UNDER (Real array) is a work array required by

surfac to maintain the lower horizon and must
be dimensioned (NX)

XS, YS (Real inches) is the degree of shift applied
to the entire surface relative to the origin

EXAMPLE CALL SURFAC(C, 6.0, 50, 6.0, 50, 30.0, 3.0, 3.0, ARRAY, OVER, UNDER, 0.0, 0.0)

This statement rotates a contour stored in C
about the point (3.0, 3.0) and draws a non-

transparent surface

(1) 26.03.75
NOTES

1. The following diagram illustrates the significance of the arguments:
NAME: TAKE (Choose a real random number)

FUNCTION: Chooses a random number from a specified range

ARGUMENTS: \((A, B)\)
- \(A\) (Real) defines low-order range
- \(B\) (Real) defines high-order range

EXAMPLE: \(X = \text{TAKE}(-2.0, 10.0)\)
This statement sets \(X\) a random number between the range \(-2.0\) to \(10.0\) inclusive.
NAME TRACE

FUNCTION Traces around a 2-d Picaso shape a given distance

ARGUMENTS (ARRAY, DIST, ARRAYA)
- ARRAY (Real array) stores a 2-d Picaso shape
- DIST (Real inches) is the tracing distance
- ARRAYA (Real array) receives the traced shape

EXAMPLE CALL TRACE(A, 0.2, B)
This statement traces 0.2" away from A
and stores the shape in B.

NOTES
1. No plotting takes place
2. Open contours are treated as illustrated above
NAME TRANSH (Transform shapes)

FUNCTION Transforms one 2-dimensional shape into another in a given number of steps

ARGUMENTS \((\text{ARRAYA}, \text{LA}, \text{ARRAYB}, \text{LB}, \text{ARRAYC}, \text{F})\)

- \(\text{ARRAYA}\) (Real array) stores the 2-d Picaso shape to be transformed
- \(\text{LA}\) (Integer) references the contour on Arraya that is transformed
- \(\text{ARRAYB}\) (Real array) stores the 2-d Picaso shape guiding the transformation
- \(\text{LB}\) (Integer) references the contour on Arrayb that guides the transformation
- \(\text{ARRAYC}\) (Real array) stores the transformed shape
- \(\text{F}\) (Real) is the degree of transformation

EXAMPLE \(\text{CALL TRANSH}(A, 1, B, 1, C, 0.5)\)

This statement transforms the first contour of \(A\) into the first contour of \(B\) by 50% and stores it in \(C\).

\[\text{A} \quad \text{C} \quad \text{B}\]

NOTES 1. No plotting takes place

(1) 26.03.75
NAME TRANS (Transpose)

FUNCTION Transposes a 2-d Picaso shape into a 3-d object

ARGUMENTS (ARRAY, ARRAYA)
- ARRAY (Real array) stores a 2-d Picaso shape
- ARRAYA (Real array) receives the 3-d Picaso shape

EXAMPLE CALL TRANS(A, B)

This statement transposes the 2-d shape in A, into a 3-d object by including a Z coordinate set to 0.0 and stores it in B.
NAME TURN3D (Turn in 3 dimensions)

FUNCTION Turns a 3-d Picaso object about one axis

ARGUMENTS (ARRAY, THETA, CH, CV, N, ARRAYA)
- ARRAY (Real array) stores a 3-d Picaso object
- THETA (Real degrees) is the angular rotation
- CH, CV (Real) are the coordinates of the rotational centre
- N (Integer) references the axis unrotated, X=1, Y=2 & Z=3
- ARRAYA (Real array) receives the rotated object

EXAMPLE CALL TURN3D(B, 20.0, 0.0, 0.0, 1, C)
This statement rotates 'B' about the X axis and the result in C.

NOTES 1. No plotting takes place

(1) 26.03.75
NAME WINDOW

FUNCTION Establishes a boundary to confine plotting to occur only within a specified shape

ARGUMENTS (ARRAY)
ARRAY (Real array) stores a 2-d Picasso shape

EXAMPLE CALL WINDOW(A)
This statement arranges the shape stored in A to window all future plotting. If 'A' stores a hexagon the following could result

NOTES
1. The window shape is not drawn
2. The windowing is canceled by calling call window (0.0)
3. The origin of the window coincides with the plotting origin
4. Clockwise window contours create holes anti-clockwise window contours mask
5. A window shape may have many contours so long as any line drawn does not intersect with more than 10 lines
6. The window shape cannot be drawn whilst windowing occurs

(1) 26.03.75
NAME            XARC (X-coordinate of an Arc)

FUNCTION       Returns the X-coordinate of a point on an Arc

ARGUMENTS      \((X, Y, RADIUS, THETA, PHI, I, N)\)

\(X, Y\)        (Real inches) are the coordinates of the centre of revolution

\(RADIUS\)     (Real inches) is the radius of curvature

\(THETA\)      (Real degrees) is the starting angle of the arc

\(PHI\)        (Real degrees) is the swept angle in a clockwise direction

\(I\)          (Integer) is the position on the curve relative to \(N\).

\(N\)          (Integer) is the number of points on the curve.

EXAMPLE        \(X = XARC(0.0, 0.0, 2.0, 180.0, 90.0, 2, 100)\)

This statement sets \(X\) to the X-coordinate of the 2nd of an arc consisting of 100 points.

NOTES           1. No plotting takes place

(1)26.03.75
NAME XCOORD (X coordinate)

FUNCTION Is a function returning the X coordinate of a specified point in a Picaso structure

ARGUMENTS (ARRAY, NL, NP)
- ARRAY (Real array) stores a Picaso structure
- NL (Integer) references the contour or surface containing the point
- NP (Integer) references the point

EXAMPLE X=XCOORD(BOX, 1, 1)
This statement returns the X coordinate of the first point on the first contour of BOX.

NOTES
1. No plotting takes place
2. Xcoord is a real type function

(1)26.03.75
NAME       XELLIP  (X coordinate of an ellipse)

FUNCTION  Returns the X-coordinate of a specific point on an ellipse

ARGUMENTS (MAJOR, MINOR, I, N)

- **MAJOR** (Real inches) is the major radius of the ellipse.
- **MINOR** (Real inches) is the minor radius of the ellipse.
- **I** (Integer) is the number of the point on the ellipse.
- **N** (Integer) is the maximum number of points on the curve.

EXAMPLE     \( C \equiv \text{XELLIP}(3.0, 2.0, 20, 100) \)

This statement sets \( C \) to the 20th X-coordinate of an ellipse containing 100 points.

NOTES          1. No plotting takes place
                2. The curve is clockwise

\( (1)26.03.75 \)
NAME XEPIT (X-coordinate of an epitrochoid)

FUNCTION Returns the X-coordinate of a specific point on an epitrochoid

ARGUMENTS \((RADFIX, RADMOV, DIST, I, N)\)

\(RADFIX\) (Real inches) is the radius of the fixed circle

\(RADMOV\) (Real inches) is the radius of the moving circle

\(DIST\) (Real inches) is the distance of the traced point that exists on the moving circle to its centre

\(I\) (Integer) is the number of the point on the curve

\(N\) (Integer) is the maximum number of points on the curve

EXAMPLE \(C = XEPIT(3.0, 1.0, 0.5, 20, 100)\)

This statement sets \(C\) to the 20th X-coordinate of an epitrochoid containing 100 points

NOTES
1. No plotting takes place
2. The curve is clockwise

(1)26.03.75
NAME XHYPER (X-coordinate of a hyperbola)

FUNCTION Returns the X-coordinate of a specific point on a hyperbola

ARGUMENTS (SPAN, DIST, I, N)

SPAN (Real inches) is the width and height of the curve relative to the X & Y axes
DIST (Real inches) is the distance of the curve from the X or Y axis at the limits of the curve
I (Integer) is the number of the point on the curve
N (Integer) is the maximum number of points on the curve

EXAMPLE C=XHYPER(2.0, 0.5, 20, 100)
This statement sets C to the 20th X-coordinate of a hyperbola consisting of 100 points

NOTES 1. No plotting takes place

(1) 26.03.75
NAME XHYPOT (X-coordinate of a hypotrochoid)

FUNCTION Returns the X-coordinate of a specific point on a hypotrochoid

ARGUMENTS (RADFIX,RADMOV,DIST,I,N)
RADFIX (Real inches) is the radius of the fixed circle
RADMOV (Real inches) is the radius of the moving circle
DIST (Real inches) is the distance of the traced point on the moving circle to its centre
I (Integer) is the number of the point on the curve
N (Integer) is the maximum number of points on the curve

EXAMPLE C=XHYPOT(3.0,1.0,0.5,20,100)
This statement sets C to the 20th X-coordinate of a hypotrochoid consisting of 100 points

NOTES
1. No plotting takes place
2. The curve is clockwise

(1)26.03.75
NAME       XLISS (X-coordinate of a lissajous curve)

FUNCTION  Returns the X-coordinate of a specific point on a lissajous curve

ARGUMENTS (XSPAN, YSPAN, F, D, I, N)

XSPAN     (Real inches) is the width of the curve
YSPAN     (Real inches) is the height of the curve
        (Real) is a factor affecting the number of loops in the 'X' direction
D        (Real radians) is a displacement value
I        (Integer) is the number of the point on the curve
N        (Integer) is the maximum number of points on the curve

EXAMPLE   C=XLISS(2.0, 2.0, 1.0, 0.0, 20, 100)
           This statement sets C to the 20th X-coordinate of a lissajous curve consisting of 100 points

NOTES     1. No plotting takes place
          2. The curve is clockwise

(1) 26.03.75
NAME  XLSPIR (X-coordinate of a logarithmic spiral)

FUNCTION Returns the X-coordinate of a specific point on a logarithmic spiral

ARGUMENTS (RADIUS, CYCLES, I, N)
- **RADIUS** (Real inches) is the final radius of the spiral
- **CYCLES** (Real) is the number of cycles in the spiral
- **I** (Integer) is the number of the point on the curve
- **N** (Integer) is the maximum number of points on the curve

EXAMPLE  \( C = \text{XLSPIR}(3.0, 2.0, 20, 100) \)
This statement sets \( C \) to the 20th X-coordinate of a spiral consisting of 100 points

NOTES  
1. No plotting takes place
2. The curve is clockwise

(1)26.03.75
NAME XMIRRO (X-coordinate of a mirror-image)

FUNCTION Returns a mirror-image X-coordinate of a given point

ARGUMENTS \((X, Y, X_1, Y_1, X_2, Y_2)\)
- \(X, Y\) (Real inches) are the coordinates of the point to be reflected
- \(X_1, Y_1\) (Real inches) are the coordinates of one point on the mirror
- \(X_2, Y_2\) (Real inches) are the coordinates of another point on the mirror

EXAMPLE \(C = XMIRRO(1.0, 2.0, 2.0, 0.0, 2.0, 1.0)\)
This statement sets \(C\) to the X-coordinate of the reflected point \(1.0, 2.0\)

NOTES 1. No plotting takes place

(1) 28.03.75
NAME
XPARAB (X-coordinate of a parabola)

FUNCTION
Returns the X-coordinate of a point on a parabola

ARGUMENTS
(XSPAN, YSPAN, I, N)

XSPAN (Real inches) is the width of the curve
YSPAN (Real inches) is the height of the curve
I (Integer) is the number of the point on the curve
N (Integer) is the maximum number of points on the curve

EXAMPLE
C=XPARAB(4.0, 4.0, 20, 100)

This statement sets C to the X-coordinate of the 20th point on a parabola consisting of 100 points

NOTES
1. No plotting takes place

(1) 28.03.75
NAME XPERSP (X-perspective)

FUNCTION Transforms a 3-d point onto a 2-d picture plane

ARGUMENTS $(X,Y,Z)$

$X,Y,Z$ (Real) are the coordinates of a point in 3-d space

EXAMPLE $XW=\text{XPERSP}(1.0,1.0,1.0)$

This statement sets $XW$ to the $X$-coordinate of the point $(1.0,1.0,1.0)$ on the picture plane

NOTES

1. The position of the eye must be specified by a call on eye or fishi
2. Xpersp is a real function

(1)26.03.75
NAME XPOLY (X-coordinate of a Polygon)

FUNCTION Returns the X-coordinate of a point on a polygon

ARGUMENTS (RADIUS, I, N)

RADIUS (Real inches) is the radius of the polygon
I (Integer) is the position on the polygon
N (Integer) is the number of points on the polygon

EXAMPLE X=XPOLY(2.0,6,8)
This statement sets X to the X-coordinate of the 6th point on a polygon consisting of 8 points

NOTES
1. No plotting takes place
2. If there are N points on the polygon there are N-1 sides
NAME XROT (X-coordinate rotated)

FUNCTION Returns the X-coordinate of a point rotated theta degrees

ARGUMENTS \((X,Y,XP,YP,\theta)\)
- \(X,Y\) (Real inches) are the coordinates of the point to be rotated
- \(XP,YP\) (Real inches) are the coordinates of the centre of rotation
- \(\theta\) (Real degrees) is the angle of rotation
  - \(-\text{VE} = \text{clockwise} \quad +\text{VE} = \text{anti-clockwise}\)

EXAMPLE \(X=X\text{ROT}(1.0,1.0,0.0,0.0,45.0)\)
This statement sets \(X\) to the X-coordinate of the point \((1.0,1.0)\) rotated about the origin 45.0 degrees

NOTES
1. No plotting takes place

\((1)26.03.75\)
NAME   XSIN (X-coordinate of a sinewave)

FUNCTION Returns the X-coordinate of a sinewave

ARGUMENTS \((\text{AMP, WAVE, ANGST, ANGFIN, I, N})\)

\text{AMP} (Real inches) is the amplitude of the sinewave
\text{WAVE} (Real inches) is the length of the sinewave
\text{ANGST} (Real degrees) is the starting angle of the sinewave
\text{ANGFIN} (Real degrees) is the finishing angle of the sinewave
\text{I} (Integer) is the number of the point on the sinewave relative to \text{N}
\text{N} (Integer) is the number of points on the sinewave

EXAMPLE \(x = \text{XSIN}(2.0, 4.0, 0.0, 360.0, 20, 100)\)
This statement sets \(x\) to the X-coordinate of the 20th point on a sinewave containing 100 points

NOTES 1. No plotting takes place

(1)26.03.75
NAME  XYZLIN (X, Y or Z-coordinate of a line)

FUNCTION  Returns the X, Y or Z-coordinate of a point on a line

ARGUMENTS  \((C1, C2, I, N)\)

- \(C1\) (Real inches) is the X, Y or Z-coordinate of one end of the line
- \(C2\) (Real inches) is the X, Y or Z-coordinate of the other end of the line
- \(I\) (Integer) is the number of the point on the line
- \(N\) (Integer) is the number of points on the line

EXAMPLE  \(Y = \text{XYZLIN}(0.0, 4.0, 20, 100)\)

This statement sets \(Y\) to the 20th value between 0.0 and 4.0 when there exists 100 values

NOTES  1. No plotting takes place
        2. The returned value may be interpreted as an X, Y or Z value
NAME YARC (Y-coordinate of an arc)

FUNCTION Returns the Y-coordinate of a point on an arc

ARGUMENTS \((x, y, \text{RADIUS, THETA, PHI, I, N})\)

\(x, y\) (Real inches) are the coordinates of the centre of revolution

\(\text{RADIUS}\) (Real inches) is the radius of curvature

\(\text{THETA}\) (Real degrees) is the starting angle of the arc

\(\text{PHI}\) (Real degrees) is the swept angle in a clockwise direction

\(I\) (Integer) is the position on the curve relative to \(N\)

\(N\) (Integer) is the number of points on the curve

EXAMPLE \(Y = \text{YARC}(0.0, 0.0, 2.0, 180.0, 90.0, 2, 100)\)

This statement sets \(Y\) to the Y-coordinate of the 2nd point of an arc consisting of 100 points

NOTES 1. No plotting takes place

(1)28.03.75
PICASO SYSTEM

YCOORD

NAME
YCOORD (Y-coordinate)

FUNCTION
Is a function returning the Y-coordinate of a specified point in a Picaso structure

ARGUMENTS (ARRAY, NL, NP)

ARRAY (Real array) stores a Picaso structure
NL (Integer) references the contour or surface containing the point
NP (Integer) references the point

EXAMPLE
Y = YCOORD(BOX, 1, 1)
   This statement returns the Y-coordinate of the first point on the first contour of box

NOTES
1. No plotting takes place
2. Ycoord is a real type function
NAME  YELLIP (Y-coordinate of an ellipse)

FUNCTION  Returns the Y-coordinate of a specific point on an ellipse

ARGUMENTS (MAJOR, MINOR, I, N)

MAJOR  (Real inches) is the major radius of the ellipse
MINOR  (Real inches) is the minor radius of the ellipse
I  (Integer) is the number of the point on the ellipse
N  (Integer) is the maximum number of points on the curve

EXAMPLE  C = YELLIP(3.0, 2.0, 20, 100)
           This statement sets C to the 20th Y-coordinate of an ellipse containing 100 points

NOTES  1. No plotting takes place
        2. The curve is clockwise

(1) 26.03.75
NAME YEPIT (Y-coordinate of an epitrochoid)

FUNCTION Returns the Y-coordinate of a specific point on an epitrochoid

ARGUMENTS (RADFIX, RADMOV, DIST, I, N)
- RADFIX (Real inches) is the radius of the fixed circle
- RADMOV (Real inches) is the radius of the moving circle
- DIST (Real inches) is the distance of the traced point that exists on the moving circle to its centre
- I (Integer) is the number of the point on the curve
- N (Integer) is the maximum number of points on the curve

EXAMPLE C=YEPIT(3.0,1.0,0.5,20,100)
This statement sets C to the 20th Y-coordinate of an epitrochoid containing 100 points

NOTES
1. No plotting takes place
2. The curve is clockwise
NAME YHYPER (Y-coordinate of a hyperbola)

FUNCTION Returns the Y-coordinate of a specific point on a hyperbola

ARGUMENTS (SPAN,DIST,I,N)
  SPAN (Real inches) is the width and height of the curve relative to the X and Y axes
  DIST (Real inches) is the distance of the curve from the X or Y axis at the limits of the curve
  I (Integer) is the number of the point on the curve
  N (Integer) is the maximum number of points on the curve

EXAMPLE C= YHYPER(2.0,0.5,20,100)
  This statement sets C to the 20th Y-coordinate of a hyperbola consisting of 100 points

NOTES 1. No plotting takes place

(1)26.03.75
NAME

YHYPOT (Y-coordinate of an hypotrochoid)

FUNCTION

Returns the Y-coordinate of a specific point on a hypotrochoid

ARGUMENTS \((RADFIX, RADMOV, DIST, I, N)\)

\(RADFIX\) (Real inches) is the radius of the fixed circle

\(RADMOV\) (Real inches) is the radius of the moving circle

\(DIST\) (Real inches) is the distance of the traced point on the moving circle to its centre

\(I\) (Integer) is the number of the point on the curve

\(N\) (Integer) is the maximum number of points on the curve

EXAMPLE

\(C = \text{YHYPOT}(3.0, 1.0, 0.5, 20, 100)\)

This statement sets \(C\) to the 20th Y-coordinate of a hypotrochoid consisting of 100 points

NOTES

1. No plotting takes place
2. The curve is clockwise

(1) 26.03.75
NAME  YLISS (Y-coordinate of a lissajous curve)

FUNCTION  Returns the Y-coordinate of a specific point on a lissajous curve

ARGUMENTS  \((XSPAN, YSPAN, F, D, I, N)\)

- \(XSPAN\) (Real inches) is the width of the curve
- \(YSPAN\) (Real inches) is the height of the curve
- \(F\) (Real) is a factor affecting the number of loops in the 'X' direction
- \(D\) (Real radians) is a displacement value
- \(I\) (Integer) is the number of the point on the curve
- \(N\) (Integer) is the maximum number of points on the curve

EXAMPLE  \(C=\text{YLISS}(2.0, 2.0, 1.0, 0.0, 20, 100)\)

This statement sets \(C\) to the 20th Y-coordinate of a lissajous curve consisting of 100 points

NOTES  1. No plotting takes place
        2. The curve is clockwise

(1) 26.03.75
NAME YLSPIR (Y-coordinate of a logarithmic spiral)

FUNCTION Returns the Y-coordinate of a specific point on a logarithmic spiral

ARGUMENTS \( (RADIUS, CYCLES, I, N) \)
- \( RADIUS \) (Real inches) is the final radius of the spiral
- \( CYCLES \) (Real) is the number of cycles in the spiral
- \( I \) (Integer) is the number of the point on the spiral
- \( N \) (Integer) is the maximum number of points on the spiral

EXAMPLE \( C=YLSPIR(3.0, 2.0, 20, 100) \)
This statement sets \( C \) to the 20\textsuperscript{th} Y-coordinate of a spiral consisting of 100 points

NOTES
1. No plotting takes place
2. The curve is clockwise

(1) 26.03.75
NAME
YMIRRO (Y-coordinate of a mirror-image)

FUNCTION
Returns the Y-coordinate of a reflected point

ARGUMENTS $(X, Y, X_1, Y_1, X_2, Y_2)$
- $X, Y$ (Real inches) are the coordinates of the point to be reflected
- $X_1, Y_1$ (Real inches) are the coordinates of one point on the mirror
- $X_2, Y_2$ (Real inches) are the coordinates of another point on the mirror

EXAMPLE
$c = YMIRRO(1.0, 2.0, 2.0, 0.0, 2.0, 1.0)$
This statement sets $c$ to the Y-coordinate of the reflected point $1.0, 2.0$.

NOTES
1. No plotting takes place

(1) 26.03.75
NAME YPARAB (Y-coordinate of a parabola)

FUNCTION Returns the Y-coordinate of specific point on a parabola

ARGUMENTS (XSPAN, YSPAN, I, N)
- XSPAN (Real inches) is the width of the curve
- YSPAN (Real inches) is the height of the curve
- I (Integer) is the number of the point on the curve
- N (Integer) is the maximum number of points on the curve

EXAMPLE C = YPARAB(4.0, 4.0, 20, 100)
This statement sets C to the Y-coordinate of the 20th point on a parabola consisting of 100 points

NOTES 1. No plotting takes place

(1) 26.03.75
NAME YPERSP (Y-perspective)

FUNCTION Transforms a 3-d point onto a 2-d picture plane

ARGUMENTS $(X,Y,Z)$

$X,Y,Z$ (Real) are the coordinates of a point in 3-d space

EXAMPLE $YW = YPERSP(1.0,1.0,1.0)$

This statement sets $YW$ to the $Y$-coordinate of the point $(1.0,1.0,1.0)$ on the picture plane

NOTES
1. The position of the eye must be specified by a call on `eye` or `fishi`
2. Ypersp is a real function

(1)26.03.75
NAME  
YPOLY (Y-coordinate of a polygon)

FUNCTION  
Returns the Y-coordinate of a point on a polygon

ARGUMENTS (RADIUS, I, N)

RADIUS  
(Real inches) is the radius of the polygon

I  
(Integer) is the position on the polygon relative to N

N  
(Integer) is the number of points on the polygon

EXAMPLE  
Y=YPOLY(2.0, 6, 8)

This statement sets Y to the Y-coordinate of the 6th point on a polygon consisting of 8 points

NOTES  
1. No plotting takes place

2. If there are N points on the polygon there are N-1 sides

(1)26.03.75
NAME YROT (Y-coordinate rotated)

FUNCTION Returns the Y-coordinate of a point rotated theta degrees

ARGUMENTS \((X,Y,XP,YP,\text{THETA})\)

\(X,Y\) (Real inches) are the coordinates of the point to be rotated

\(XP,YP\) (Real inches) are the coordinates of the centre of rotation

\(\text{THETA}\) (Real degrees) is the angle of rotation

-VE = clockwise +VE = anticlockwise

EXAMPLE \(Y=YROT(1.0,1.0,0.0,0.0,45.0)\)

This statement sets \(Y\) to the Y-coordinate of the point \((1.0,1.0)\) rotated about the origin 45.0 degrees

NOTES 1. No plotting takes place

(1) 26.03.75
NAME  YSIN (Y-coordinate of a sinewave)

FUNCTION Returns the Y-coordinate of a sinewave

ARGUMENTS \((\text{AMP}, \text{WAVE}, \text{ANGST}, \text{ANGFIN}, I, N)\)

- **AMP** (Real inches) is the amplitude of the sinewave
- **WAVE** (Real inches) is the length of the sinewave
- **ANGST** (Real degrees) is the starting angle of the sinewave
- **ANGFIN** (Real degrees) is the finishing angle of the sinewave
- **I** (Integer) is the number of the point on the curve
- **N** (Integer) is the number of points on the sinewave

EXAMPLE  \(Y = \text{YSIN}(2.0, 4.0, 0.0, 360.0, 20, 100)\)

This statement sets \(Y\) to the Y-coordinate of the 20th point on a sinewave containing 100 points

NOTES  1. No plotting takes place

(1) 26.03.75
NAME ZCOORD (Z-coordinate)

FUNCTION Returns the Z-coordinate of a specified vertex on a 3-d Picaso object

ARGUMENTS (ARRAY, NL, NP)
- ARRAY (Real array) stores a 3-d Picaso object
- NL (Integer) is the surface containing the point
- NP (Integer) is the number of the point on the nl'th surface

EXAMPLE Z=ZCOORD(S, 1, 1)
This statement sets Z to the Z-coordinate of the first point on the object

(1)28.03.75
PICASO SYSTEM ERROR MESSAGES

1. DRAW DIMENSION OF SHAPE ARRAY ≠ 2.0
2. DRAW CONTOURS IN SHAPE ARRAY < 1.0
3. DSHAPE DIMENSION OF SHAPE ARRAY ≠ 2.0
4. ISOMET DIMENSION OF CONTOUR ARRAY ≠ 2.0
5. MODSH DIMENSION OF MODULATING SHAPE ARRAY ≠ 2.0
6. MODSH DIMENSION OF SHAPE ARRAY ≠ 2.0
7. XCOORD INVALID VERTEX NUMBER
8. YCOORD INVALID VERTEX NUMBER
9. ZCOORD INVALID VERTEX NUMBER
10. GROW2D DIMENSION OF SHAPE ARRAY ≠ 2.0
11. GROW2D CONTOURS IN SHAPE ARRAY < 1.0
12. TURN3D DIMENSION OF SHAPE ARRAY ≠ 3.0
13. ROTATE DIMENSION OF SHAPE ARRAY ≠ 2.0
14. TRANSH DIMENSION OF SHAPE ARRAYS UNEQUAL
15. TRANSH DIMENSION OF SHAPES ≠ 2.0 OR 3.0
16. TRANSH INVALID CONTOUR NUMBERS IN FIRST SHAPE
17. TRANSH INVALID CONTOUR NUMBERS IN SECOND SHAPE
18. CLOK3D DIMENSION OF SHAPE CONTOUR ≠ 3.0
19. REMOVE INVALID CONTOUR NUMBER
20. NORMAL DIMENSION OF SHAPE ARRAY ≠ 2.0
21. NORMAL INVALID LINE NUMBER
22. NORMAL INVALID POINT NUMBER
23. EXTEND DIMENSION OF SHAPE ARRAYS UNEQUAL
24. EXTEND INVALID CONTOUR NUMBER
25. SHIFT DIMENSION OF SHAPE ARRAY ≠ 2.0
26. EXPLOD DIMENSION OF SHAPE ARRAY ≠ 2.0
27. MIRROR DIMENSION OF SHAPE ARRAY ≠ 2.0
28. MIRROR CONTOURS IN SHAPE ARRAY < 1.0
29. PULL DIMENSION OF SHAPE ARRAY ≠ 2.0
30. PULL CONTOURS IN SHAPE ARRAY < 1.0
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Hull dimension of shape array ≠ 2.0</td>
</tr>
<tr>
<td>32</td>
<td>Hull contours in shape array &lt; 1.0</td>
</tr>
<tr>
<td>33</td>
<td>Cycle dimension of shape array ≠ 2.0</td>
</tr>
<tr>
<td>34</td>
<td>Cycle contours in shape array &lt; 1.0</td>
</tr>
<tr>
<td>35</td>
<td>Extsh dimension of shape array ≠ 2 or 3</td>
</tr>
<tr>
<td>36</td>
<td>Fill dimension of shape array ≠ 2.0</td>
</tr>
<tr>
<td>37</td>
<td>Fill contours in shape array &lt; 1.0</td>
</tr>
<tr>
<td>38</td>
<td>Object dimension of shape array ≠ 3.0</td>
</tr>
<tr>
<td>39</td>
<td>Object contours in shape array &lt; 1.0</td>
</tr>
<tr>
<td>40</td>
<td>Stick dimension of shape array ≠ 2.0</td>
</tr>
<tr>
<td>41</td>
<td>Stick dimension of 'stick' shape array ≠ 2.0</td>
</tr>
<tr>
<td>42</td>
<td>Smooth periodic spline error</td>
</tr>
<tr>
<td>43</td>
<td>Row dimension of shape array ≠ 2.0</td>
</tr>
<tr>
<td>44</td>
<td>Join dimension of shape arrays unequal</td>
</tr>
<tr>
<td>45</td>
<td>Join dimension of shapes ≠ 2,0 or 3,0</td>
</tr>
<tr>
<td>46</td>
<td>Conect dimension of shape array ≠ 2,0</td>
</tr>
<tr>
<td>47</td>
<td>Conect contours in shape array &lt; 1,0</td>
</tr>
<tr>
<td>48</td>
<td>Draw2d dimension of shape array ≠ 2,0</td>
</tr>
<tr>
<td>49</td>
<td>Draw2d contours in shape array &lt; 1,0</td>
</tr>
<tr>
<td>50</td>
<td>Draw3d dimension of shape array ≠ 3,0</td>
</tr>
<tr>
<td>51</td>
<td>Draw3d contours in shape array ≠ 1,0</td>
</tr>
<tr>
<td>52</td>
<td>Mpysb dimension of shapes unequal</td>
</tr>
<tr>
<td>53</td>
<td>Mpysb invalid contour for arraya</td>
</tr>
<tr>
<td>54</td>
<td>Mpysb invalid contour for arrayb</td>
</tr>
<tr>
<td>55</td>
<td>Slope dimension of shape array ≠ 2,0</td>
</tr>
<tr>
<td>56</td>
<td>Slope invalid contour number</td>
</tr>
<tr>
<td>57</td>
<td>Slope invalid vertex number</td>
</tr>
<tr>
<td>58</td>
<td>Mpysb invalid operator number</td>
</tr>
<tr>
<td>59</td>
<td>Matop dimension of shape array ≠ 2,0</td>
</tr>
<tr>
<td>60</td>
<td>Surfacer dimension of shape array ≠ 2,0</td>
</tr>
</tbody>
</table>

(1) 26.03.75
<table>
<thead>
<tr>
<th></th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>61</td>
<td>REVERS   DIMENSION OF SHAPE ARRAY ≠ 2.0</td>
</tr>
<tr>
<td>62</td>
<td>REVERS   INVALID CONTOUR NUMBER</td>
</tr>
<tr>
<td>63</td>
<td>MATOP2   CONTOURS IN SHAPE ARRAY &lt; 1.0</td>
</tr>
<tr>
<td>65</td>
<td>INSIDE   DIMENSION OF SHAPE ARRAY ≠ 2.0</td>
</tr>
<tr>
<td>66</td>
<td>INSIDE   CONTOURS IN SHAPE ARRAY &lt; 1.0</td>
</tr>
<tr>
<td>67</td>
<td>TRACE    DIMENSION OF SHAPE ARRAY ≠ 2.0</td>
</tr>
<tr>
<td>68</td>
<td>TRACE    CONTOURS IN SHAPE ARRAY &lt; 1.0</td>
</tr>
<tr>
<td>69</td>
<td>SILUET   DIMENSION OF SHAPE ARRAY ≠ 2.0</td>
</tr>
<tr>
<td>70</td>
<td>SILUET   CONTOURS IN SHAPE ARRAY &lt; 1.0</td>
</tr>
<tr>
<td>71</td>
<td>SIZE     DIMENSION OF SHAPE ARRAY ≠ 2.0</td>
</tr>
<tr>
<td>72</td>
<td>SIZE     CONTOURS IN SHAPE ARRAY &lt; 1.0</td>
</tr>
<tr>
<td>73</td>
<td>SNOW     DIMENSION OF SHAPE ARRAY ≠ 2.0</td>
</tr>
<tr>
<td>74</td>
<td>SNOW     CONTOURS IN SHAPE ARRAY &lt; 1.0</td>
</tr>
<tr>
<td>77</td>
<td>FILL2D   DIMENSION OF SHAPE ARRAY ≠ 2.0</td>
</tr>
<tr>
<td>78</td>
<td>FILL2D   CONTOURS IN SHAPE ARRAY &lt; 1.0</td>
</tr>
<tr>
<td>79</td>
<td>CLOCK    DIMENSION OF SHAPE ARRAY ≠ 2.0</td>
</tr>
<tr>
<td>80</td>
<td>SMOOTH   DIMENSION OF SHAPE ARRAY ≠ 2.0</td>
</tr>
<tr>
<td>81</td>
<td>SMOOTH   CONTOURS IN SHAPE &lt; 1.0</td>
</tr>
<tr>
<td>82</td>
<td>SMOOTH   CHORDAL DISTANCES NOT INCREASING</td>
</tr>
<tr>
<td>83</td>
<td>SMOOTH   CHORDAL DISTANCES NOT INCREASING</td>
</tr>
<tr>
<td>84</td>
<td>CLOSED   DIMENSION OF ARRAY ≠ 2.0</td>
</tr>
<tr>
<td>85</td>
<td>CLOSED   INVALID CONTOUR NUMBER</td>
</tr>
<tr>
<td>86</td>
<td>WINDOW   DIMENSION OF ARRAY ≠ 2.0</td>
</tr>
<tr>
<td>87</td>
<td>WINDOW   CONTOURS IN SHAPE ARRAY &lt; 1.0</td>
</tr>
<tr>
<td>88</td>
<td>SHIFT3   CONTOURS IN OBJECT ARRAY &lt; 1.0</td>
</tr>
<tr>
<td>89</td>
<td>SHIFT3   DIMENSION OF ARRAY ≠ 3.0</td>
</tr>
</tbody>
</table>

(1)26.03.75
<table>
<thead>
<tr>
<th></th>
<th>PICASO SYSTEM ERROR MESSAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>91</td>
<td>TRANSP Dimension of array = 2.0</td>
</tr>
<tr>
<td>92</td>
<td>TRANSP Contours in shape array &lt; 1.0</td>
</tr>
<tr>
<td>93</td>
<td>NYSUB Dimension of array ≠ 2 or 3</td>
</tr>
<tr>
<td>94</td>
<td>NYSUB Contour argument too large</td>
</tr>
<tr>
<td>95</td>
<td>NXSUB Dimension of array ≠ 2 or 3</td>
</tr>
<tr>
<td>96</td>
<td>NXSUB Contour argument too large</td>
</tr>
<tr>
<td>97</td>
<td>FIT Dimension of array ≠ 2</td>
</tr>
<tr>
<td>98</td>
<td>FIT Contours in shape &lt; 1.0</td>
</tr>
<tr>
<td>99</td>
<td>NZSUB Dimension of array ≠ 2 or 3</td>
</tr>
<tr>
<td>100</td>
<td>NZUB Contour argument too large</td>
</tr>
<tr>
<td>101</td>
<td>HATCH Dimension of shape array ≠ 2.0</td>
</tr>
<tr>
<td>102</td>
<td>HATCH Contours in shape &lt; 1.0</td>
</tr>
<tr>
<td>103</td>
<td>MASK Dimension of shape array ≠ 2.0</td>
</tr>
<tr>
<td>104</td>
<td>MASK Contours in shape &lt; 1.0</td>
</tr>
<tr>
<td>105</td>
<td>RIPPLE Dimension of array ≠ 2.0</td>
</tr>
<tr>
<td>106</td>
<td>RIPPLE Invalid contour</td>
</tr>
<tr>
<td>107</td>
<td></td>
</tr>
<tr>
<td>108</td>
<td></td>
</tr>
<tr>
<td>109</td>
<td>THICK Dimension of shape array ≠ 2.0</td>
</tr>
<tr>
<td>110</td>
<td>THICK Contours in shape array &lt; 1.0</td>
</tr>
<tr>
<td>111</td>
<td>DOTUET Dimension of contour array ≠ 2.0</td>
</tr>
<tr>
<td>112</td>
<td>DOTUET Contours on contour array &lt; 1.0</td>
</tr>
<tr>
<td>113</td>
<td>PERSHP Dimension of shape array ≠ 2.0</td>
</tr>
<tr>
<td>114</td>
<td>PERSHP Contours in shape &lt; 1.0</td>
</tr>
<tr>
<td>115</td>
<td>MIX2D Dimension of shape array ≠ 2.0</td>
</tr>
<tr>
<td>116</td>
<td>MIX2D Contours in shape &lt; 1.0</td>
</tr>
<tr>
<td>117</td>
<td>MIX3D Dimension of object array ≠ 3.0</td>
</tr>
<tr>
<td>118</td>
<td>MIX3D Contours in object array &lt; 1.0</td>
</tr>
<tr>
<td>119</td>
<td>ROW3D Dimension of object array ≠ 3.0</td>
</tr>
<tr>
<td>120</td>
<td>ROW3D Contours in object &lt; 1.0</td>
</tr>
<tr>
<td>121</td>
<td>FILL3D Dimension of object array ≠ 3.0</td>
</tr>
<tr>
<td>122</td>
<td>FILL3D Contours in object &lt; 1.0</td>
</tr>
</tbody>
</table>

(1)26.03.75