A COMPUTER AIDED DESIGN SYSTEM WITH

PARAMETRIC DIMENSIONING

A thesis presented

in partial fulfilment of the requirements

for the degree

of Master of Philosophy in Industrial Technology

at

Massey University

Brian Morris Meads

1987

ABSTRACT

This thesis develops the concept of a parametrically dimensioned CAD system. Conventional CAD systems require the actual dimensions of all objects drawn to be defined during the drawing process. To alter any dimension requires manual modification of all affected objects in the drawing. Parametrically dimensioned CAD systems would allow drawings to be constructed containing dimensions defined using variable parameters. These parametric drawings could then be fully specified at some later stage by supplying actual values for the parameters. Such systems would allow drawings of families of components (that varied only in their dimensions) to be easily produced from a single parametric drawing, would simplify dimensional modifications to drawings, and would permit the drawing production to be part of an automated design process.

The general requirements for such a parametric CAD system are developed in the thesis and the implementation of a limited package based on these ideas is described. On the basis of this work, it has been concluded that such systems are viable, could have successful user interfaces and would be a valuable extension to conventional CAD packages.

ACKNOWLEDGEMENTS

The production of this thesis has been made posssible by the co-operation and help of a number of people.

First and foremost, my sincere thanks go to my supervisor, Professor Mark Apperley, for his guidance and suggestions during the investigations into parametric CAD and for his helpful criticisms in the presentation of this thesis.

I would like to extend special thanks to Mr Len Chisholm for his assistance with the Assembler routines and the hardware interfacing in Paracad and for his helpful suggestions during the development of Paracad.

My thanks also go to Mr Ralph Ball and Mr David Morgan who took over a considerable amount of my lecturing related workload thus enabling me to devote sufficient time to carry out my studies towards this thesis.

Finally I would like to extend a very special thank you to my wife Eileen and my children Jason, Darron and Kirsty for their patience, love and understanding. For this I am truly grateful.

Brian Meads

CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Computer Aided Design (CAD) systems are responsible for major productivity gains in drawing and design operations. Up until the 1980's CAD systems ran only on mainframes and were expensive to purchase and run. This limited their use to such areas as the aerospace, automobile and electronics industries. Major increases in the performance of computer systems over the last decade have resulted in real time CAD functions that were previously only performed on mainframe computers migrating down through minicomputers to microcomputers. This has caused a substantial increase in the number of potential computers on which CAD packages can be run and has resulted in strong competition between CAD software suppliers. This competition is manifesting itself in increasingly sophisticated CAD features on microcomputer systems that are tailored to the end user's requirements becoming available **[Wohl. 1984, Myer. 1985]**. The basis of this thesis is the investigation of one such feature about which there has been little published research.

The three major areas of use of CAD packages are in electrical and electronic design, mechanical engineering design and architectural/layout design **[Merm. 1980]**. A common

output from each of these areas is the production of a drawing from a plotter.

Many drawings that are produced in practice are similar, varying only in their dimensions. This is especially true of component drawings. As an example, consider the the two drawings shown in Figure 1-1. These show the front cabinet shape of two different sized television sets. The cabinets have different widths, heights, screen sizes and speaker cover sizes and placements.

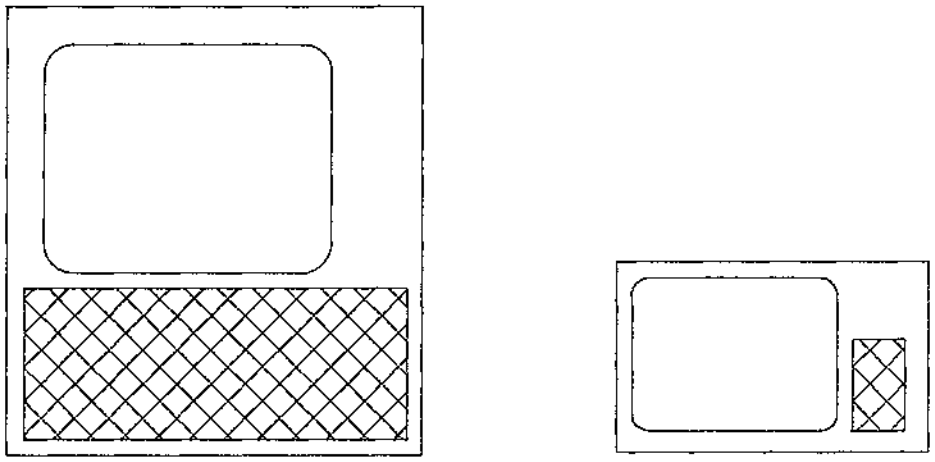


Figure 1-1 Front Cabinet of Two Different Television Sets

This thesis examines the feasibility of having a CAD package that allows a designer to prepare a template or "parametric" drawing with some or all dimensions defined in terms of variable parameters. Hereafter such a drawing will be referred to as a **parametric drawing**. Final specific drawings could then be produced by supplying values for each of the

parameters for that particular drawing. Such a drawing will hereafter be referred to as a **particular drawing**.

A parametric drawing covering the family of television sets similar to those in Figure 1-1 might appear something like that in Figure 1-2. Each different model of television set would have its own particular drawing with the actual values entered for the various dimensions a, b, c etc. being different in each case.

In addition to allowing easy generation of particular drawings for families of components (or models) from a single parametric drawing, with the consequent time savings, parametric CAD could also be used as part of an automated design system. It could also permit rapid "what if" tests to be made on designs.

There appears to be no published evidence of research in the area of parametric dimensioning. It is suspected that this is because the only research in the area has been done by CAD software houses who wish to keep their results confidential. Because of the lack of published research in the area, this thesis attempts to lay general foundations for a parametric CAD system rather than concentrating on narrow specialised areas within such a system.

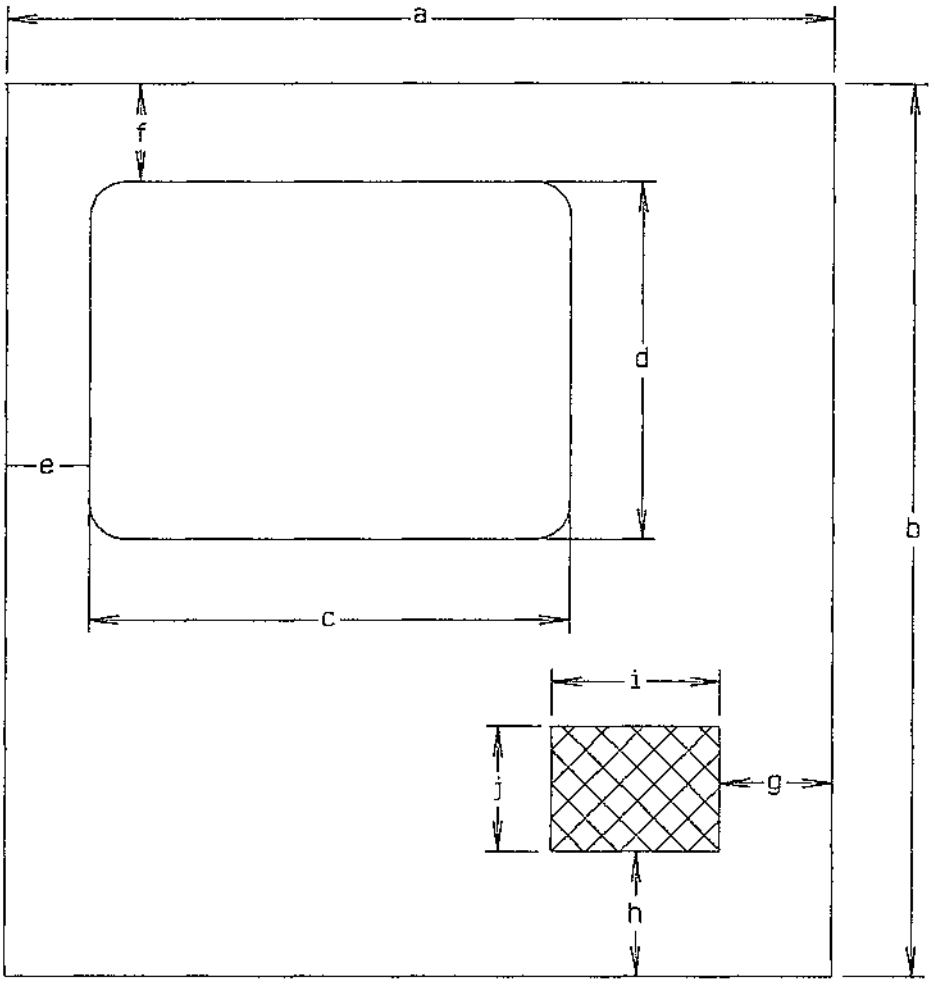


Figure 1-2 Parametric Drawing of Television Cabinet

In Chapter 2 the hardware and software features of conventional CAD packages are examined where these are relevant to a parametric CAD system.

In Chapter 3 the extra design considerations and decisions necessary for parametric CAD packages are considered and other non-essential but highly desirable additional features are contemplated.

In Chapter 4 design decisions for a specific implementation of a parametric CAD package known as **Paracad** are discussed. Paracad is used as a basis for investigating the feasibility of parametric CAD.

The Paracad environment is described in Chapter 5. This covers the hardware and software environment, interfacing between hardware elements, data structures used and the method of storing these data structures.

In Chapter 6 the interface between Paracad and the user is explored. This includes the way Paracad responds to user requests and the algorithms used to perform the actions required by the user.

The performance of Paracad is discussed in Chapter 7 in terms of its speed of operation and user friendliness. Future Paracad developments and areas for further parametric CAD research are also described.

In Chapter 8 conclusions are made as to the feasibility of parametric CAD and the advantages of such parametric CAD systems.

Appendix A contains a description of the notation used in the illustrations in this thesis.

CHAPTER 2

REVIEW OF CONVENTIONAL CAD PACKAGES

Before making any decisions on the design of a parametric based CAD system it is pertinent to examine the typical features of current conventional CAD packages.

Conventional CAD packages provide a host of design features, some of which are very specialised (eg. printed circuit board routing) and some of which are general to most packages (eg. adding a line to a drawing). For the purposes of this thesis, only those CAD features that are of a general nature will be considered.

The components of a general CAD system can be broken down into hardware features and software features. In addition to these, the method of database storage used is pertinent to this thesis [Bes. 1983].

2.1 Hardware Features

The major hardware components of a CAD system are a graphics display system, a central processing unit, a graphics input device and a graphics output device for hardcopy output.

2.1.1 Graphics Display

By far the most common graphics display is a raster scan cathode ray tube with some form of graphics processing hardware controlling it. There is a strong tendency towards colour systems for serious CAD work with screen resolution being anything from 640 by 200 pixels upwards. Fast display processing ability is a primary requirement for real time CAD systems. Many CAD systems include an option to run a dual screen arrangement in which a fast, high resolution colour display is used for graphics while a standard monochrome alphanumeric display is used for textual information (eg. Versacad by T&W Systems).

2.1.2 Central Processing Unit

The central processing unit is some form of computer (mainframe, mini or micro) with keyboard, primary and secondary storage etc. CAD systems are notoriously "processor hungry" - that is they make considerable demands on the processor and so a fast central processing unit is a high priority in a CAD installation.

2.1.3 Graphics Input Device

The graphics input device most commonly used in CAD systems is some form of digitising tablet. This usually consists of a flat rectangular base with a movable puck or stylus. The position of the puck (or stylus) on the tablet base is detected either magnetically, electrically or optically and this positional information is passed to the CAD program - typically to control the position of a cursor on the graphics screen or to select a command for the software to action. Tablet base sizes range from 25cm x 25cm (12" x 12") through to 121cm x 121cm (48" x 48") and beyond with the tendency towards the smaller sizes for most non-specialist CAD work. Positional accuracy is typically 0.002cm (0.001"). Lower positional accuracy would generally be acceptable since graphics display device resolutions cannot approach digitiser resolutions of this accuracy. Ergonomic design of the graphics input device can be an important factor during extended sessions using a CAD system. Digitiser response speed is virtually instantaneous and is usually insignificant when compared to the time taken in analysis of the digitiser data and displaying of the graphics.

2.1.4 Graphics Output Device

The graphics hardcopy output device is generally a pen plotter. These devices range in paper size from A4 through to

A1 and some models provide multiple colours. Accuracy can be up to 0.002cm with pen speeds in excess of 1 m/s and accelerations of 4g (40m/s/s). Slower plotter speeds are often acceptable as plotter files can be batched for later spooling. Many installations have found the output quality and general readability of plotter generated output to be such an improvement over manual drawing that a smaller drawing size can be used (eg. A2 size drawings are acceptable from a plotter where A1 size drawings were needed for manual drawings).

2.2 Software Features

For the purposes of this thesis, only two dimensional CAD systems will be considered. The major software aspects of general 2D CAD packages can be broken down into the following areas: primitives, input of primitives, groups, modifying primitives or groups, copying primitives or groups, transformations on primitives or groups, viewing control, output and method of driving. Most CAD packages also provide various extra features that are not mentioned here.

2.2.1 Primitives

Primitives (also called entities or objects) are the basic graphic entities that may be added to a drawing. These

typically include lines, circles, arcs, rectangles, polygons, Bezier or B-spline curves and text. Many packages allow primitives to have various properties attached to them such as colour, linestyle (solid, dotted etc.), plotter pen number and textual information. Primitives are generally stored as sets of real-numbered co-ordinates, for example a straight line would be stored as its starting co-ordinates and its ending co-ordinates together with any associated properties. Often double precision real numbers are used to allow storage of a wide range of potential primitive sizes with high accuracy.

2.2.2 Input of Primitives

A variety of different methods of specifying the co-ordinates of a primitive during the input stage are usually provided to enable flexibility and ease of data entry. The most common methods used for specifying the co-ordinates of a particular point are :

digitiser - the position of the digitiser puck or stylus corresponds to some real world position, and the co-ordinates of this position are used for the co-ordinates of the point.

absolute - two real numbers are given, via the keyboard, for the actual co-ordinates of the required point.

relative - two real numbers are given, via the keyboard, for the x and y offset of the required point from some reference point (usually the last point entered).

polar - two real numbers are given, via the keyboard, for the angle and distance of the required point from some reference point (usually the last point entered).

primitive snap - the required point is specified as being some particular point on an existing primitive (eg. one end of a selected line - or its midpoint).

In addition to having a number of different ways of specifying a particular point, a choice is often provided in the way a primitive is entered. The normal entry method for a line is to specify each endpoint, but an alternative is to specify a start point, give another line it is to be parallel (or normal) to, and specify its length. For a circle it is possible to specify the endpoints of a diameter, specify the centre and a point on the circumference or specify three points on the circumference. Other such optional input methods exist for other primitives.

2.2.3 Groups

An arbitrary collection of primitives can be collected together to form a group (also called a segment). Groups can also be allocated certain properties in some systems. Often it is more convenient to work with groups rather than the separate primitives that make up the group. For example a collection of lines might form a chair and so when positioning the chair it is a natural action to group these lines together and manipulate the chair as a whole rather than each individual line. Forming groups can be done by "picking" each of the individual primitives that make the group up, drawing an imaginary fence around the required primitives, or selecting all primitives with a particular property.

2.2.4 Modifying Primitives or Groups

Modifications may be made to selected primitives or groups. Common modifications are deleting the primitive or group, changing properties of the primitive or group, or applying some transformation to the primitive or group (discussed later).

2.2.5 Copying Primitives or Groups

Many drawings have repeated features in them and so most CAD packages provide the facility to copy any primitive or group. Copies can be placed at any specified point(s) or may be arranged in a line, in a matrix, or radially in a circle.

2.2.6 Transformations on Primitives and Groups

The common transformations provided are translation (moving in a straight line), rotation, reflection, enlargement (and reduction) and stretching in one dimension.

2.2.7 Viewing Control

A CAD drawing worksheet provides a very large area to draw in. Windowing (also called zooming) in and out on selected areas of the worksheet and panning in any direction is generally provided to allow the designer to keep only the area of interest on the screen. In addition, another property that can be associated with primitives is a level (or layer). Levels can be considered to be overlays, and any particular level can be made visible or invisible. If a level is visible, all primitives on that level (and all other visible levels) will be displayed, otherwise they will not be displayed. This allows selected features of a drawing to be

displayed or hidden. For example a drawing of a building may have all plumbing on one level, electrical wiring on another, structural members on another, furniture on another and so on. The visibility of any of these levels can then be individually controlled to allow only the required features on the screen.

2.2.8 Output

Output involves sending part, or all, of the drawing to the output device (usually a plotter). Control over scaling factors, where the drawing will appear on the paper and orientation on the paper is generally provided. For multipen plotters a link is provided between the plotter pen property for primitives and the various plotter pens.

2.2.9 User Interface

Most CAD packages are either menu-driven or command-driven. With a menu-driven system (eg. Autocad by Autodesk Inc.) a menu of all the allowable commands is displayed on screen and the user selects one (generally by typing the first letter, a number, or selecting it with the digitiser). Menu-driven packages generally have a large number of different menus arranged in a tree structure, so selecting a particular choice from one menu produces a new menu of possible actions.

A command-driven system (eg. Personal Designer by Computervision) does not use menus - instead it has a number of valid commands that are directly entered either from the keyboard or from a command area on the digitiser. Menu-driven systems are generally far easier to learn as all the allowable options at any particular stage are displayed on screen. For experienced users the command-driven systems can provide greater speed as it is not necessary to traverse the menus to get to the required command.

2.3 CAD Database Storage

While many different methods of CAD database storage are used, an important factor is that information is not stored regarding physical interconnections between primitives. For example, suppose line 1 is added to a drawing, then line 2 is added where the startpoint of line 2 is specified as the endpoint of line 1. At entry time the co-ordinates of the startpoint of line 2 can be readily obtained and these are stored with the primitive line 2. No information indicating line 1 and line 2 are connected is stored apart from the common co-ordinate value - and in fact none is needed as once a primitive's position has been specified it is completely defined.

CHAPTER 3

DESIGN CONSIDERATIONS FOR PARAMETRIC CAD SYSTEMS

While a full implementation of a parametric CAD system would embody all the conventional CAD features, it would require a number of enhancements and additional elements. The main additional requirements are a two phase drawing procedure, a method of storing data describing the interconnections between primitives and a check that an "impossible" drawing is not constructed. Also, although it is not an absolute necessity, a provision for allowing "floating" primitives that can "fixed" at a later stage provides a considerable increase in flexibility.

This chapter will consider each of these requirements in greater depth.

3.1 Two Phase Drawing Procedure

With a conventional CAD package, the drawing process is essentially a single phase whereby primitives are added in various positions to make up a drawing and the finished drawing is then plotted out to some required scale.

With a parametric CAD system two distinct phases are required. The first phase involves the construction of the parametric drawing by adding primitives together with their parametric descriptions. The second stage then involves the construction of a particular drawing which takes the parametric drawing as input, gets the actual values required for each parameter value and then constructs the particular drawing which is plotted out to some required scale.

Obviously several different particular drawings may be obtained from a single parametric drawing by specifying different actual values for the parameters - this is the essence of a parametric CAD system. The construction of particular drawings may also continue to occur for many years after the parametric drawing has been constructed.

3.2 Primitive Interconnection Data

A parametric CAD system would allow the positions and dimensions of primitives in a particular drawing to change depending on the actual values supplied for the parameters. This does not happen with conventional CAD systems and it causes some difficulties. For example, consider the parametric drawing shown in Figure 3-1. When a particular drawing is produced, the values x and y need to be supplied. A number of possibilities then exist for the particular drawing depending on whether line connection points (at the

intersection of the line endpoints or along the locus of a line), line positions, line lengths (for the non-parametric lines 3 and 4), or vertex positions (a, b, c and d) are preserved. If the supplied values for each of x and y are such that each line using them will be longer in the particular drawing than in the parametric drawing then some of the possible particular drawings are shown in Figure 3-2.

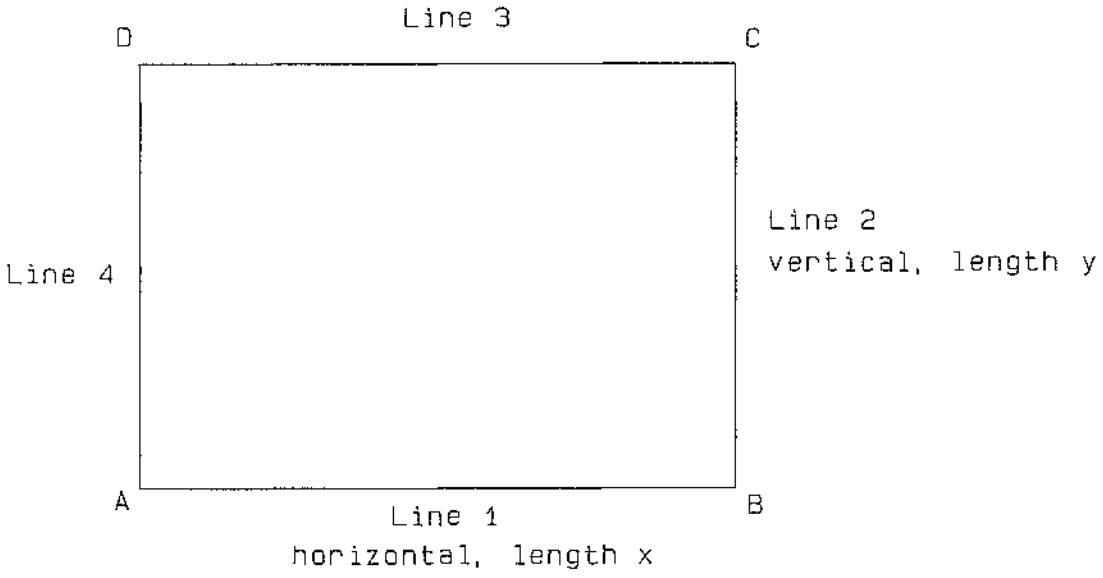


Figure 3-1 Parametric Drawing of four lines

Figure 3-2 part (a) shows the result of preserving the endpoint intersection connections between lines 1 and 2, between lines 2 and 3, between lines 3 and 4 and between lines 4 and 1. The lengths of lines 3 and 4 are not preserved and the positions of lines 2, 3 and 4 are not preserved. The position of point A is preserved while B, C and D are not.

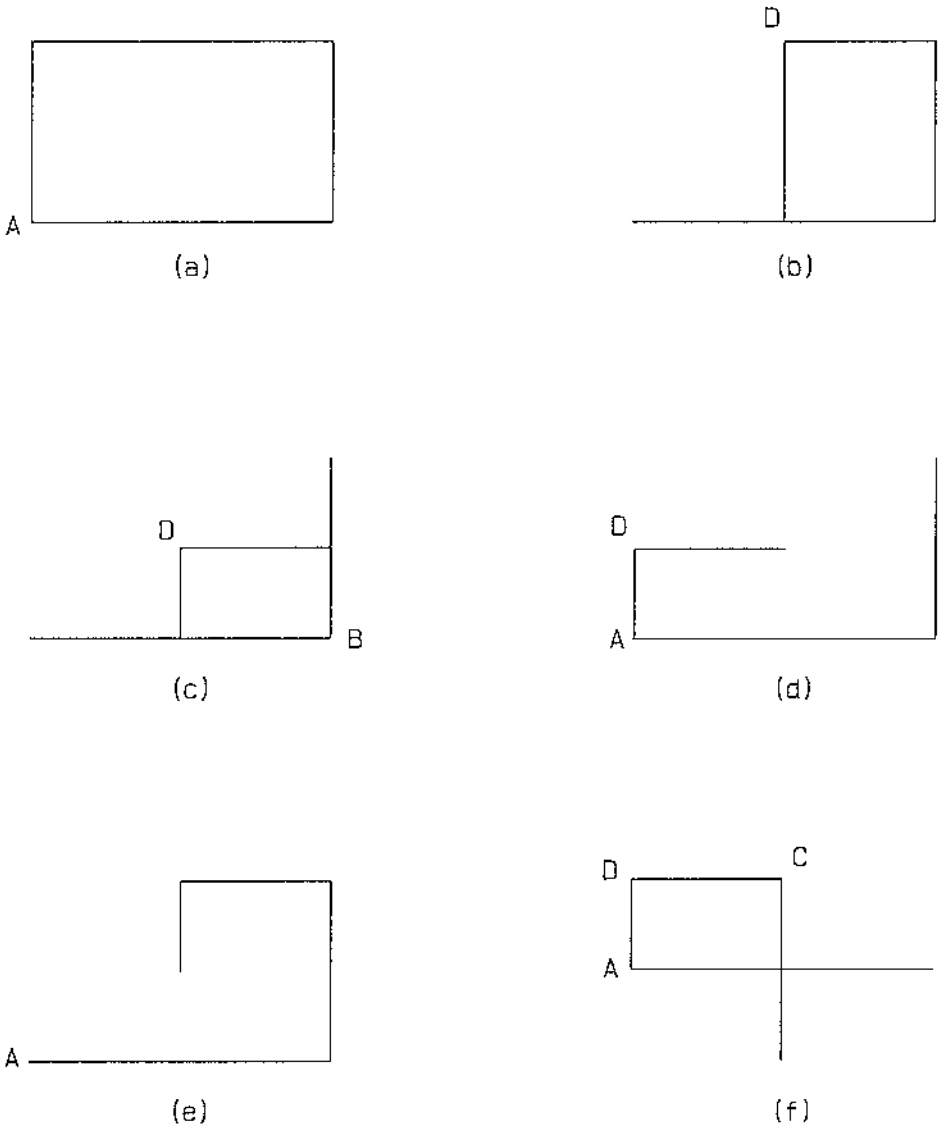


Figure 3-2 Different Interpretations Of Figure 3-1

Figure 3-2 part (b) shows the result of preserving the endpoint intersection connections between lines 1 and 2, 2 and 3, and 3 and 4. The connection between lines 4 and 1 is preserved between the endpoint of line 4 and anywhere along

the locus of line 1. The length of line 3 is preserved while that of line 4 is not. The position of point D is preserved.

Figure 3-2 part (c) shows the result of preserving the endpoint intersection connections between lines 1 and 2, and lines 3 and 4. The positions of points B and D are preserved while A and C are not. The lengths and positions of lines 3 and 4 are preserved.

Figure 3-2 part (d) shows the result of preserving the endpoint intersection connections between lines 1 and 2, lines 3 and 4, and lines 4 and 1. The lengths and positions of lines 3 and 4 are preserved. The position of points A and D are preserved.

Figure 3-2 part (e) shows the result of preserving the endpoint intersection connections between lines 1 and 2, lines 2 and 3, and lines 3 and 4. The lengths of lines 3 and 4 are preserved, but their positions are not.

Figure 3-2 part (f) shows the result of preserving the endpoint intersection connections between lines 2 and 3, lines 3 and 4, and lines 4 and 1. The lengths and positions of lines 3 and 4 are preserved.

These are only six possible interpretations of this parametric drawing - there are many more. Only one of these different interpretations will be the correct one (ie. the

way the designer intended the parametric drawing to be interpreted) but all represent valid possible design requirements. Obviously it is essential to ensure that such ambiguities do not occur in any implementation of a parametric CAD system. This can either be done by having rules laid down by the package which govern the way a parametric drawing is interpreted or else sufficient information must be obtained from the designer during construction of the parametric drawing to ensure all such possible ambiguities are resolved, and this information needs to be stored with the parametric drawing so it can be used when a particular drawing is constructed.

3.3 Prevention of Impossible Drawings

With a conventional CAD system each primitive is fully defined as it is entered and so checks can easily be made at entry time to ensure only valid primitive points are entered. This is not the case with a parametric CAD system as the ultimate drawing shape and size is not determined until the parametric values are supplied. This could give rise to impossible drawings unless additional checks, or design rules, are made. For example consider Figure 3-3. Suppose each line has been specified so that the endpoints connect together as shown to form a quadrilateral. It would be impossible to construct a particular drawing of this parametric drawing unless lengths a and c are the same value

since the drawing must form a rectangle. If line 4 had not been declared a vertical line no problem would have occurred. This is a somewhat simple example of an impossible drawing. With a more complex drawing the "impossibility" can be much more difficult to detect.

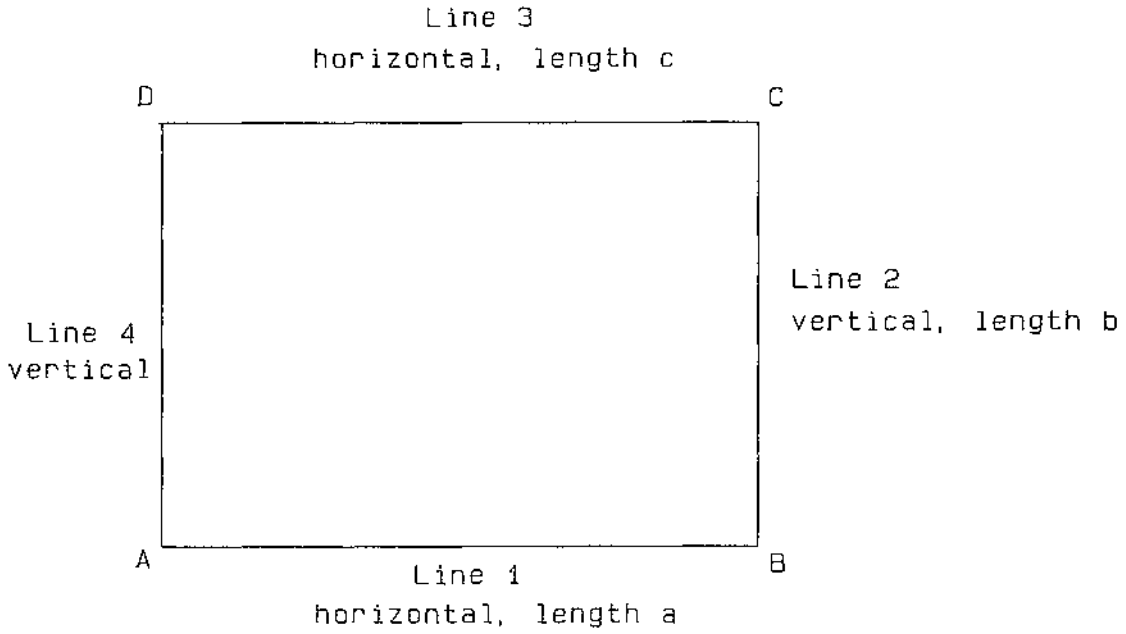


Figure 3-3 A Potentially "impossible" Drawing

3.4 Floating Primitives

Since a parametric CAD system is a conceptual representation of a family of drawings rather than an actual drawing, its construction may well be approached in a different way from a conventional CAD drawing. An aid to drawing such diagrams would be the facility to be able to leave the endpoint of a

line "floating" temporarily. For example a vertical line could have its start point defined, but its endpoint (and hence length) might be left floating until other required primitives have been added, at which stage a "float connection" can be made. This floating line concept would also aid many conventional CAD packages.

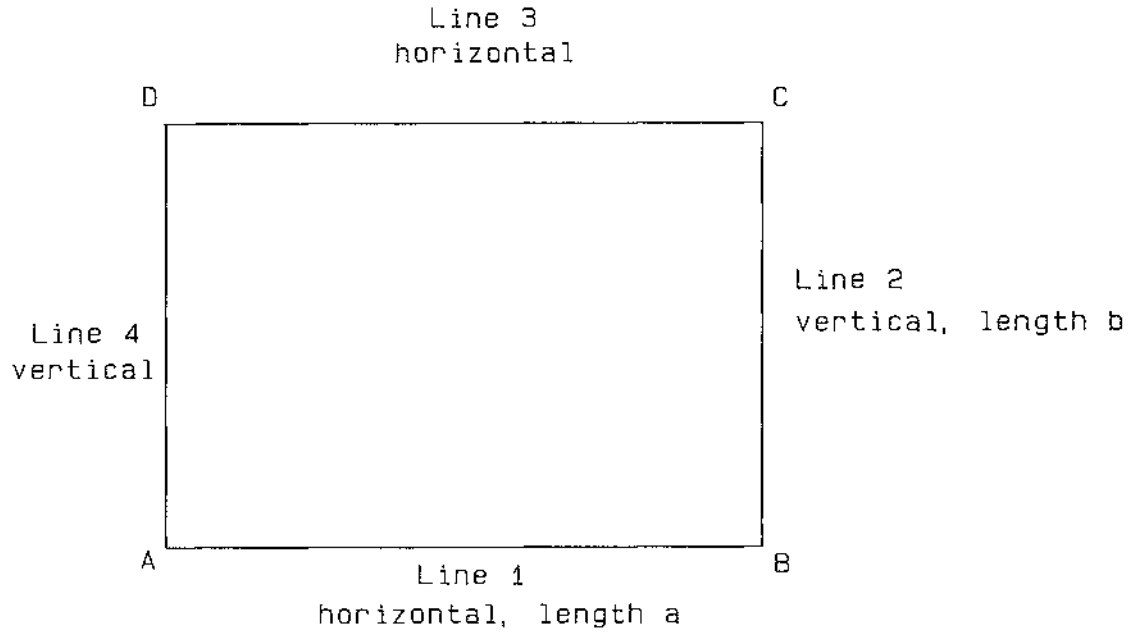


Figure 3-4 A Case For Floating Lines

As an example of floating primitives, consider the construction of the parametric drawing shown in Figure 3-4. Assume that a rectangle primitive is not available so the rectangle shown has to be constructed using lines only. Line 1 would be entered as a horizontal line with parametric length a, line 2 would then be added as a vertical line of

parametric length b with its startpoint defined as the endpoint of line 1. Line 3 would then be entered as a horizontal line with startpoint at the endpoint of line 2 and its end point floating. Line 4 would be entered as a vertical line with its startpoint at the startpoint of line 1 and its endpoint floating. The floating endpoints of lines 3 and 4 could then be connected. The alternative method of entering this, without floating lines, would be to specify line 3 as having length a also. This would then add the complication of having the package perform some form of parameter checking to ensure line 4 would remain vertical with its endpoints at the endpoints of lines 1 and 3. Obviously this is a fairly simple check in this case since the drawing is so simple, but in more complex drawings such checks rapidly become very involved and time consuming.

CHAPTER 4

GENERAL DESIGN DECISIONS FOR PARACAD

Having identified the main features of conventional CAD systems and examined some of the extra requirements of a parametric system it is possible to come up with some general design decisions for the implementation of a parametric system. Many of the design decisions made would apply to any parametric CAD system, but others are specific to a particular implementation that will be referred to as **Paracad** for the rest of this thesis. Paracad has been specifically developed to test the feasibility of a parametric CAD system. Whenever a design decision is discussed a mention will be made as to whether it is a general or specific decision, and in the case of specific decisions the alternatives will be discussed.

4.1 Two Drawing Phases

Because of the natural way in which a parametric CAD system falls into two distinct phases, that of constructing the parametric drawing and that of constructing a particular drawing from the parametric drawing, it was decided to break Paracad into two separate parts - one to cover each phase.

This would seem the logical approach for any parametric CAD system as the two phases would almost always be carried out at completely different times and often by different people.

4.2 Drawing Saving/Loading

Some means of saving and loading partially completed parametric drawings is essential. These drawings could well be very complex and take considerable time to design so a user must be able to stop a design session and resume it at a later stage. In addition to this it is obvious that there must be a means of saving the finished parametric drawing. It was decided Paracad should have both these features (as should any other parametric CAD system).

The need for saving/loading partially completed particular drawings is not so vital since constructing these drawings essentially involves merely specifying parameters. This is a fairly straightforward procedure and should not be particularly time consuming. It was therefore decided that the ability to save a partially completed particular drawing was not a requirement for Paracad, although it could have some use if included in a full implementation of a parametric CAD system. Similarly the saving of a completed particular drawing is not essential if the drawing can be immediately plotted, but this would be a definite advantage (and would be a requirement in any full implementation). Since Paracad is

only a feasibility testing system, it was decided that this feature could be omitted.

4.3 Menus

Because of the advantages for unfamiliar users a menu system is the obvious choice for Paracad since a test system is prone to frequent change and modification. The menu system provides an inherent help method with the menu choices (assuming they are suitably descriptive). The argument over whether a menu-driven or command-driven system would be best in a full implementation is not changed by the addition of parametric facilities to a CAD package and so each method would have its supporters.

4.4 Co-ordinate Storage and Viewing Control

Since all CAD systems of reasonable accuracy employ real variables to store co-ordinate data, it was decided that this should also be the case with Paracad.

Viewing control (windowing and panning) would be necessary for a full implementation, but this does not in any way affect the problems encountered by adding parametric facilities and so it was decided to omit viewing control from Paracad.

4.5 Primitives Supported

The most severe restriction placed on Paracad was in the primitives it supports. While most CAD systems support lines, rectangles, circles, arcs, text, dimensions, polygons and so on it was decided to limit Paracad to lines only. The reason for this is that to implement a large number of primitives into a CAD package takes a number of man years of programming effort. In the case of Paracad the aim is to examine the feasibility of parametric CAD systems rather than to provide a full implementation of a running version and so while restricting Paracad to lines only reduces its usefulness considerably, it does provide a sufficient subset for initial feasibility studies. The additional problems that might be posed by including other primitives in a parametric CAD system will be considered in a Chapter 7. All remaining design decisions will relate to this line-only restriction.

It was also decided that the only method of line entry in Paracad would be by specifying the position of the line's startpoint and endpoint. For a full implementation it would be advantageous to include other methods of line entry, (such as defining a startpoint, a length, and another parallel line).

4.6 Startpoint Types

It was decided that the different possible ways of specifying the startpoint of a line for Paracad should be to give the actual point by using the digitiser, to give the actual co-ordinates of the point from the keyboard, or to specify a point on another line to connect to. While full implementations could add some extra methods such as giving cartesian offsets relative to the last point entered or giving a distance and direction (polar offset) relative to the last point entered, these do not have any additional effect on a parametric CAD system unless the offset values are parametrically defined (ie. the start point position is determined by some parameter). In fact these possibilities are catered for within Paracad by the use of construction lines (described in Section 4.8).

4.7 Endpoint Types

Different CAD packages provide various methods of defining the endpoint of a line. It was resolved that Paracad should have a fairly comprehensive range of methods for line endpoint definition. The methods it was decided to implement were:-

a) specifying the actual position with the digitiser

b) entering the actual co-ordinates from the keyboard

c) specifying a vertical line (with length defined by digitiser, keyboard, parametrically, connecting to some point on another line, or left floating)

d) specifying a horizontal line (with length defined by digitiser, keyboard, parametrically, connecting to some point on another line, or left floating)

e) specifying a polar line (with angle from startpoint defined either parametrically or actual angle entered from keyboard and distance from startpoint also defined either parametrically or from keyboard)

f) specifying a relative line (with x and y offset from line's startpoint defined from keyboard)

g) specifying a connection to some particular point on another line

h) leaving the endpoint floating

These provide a greater selection of methods than is strictly necessary in a full implementation, but they do allow more design flexibility and permit comprehensive testing of the feasibility of parametric line drawing.

A rather insidious problem that may occur is when the parameters supplied to a parametric drawing are such that they result in a particular drawing that has a polar line that ends up with the startpoint and endpoint being in the same place. Since most formulae for calculating the endpoint of a polar line rely on dividing by either dx (the change in x) or dy (the change in y) an attempted divide by zero could result if care is not taken. This problem does not arise in conventional CAD packages as a test is made at entry time to ensure a line has non-zero length.

4.8 Construction Lines

One of the requirements of a parametric CAD system is to allow variable placement of primitives (depending on the value provided for some parameter). One of the most flexible methods for providing this is to allow the use of a "construction line". This is a line that appears only in the parametric drawing and is used as a design aid. It can be drawn as any other line, but is depicted in a different way (eg. different colour or linestyle) to readily identify it. This then allows adaptable defining of a primitive's location.

As an example, suppose a designer needs to specify a primitive as being at an angle of 30^{o} from some point P and a distance r from P (where r is some parametric value that

varies between particular drawings). This can be done as in Figure 4-1 which shows a construction line with startpoint at P. The construction line is defined as being polar with actual angle of 30° and parametric length b. The primitive to be drawn is now drawn connected to the endpoint of this construction line, Q, so altering the value of b will alter the position of this primitive. On any particular drawing the construction line is not visible.

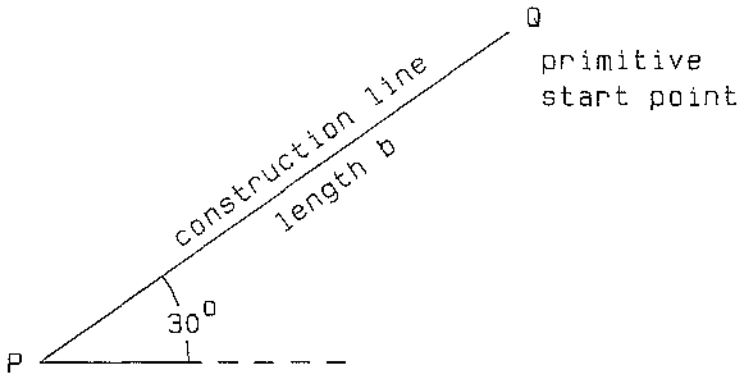


Figure 4-1 Use of a Construction Line

The use of construction lines is not essential in a parametric CAD system, but it does provide considerable flexibility in specifying the startpoint of primitives without increasing the complexity of the startpoint definition. To not have construction lines but still retain the features provided by them by adding different choices to the way of defining the startpoint of a primitive would be a more difficult approach to implement and to use (consider how

the example shown in Figure 4-1 could be done by startpoint definition only - the menu for adding the startpoint would be far more complex).

Construction lines do not really have a great deal of usefulness in conventional CAD packages, although some systems (eg. Versacad) do provide a form of temporary construction line, because a normal line can be drawn instead of the construction line and then erased after it is no longer needed. This is not possible with a parametric CAD system as the linkage between the startpoint and endpoint of the construction line needs to be retained.

Because of the flexibility provided by construction lines it was decided they should be included in Paracad.

4.9 Connect Points

One method permitted by many CAD packages for defining a cursor position is to "snap" to a particular point on an existing primitive. For a line this position is typically the startpoint, endpoint or midpoint of the line. For a rectangle it is typically any of the four vertices or the centre. Such a "snapped" cursor position can then be used to define the startpoint or endpoint of the primitive currently under construction. With a parametric CAD system this concept of snapping to a particular point on a primitive is taken a step

further since it is often necessary to define a connection to the chosen point. For the rest of this thesis such a point will be referred to as a **connect point**.

For Paracad the only primitives allowed are lines and so the obvious connect points to provide on lines are the startpoint, endpoint and midpoint. In addition to these the provision of a fourth point, the equation point, was decided on.

The startpoint, endpoint and midpoint connect points are self-explanatory in most instances (ie. the connect point is at the startpoint, endpoint or midpoint of the line to connect to), but the decisions made for Paracad when the connect point is used to define vertical, horizontal and polar endpoints may require some elucidation. For vertical line endpoints, the connect point defines the y co-ordinate of the vertical line (the x co-ordinate is the same as that for the line's startpoint). This is shown in Figure 4-2 which illustrates three different vertical lines connected in turn to the startpoint, midpoint and endpoint of another line. A similar interpretation applies to connects used for horizontal line endpoints except now the connect point defines the x co-ordinate of the horizontal line's endpoint as shown in Figure 4-3.

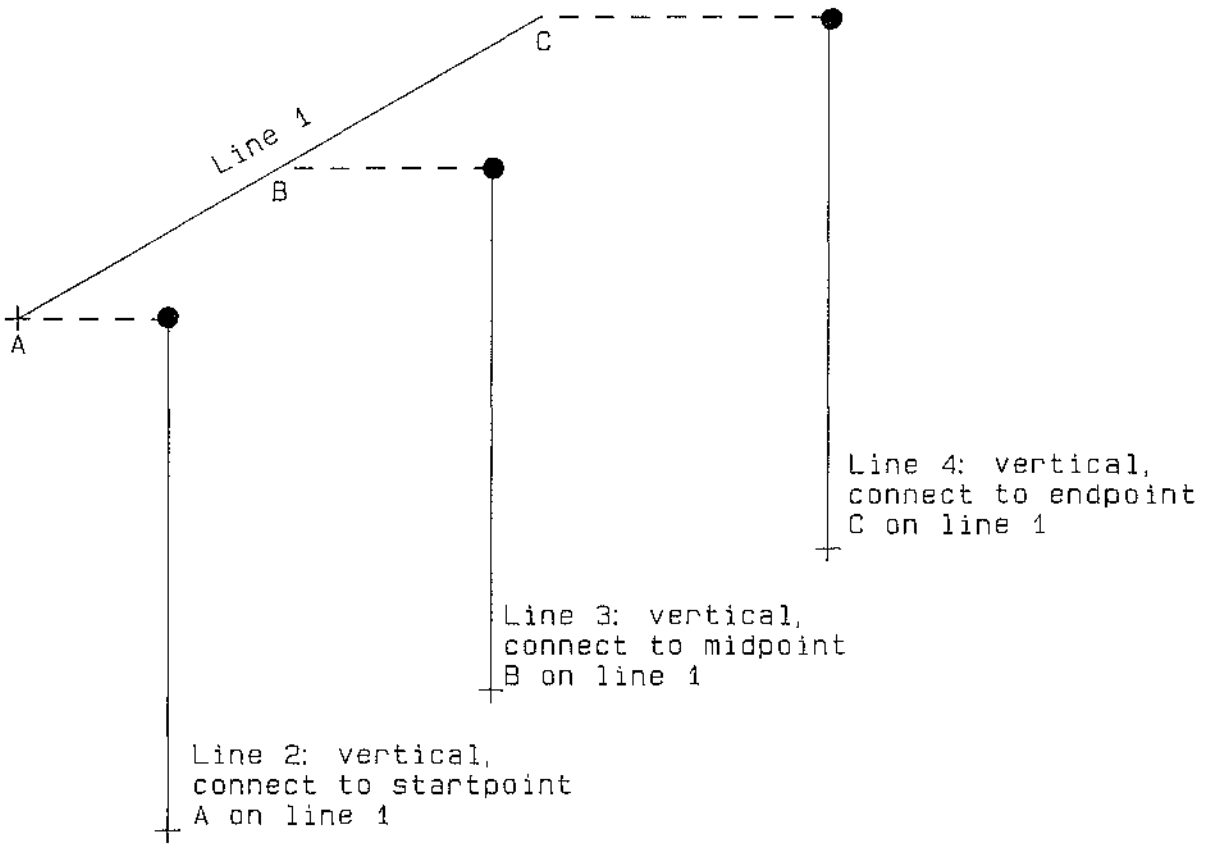


Figure 4-2 Connect Points for Vertical Line Endpoints

For the endpoint of a polar line, the connect point specifies either the y co-ordinate of the polar line's endpoint or the x co-ordinate - depending on whether the line is more nearly vertical or horizontal. For lines that are more vertical, the y co-ordinate is given by the connect point, and the x co-

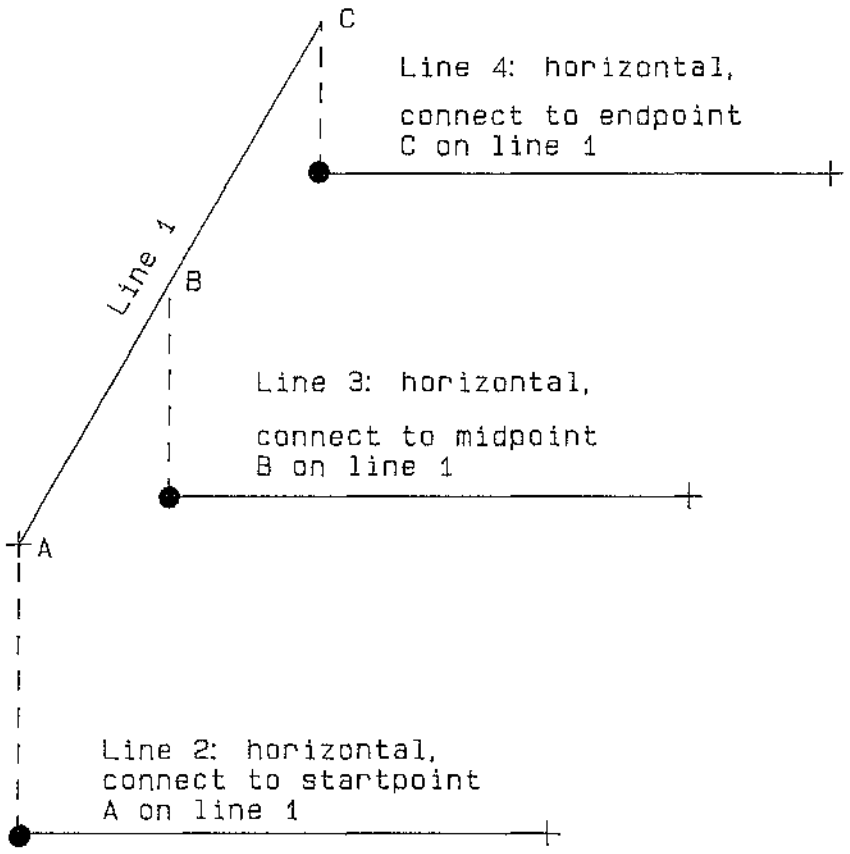


Figure 4-3 Connect Points for Horizontal Line Endpoints

ordinate can thus be calculated. For lines that are more horizontal the x co-ordinate is given by the connect point and the y co-ordinate is then calculated. This is illustrated in Figure 4-4 which shows lines 2 and 3 connected to the

endpoint of line 1. Line 2 is a more vertical case and line 3 is a more horizontal one. Since the angle of a polar line can be parametrically defined and can thus be different for different particular drawings, it is quite feasible that for one particular drawing a polar line takes a connect point to give its endpoint x co-ordinate, while in another particular drawing the same connect point gives its y co-ordinate.

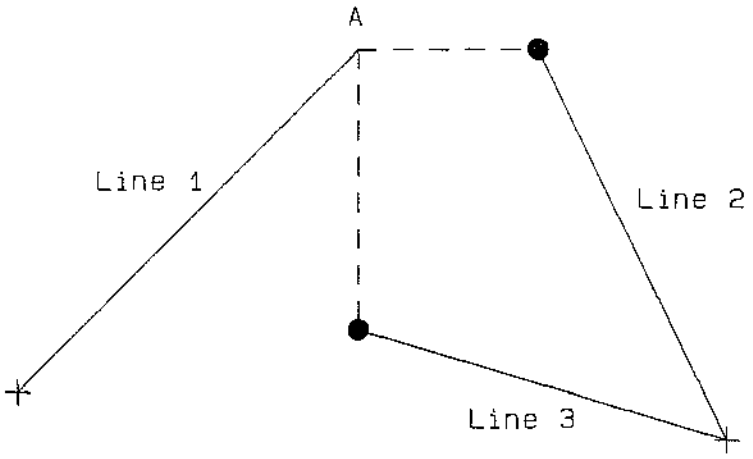


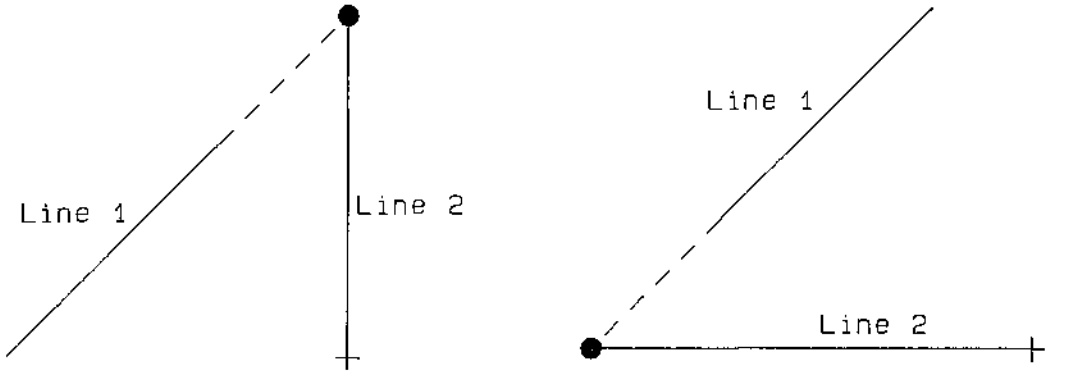
Figure 4-4 Connect Points for Polar Lines

The so-called "equation" connect point needs some amplification. The interpretation of an equation connect point is context dependent. If a user is about to enter the startpoint of any line, or the endpoint of a line unconstrained in direction (ie. not horizontal, vertical or polar), an equation connect indicates that the cursor is to lock onto the equation of the line selected for connection

to. That is the cursor moves along the chosen line (the line is considered to extend to the edge of the screen in both directions). If a user is about to enter the endpoint of a directionally constrained line, the equation connect point is calculated to be the point of intersection of the two lines. Examples of this are shown in Figure 4-5. In each case the line to connect to is line 1, and the line for which the endpoint is being specified is line 2.

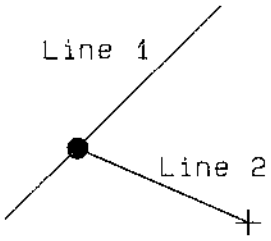
Note that if two lines are parallel there is no intersection point and so an equation connect is not possible. For a parametric drawing the software can prevent the user from being offered the equation connect option for two lines that are parallel at that time, but the parameters supplied for a particular drawing may make a previously possible case (non-parallel lines) into an impossible one (lines parallel). If this occurs the package needs to indicate to the user that the supplied parameters lead to an invalid drawing.

It is convenient for the designer to have connect points indicated in some way so he is aware of the fact that the points may well move in response to the repositioning of another primitive, so a requirement for Paracad was to have some form of connect point identification. It was decided this identifier should be on the point being connected rather than the point being connected to - ie. if the endpoint of line 1 is being connected to the midpoint of line 2, the connect identifier appears on the endpoint of line 1 (which



a) Vertical Line Endpoint

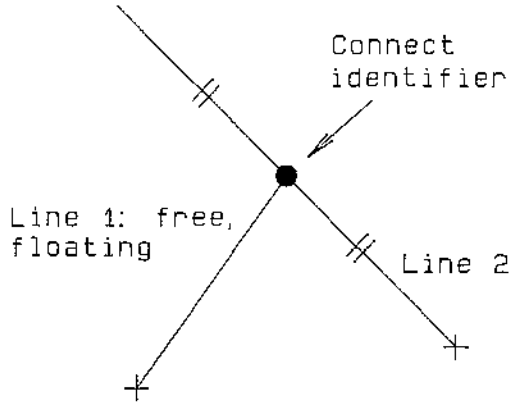
b) Horizontal Line Endpoint



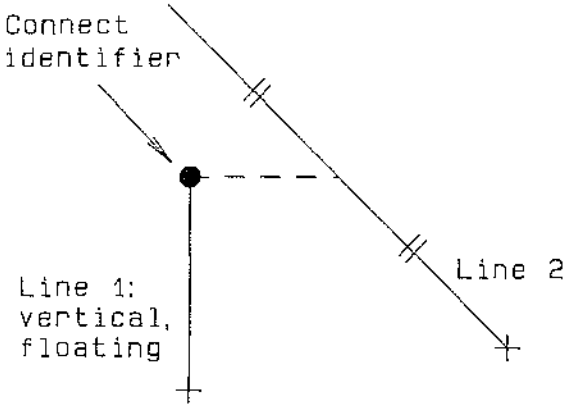
c) Angular Line Endpoint

Figure 4-5 Equation Connect with Constrained Line Endpoint

may or may not be in the same location as the midpoint of line 2). Figure 4-6 part (a) shows a case where the two points are in the same location while part (b) shows a case where they are in different locations and the midpoint of line 2 is used to give just the y co-ordinate of the line 1 endpoint.



a) Same location



b) Different locations

Figure 4-6 Position of Connect Identifier

4.10 Floating Endpoints

The concept of floating line endpoints was discussed in Section 3.4. Providing floating lines adds a number of extra problems for a parametric CAD system, so it was considered necessary to include this feature in Paracad. Since the author is unaware of any CAD package that includes floating lines (or any similar concept) all the development techniques for handling them needed to be developed from scratch. For Paracad it was decided to provide floating endpoints for lines unconstrained in direction and for the length of vertical and horizontal lines. It was decided not to include floating endpoints for the length or angle of polar lines or for either offset in relative lines in Paracad since these do not add any new problems in terms of parametric CAD considerations but do add significantly to the complexity of the software required for implementation.

It was also decided not to include polar lines with fixed (or parametric) lengths and floating angles for reasons of ambiguity. An example of this ambiguity is indicated in Figure 4-7. Line 1 is specified as having length **a** and line 2 has length **b**. Both lines have floating angles and their endpoints are to be connected. The endpoint of line 1 must lie on the circle centred at the startpoint of line 1 with radius **a** and similarly the line 2 endpoint lies on the circle centred on its startpoint with radius **b**. These two circles will not intersect at all if the distance between their

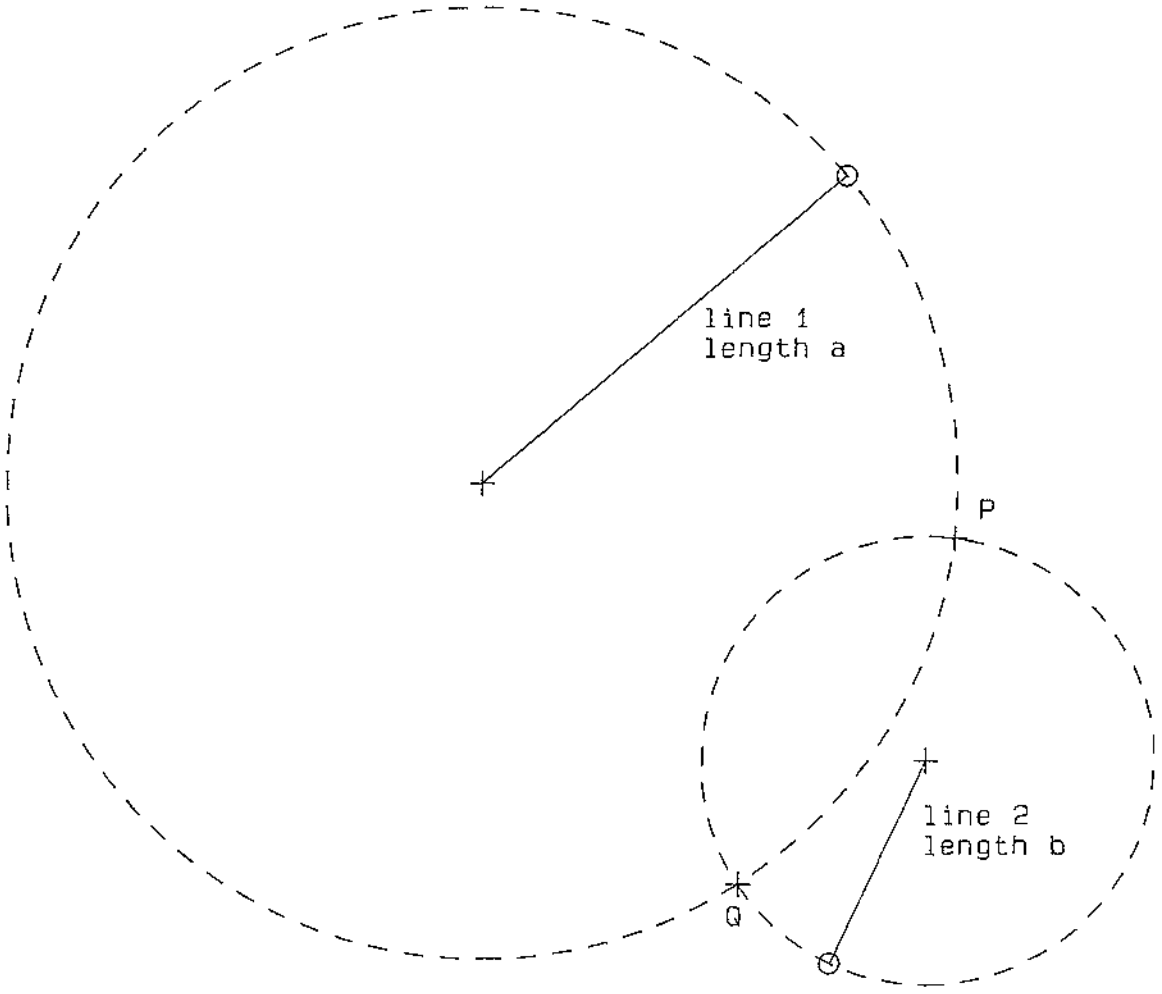


Figure 4-7 Ambiguity With Floating Angles

centres is greater than **a** + **b**, they will touch at one point if the distance between their centres equals **a** + **b**, they will intersect at two points if the distance between their centres is less than **a** + **b** but greater than zero, and if they are concentric they will not intersect at all if **a** ≠ **b** and they

will intersect in an infinite number of places if **a = b**. Figure 4-7 shows the case where they intersect in two places, **P** and **Q**.

This form of ambiguity is difficult to resolve. The case where the circles do not intersect could be considered an error if it occurs in a particular drawing, as could the concentric circle cases. The difficulty arises in the case where there are two possible intersection points. It is almost impossible to come up with some easy to use method for defining which point of the two is required at parametric time since particular drawings can look radically different from the parametric drawings they derive from - thus requesting "the leftmost point" of the two is not satisfactory. For this reason it was decided not to include floating angles in Paracad.

For future reference, a line that is unconstrained in direction and that has a floating endpoint will be referred to as a **freefloat** line, while a horizontal or vertical line with a floating endpoint will be referred to as a **vertically** or **horizontally floating** line. Collectively they will be referred to as **floating lines** while lines that are not floating will be referred to as **fixed lines**.

It is convenient, for the designer, to have a floating endpoint indicated in some way so he is aware that the endpoint has not yet been fixed and thus the drawing is

incomplete, so a requirement for Paracad was to have a form of floating point indication. This identifier should be on the endpoint of the floating line.

It was further decided that no connect would be allowed to be made to the midpoint, endpoint or equation point of any line with a floating endpoint (the only exception to this is a special float connect which is discussed in Section 4.11). This restriction is necessary to ensure that the startpoint of every line is always fully defined (either directly or parametrically). This considerably reduces the problems of resolving floating lines when they finally become connected and it helps prevent the construction of impossible drawings. This is a very minor restriction in practice since once the endpoint of a floating line has been fixed it is then possible to connect to it.

4.11 Floating Endpoint Connections

The provision of floating endpoints on lines means some method must be provided for eventually fixing these points. There are two possibilities for the fixing point - either the point will be some connect point based on another primitive or else the point will be some actual point selected by the digitiser or keyboard input. The latter of these two possibilities is really an unnecessary option - there is no need to have a floating line if it is merely fixed to a known

actual point at a later time since it could have been fixed at the time it was first added (or else not been added until the time it was fixed) and so this possibility was not required in Paracad, although in a full implementation it may well be incorporated.

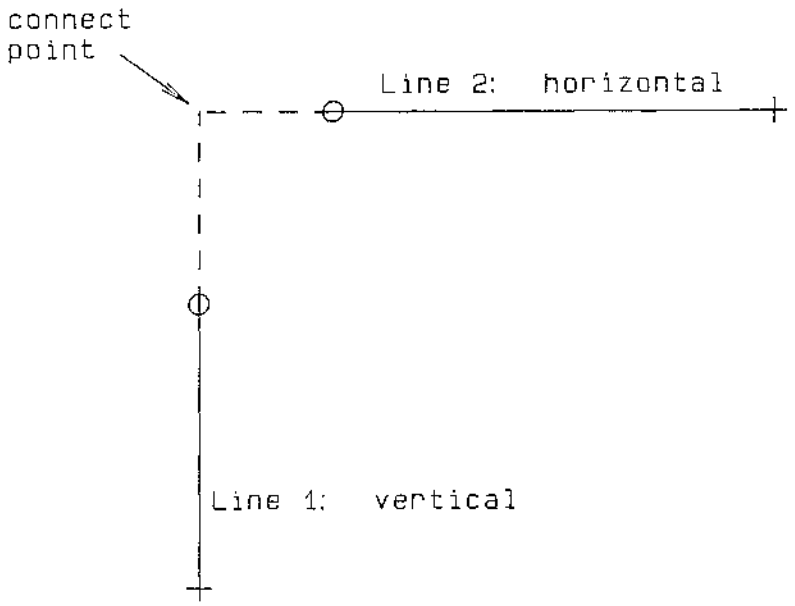


Figure 4-8 Float Connect of Horizontal and Vertical Lines

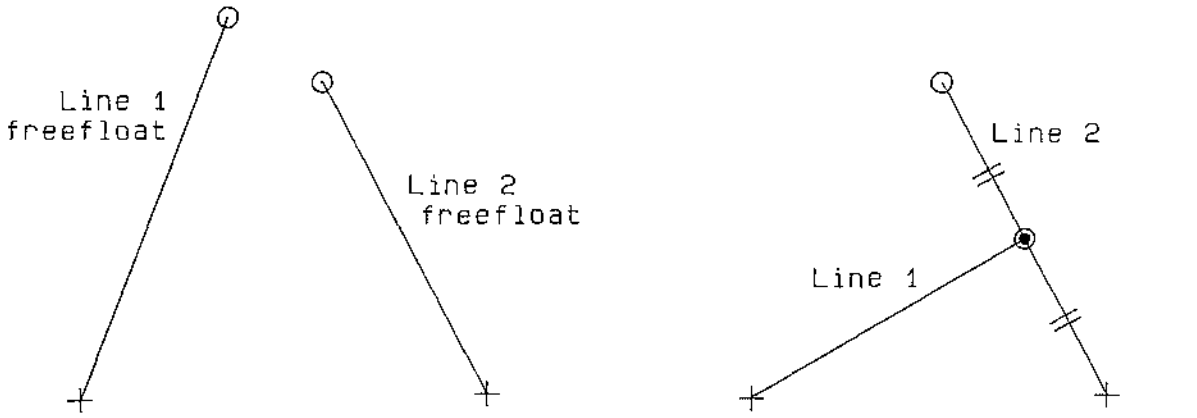
The ability to connect a floating endpoint to some other primitive was a requirement for Paracad. The possible connect points allowed should, for consistency, be the same as those allowed for other connects - ie. the startpoint, endpoint, midpoint and equation point. In fact since normal connects can be used to fully test all these options, it was decided to only support startpoint, endpoint and midpoint for float connects in Paracad. This provides a saving in program

complexity at no cost to parametric feasibility testing.

Float connecting ideally needs to allow connecting to other floating lines as well as fixed lines. An example of the use of this is shown in Figure 4-8 where it is required to connect the two floating endpoints together. Note that doing this will then fully define the position and lengths of both lines. This would not always be the case as is shown in Figure 4-9. In this case the floating end of freefloat line 1 is connected to the midpoint of freefloat line 2. Before this float connection was made, both lines were unconstrained in both direction and length. After the float connection is made the lines are still not fully defined in either length or direction, but they do have a form of constraint in that once one of the lines becomes fully defined the other one also becomes fully defined. Since extra problems are imposed by allowing float connection to lines that are themselves floating, and the concept of floating lines has been developed for parametric CAD systems, it was decided that this feature needed to be included in Paracad.

It is worth considering the different possibilities that may occur in float connection:

a) Any float connection to any point on a fixed line will result in the floating line becoming fixed.



a) before connection

b) after line 1 float connected to midpoint of line 2

Figure 4-9 Float Connect of Two Freefloat Lines

b) Any float connection to the startpoint of another line will result in the floating line becoming fixed, regardless of whether the line being connected to is fixed or floating. This is because startpoints are always fully defined for all lines in Paracad (see Section 4.6).

c) Any float connection from a floating horizontal line endpoint to either the midpoint or endpoint of a floating vertical line will result in both lines becoming fixed as shown in Figure 4-10. Notice that each line adjusts its length accordingly to fit the type of connection.

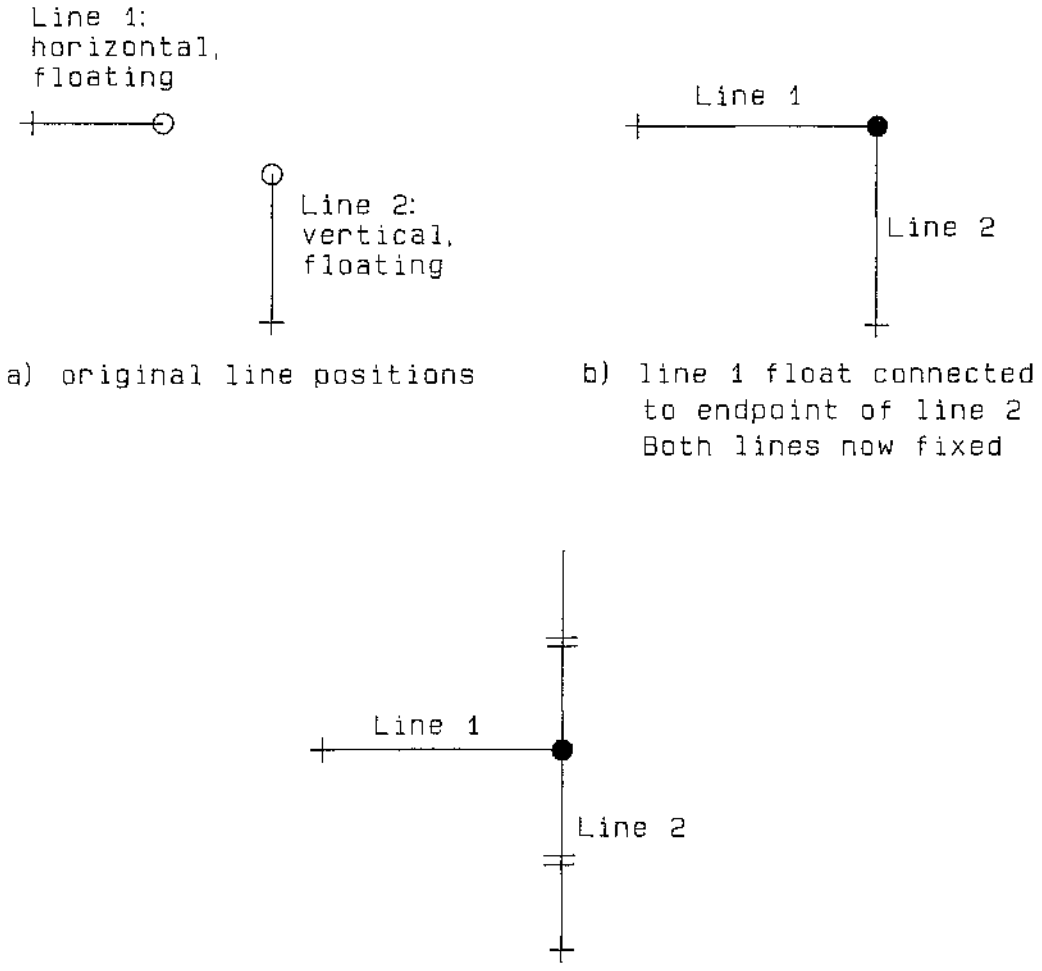


Figure 4-10 Float Connect From Horizontal To Vertical Line

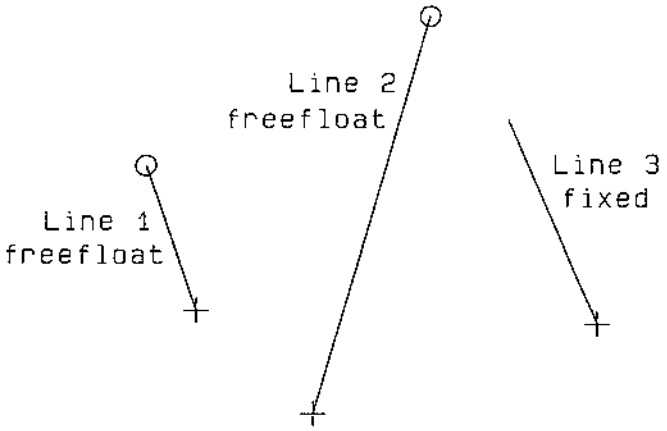
d) Any float connection from a floating vertical line to either the midpoint or endpoint of a floating horizontal line will result in both lines become fixed in a similar fashion to that illustrated in (c) above.

e) Any float connection from a floating vertical line to the midpoint or endpoint of another floating vertical line, or from a floating horizontal line to the midpoint or endpoint of another floating horizontal line, results in both lines remaining floating, but as soon as one of the lines becomes fixed the other will also become fixed.

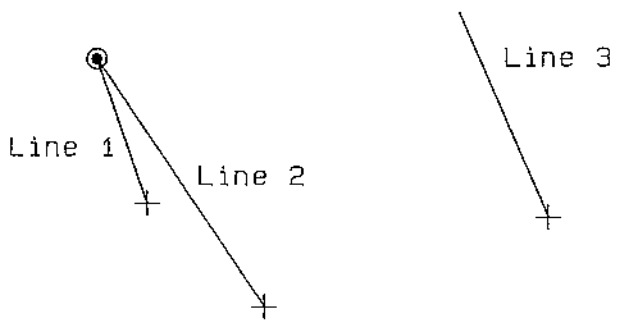
f) Any float connection from a freefloat line to the midpoint or endpoint of any other float line (vertical floating line, horizontal floating line, or freefloat line) results in both lines remaining floating (although they would be locked together in some way) but as soon as one line becomes fixed the other will also become fixed. This is illustrated for two freefloat lines in Figure 4-11.

g) Any float connection from a floating vertical, horizontal or polar line to the midpoint or endpoint of a freefloat line results in both lines remaining floating but as soon as one line becomes fixed the other will also become fixed.

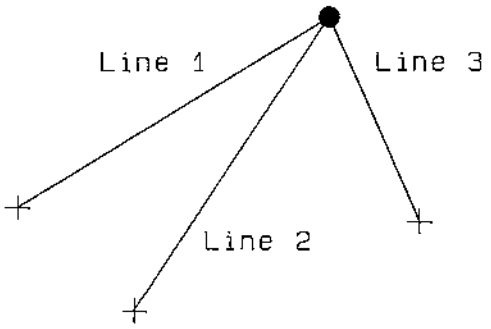
The diagrams in Figure 4-11 illustrate another design decision needed for Paracad. It would impose a somewhat severe restriction on a user if he was only allowed to make one float connection with each floating line for two reasons. Firstly he would need to remember which lines had been float connected in cases like that shown in Figure 4-11 as it would



a) original line positions



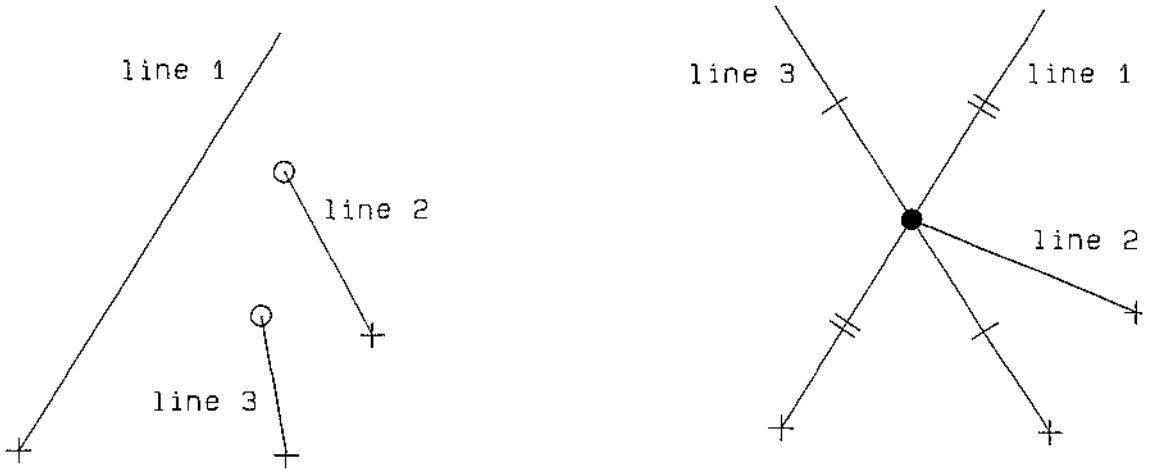
b) line 2 float connected to endpoint of line 1. Both lines remain floating with endpoints locked together



c) line 2 float connected to endpoint of line 3. Both lines are now fixed

Figure 4-11 Float Connect Between Two Freefloat Lines

not be permitted to connect line 2 to line 3 as in part 3, rather he would have to connect line 1 to line 3 - an action that would give exactly the same result. Secondly, it would be restrictive in that a user may become trapped and not be allowed to produce the type of drawing he requires easily. As an example consider the drawing shown in Figure 4-12. The easiest way to construct the required drawing is to float connect the endpoint of line 2 to the midpoint of line 3, then float connect the endpoint of line 2 to the midpoint of line 1. If only a single float connect is permitted for each floating endpoint this would not be possible and a more roundabout way would be needed to construct the drawing.



a) original line positions b) required drawing

Figure 4-12 The Case For Multiple Float Connects

Since floating lines have been created for parametric CAD systems it is necessary to fully investigate the difficulty in implementing them, so it was decided to place no limit on the number of times a floating endpoint can be float connected within Paracad - the process can continue until such time as the endpoint becomes fixed.

Since a Paracad requirement is to have some form of identification on floating endpoints and connect points it was decided that a floating endpoint that has been fixed (by some connection) should no longer have a floating endpoint identifier, but rather should have a connect point identifier. A floating endpoint that is connected to another line but still remains floating should have some identifier that shows it is both floating and connected.

4.12 Cursor Indication

For any interactive CAD package using an input device such as a digitiser it is essential to have some form of visual feedback as to the relationship between input device position (eg. position of stylus on the digitiser tablet) and screen position. This is normally done by displaying some form of cursor on the screen that "follows" the input device around. Such a cursor is a requirement for Paracad.

4.13 Rubberbanding

When selecting the endpoint of a line it is useful to have the line follow the cursor about so the user can immediately see the effect if he were to select a particular point for the line endpoint. This feature is called rubberbanding (or tracking). While it is not strictly necessary for parametric feasibility testing, its inclusion makes a package easier to use. For this reason it was decided to include rubberbanding within Paracad where appropriate.

4.14 Parameter Setting

For a parametric CAD system some method of indicating what dimensions are to be parametric is needed during the parametric drawing construction phase and some method is necessary for entering values for these parameters during the particular drawing construction phase. Since Paracad is to examine the feasibility of parametric CAD rather than the various possible mechanisms for supplying and linking parameters it was decided that a simple method could be used for each of these requirements. Indicating that a particular dimension should be parametric can be done at entry time for the affected primitive - the parameter name can be entered and a sample value can be provided for the purposes of constructing the parametric drawing. During construction of the particular drawing a scan can be made of each primitive

in turn, and as parameters are encountered the user can be prompted for keyboard entry of a value for the parameter. In a full implementation providing these parameter values could be done from keyboard entry, lookup tables, input files, decision tables or output from other programs.

4.15 Parameterisation

Parameterisation refers to the process of actually constructing the particular drawing once all the parameter values specific to that drawing have been ascertained. To reduce the software-writing burden in producing a parametric CAD system the most convenient approach is to use the same routines written for constructing the parametric drawing to construct the particular drawing. While many features will not be required, such as input prompts, the underlying logic will be the same. For this reason the routines written for the parametric construction phase should be written in such a way that they can be readily used in the construction phase. This philosophy is to be incorporated in Paracad and the problems encountered noted to give some indication of the difficulty of this approach.

4.16 Plotting

Since Paracad is not a full implementation it was decided that the plotting facilities required would be as simple as practicable. These should provide the output of particular drawings only, in one colour, with a fixed scale and no user control over the position of the drawing on the paper.

4.17 Miscellaneous

In order to reduce the complexity of Paracad it was decided not to provide a number of miscellaneous features that would be expected in a full implementation. None of these features affect the parameter feasibility testing. They include the ability to create or use groups, perform transformations (such as enlargement), modify primitives (such as deleting), copy primitives, have properties attached to primitives (such as a primitive name) and the provision of levels (or layers).

CHAPTER 5

PARACAD ENVIRONMENT AND STRUCTURE

This chapter covers the environment of the Paracad system. Both hardware and software environments are covered as are the data structures used and their method of storage.

5.1 Hardware and Software Environment

The hardware used for implementing Paracad was an IBM AT microcomputer with 512 kbytes of RAM, 80287 numerical co-processor chip, 20 Mbyte hard disk, serial communications card (RS232), monochrome display adapter with monitor and professional graphics controller (PGC) and display. The digitiser used was a Kurta 12" x 12" and the plotter used was a Hewlett Packard 7475A A3/A4 model.

Paracad was written in Turbo Pascal (co-processor version) with some in-line machine language routines developed using 80286 Assembler.

Paracad would run unmodified on any IBM PC, XT, AT or compatible computer that had a PGC system and numerical co-processor. For machines without a co-processor the Turbo

Pascal source code could be recompiled, but would run more slowly. Driver routines for the PGC are written as separate procedures to enable easy modification to other graphics displays. These driver routines provide for the drawing of solid and dotted lines in different colours, solid and filled in circles in different colours, text, and allow windows on the screen for a menu area and a graphics area.

5.2 PGC System

The IBM Professional Graphics Controller (PGC) provides high-function graphics capability. It supports a screen resolution of 640 x 480 pixels with 256 colours from a palette of 4096 colours. It has an on-board 8088 microprocessor and 300 kbytes of RAM for memory-mapped graphics **[IBM.1]**. The host system provides command strings to the PGC which then carries out the appropriate action on its bit-mapped image. No direct access is provided to the bit-image area of the PGC memory - commands are available to transfer data to and from this area but generally this type of use does not match the philosophy behind the PGC design.

The PGC provides drawing facilities for many primitives including lines, text and circles, co-ordinate transformations with modelling and viewing transformations, user-definable look up tables (LUT) and window and viewport control. Communication can either be in ASCII or HEX format.

ASCII format enables commands to be sent in text form – for example the command

DRAW 10 20

will draw a line from the current point to the point with co-ordinates (10,20). In Hex format the commands are sent as a single byte followed by the arguments (if any). The PGC effectively converts ASCII format commands into Hex format before processing them. Since Paracad is to run interactively in real-time the Hex format is used as it is significantly faster than ASCII format.

5.3 Digitiser and Interface

Communications with the Kurta digitiser are via an RS232 serial card running at 9600 baud. Data is frequently required from the digitiser regarding stylus and button (switch) position. Further, this data is often required as rapidly as possible (for example when the stylus is being moved across the digitiser and its position is being displayed on the screen) so drivers were written in 80286 Assembler for maximum speed and included within Paracad as in-line code. The digitiser has four different operating modes for positional data output - auto (continuous data output whenever stylus is close to digitiser surface or stylus button is activated), draw (continuous data output only when button is activated), point (single point data output upon button activation) and delta (single point data output

whenever stylus moves more than 0.01" or button is activated) **[KURT]**. The mode selected for Paracad was auto so it was possible to obtain information on the stylus position without the button being pressed. This enables the button to be used for such actions as selecting primitives, indicating the current point is the one required, responding to questions and so on.

The Paracad digitiser drivers consist of two in-line machine code procedures. One procedure software configures the digitiser to be in the auto mode, with resolution of 200 points per inch and high resolution packed binary output. The other procedure reads the digitiser data stream and returns the current stylus position (x and y co-ordinates) and the button status (switch open or closed). Some spurious readings occasionally come from the digitiser so for the most critical positions (the ones when the button is pressed) the procedure waits until it receives two successive positional data items that have the same co-ordinates before passing back the results.

In order to use a different digitiser, two new machine code interface routines would need to be developed.

5.4 Cursor Display

Initial tests were done using a machine code routine for drawing a cursor on the screen, but it was found that because of the nature of the PGC's operation and the speed of the AT microcomputer that it was just as fast to code the cursor drawing routine as a Turbo Pascal procedure. The procedure is supplied with the screen co-ordinates of the required position and draws a cursor at this point.

The cursor chosen is of the cross-hair type with the size specified as a program constant for easy alteration. The cursor is drawn by complementing the value of any pixels at the locations of the cursor pixels. This has two main advantages. Firstly, it allows easy erasing of the cursor by a second call to the procedure with the same screen co-ordinates. Secondly, it permits the cursor to pass over existing primitives without deleting or altering the primitives when it has gone - also judicious choice of the PGC LUT allows the cursor to still be visible (as some different colour) when over a primitive.

5.5 PGC Interface

As discussed in Section 5.2 communication with the PGC is in Hex format. Since PGC commands are of different lengths (different numbers of bytes) Paracad uses a somewhat unusual

approach to the method of interfacing to the PGC. This involves setting up a data type structure for each group of commands with the same length (ie. one data type for one byte commands, one for two byte commands and so on). The first byte of each structure stores the number of bytes that follow, then comes the hex command byte, then the actual bytes that make up rest of the command (if there are any). To pass these data structures to the PGC the Turbo Pascal extension that allows procedures to have untyped variable parameters is used. This allows a procedure to be declared with a variable parameter that has no specified type. Whenever a call is made to the procedure, what is passed across is the address of the actual parameter [TURB.1985].

In Paracad an in-line machine code routine with an untyped variable parameter is used to send commands to the PGC. Any call to this procedure uses the required data structure name as the actual parameter, so the machine code routine is given the address of the data structure containing the information to be sent. The first byte of this information is the length (ie. the number of bytes following) so the routine knows how many bytes to send. While this may seem a rather convoluted approach, it does allow sending of variable length groups of bytes in hex format. The alternatives are to either have a separate routine for sending each different type of data structure or else to send data in ASCII form (as a string). Paracad uses eight different data structures so eight different sending routines would be required if that approach

was used, and using the ASCII mode of communication is considerably slower.

5.6 Identification and Colours

All colours used in Paracad are easily alterable by changing the LUT initialisation area. This involves specifying three integers in the range 0 - 15. These integers specify the required intensity of the Red, Green and Blue electron guns respectively. Each of the LUT colours specified has a complement that provides a sensible display when the cursor moves over the pixels of that colour, or when another line crosses it. The various identifiers and colours used will now be described briefly.

Menu area: Paracad uses a rectangular area at the top of the colour screen for a menu display and message area. This area has a blue background and text is in white. The menu area can accommodate up to 3 lines of 80 characters of text.

Graphics area: The area of the screen used for the drawing comprises all the screen below the menu area. A black background is used.

Cursor: Paracad uses a small red cross-hair cursor.

Floating endpoints: Paracad uses a small blue open circle on any line endpoint that is floating.

Connect points: Paracad uses a small yellow filled-in circle on any line connect points. If a point is both connected and floating a blue circle circumference that is filled in with yellow is used.

Lines: A line that has been fully drawn is shown in white.

Construction lines: A construction line that has been fully drawn is shown in purple.

Rubberbanded lines: A line (or construction line) that is currently being drawn and is being rubberbanded is shown in red until its endpoint position is specified.

Selection lines: A line that is being displayed for possible selection (eg. as a line to connect to) is shown in green.

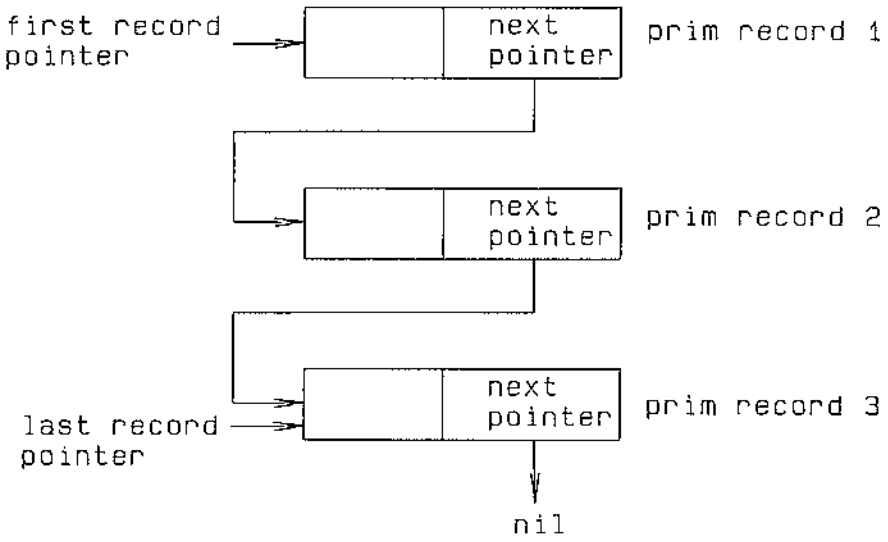
Float connect lines: A line with a floating endpoint that has been selected in order to connect (or fix) its endpoint is shown in red.

5.7 Data Structures and Database Storage

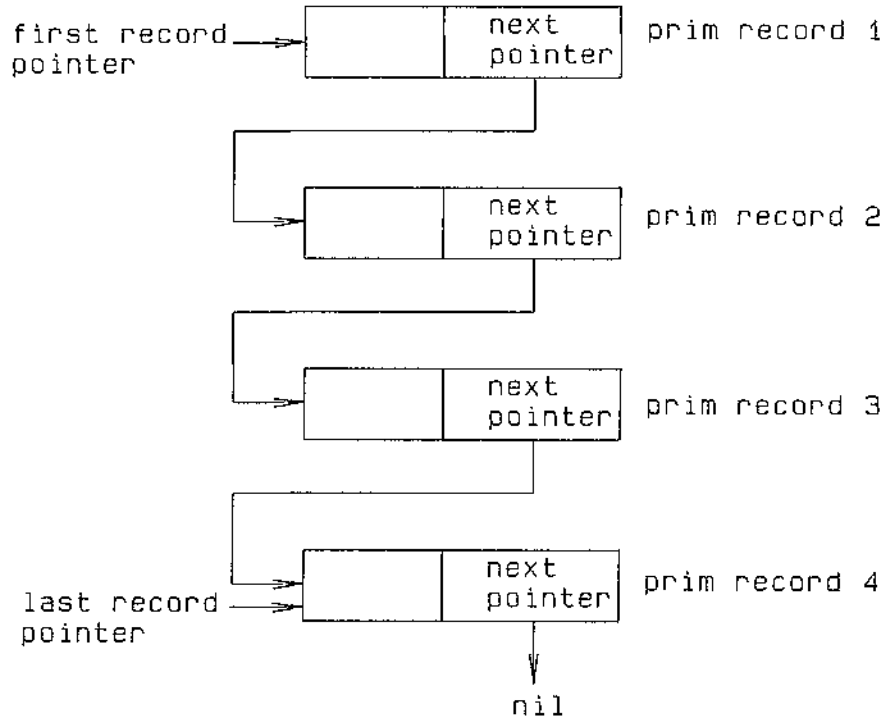
5.7.1 Primitive Data Structure

Each line drawn in Paracad forms a record (referred to as a primitive record). Dynamic variables are used for these primitive records to allow efficient memory use and to avoid being constrained by the Turbo Pascal limit of 64 kbytes total memory for static variables and program. Dynamic variables can use the entire memory space of the machine.
The primitive records form a linked list. Each primitive record has a pointer to the next record in the list and there are separate pointers to the first and last records in the list. As a new line is added to the drawing its corresponding primitive record is added to the end of the list. A list containing three primitive records is shown in Figure 5-1 part (a) and the same list with a fourth primitive added is shown in Figure 5-1 part (b). No deletion of records is necessary since Paracad does not support line deletion.

The format of each primitive record is shown in Figure 5-2. Each of the 13 different data fields will now be considered in more detail:



a) list containing 3 primitive records



b) list with a 4th primitive record added

Figure 5-1 Primitive Record List Structure

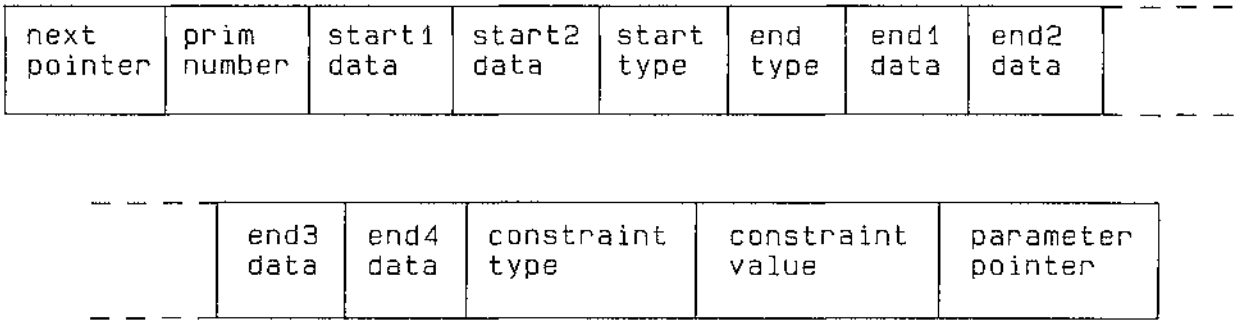


Figure 5-2 Primitive Record Format

1. **Next pointer**

This field is a pointer variable that contains the pointer to the next primitive record in the linked list.

2. **Prim number**

This field is an integer variable that contains the number of the current primitive. Each primitive has an associated unique number that is used to identify the primitive.

3. **Start1 data**

This field is a real variable that is used to store the x co-ordinate of the line's current startpoint.

4. **Start2 data**

This field is a real variable that is used to store the y co-ordinate of the line's current startpoint.

5. **Starttype**

This field is an integer variable used for storing flags to indicate the type of startpoint the line has. Currently Paracad only supports actual positions or connect positions so only two different values - 0 for actual startpoint, $8000 (hexadecimal 8000) for connect startpoint.

6. **Endtype**

This field is an integer variable used for storing flags to indicate the type of endpoint the line has. An integer takes two bytes of storage in Turbo Pascal and so this allows 16 one bit flags. These flags are as follows:

Bit 16 - If the line is a construction line this bit is 1 otherwise it is 0

Bit 15 - If there is an unresolved floating endpoint for the line this bit is 1 otherwise it is 0.

Bit 14 - If the line has a connect endpoint and is not directionally constrained (ie. not polar, horizontal or vertical) this bit is 1 and the 2nd byte of the endtype field is unused (all bits set to 0). If the endpoint is not a connect or the line is directionally constrained this bit is 0.

Bit 13 - If the line has an endpoint specified as an absolute value (absolute co-ordinates entered from either the keyboard or digitiser position) this bit is 1 and the second byte of the field is unused (all bits set to 0), otherwise this bit is 0.

Bit 12 - If the endpoint of the line is of the freefloat type (ie floating and not directionally constrained) this bit is 1 and the second byte of the field is unused (all bits set to 0), otherwise this bit is 0.

Bit 11 - If the line endpoint has been defined as relative this bit is 1 otherwise it is 0. If this bit is 1, the second byte of the field is used as follows:- bits 8 and 7 are 0 (unused), bit 6 is 1 if the x offset is given as an actual value (otherwise it is 0), bit 5 is 1 if the y offset is given as an actual value (otherwise it is 0), bit 4 is 1 if the x offset is a parametric value (otherwise it is 0), bit 3 is 1 if the y offset is a parametric value (otherwise it is 0), bit 2 is 1 if the x offset is floating (otherwise it is 0) and bit 1 is 1 if the y offset is floating (otherwise it is 0). Paracad currently does not allow relative lines with floating endpoints so the last two of these flags are not currently used.

Bit 10 - If the line has been defined as polar this bit is 1 otherwise it is 0. If this bit is 1, the second byte of the field is used as follows:- bit 8 is 0 (unused), bit 7 is 1 if the line has a connect endpoint (otherwise it is 0), bit 6 is 1 if the line angle is given as an actual value (otherwise it is 0), bit 5 is 1 if the line length is given as an actual value (otherwise it is 0), bit 4 is 1 if the line angle is a parametric value (otherwise it is 0), bit 3 is 1 if the line length is parametrically value

(otherwise it is 0), bit 2 is 1 if the line angle is floating (otherwise it is 0) and bit 1 is 1 if the line length is floating (otherwise it is 0). Paracad currently does not allow polar lines with floating angles or lengths so the last two of these flags are not currently used.

Bit 9 - If the line has been defined as vertical or horizontal this bit is 1 otherwise it is 0. If this bit is 1, the second byte of the field is used as follows:- bit 8 is 1 if the line is vertical or 0 if horizontal, bit 7 is 1 if the line has an endpoint connect (0 otherwise), bit 6 is 0 (unused), bit 5 is 1 if the line length is given as an actual value (otherwise it is 0), bit 4 is 0 (unused), bit 3 is 1 if the line length is parametrically defined (otherwise it is 0), bit 2 is 0 (unused) and bit 1 is 1 if the line endpoint is left floating.

Bits 8 to 1 - If bits 11, 10 or 9 are set to 1, these bits are set as described for bits 11, 10 and 9 above, otherwise they are all set to 0 (unused).

7. **End1 data**

This field is a real variable that is used to store the x

co-ordinate of the line's current endpoint.

8. **End2 data**

This field is a real variable that is used to store the y co-ordinate of the line's current endpoint.

9. **End3 data**

This field is a real variable that is used to store endpoint information depending on the type of line. For a polar line it stores the current angle (in radians), for a line with a relative endpoint it stores the current x offset, and for all other lines it is unused.

10. **End4 data**

This field is a real variable that is used to store endpoint information depending on the type of line. For a vertical, horizontal or polar line it stores the current line length, for a line with a relative endpoint it stores the current y offset, and for all other lines it is unused.

11. **Constraint type**

This field is an enumerated type variable that indicates the type of constraint on the line's endpoint. The possible values are vertically_constrained, horizontally_constrained, fixed and freefloat. The use of this field will be described in more detail in Section 6.7.

12. **Constraint value**

This field is a real number variable that stores the currently constrained value for directionally constrained line. The use of this field will be described in more detail in Section 6.7.

13. **Parameter pointer**

This field is a pointer variable that contains a pointer to the first parametric variable used by this line (if any are used).

5.7.2 Parameter Record

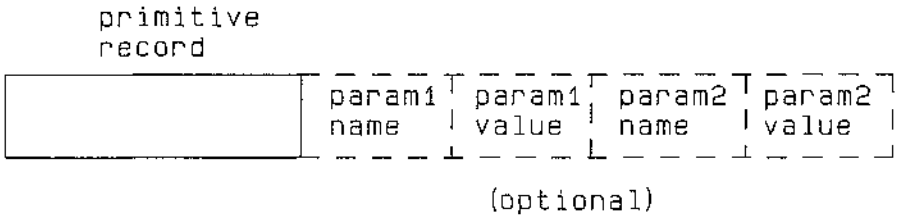
Each primitive drawn in Paracad may have 0, 1 or 2 associated parameters. For example, a line with an absolute startpoint and absolute endpoint would have no associated parameters, a vertical line with parametric length would have one parameter, and a polar line with parametric length and parametric angle would have two parameters. To store these parameter names and link them to the corresponding primitive record could be done in three possible ways.

The first method is to have variable size primitive records depending on the number of parameters involved (using Pascal's variant records for example), but to cover all the possible cases where a parameter could occur would require several different record forms, and if the package was extended to cover other primitives the number of cases would be even greater. This could be simplified somewhat by allowing only three variant record types - one with no parameter name field, one with one parameter name field and one with two parameter name fields. If this method is used it is important to ensure parameter names are correctly matched with the dimension they represent in the two parameter field case - for example for a polar line with parametric angle and parametric length it is necessary for the program logic to have a specific order of parameter name storage so they can be correctly retrieved and the angle parametric name does not get used as the length parametric name. An added complication

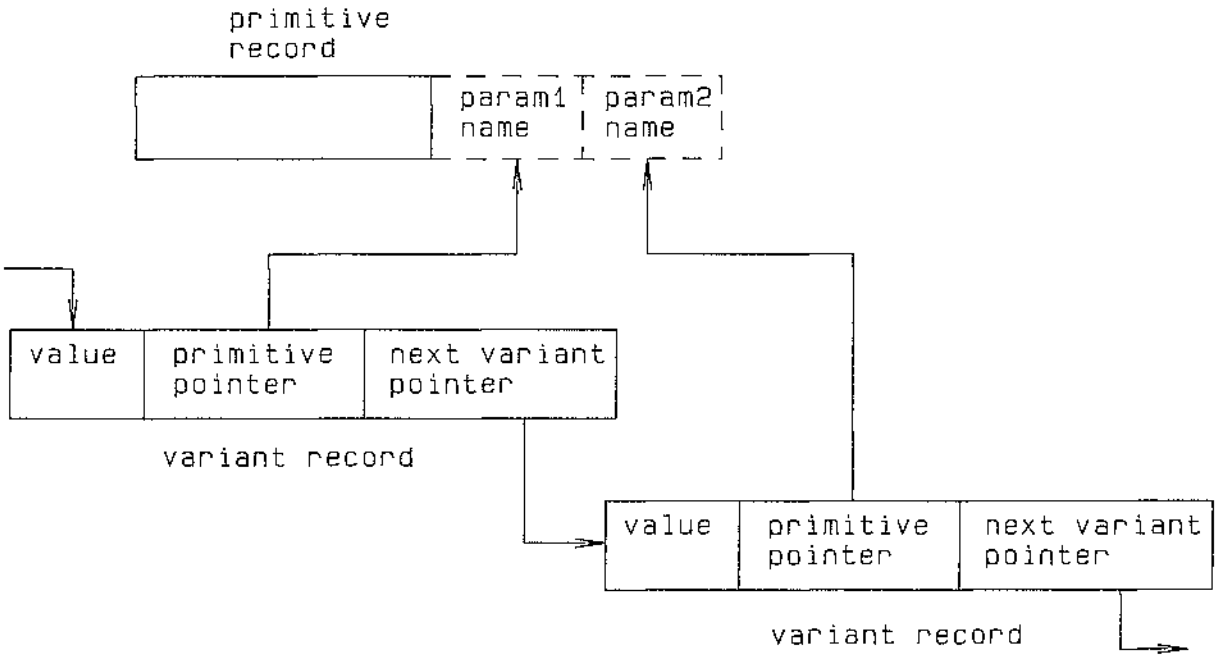
arises when the parameters are given their actual values - some form of linkage is required between the parameter names and their values. This necessitates either an extra field in the primitive record for each parameter value, or separate parameter value records either with pointers back to the primitive records or else parameter name fields within them. The different possibilities are illustrated in Figure 5-3 with the two forms of parameter value storage indicated. If the same parameter name (and value) is to be used for more than one primitive, which is a reasonable requirement, modifications would be needed to this storage mechanism.

The second method is to append each primitive record with two data fields for the names of the parameters and only use as many as are necessary for that primitive. This has the overhead of having two data fields associated with every primitive record, and one (or both) may not be used in many cases. As with the first method, the order the parameters are stored in is critical to ensure correct matching of names on retrieval, the linkage of names to values is still required, and modifications would be required to allow the same parameter name for more than one primitive.

The third method is to have a single parameter pointer field in the primitive record that points to a separate parameter record (if one is needed) that in turn points to a second parameter record (if two are needed). Again the order of parameter name storage is important, but no problems exist



a) storing all parameter data with primitive record



b) storing parameter name only with primitive record

Figure 5-3 Possible Methods Of Storing Parameters

with linkage between the parameter name and value as both would be stored as part of the parameter record. This also simplifies processing and storage when the same parameter is used for more than one primitive and allows extra fields to be easily added to the parameter records (if required for future expansion). It also allows any number of parameters to be associated with a single primitive record without any modification and so lends itself to future expansion to other primitive types. This was considered the most attractive method of the three and so it is the one implemented in Paracad. Figure 5-4 illustrates the method. The parameter record in the parametric drawing program does not contain a value field, while the parameter record in the particular drawing program does.

5.7.3 Affected Record

The linkage between connect points is an important one in any parametric CAD system. Since line lengths and angles can vary with their parameters, so the positions of other lines connected to them also changes. It is possible for a single parameter change for one line to have a ripple-through effect causing changes in the position of every other line in the drawing. Further, during floating endpoint connection the fixing or connecting of one floating endpoint has an effect on all other interconnected floating lines. The processing involved in calculating these effects can be considerable and

so it is essential that the mechanism used to indicate line interconnections permits fast access.

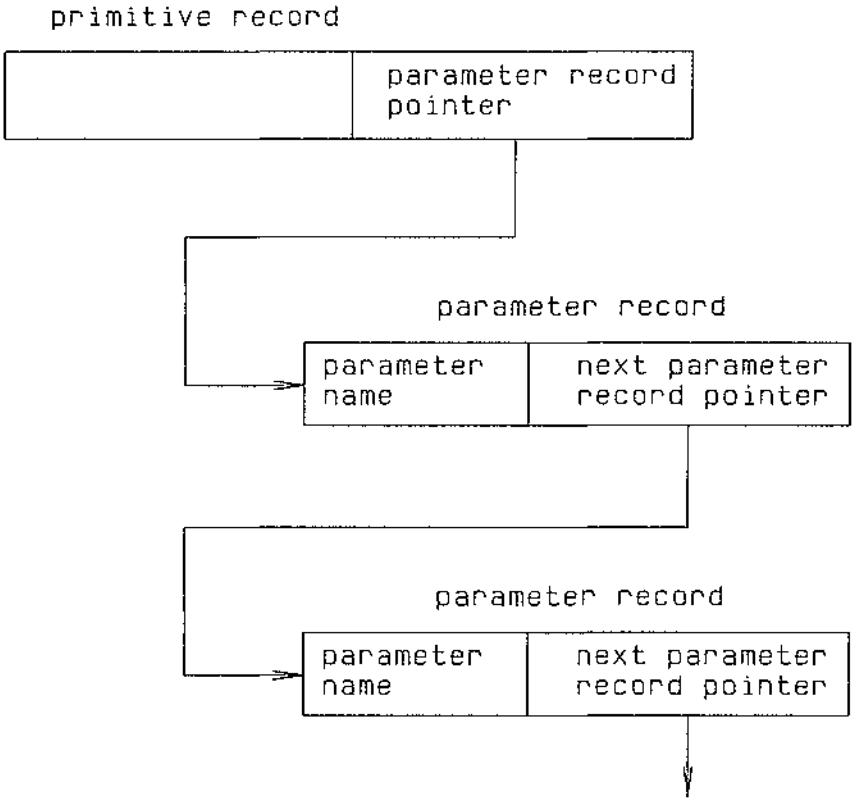


Figure 5-4 Method Of Storing Parameters In Paracad

Initially Paracad was designed with line interconnection data stored as part of the primitive record, but this became too unwieldy for floating line connection processing and also required considerable storage overhead in fields needed in the primitive record - fields that would be unused if the primitive was not connected to another. To find out what lines were connected to each other it was necessary to scan

all primitive records (several times in many cases) and then deduce interconnections. For example, to find out what lines were affected by line 1 it was necessary to scan all primitive records to find connections to line 1 and if any line was connected, line 4 say, it was then necessary to scan all primitive records again for a reference to line 4 and so on.

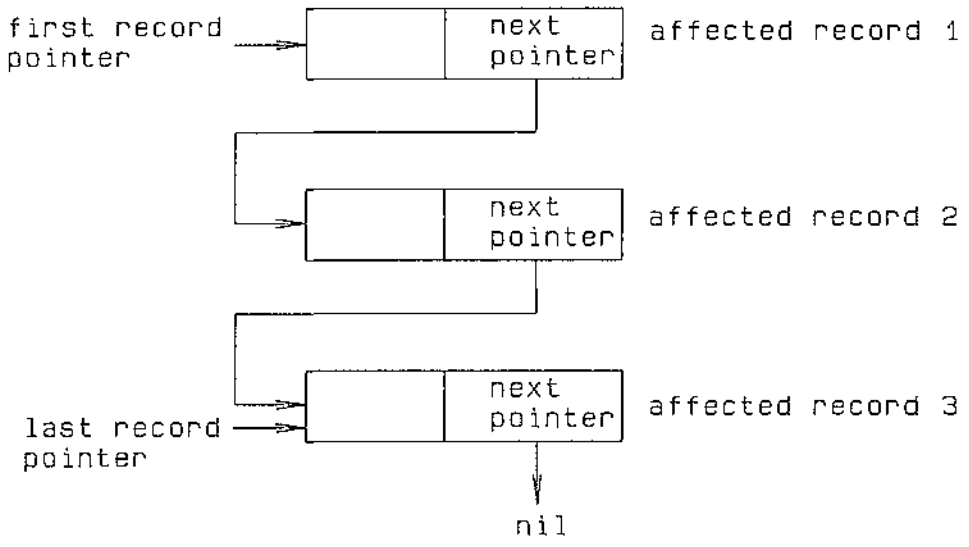


Figure 5-5 Affected Record List Structure

It was quickly apparent that this method of connect data storage was unsatisfactory, so an alternative method is now implemented. This involves having separate records containing connection data. These records are called **affected records** and are dynamic records in a linked list. This affected records list has been set up to enable fast processing of

both normal connections and floating endpoint connections as described in Sections 6.6 and 6.7. The list structure has a separate pointer to the first and last records and each record in the list has a pointer to the next record as shown in Figure 5-5.

controlling primitive	affected primitive	controlling point	affected point	lineequation ratio	next affected pointer

Figure 5-6 Affected Record Format

The format of each affected record is shown in Figure 5-6. An explanation of each of the 6 fields follows:

1. **Controlling primitive**

   This field is an integer variable that stores the primitive number of the primitive that causes some other primitive to be affected because of this particular connection.

2. **Affected primitive**

   This field is an integer variable that stores the

primitive number of the primitive that is affected by this particular connection.

3. **Controlling point**

This field is an enumerated type variable that indicates the position on the controlling primitive where the affected primitive is connected. Current allowable positions are linestart, lineend, line middle and line equation.

4. **Affected point**

This field is an enumerated type variable that indicates the point on the affected primitive that is connected to the controlling point. Current allowable positions are linestart, lineend, line middle and line equation.

5. **Line-equation ratio**

This field is a real variable that stores the ratiometric position of a connect point for a line equation connection. This is described in more detail in Section 6.6.

6. **Next affected pointer**

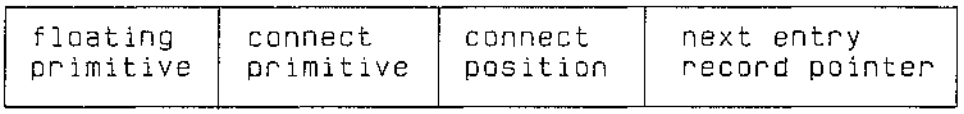
This field is a pointer variable that stores the pointer address to the next affected record (if there is one).

For more information on the method of primitive connection, refer to Sections 6.6 and 6.7.

5.7.4 Entry Record

The method used for parameterisation in Paracad follows through each primitive addition and float connection in the order they were performed during construction of the parametric drawing (refer Section 6.8). This requires the order these actions are performed in to be recorded. This is done by using **entry records**. An entry record is a variant record with two different forms - one for adding primitives and one for float connections. These records are stored in a linked list with a separate pointer to the first and last record in the list. The two different formats for entry records are shown in Figure 5-7. The various fields in these records are as follows:

a) primitive addition form



b) float connect form

Figure 5-7 Entry Record Formats

1. **Add primitive**

This field is an integer variable that stores the primitive number of the primitive that is being added.

2. **Next entry record pointer**

This field is a pointer variable that stores the pointer address of the next entry record (if there is one).

3. **Floating primitive**

This field is an integer variable that stores the primitive number of the primitive with the floating endpoint that is being connected.

4. **Connect primitive**

This field is an integer variable that stores the primitive number for the primitive that the floating endpoint is being connected to.

5. **Connect position**

This field is an enumerated type variable that stores the position on the connect primitive where the floating endpoint of the floating primitive is being connected to. The positions allowed within Paracad are linestart, lineend, linemiddle and line equation.

5.7.5 Queue record

The method of connecting floating endpoints adopted in Paracad is described in detail in Section 6.7. This method involves searching the affected primitives list to find any

primitives affected by the one that is being connected. If any are, they are added to a queue (that is initially empty). Once the search reaches the end of the affected records list, the primitive that is first in the queue is taken and another scan is made of the affected records to see if any are affected by this primitive and so on until the queue is empty. The records stored in the queue are called **queue records** and are dynamic records that are stored in a linked list with a pointer to the first and last record in the list. The format of each queue record is illustrated in Figure 5-8.

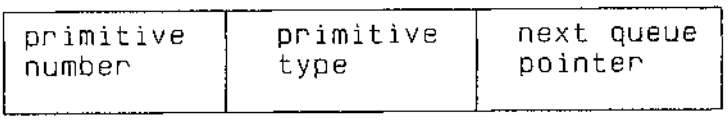


Figure 5-8 Queue Record Format

The various fields in these queue records are as follows:

1. **Primitive number**

   This field is an integer variable that stores the primitive number for the primitive being added to the queue.

2. **Primitive type**

This field is an enumerated type variable that indicates how the primitive endpoint is constrained. Possible values for this field are vertically constrained, horizontally constrained, fixed and free floating.

3. **Next queue pointer**

This field is a pointer variable that stores the pointer address of the next queue record (if there is one).

5.7.6 Primscovered Record

During the float connect stage (ie. when floating endpoints are being connected) scans are made of the affected primitives records to find any primitives affected by the current queue record (this process is discussed in more detail in Section 6.7). Once all affected records have been scanned for the current queue primitive the record is removed from the queue. To prevent it being put back in the queue by a subsequent affected record Paracad employs a list containing primitives that have been covered by an affected records scan. This list, called the **primscovered list**, consists of dynamic record variables in a linked list with

pointers to the first and last elements in the list. The format of each primscovered record is shown in Figure 5-9, and each field is described in detail below.

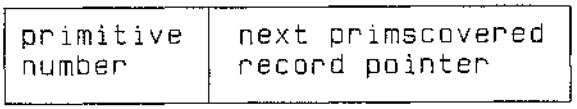


Figure 5-9 Primscovered Record Format

1. **Primitive number**

   This field is an integer variable that stores the number of the primitive that has been covered by an affected records list scan.

2. **Next primscovered record pointer**

   This field is a pointer variable that stores the pointer address of the next primscovered record (if there is one).

5.7.7 Check Record

When a floating endpoint is being connected to another primitive it is important to ensure that a connection causing a circular float conflict is not made. A simple example of a circular float conflict is if lines 1 and 2 are both floating, the endpoint of line 1 is float connected to the midpoint of line 2, and then an attempt is made to float connect the endpoint of line 2 to the midpoint of line 1. This circular float conflict involves only two primitives, but in practice any number of primitives can be involved (circular float conflicts are discussed in more detail in Section 6.7). In order to detect such conflicts, Paracad builds up a list of circularly affected primitives during a float connect. The elements of this list are called **check records** and are dynamic record variables. The format of a check record is shown in Figure 5-10 and the contents of each field is described below.

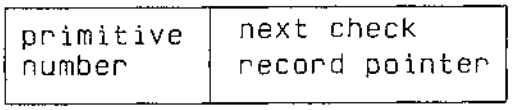


Figure 5-10 Check Record Format

1. **Primitive number**

This field is an integer variable that stores the number of the primitive that is being added for check purpose. list scan.

2. **Next check record pointer**

This field is a pointer variable that stores the pointer address of the next check record (if there is one).

5.8 File Handling (Saving and Loading)

During the parametric drawing construction phase Paracad provides the ability to save and/or reload any partially or fully completed parametric drawing. During the particular drawing construction phase it allows the loading of any parametric drawing, but will not permit processing if the drawing has any lines with floating endpoints that have not been fixed.

Saving a partially or fully completed parametric drawing involves saving, as separate files, the primitive records, the entry records, the affected primitive records and the parameter records. Since all pointer variable linkages that exist when a drawing is saved will be incorrect when the

drawing is reloaded again, an alternative to pointers is necessary in the files saved. For the entry record, affected primitive record and parameter record files the records are stored, without pointers, in their linked list order. The primitive records are also stored in their linked list order, and the parameter pointer field that points to the first parameter record associated with this primitive is changed to store an integer that gives the position of the parameter record in the parameter file (it is set to zero for primitives that have no associated parameters).

When a parametric drawing is to be loaded, either during the parametric drawing construction phase or the particular drawing construction phase, all these files are read back in again and the pointer records re-established.

CHAPTER 6

PARACAD USER INTERFACE

The way the user operates Paracad and the way it responds to the user is discussed in this chapter. Direct responses (such as the menu structures) are described as are the algorithms used for implementing these responses (such as the method of selecting primitives, the way line connects operate and so on). Both the parametric drawing phase and the parameterisation phase (and its output) are covered.

6.1 Menu Structure

Paracad is a menu-driven package with the various menus forming a tree structure. The top three lines of the graphics display screen are reserved for menus and text. For menu displays, the top line is used for a description of the menu (eg. Main Menu, Add Line Endpoint etc.) while the second and third line display allowable choices. A selection is made by typing the first letter of the required menu item. To exit from a menu to the menu above the user selects the Quit menu item. An outline of the menu structure employed in Paracad is displayed in Appendix B.

6.2 User Friendliness

Since Paracad is an interactive package it is important that the user feels comfortable with its use. Necessary consequences of this are to have a sensible menu structure with easy to understand options, consistency in software actions, user feedback to ensure the user is kept informed of what is happening and fast response to all user actions wherever possible or an indication of what is happening if this is not practicable.

Considerable efforts have been made to make the menu structure easy to understand and use and to ensure consistency throughout. An improvement that should be considered on any full implementation is an on-line help facility to explain in detail the actions of any menu choice.

User feedback is employed in a number of ways. Whenever an invalid menu choice is made or an illegal action is attempted the system emits an audible error "beep" and displays a message (if appropriate).

If the user is required to enter data either from the keyboard or the digitiser the system emits an audible prompt "pip" of shorter duration than the error beep to indicate to the user that some action is required. Whenever digitiser action is permitted the cursor is displayed on the screen and moves in synchronisation with the digitiser stylus. If

digitiser input is not permitted the cursor is not displayed.

Rubberbanding is used when a primitive is being drawn to show how the primitive would appear if the current digitiser stylus position was selected. The primitive being rubberbanded is also drawn in a different colour from the other primitives to make it stand out. Once the endpoint position is selected the primitive colour reverts back to the same colour as the other primitives.

Rubberbanding of some primitives involves more calculation than others. For example the endpoint of a primitive that has no directional constraints can follow the cursor anywhere, while a polar line say has only its x or y endpoint co-ordinate defined by the cursor and the other co-ordinate must be calculated by some formula. The algorithms used in Paracad for calculating such co-ordinates have been optimised for speed and there is no noticeable difference in the speed at which any type of line endpoint tracks the cursor.

Initially Paracad flashed the primitive being drawn on and off during rubberbanding, but the flash rates were so fast for some lines that they appeared to be invisible at times. This is now changed so that during rubberbanding a line is kept continuously displayed unless the digitiser moves the cursor by at least a pixel. When this occurs the old rubberband line is erased and a new one drawn. This results in a far steadier and more aesthetically pleasing display.

Any primitive that is selected for some action in Paracad (for example for a connection to be made to) is displayed in a different colour from other primitives to enable it to stand out. During a connect the cursor becomes "locked" on the line selected to connect to.

6.3 Directionally Constrained Lines

At all stages within Paracad, the endpoint position specified for a vertical line determines the lines y co-ordinate only - the x co-ordinate is made the same as that for the lines startpoint. This is true regardless of whether the endpoint is specified by digitiser, absolute co-ordinates, connection to another line, by parametric definition or whether the line is tracking the cursor during rubberbanding. Similarly for horizontal lines the endpoint position specified defines the x co-ordinate only at all times.

For polar lines the endpoint position always specifies the x co-ordinate for lines that are more nearly horizontal than vertical or the y co-ordinate for lines more vertical than horizontal.

The above rules for endpoint determination of directionally constrained lines were implemented to provide a consistent behaviour pattern for such lines and to avoid any conflicting problems that may arise in the construction of parametric

drawings such as that shown in Figure 3-3 and discussed in Section 3.3. In this case, if the endpoint of line 4 was defined as being connected to the startpoint of line 1 (at point A) then point A would merely be supplying the y co-ordinate of line 4 and line 4 would not actually go to point A except in the special case where parameters **a** and **c** were supplied with the same values.

6.4 Primitive Selection Methods

During the parametric drawing construction phase in a Paracad session it is often a requirement to select a primitive - for example to connect a line to. The method adopted for this is to select the required line by first using the digitiser to move the cursor to a point close to the line. The software then uses a two-level elimination process to locate the primitive.

The first level elimination involves taking the first primitive stored and checking to see whether the cursor lies within its extents - ie. whether the cursor falls within the rectangle enclosing the primitive. This is performed by testing whether the cursor's x value falls within the x values covered by the primitive and the cursor's y value also lies within the y range covered by the primitive). If the cursor is not within both x and y extents, the line is rejected and the next line is tested. This provides a very

rapid method of eliminating almost all unwanted lines, but can lead to some primitives being accepted that the user may find rather strange **[FOL. 1984]**. As an example of this consider Figure 6-1.

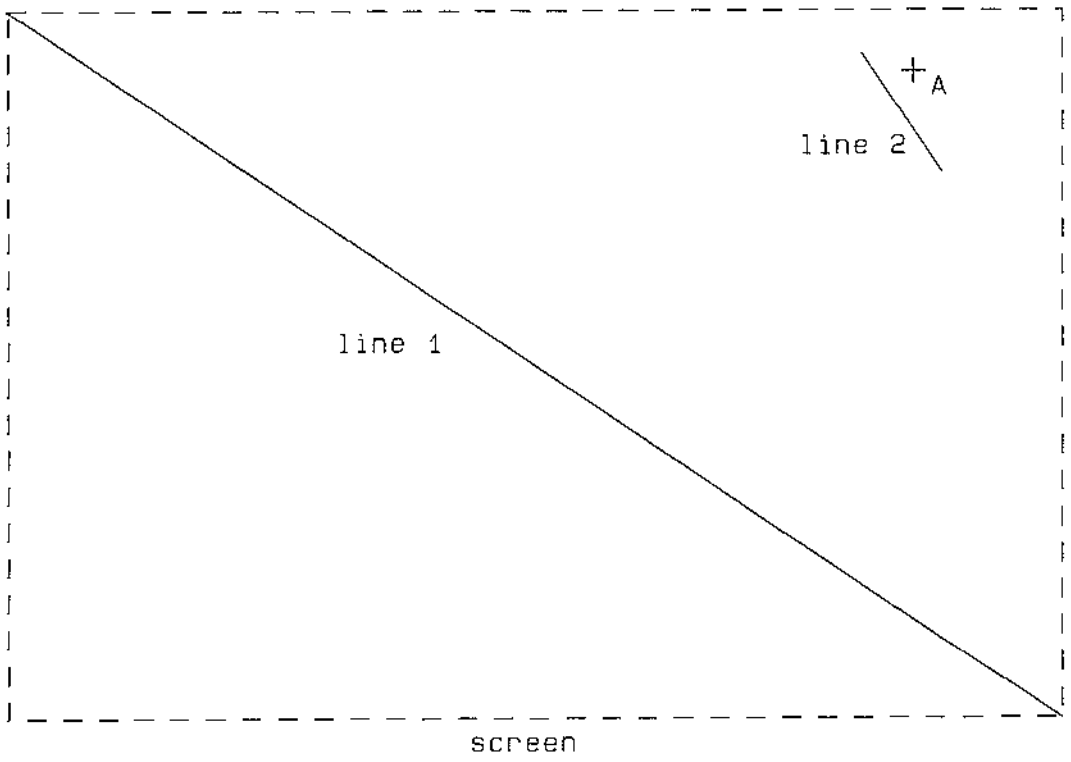


Figure 6-1 Unusual Extent Selection

The cursor is placed at point A to select line 2. If line 1 is stored first in the database then it will come up as the first line that passes the extent test. As far as users are concerned this is a poor action since it is obvious that line 2 is required. Users should not need to be aware of the strategy used by the software for primitive selection, they

merely want actions that seem reasonable.

To avoid problems like that described above a second level of elimination is provided. If a primitive passes the extent test, a calculation is then made to see whether the cursor is within a certain number of screen pixels, called a **trap range**, of the line. If not, the line is rejected and the search moves on by extent testing the next line. If the trap range test is successful, the primitive is displayed in a different colour and the user is asked whether it is the required primitive. If not, the search continues. The trap distance, in pixels, can be readily changed in Paracad. For calculating whether the cursor is within the lines trap distance, the equation of the line is deduced (since its current startpoint and endpoint are known) and the cursor's x value put into this equation (vertical and horizontal lines are treated as special cases) and a y value is obtained. This is then tested to see whether it is within the trap range of the cursor's y value.

This search strategy has the advantage of eliminating the majority of primitives that are not near the cursor by the very quick extent test, and the few that pass this test but are still a long way from the cursor are then rejected by the much slower but more precise trap range test. To the user the selection process appears almost instantaneous.

6.5 Construction Lines

During the parametric drawing construction phase in Paracad, construction lines are added in the same way as ordinary lines - they can have connect, polar, floating etc. endpoints and can be parametrically defined. The major difference is that construction lines are shown in a different colour from other lines to distinguish them once they have been drawn.

During the particular drawing construction phase construction lines are only displayed while the parameter values are being obtained. Once this is done, the particular drawing is displayed with no construction lines visible. In a full implementation it may be worthwhile having a user-selectable visibility option to allow the optional display of construction lines on a particular drawing (and possibly even to control their visibility on a parametric drawing).

6.6 Connects

Any new line that is added in Paracad may have its startpoint or endpoint (or both) connected to existing lines. As described in Section 4.9 a connect may be made to a startpoint, endpoint, midpoint, or equation point (in the case of a vertical, horizontal or polar line the equation point is the point of intersection; for all other lines it is the locus of all points on the line to connect to - extending

to the screen extremities). For later processing of connected lines, particularly during float connects and parameterisation, it was decided to have a linked list of records describing the interconnections. Each record stores the primitive number of the line being connected, the line it is being connected to, the relevant point on the line being connected (ie. whether it is the line startpoint or endpoint being connected) and the point it is being connected to (ie. whether it is linestart, lineend, linemiddle or line equation).

As an example of the use of this affected records list, suppose line 1 has a parametric length and line 2 has its startpoint connected to the midpoint of line 1. When a particular drawing is constructed, the value provided for the length of line 1 will cause line 1 to change its length. A scan can then be made of all affected records to see which lines are affected by line 1 and these lines would in turn have their corresponding positions changed, so this scan would indicate that the start of line 2 must be moved to the new position of the midpoint of line 1. This in turn may lead to changes in the endpoint of line 2 (eg. if it is defined as a vertical line). Another scan would then be made of the affected records to find any lines affected by line 2 and so on.

Whenever a connect is made in Paracad an affected record is also created and added to the end of the linked list.

Because the positions and lengths of lines can change in a parametric drawing a design decision was made for Paracad that a line equation connect to a line (by a freefloat line) will result in the line being connected retaining its proportional position on the line it is being connected to if this line should change its length or position. Figure 6-2 illustrates an example of a line equation connect where line 2 is connected to line 1 at point P.

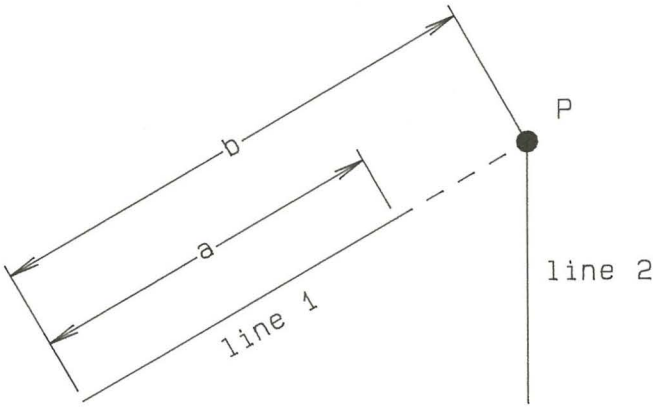
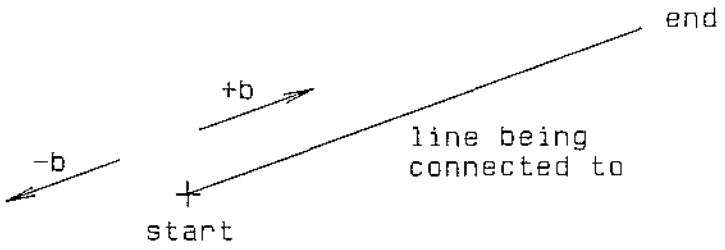


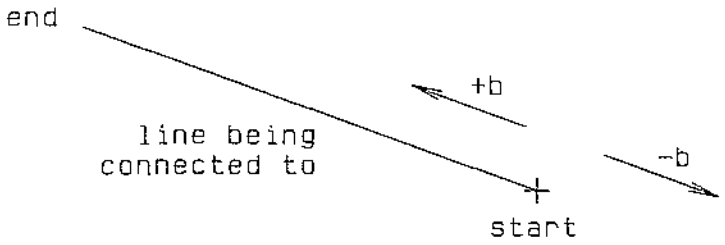
Figure 6-2 Line Equation Connect Position

The ratio **b/a** is saved as part of the affected record (where **a** is the length of the line being connected to and **b** is distance from the startpoint of the line being connected to to the connect position). For any particular drawing, this ratio is maintained regardless of any change in the position

and length of line 1. The length **b** is considered to be a directed length (ie. a negative value for b indicates the connect point is in the opposite direction from the startpoint to endpoint direction. This is shown in Figure 6-3. The ratio **b/a** will always have the same sign as **b**.



a) example 1



b) example 2

Figure 6-3 Sign Notation For Line Equation Ratio

This means if a connect point is placed 2/3 of the way along a line in a parametric drawing, it will always be 2/3 of the way along that line in any particular drawing. In Figure 6-2 if length **a** becomes **a'** in a particular drawing, **b'** is easily calculated by similar triangles to give the formula

$b' = (a' \times b)/a$

Whenever a vertical, horizontal or polar line endpoint is connected to another line some form of calculation is required. For a vertical line endpoint connect to the startpoint, endpoint or midpoint of another line the calculation is trivial since the point being connected to provides the y value of the vertical line's endpoint and the x value of the vertical line's endpoint is the same as its startpoint. For a connect to the line equation the intersection point between the vertical line and the line being connected to is required. First a check is made to ascertain whether the line being connected to is also vertical (if it is Paracad does not permit the line equation connect as there is no intersection point).

Any line can be defined by the linear equation $y = mx + c$ where **m** is the slope of the line and **c** is the intersection point of the line to connect to and the y axis. The equation is also given by $y - y_1 = m(x - x_1)$ where (x_1,y_1) is any point on the line. Further, $m = (y_2 - y_1)/(x_2 - x_1)$ where (y_1,x_2) is any other point on the line. From this we get

$$y = (y_2 - y_1)/(x_2 - x_1)x + (y_1 - x_1((y_2 - y_1)/(x_2 - x_1))$$

and so we get the following:

$$m = (y_2 - y_1)/(x_2 - x_1)$$

and $c = y_1 - mx_1$

for the formula $y = mx + c$

For calculating the intersection point in Paracad, the co-ordinates of the startpoint of the line being connected to are taken as (x_1,y_1) and the endpoint co-ordinates are taken as (x_2,y_2), **y** is the y co-ordinate of the connect point and **x** is the x co-ordinate of the connect point (this is the same as the x co-ordinate of the vertical line's startpoint, so it is known). Note that if the line being connected to is also vertical, this method will fail since it will give a zero denominator for **m**. To avoid this a check is made at the start to ensure the line is not vertical (which would make the two lines parallel).

For a horizontal line endpoint connect a similar approach is used - for linestart, lineend or linemiddle connects the point provides the **x** co-ordinate of the connect point and the **y** co-ordinate is the same as the **y** co-ordinate for the horizontal line startpoint. For a line equation connect the lines are again checked to ensure they are not parallel, but the above formula cannot be used because the line being connected to could now be vertical and thus the $x_2 - x_1$ denominator would be zero. The formula used in this case is:

$$x = m'y + c'$$

where $m' = (x_2 - x_1)/(y_2 - y_1)$

and $c' = x_1 - m'y_1$

(x_1,y_1) and (x_2,y_2) are the same as for the vertical case, and y is the y co-ordinate of the horizontal line startpoint. This is also the same as its endpoint. Note that this formula cannot be used if the line being connected to is horizontal as this would make the denominator for m' zero. This prevents this formula also being used for the vertical line case above.

For polar line endpoint connects the polar line is first checked to see if it is either vertical or horizontal. If so, the connect is processed as for vertical and horizontal connects described above. If not, two different sets of formulae are used depending on whether the polar line is more vertical than horizontal or vice versa. In the case of a polar line that is more vertical the following formulae are used:

For linestart, lineend or linemiddle connects the y co-ordinate of the point selected provides the y co-ordinate of the connect point. The x co-ordinate of the connect point is then found using the formula

$$x = x_3 + (y_1 - y_3)/\tan\theta$$

where (x_1,y_1) is the startpoint of the line to connect to, (x_3,y_3) is the startpoint of the polar line, y is the y co-

ordinate of the polar line endpoint and $\tan\theta$ is the slope of the polar line. Note that there will never be a zero denominator since the line is more vertical than horizontal hence θ cannot be zero. Similarly $\tan\theta$ will not be infinite since this only occurs for a vertical line which is treated as a separate case.

For a line equation connect the line being connected to is checked to see if it is parallel to the polar line. If so, no line equation connect is possible. If not, the formula used to calculate the x co-ordinate of the intersection point is

$$x = (y_3 - y_1 + x_1\tan\beta - x_3\tan\theta)/(\tan\beta - \tan\theta)$$

and for the y co-ordinate it is

$$y = y_3 + \tan\theta(x - x_3)$$

For the case when the polar line is more horizontal than vertical the same formula is used for a line equation connect as that shown above for the more vertical case. For linestart, lineend or linemiddle connects the x co-ordinate of the point selected provides the x co-ordinate of the connect point. The y co-ordinate of the connect point is then found using the formula

$$y = y_3 + (x_1 - x_3)\tan\theta$$

6.7 Float Connects

Float connect refers to the process whereby a floating line endpoint is connected to some other line (which in turn may or may not be floating). It is the only time Paracad permits a floating endpoint to be connected to some other primitive. In Paracad floating endpoints are shown on the graphics display with an open circle. When they are float connected to some other point the circle is filled in to indicate a connect, and if the floating point becomes fully defined or **fixed** then the colour of the circle is changed.

6.7.1 The Float Connect Problems

Two major difficulties arise in the processing of float connects. The first problem is that many floating lines may have their endpoints and midpoints interconnected until eventually one of them becomes fixed. This will then fix all the other lines and so some means of updating all these other affected lines is required.

The second problem is that normally when a floating endpoint is float connected to a freefloat line midpoint or endpoint the two lines still remain floating, but sometimes other interconnections may have been made to the freefloat line that could have made it partially constrained in such a way that the floating lines now become fully defined. This is

shown in Figure 6-4 where part (a) shows line 1 (a vertical line with floating length) float connected to the endpoint of line 2 (a freefloat line).

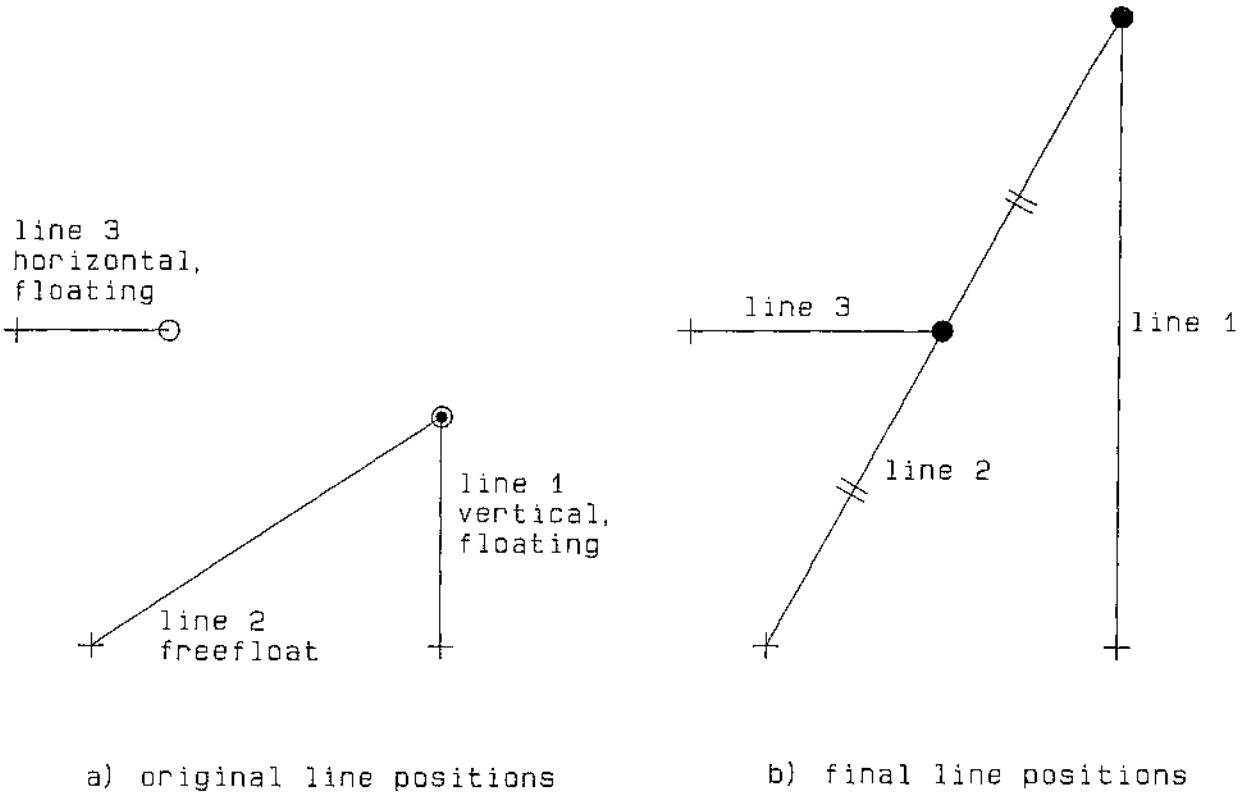


Figure 6-4 A Float Connect Problem

This causes line 2 to have its endpoint x co-ordinate fixed to that of line 1. If line 3 (a horizontal line with floating length) is now float connected to the midpoint of line 2 then the y co-ordinate of the midpoint of line 2 is now fixed. Since the startpoint of line 2 is fixed, this now completely defines line 2 and so it is now a fixed line and it in turn fixes lines 1 and 3 as shown in part (b) of Figure

6-4. Since a much larger number of floating lines than the three shown in Figure 6-4 could be interconnected together some rapid method is needed to find out what partial constraints, if any, exist on any floating lines that are being connected together. This is done by using two fields in the primitive record for constraint type and constraint value.

6.7.2 Constraint Fields

Since Paracad does not permit polar lines with floating angles (for reasons of ambiguity discussed in Section 4.10) the only partial constraint types required are vertical, horizontal, fixed or freefloat.

A vertical constraint type indicates the line has its endpoint x co-ordinate defined. This is the case for lines that have been initially entered as vertical floating lines or for freefloat lines that have a connection between their endpoint or midpoint and another line that is vertically constrained. The constraint value field of the primitive record contains the value of this x co-ordinate.

A horizontal constraint type indicates the line has its endpoint y co-ordinate defined. This is the case for lines that have been initially entered as horizontal floating lines or for freefloat lines that have a connection between their

endpoint or midpoint and another line that is horizontally constrained. The constraint value field of the primitive record contains the value of this y co-ordinate.

A fixed constraint type indicates the line's endpoint is not floating. This is the case for all lines entered without a floating endpoint and for all floating lines that have become fully defined by float connects. The constraint value field is not used for fixed constraint types.

A freefloat constraint type indicates the line's endpoint is unconstrained directionally. This is the case for any line entered as a freefloat line that has either not been float connected to any other line or else only float connected to the midpoint or endpoint of other lines that have freefloat constraint types. The constraint value field of the primitive record is not used for freefloat constraint types.

Whenever a primitive is added, its constraint type and value fields are set to the appropriate values. If a float connect is made these fields are updated accordingly for both the lines involved in the connection. In the example shown in Figure 6-4 the original constraint types would have been vertically constrained, freefloat and horizontally constrained for lines 1, 2 and 3 respectively. Lines 1 and 3 would also have constraint values representing their constant x and y values respectively. When line 1 is float connected to the endpoint of line 2 there is no change to the

constraints on line 1, but line 2 now has constraint type vertically constrained with the same constraint value as line 1. When line 3 is float connected to line 2 the constraint type of all three lines is changed to fixed.

6.7.3 Constraint Field Updating

When a float connect is made the constraint fields may need updating for one or both of the lines involved. The different possible combinations are connecting a floating endpoint (regardless of constraint type) to a fixed line or to a line startpoint, a line endpoint with freefloat constraint type to (the middle or endpoint of) a freefloat line, a freefloat endpoint to a horizontally constrained line or vice versa, a freefloat to a vertically constrained line or vice versa, a vertically constrained endpoint to a horizontally constrained line or vice versa, a vertically constrained endpoint to a vertically constrained line and a horizontally constrained endpoint to a horizontally constrained line. The action taken in Paracad for each of these cases is:

Any floating endpoint to fixed line midpoint or endpoint - the endpoint becomes fully defined so its constraint type is changed to fixed and the floating line is redrawn with its new endpoint.

Any floating endpoint to line startpoint - since startpoints are all fixed in Paracad the endpoint becomes fully defined so its constraint type is changed to fixed and the floating line is redrawn with its new endpoint.

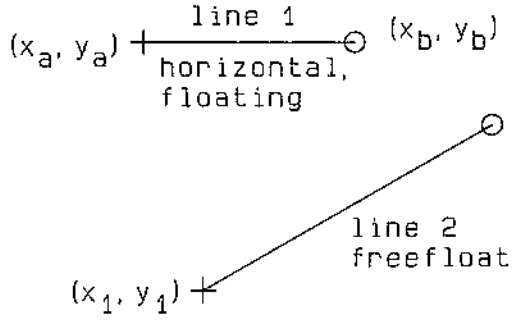
Freefloat endpoint to midpoint or endpoint of freefloat line - both lines remain freefloat although they now are linked together. The line being connected is redrawn with its endpoint moved to (the midpoint or endpoint of) the line being connected to.

Freefloat endpoint to midpoint or endpoint of horizontally constrained line (or vice versa) - the freefloat line endpoint becomes horizontally constrained so its constraint type is changed accordingly and the constraint value is updated to store the y co-ordinate of its line endpoint. No change is made to the horizontal line's constraint fields. The freefloat line is then redrawn in its connected position. Note that if the connect was a horizontally constrained endpoint connect to a freefloat line midpoint the constraint value will not be the same as the y co-ordinate of the horizontal line. This is illustrated in Figure 6-5. If line 1 with startpoint co-ordinates (x_a, y_a) and endpoint co-ordinates (x_b, y_b) is float connected to the midpoint of line 2, which has startpoint co-ordinates of (x_1, y_1) then the co-ordinates of the new endpoint of line 1 will be (x_2, y_2)

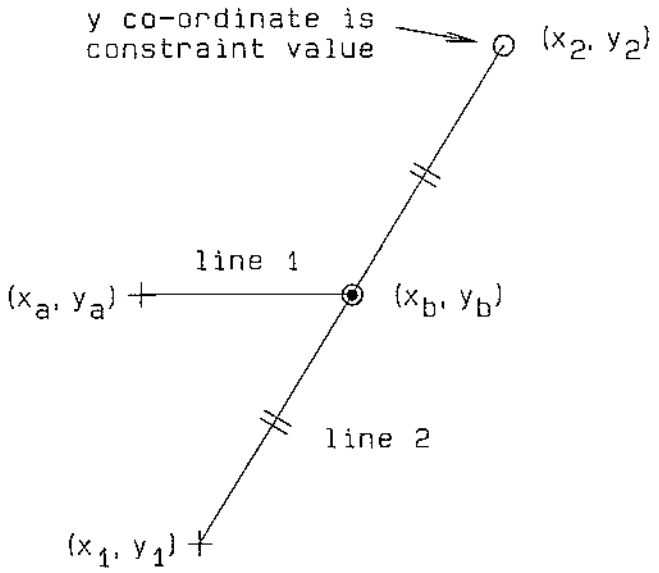
where $x_2 = 2x_b - x_1$

and $y_2 = 2y_b - y_1$.

The constraint value stored will be y_2.



a) original line positions



b) connected line positions

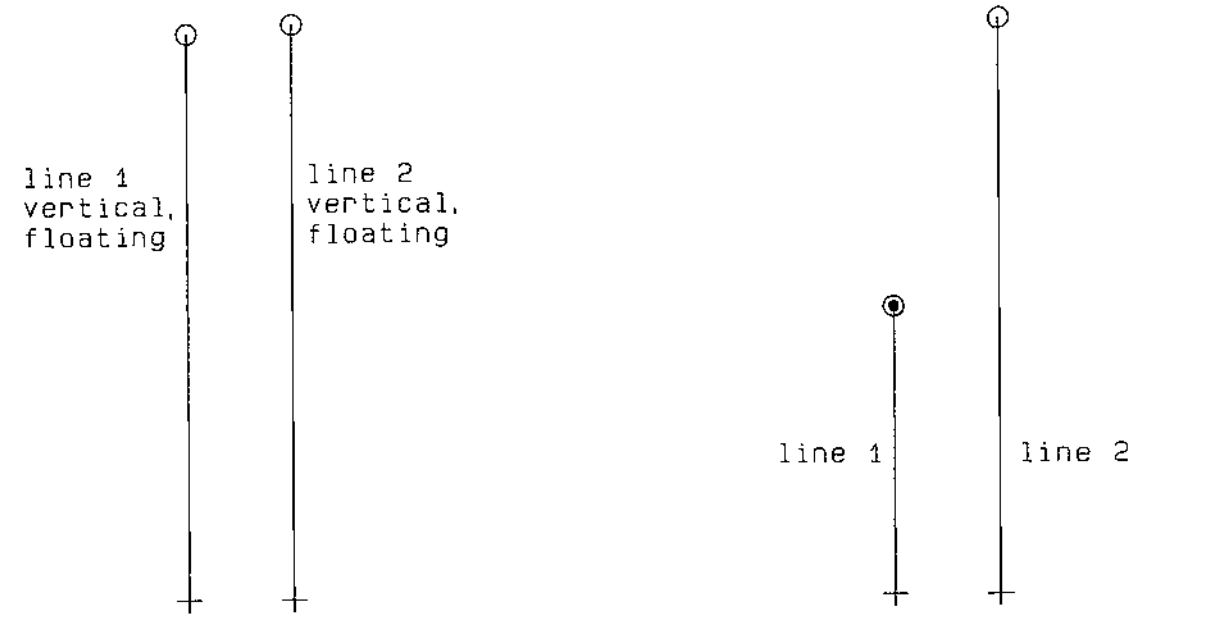
Figure 6-5 Constraint Values Stored

Freefloat endpoint to midpoint or endpoint of vertically constrained line (or vice versa) - the freefloat line endpoint becomes vertically constrained so its constraint type is changed accordingly and the constraint value is updated to store the x co-ordinate of its line endpoint. No change is made to the vertical line's constraint fields. The freefloat line is then redrawn in its connected position. Note that if the connect was a vertically constrained endpoint connect to a freefloat line midpoint the constraint value will not be the same as the x co-ordinate of the vertical line as described in the horizontal line case above.

Vertically constrained endpoint to midpoint or endpoint of horizontally constrained line (or vice versa) - both lines will become fully defined since the horizontal line provides the y co-ordinate of the connect point and the vertical line provides the x co-ordinate. Both lines have their constraint types changed to fixed. Note that if the connect was to the midpoint of one line rather than its endpoint the new endpoint will need to be calculated for this line.

Vertically constrained endpoint to midpoint or endpoint of vertically constrained line - both lines still remain floating. No change is made to their constraint types. The line being connected is changed in length so its endpoint has the same y co-ordinate as the midpoint or endpoint of the line it is being connected to and no change is made to its constraint value. This is shown in Figure 6-6 where line 1 is

float connected to the midpoint of line 2.

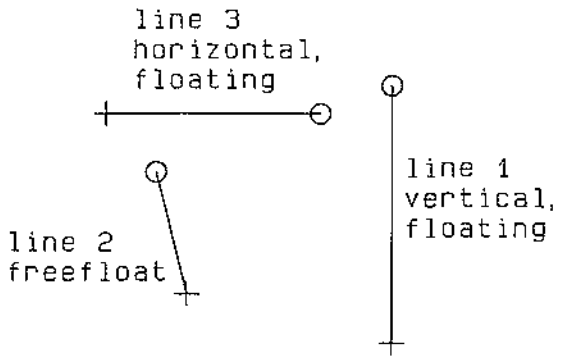


a) original line positions

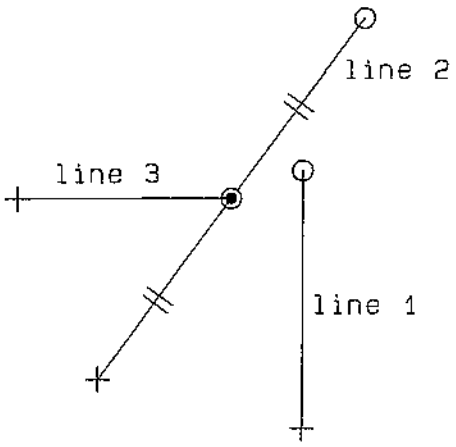
b) line 1 float connected to line 2 midpoint

Figure 6-6 Vertical To Vertical Float Connect

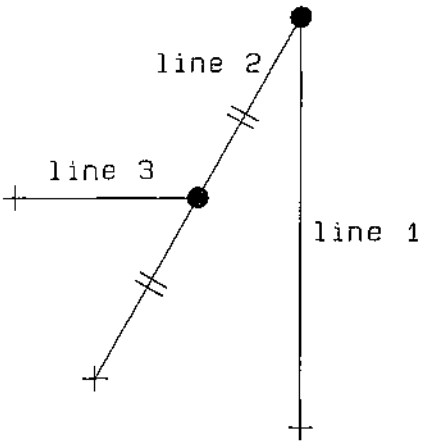
Horizontally constrained endpoint to midpoint or endpoint of horizontally constrained line - both lines still remain floating. No change is made to their constraint types. The line being connected is changed in length so its endpoint has the same x co-ordinate as the midpoint or endpoint of the line it is being connected to and no change is made to its constraint value.



a) original line positions



b) line 3 float connected to midpoint of line 2



c) line 2 float connected to endpoint of line 1

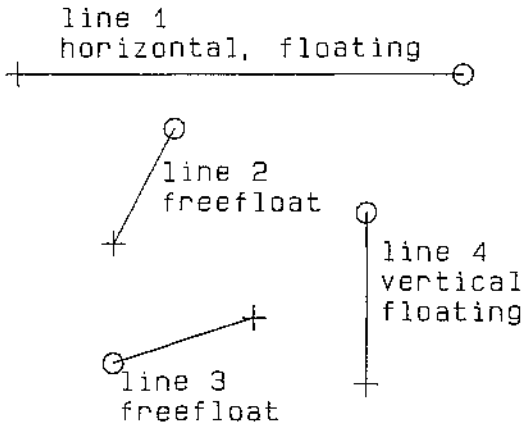
Figure 6-7 A Float Connect Example

6.7.4 Examples Of Float Connects

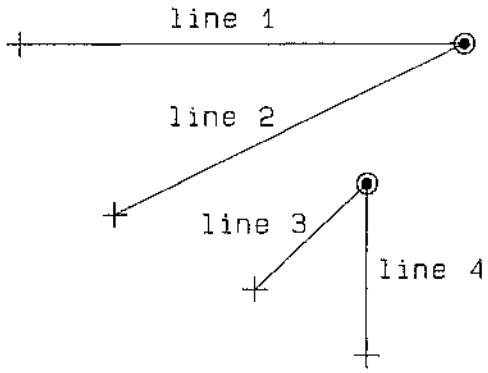
Two additional examples are appropriate to further illustrate the float connect process.

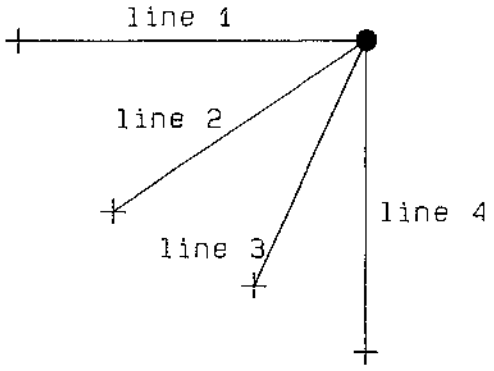
Figure 6-7 shows the same lines used in Figure 6-4 with the connects made in the reverse order. Part (a) shows the original lines. Part (b) shows the result when line 3 is float connected to the midpoint of line 2 - this causes line 2 to move and its endpoint becomes horizontally constrained. Part (c) shows line 2 float connected to the endpoint of line 1 which causes all three lines to be fixed. Notice that the final positions and lengths of the lines are the same in both Figure 6-4 and Figure 6-7.

Figure 6-8 shows a case when connecting two lines that are originally freefloating and then become constrained can cause several lines to become fixed. Part (a) shows the original lines. Part (b) shows the effect of connecting line 2 to the endpoint of line 1 and line 3 to the endpoint of line 4, causing line 2 to become horizontally constrained and line 3 to become vertically constrained. Part (c) shows the effect of connecting line 2 to the endpoint of line 3, causing lines 2 and 3 to become fixed and thus also fixing lines 1 and 4.



a) original line positions





b) line 2 connected to line 1
line 3 connected to line 4

c) line 2 connected to line 3

Figure 6-8 A Second Float Connect Example

6.7.5 Sphere of Influence

Float connects require fast checking of the primitives affected by any selected primitive. To enable this a "sphere of influence" system is employed in Paracad. This involves having a list of all primitive points that are affected by some other primitive. The records in this list are the **affected records** outlined in Section 5.7.3. New records are added to the affected records list whenever a connect is made – whether it is a normal connect or a float connect. The action taken for a normal connect is outlined in Section 6.6.

For a float connect, a record is always set up with the floating point being connected as the **affected point** and the point it is being connected to as the **controlling point**. In addition, a further record is set up if the point being connected to is not fixed (or does not become fixed by the float connect that has just been made). In this record the **affected point** is the point being connected to, and the **controlling point** is the point being connected. An example will indicate the need for this step. Suppose lines 1 and 2 both have freefloat endpoints and line 1 is float connected to the endpoint of line 2. If line 1 is then float connected to some other fixed primitive then line 1 controls the position of line 2 rather than the other way around, while if line 2 is float connected to a fixed primitive then it becomes the controlling primitive over line 1.

Whenever the endpoint of any floating primitive becomes fully defined (fixed) a scan is made of all affected records to find any other primitives affected by this primitive and corresponding changes are made to them. These affected primitives may in turn affect other primitives and so the affected records list is also scanned with them as controlling primitive.

It is not sufficient to have a single scan of the affected records list in an attempt to locate all affected primitives. This can be shown with the aid of Figure 6-9. If the float connect sequence is A to B then A to C then D to E then we get the following sequence of affected records:

a) line 3 point A controlling line 2 point B

b) line 2 point B controlling line 3 point A

c) line 3 point A controlling line 1 point C

d) line 1 point C controlling line 3 point A

e) line 4 point E controlling line 1 point D

At this stage point D becomes fixed and so line 1 is fully defined and can be redrawn. If we now scan through this affected records list in reverse order to update other affected lines we can ignore record (e) since we have just processed it. Record (d) indicates line 1 point C controls line 3 point A so line 3 also becomes fixed and can be redrawn. We are now interested in both lines 1 and 3 as controlling primitives. Record (c) is ignored as the affected primitive (line 1) has been fixed. Record (b) is ignored since the controlling primitive (line 2) is not yet in the

fixed list. Record (a) causes line 2 to become fixed and so it is redrawn. So in this example, a single scan, with a growing list of possible controlling primitives, was sufficient to catch all primitives affected.

Suppose instead the connecting sequence was B to C then A to B then D to E. This gives exactly the same drawing as before, but now the affected records list is:

a) line 2 point B controlling line 1 point C

b) line 1 point C controlling line 2 point B

c) line 3 point A controlling line 2 point B

d) line 2 point B controlling line 3 point A

e) line 4 point E controlling line 1 point D

Point D is now fixed so line 1 is fully defined. Record (d) is ignored as the controlling primitive (line 2) is not yet in the fixed list. Record (c) is similarly ignored as the controlling primitive (line 3) is not yet in the fixed list. Record (b) results in line 2 becoming fixed and being redrawn. Record (a) is ignored as the affected primitive (line 1) is fixed. Note that this scan has not fixed line 3 at any stage and this indicates that a single scan of the affected records is not sufficient to ensure all lines that should be fixed will be located.

The approach used in Paracad to overcome this problem is by having a **queue list** as explained in the next section.

6.7.6 Queue List

Paracad has a list (called a **queue list**) for each new line that becomes fixed during an affected records scan. Initially the queue list will have only the line that was fixed by the float connect. A scan is made of the affected records list and every new primitive affected by this line is added to the queue list. At the end of the scan the next record is taken off the queue list and a new scan is made of the affected records and so on until the queue list is empty. This ensures every line that should become fixed does get fixed, but a method is needed to ensure primitives that have been processed are not added to the list again. For primitives that are being fixed this is not a problem, but if the primitives are still not fully fixed it would cause the queue to never empty. For example, suppose lines 1 and 2 are both freefloating and line 1 has its endpoint float connected to the endpoint of line 2. This would add line 2 to the queue list. A scan of the affected records list would then show line 1 is connected to line 2, so line 1 would be added to the queue list. When line 1 was then taken from the list and processed, line 2 would be added to the queue list again and so on.

This particular case could be prevented fairly easily, but more obscure cases of a similar nature are much harder to detect. To avoid this problem in Paracad another temporary list called the **primscovered list** is used to hold all

primitives that have been processed by the queue list. Before any primitive is added to the queue list a check is first made of the primscovered list to see if it has already been covered, and if so it is discarded.

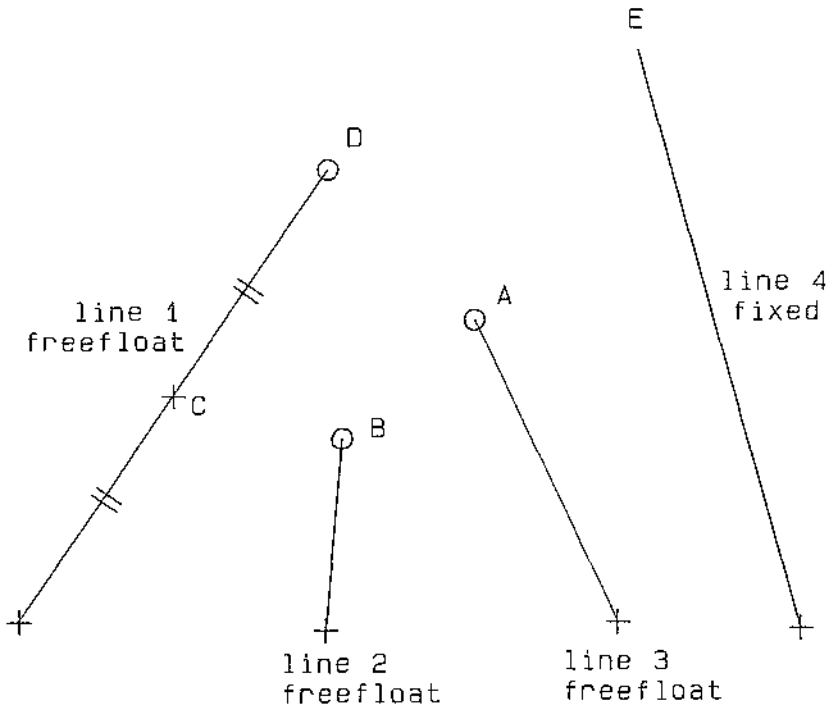


Figure 6-9 Sphere Of Influence Example

This method of handling float connects leads to the first constraints made having the highest priority. This is illustrated in Figure 6-10. Part (a) shows three lines before float connecting. Part (b) shows the result if point B is connected to A then B to C or if B is connected to A then A to C. Note that A and B, which were connected first, remain

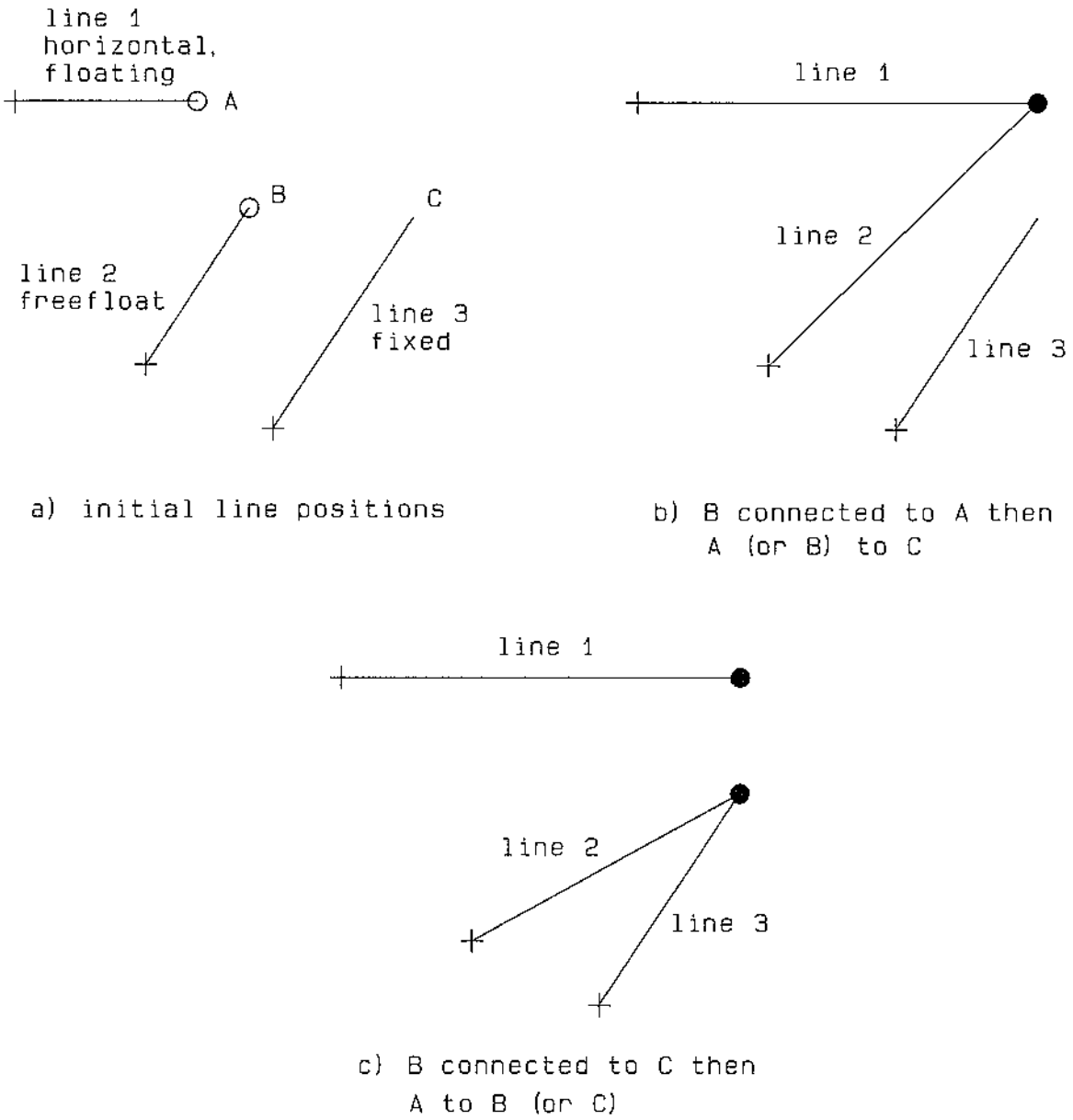


Figure 6-10 First Constraints Have Highest Priority

physically connected. Part (c) shows the result when B and C are connected first then A is connected either to B or C. Note that B and C remain physically connected in this case.

A consequence of the way float connects are handled in Paracad is that the constraint type of any line connected to the midpoint or endpoint of any other line is the same as the constraint type of the line it is connected to (ie. if line 1 is float connected to the midpoint or endpoint of line 2 then the constraint type of line 1 is the same as the constraint type of line 2). Further, if the constraint type of any line endpoint changes, all interconnected (through midpoints and endpoints) lines' constraint points will also change to the new constraint type. Note that although all interconnected lines (through midpoints and endpoints) have the same constraint type, they do not necessarily have the same constraint value.

Whenever any line has a floating endpoint fixed, the floating bit flag in the endtype field of the primitive record for that line is updated to 0 (see Section 5.7.1).

6.8 Parameterisation

The object of the parameterisation phase is to get actual values for the parameters used in a parametric drawing and reconstruct it to form a parametric drawing.

It soon became apparent that it was not possible to have just the parametric drawing files that have been discussed so far and still be able to reconstruct all drawings correctly. An example of one problem area is shown in Figure 6-11.

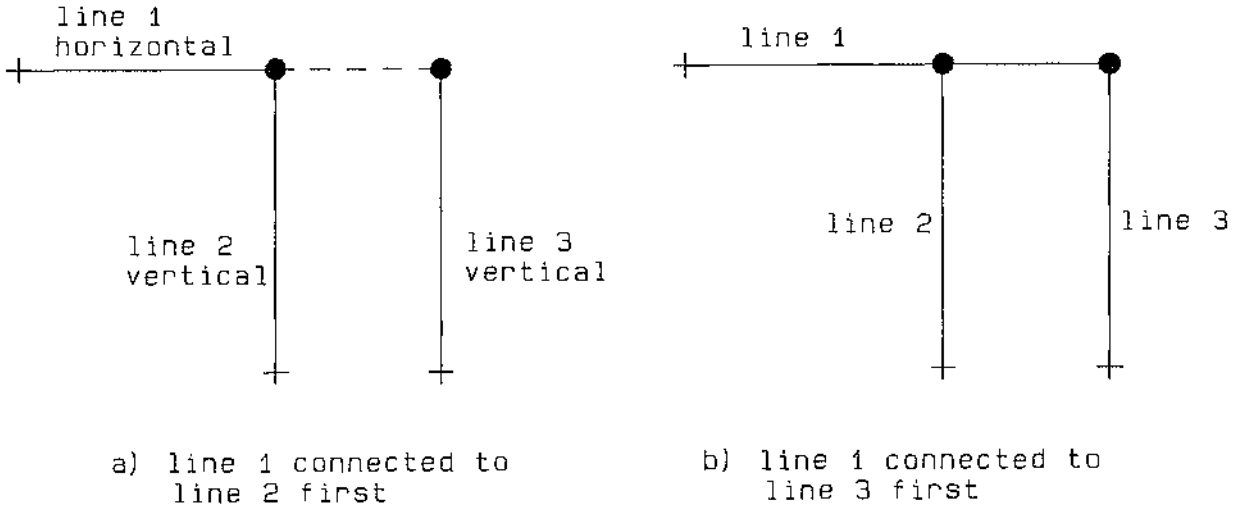


Figure 6-11 Drawing Reconstruction Ambiguity

This shows the two possibilities that can arise if the line 1 endpoint is connected to both the line 2 and line 3 endpoints. Part (a) is the result of connecting line 1 to line 2 first and part (b) is the result of connecting line 1 to line 3 first. The correct version can only be determined if there is some record of the order in which the connects were made. This problem arises whenever the same line endpoint in a drawing is connected (either by a float connect or normal connect) to more than one other point (as either a

controlling or an affected point) where it cannot be physically attached to all the points it is connected to.

Another major problem was to find an efficient strategy for deciding when and where lines should be drawn on the final drawing. With the existing files the only way possible appears to be to draw all fixed and floating lines with the actual values used for the parameters and then connect and redraw them in a process of continuous refinement until they all become fixed. Attempts were made at designing such an algorithm to do this, but they were largely unsuccessful.

Because of these two major problems it was decided the best solution was to have an additional file that stored the order actions were performed in during the parametric drawing, and this order could be followed during the construction of the particular drawing. This strategy has another highly attractive feature in that it means that the reconstruction process effectively operates in the same way as the parametric drawing construction (without the user prompts of course) so the same algorithms can be used with little modification. This provides considerable advantages for extending the package to include other primitives. This approach was adopted in Paracad. The file used to store the order actions are taken in is called the **entry file** and consists of records called **entry records**.

Using these entry records, the parameterisation process in Paracad involves loading a parametric drawing, resetting some fields in the primitive records, parameter setting and drawing reconstruction - in that order.

6.8.1 The Entry File

The entry file used in Paracad to indicate the order of actions taken during the parametric drawing construction is implemented in memory as a linked list of dynamic entry records. The structure of each record is described in Section 5.7.4. The only types of action that need to be logged are the adding of primitives and the connecting of floating endpoints. A variant record is used since the only information needed to be saved for a primitive addition is the primitive number while a float connect requires the floating primitive number and the number of the primitive it is connected to to be saved (it is also advantageous to store the connect position to save later searching of the affected records list).

6.8.2 Parametric Drawing Loading

In Paracad an existing parametric drawing is selected by the user and loaded. The same structures used in the parametric drawing phase are used in the parameterisation phase. A check

is made to ensure the parametric drawing has no unresolved floating endpoints since these would prevent a particular drawing being fully constructed. If any unresolved floats exist, processing is terminated.

6.8.3 Primitive Record Field Resetting

Since the parameterisation method used in Paracad mimics the original parametric drawing construction it is necessary to reset the **unresolved floating endpoint** flag in the endtype field and the **constraint type** field of the primitive records (refer Section 5.7.1) to the original values they had when each primitive was first entered during the parametric drawing construction phase.

6.8.4 Parameter Setting

To obtain the actual required values for the parameters used in Paracad the screen is cleared, then the parametric drawing is progressively redrawn a primitive at a time. As soon as a primitive with a parametric dimension is encountered the primitive is shown in a different colour, a prompt is displayed on the screen indicating the parameter name and a description of the dimension it represents (eg. "**The line displayed has a parametric angle. The parameter name is THETA**"). The user is then requested to supply a numeric value

for the parameter. The primitive is then redrawn in the same colour as the other primitives and the process of adding primitives continues until the original parametric drawing is completely drawn. (Note that the drawing will use all the default values and positions supplied during the parametric construction phase - although the parameter values are being entered by the user, they are not yet being used to modify the drawing). Once all parameters have been obtained, the screen is again cleared and drawing reconstruction occurs.

6.8.5 Drawing Reconstruction

In Paracad the reconstruction of a parametric drawing to create a particular drawing is achieved by scanning each of the records in the entry file in turn.

If the entry record is a primitive addition the corresponding primitive record is obtained. If it has a connected start point (ie. start point affected by some other primitive - not controlling it) the other primitive must be already recalculated and so the new start point can be computed. If the line is vertical or horizontal its constraint value is then reset accordingly (note that this could not be done at an earlier stage than this because the constraint value is dependent on the start point). If the primitive has a parametric dimension (or dimensions) the corresponding parameter is looked up in the parameter records and the value

is stored in the appropriate primitive record data field (ie. the **End3** or **End4** data fields - depending on what type of dimensions are parametric). The line endpoint, or temporary endpoint for floating lines, is then calculated using the same routines used for parametric drawing construction (with the user prompts, user inputs and display feedback areas removed).

If the entry record is a primitive connect then a float connect is done using the same routines utilised in the parametric drawing construction (with the user prompts, user inputs and display feedback areas removed).

6.9 Plotting

A plot routine is included in Paracad to allow hardcopy output of particular drawings. This routine opens up a spool file, with a name specified by the user, then writes the necessary handshaking and setup commands to prepare an HP7475A A3/A4 plotter. The routine then scans through each primitive in order and writes, to the spool file, the appropriate plotter commands to draw the primitive. Finally, the spool file is appended with commands to terminate the plot. The actual plot is then made by exiting from Paracad and using a separate spool file transfer program to send the spool file to the plotter.

The plot routine is currently only configured to output completed particular drawings but it could readily be extended to include parametric drawings as well.

CHAPTER 7

PARACAD PERFORMANCE AND FUTURE DEVELOPMENTS

The first part of this chapter considers the operation of Paracad in its current state. The speed of the package at both the entry stage (parametric drawing construction) and the output stage (particular drawing construction) is discussed and conclusions are made as to its reliability and user-friendliness.

The second part of the chapter outlines future developments that could be made to Paracad to improve its operation and indicates areas for further research into parametric CAD in general.

7.1 Paracad Performance

7.1.1 Speed

In general the performance of Paracad in terms of speed is excellent. The four main areas where speed is important are during interactive drawing, float connects, redraws and parameterisation.

1. Interactive Drawing Speed

Studies have shown the importance of the sub-second response time of computer software during interactive sessions and the effect of this response time on the user's concentration patterns [GOOD. 1978]. The time taken for Paracad to respond with a new menu whenever an existing menu choice is selected is too rapid to accurately measure, but it is estimated to be less than 0.1 second. Keypresses are buffered so even if an experienced user can type faster than the menu displays can respond there is no loss of input data and the menus will eventually "catch up" to the user. After less than an hour's experience with Paracad it is common for users to key in most required menu choices without even looking at the changing menus.

Response time to digitiser movements is limited by the 9600 baud interface and by the cursor drawing routines in Paracad. The cursor drawing routines have been written in Assembler language for optimum speed and the response speed of the cursor is sufficiently fast to feel natural to the user without having any appreciable lag evident. Stylus button selections (eg. for indicating line startpoints or endpoints) are acknowledged with an audible "pip" when they are accepted so the user has both

visual and audible feedback.

Rubberbanding also shows no speed deficiencies in Paracad - in fact as mentioned in Section 6.2 initially rubberbanding was performed by continuously flashing the line, but when the cursor was stationary this resulted in flashrates so fast that the line became difficult to see. This problem has been overcome by keeping the line displayed continuously until the cursor moves more than a pixel in any direction. Rubberbanding a line cause a reduction in the speed of the cursor following the stylus, but the speed is still fully acceptable - even for rubberbanding polar lines (which require most calculation). This is due to the rubberbanding algorithms being optimised for speed.

Once a line endpoint is selected the line appears almost instantaneously - the only change the user notices is the sudden colour change from a rubberbanded line to a drawn line (with a connect or float circle added if appropriate).

2. Float Connect Speed

Float connecting involves a significant amount of calculation, especially if a large number of floating lines are interconnected. Benchmark tests were performed

to find the times required for float connects. In these tests a vertical floating line and a horizontal floating line were drawn. A number of other floating lines were then drawn and one connected to the midpoint of the vertical line, the next one connected to the middle of the previous one and so on (with another chain of floating lines connected to the horizontal line). When the required number of lines for the benchmark test were drawn, the horizontal line was float connected to the end of the vertical line and the time to completely recalculate and draw the new line positions was recorded. Times for these tests were less than 0.1 seconds for 10 lines, 2.6 seconds for 50 lines and 7.1 seconds for 100 lines.

These times are considerably better than was originally forecast considering this is an area that was expected to be very slow. This speed is due to the efficient method employed for float connections using the various linked lists. In addition, during the float connects lines are being rapidly redrawn on the screen so the user is left in no doubt that something is happening.

3. Redraw Speed

A redraw feature was included in Paracad to allow parametric drawings loaded from file to be displayed on

the screen. This routine is also used following a float connect - all new line endpoint co-ordinates are first calculated then a complete screen redraw is done. Benchmark tests were done on redraw speeds for different numbers of lines and the results were 1.0 seconds for 100 lines and 4.8 seconds for 500 lines. Again the speed is impressive and is better than originally hoped for.

4. Parameterisation Speed

Parameterisation is the most time-intensive part of Paracad but once parameter values have been entered (which is user-speed dependent rather then computer-speed dependent) the user does not need to stay at the computer so time is not overly critical.

Benchmark parameterisation tests were taken of some worst case examples with every line float connected to another line and a time of 58.9 seconds was recorded for a 50 line drawing and 2 minutes 21.6 seconds for a 100 line drawing.

These figures are more than an order of magnitude better than originally hoped for. Again the speed is due to careful algorithm and data structure design.

7.1.2 User Friendliness

Considerable efforts were made in the design of Paracad to make it as user friendly as possible. User-friendliness is essential to a new user but can often slow down and irritate an experienced user. To try and strike a suitable medium careful consideration was given to the menu structure, menu choices, user feedback (eg. rubberbanding) and the prompts provided. The resulting system enables an experienced user to move rapidly and unhampered from task to task while the visual and audible prompts used assist the inexperienced user. Explanatory error messages are displayed whenever an invalid input is made.

Frequent modifications were made to Paracad in an effort to optimise user-friendliness, and as a result the package is fast and easy to use for both novices and experts.

The addition of an on-line help facility would go further towards assisting the beginner but this was not implemented because it was outside the scope of the original aims of Paracad.

User-friendliness in the parameterisation phase is evident in the method of prompting for parameter values - the line affected by the parameter is displayed in a different colour from other lines, the parameter name is displayed and the feature being controlled by the parameter (eg. the length) is

described on the screen. This is an important area for user-friendliness as users could easily become confused in associating parameters with their correct primitives in complex drawings containing many parameters. Further, the user supplying the parameter values in the particular drawing may well be a different user from the one who designed the parametric drawing and so unfamiliarity with the drawing could increase possible confusion.

7.1.3 Reliability

Exhaustive testing of all features of the current version of Paracad has failed to reveal any remaining "bugs". All features implemented work as required by the design specifications. The system is not yet fully protected against all forms of invalid user input - for example the entry of alphabetic characters where numeric input is required can cause the program to terminate in some places. Protecting against all forms of invalid input was not considered an essential part of Paracad since it was mainly intended for feasibility testing.

7.1.4 Parametric Variety

One of the first things that becomes apparent when using Paracad is how completely different a particular drawing can

look from the original parametric drawing. Even for drawings with only a few lines in them it can be hard to reconcile the parametric and particular drawings in some cases. This is especially true if parameter lengths of zero or negative values are supplied. Examples of some of the widely different possible combinations possible for the simple four line parametric drawing shown in Figure 7-1 are illustrated in Figures 7-2 to 7-12.

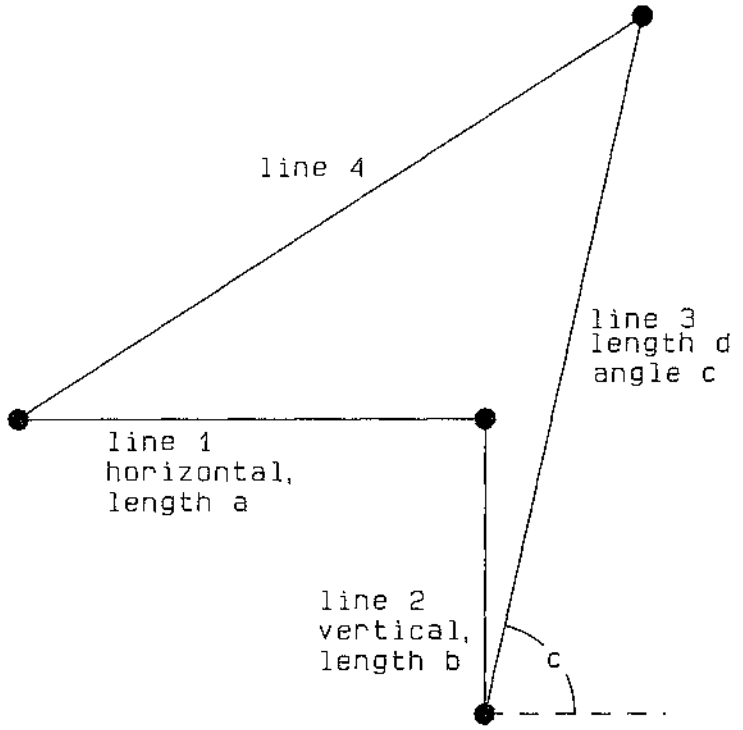
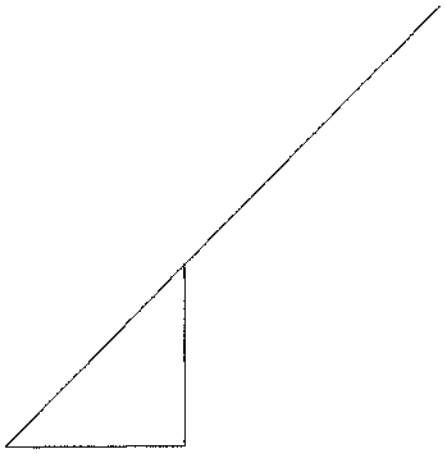


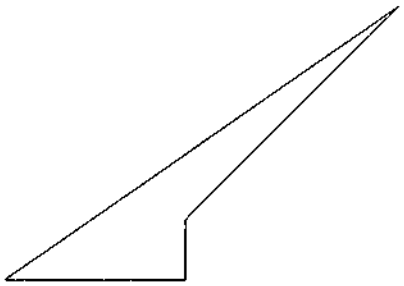
Figure 7-1 Original Parametric Drawing

All these particular drawings are produced using the inbuilt Paracad plotter routine and are just a small sample of the different possibilities. The wide variety of particular drawings conceivable from such a simple parametric drawing gives an indication of the diversity possible for more complex parametric drawings and demonstrates the flexibility provided by a parametric CAD system.



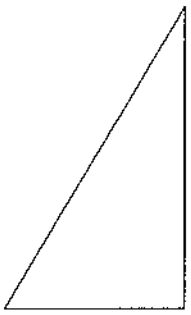
a = 300
b = 300
c = 45
d = 600

Figure 7-2 Particular Drawing



a = 300
b = 100
c = 45
d = 500

Figure 7-3 Particular Drawing

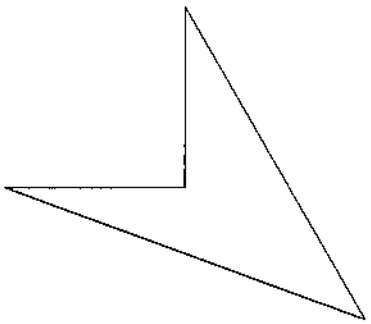


a = 300
b = 0
c = 90
d = 500

Figure 7-4 Particular Drawing

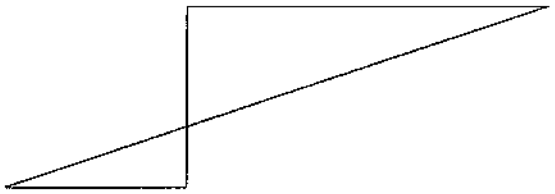
a = 300
b = 0
c = 0
d = 300

Figure 7-5 Particular Drawing



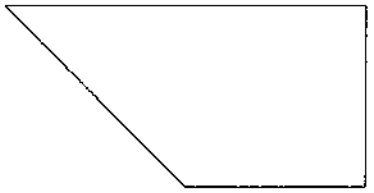
a = 300
b = 300
c = 300
d = 600

Figure 7-6 Particular Drawing



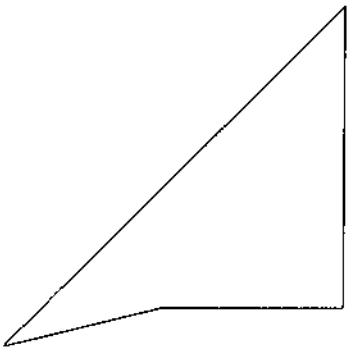
a = 300
b = 300
c = 0
d = 600

Figure 7-7 Particular Drawing



a = 300
b = 300
c = 0
d = -600

Figure 7-8 Particular Drawing

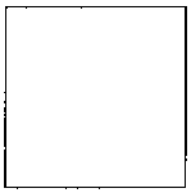


a = 300
b = 500
c = 45
d = -800

Figure 7-9 Particular Drawing

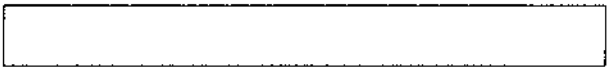
a = 0
b = 0
c = 90
d = 600

Figure 7-10 Particular Drawing



a = 300
b = 300
c = 180
d = 300

Figure 7-11 Particular Drawing



a = 1000
b = 100
c = 180
d = 1000

Figure 7-12 Particular Drawing

7.2 Future Developments

7.2.1 Deleting Primitives And Aborting Operations

Paracad currently does not allow primitives to be deleted. In addition to this it is not possible to abort an operation midway through it (eg. if a polar line has been specified then the user changes his mind before the line is finished). These two limitations mean that any errors made when entering lines cannot be altered - they either have to be left as they are or the whole drawing needs to be started again. This is a major shortcoming as input errors appear to be made rather frequently during CAD sessions. The facility to abort from any operation and a means of deleting primitives are both strong requirements.

Providing a method of aborting an operation should not pose any great difficulties. A key such as the **ESC** key could be used for this purpose and the processing involved would be to cancel the pending action and revert to the previous menu. This may involve disposing of any dynamic list variables that were set up by the pending action.

Deleting an existing line could prove a more difficult task. For an isolated line (not connected to any other line) it would merely involve disposing of the primitive record and changing the next record pointer of the previous primitive

record, but for a line that has other lines connected to it things are more complex. The line to be deleted could earlier have resulted in other floating lines becoming fixed and so if it is removed they will need to be updated. Worse still, it is possible that another line may have had its startpoint connected to one of these subsequently fixed floating lines. This would then mean that deleting the line would leave "illegal" lines remaining (ie. lines with floating startpoints). Two methods for overcoming this difficulty are to either delete all "illegal" lines as well, or to allow line startpoints to be floating. Of these the second is probably the most attractive for the user.

Resetting of various bit flags for affected lines and disposing of the relevant affected records from the affected records list would also be required for all lines that are connected to a line to be deleted.

7.2.2 Other Primitives

Any full parametric CAD package obviously needs to support more primitives than just lines. Adding circles, arcs, text etc. increases the complexity of the exercise considerably and further study is required in this area.

7.2.3 Floating Polar Line Lengths

Currently Paracad supports freefloat lines and vertical and horizontal lines with floating lengths. As discussed in Section 4.10 having polar lines with floating angles gives rise to ambiguities, but it would be useful to allow polar lines with floating lengths.

Floating line processing is performed largely using the constraint type and constraint value fields of the primitive records. Extending Paracad to include floating polar line lengths would require altering this processing. One possible method would be to have three constraint values stored rather than one - these being the coefficients **a**, **b** and **c** of the equation

$$ay + bx + c = 0$$

This equation represents the equation of the line the floating endpoint is constrained to. This would cover polar, vertical and horizontal lines. Constraint types could also be extended for polar lines to include **angularly constrained** (in fact the constraint type is no longer needed as this information can be deduced from the constraint values, but it would significantly speed up processing).

If Paracad is extended to cover other primitives as well as lines and these primitives are also able to be floating the complexity of the problem would be considerably magnified.

7.2.4 Conventional CAD Features

For Paracad to be used in production work it would be desirable to include many of the conventional CAD features that are not currently supported. These include viewing control (to set up windows, viewports, allow panning etc.), groups and transformations of groups and primitives. These features are all described in more detail in Section 2.2.

7.2.5 Methods Of Supplying Parameter Values

Paracad has only one method of supplying values for parameters during particular drawing construction. This should be extended to include some or all of look-up tables, decision tables, formula calculations and input from other programs or files. All of these should be able to be implemented without any major side effects on Paracad.

7.2.6 Formula Processor

A highly desirable feature of a parametric CAD package under certain circumstances is being able to specify formulae for parameters rather than just a parameter name. For example a line's length could be specified by the parametric formula

2a - b where **a** and **b** are parameter names used elsewhere. Implementing a formula processor for handling such parametric

expressions at particular drawing construction time is not a trivial exercise and would probably require some form of parsing of the formula. This area warrants further study.

7.2.7 Repeated Groups

Another useful extension to any parametric CAD system would be the ability to have certain sections of the drawing repeated where the number of times the section is repeated depends on the value provided for some parameter (eg. an explicit count parameter, or the length of some primitive etc.). An example of the use of such a facility might be in designing a roof where the number of supporting members depends on the roof's dimensions. Such a facility would need considerable further study.

7.2.8 Parametric Decision Making

A powerful extension to any parametric CAD system would be the ability to control the actions taken (not just the dimensions) during particular drawing construction by some parameter value. For example if a particular parameter value is greater than a specified number a certain set of primitives will be drawn, otherwise a different set will be drawn. A use of such a feature could be to control the number of holes in a flange or plate by entering the number of holes

as a parameter.

Again considerable further study would be necessary before deciding whether it was feasible to implement such a feature.

CHAPTER 8

CONCLUSIONS - IS PARAMETRIC CAD FEASIBLE?

The studies made and experiences gained with Paracad indicate that not only is it possible to implement a parametric CAD package involving line primitives, but such a package can be made easy to use, powerful and provide considerable flexibility.

The indications are that extending this to cover other primitives would also be feasible. One of the more difficult areas of Paracad to implement was the idea of floating lines, but floating lines provided a considerable increase in the power and flexibility of Paracad. If this floating concept was extended to cover other primitives then matters would be complicated considerably but the gains could be well worthwhile.

8.1 Advantages Of Parametric CAD Systems

One major advantage of a parametric CAD system is that a skilled designer can be used to generate a parametric drawing and then, with careful program design and judicious choice of parameter names, the generation of particular drawings from

this parametric drawing becomes a simple process with the manual input of parameter values able to be handled by unskilled users. All engineering decisions, safety considerations, legal aspects etc. can be incorporated at the parametric design stage.

Another major advantage is the reduction in effort required when families of components can be produced using a single parametric drawing to produce multiple different particular drawings. The savings in drawing time increase with the number of different particular drawings produced.

A third major advantage is the ability to construct a drawing without knowing its dimensions. A parametric drawing can be constructed and the actual dimensions can be left until during the particular drawing stage. This approach could be used for "what if" testing to examine the results of altering various dimensions. As an example, a parametric drawing of the side view of a prototype car (or any other product with an iterative design cycle) could be constructed and parameters changed at will to optimise the final appearance.

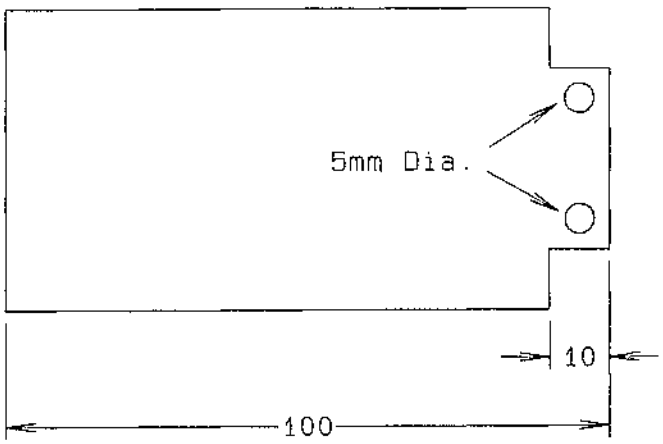
A fourth major advantage of a parametric CAD system is its potential to automate the output drawing from a combination of different input methods (once the parametric drawing has been produced). The input to the parameterisation process can be from output piped from another program, manual input, lookup tables, files and so on. A simple example of this is

to set up a parametric drawing for a bar chart using a number of bars with parametric widths, lengths and positions. Input to the parameterisation process would be the heights and widths of the bars with a constant (or variable) being added to each bar position to provide horizontal separation. A more practical example is the design of modular houses or sheds. By having parametric dimensions the size and shape of the structure can be easily changed to meet the needs of the customer.

8.2 "Usability" Of Parametric CAD

An important facet of any interactive computer package is its ease of use. Many of the concepts involved in parametric CAD are natural to humans - for example the idea of making a width or length larger or smaller is easy to comprehend. Unfortunately, as discussed earlier in the thesis, ambiguities can easily arise in such cases and what seems obvious to one person may seem ridiculous to the next. An example of such an ambiguity appears in Figure 8-1 which shows a partially dimensioned bracket. The bracket is 100mm long. If an order comes through for some of these brackets with the request "make them 120mm long instead of 100mm" an ambiguity immediately arises. Does the wide part of the bracket get extended, or does the narrower end piece get extended, or do they both get extended -and if so, how much does each change by. If the narrower part is to be extended,

what happens to the bolt holes? To the person making the order the answers to these questions are obvious since he has prior knowledge, but a parametric CAD system has no such prior knowledge.



All dimensions in mm.
Not to scale

Figure 8-1 Ambiguity Through Lack Of Prior Knowledge

To overcome such ambiguities a parametric CAD system must either enforce a particular interpretation to potentially ambiguous situations or else it must request further information from the user to resolve the ambiguity. The balance between these two methods largely determines the usability of the package. Too much enforcing of particular interpretations by the package will reduce flexibility and

limit the user. On the other hand, requiring the user to meticulously describe each action increases flexibility but slows productivity and frustrates the user.

An example of ambiguity resolution in Paracad that combines both these methods is the strategy employed for connecting line endpoints. If a vertical line endpoint is connected to some point on another line the program resolves ambiguities by always assuming the point provides the y co-ordinate only for the vertical line. User flexibility still exists as the user chooses which point on the line he wants to connect to.

In some cases the program should make the decision required to resolve ambiguitites, in some cases the user should make it, and in some cases it should be a combined effort. Deciding which of these three is appropriate in each particular case is an important decision in terms of the user friendliness, usability and general flexibility of the program.

8.3 Adapting An Existing Package

The question of how difficult it would be to adapt an existing CAD package to include parametric features can only be answered with respect to line drawings and depends largely on the current structure of the particular CAD package in question. Most of the routines could be readily adapted (eg.

windowing, panning, line drawing, cursor tracking, plotting etc.) if they exist as modular routines. The data structures would need to be extended, an overall shell would need to be written and a number of new routines for handling parametric-specific areas would have to be written.

8.4 Is Parametric CAD Feasible?

The main purpose of this thesis was to investigate the feasibility of parametric CAD. While the investigations were limited to lines only, reasonable assumptions can be based on extending this to cover other primitives.

A line-only system is definitely feasible and can be user friendly and flexible. Any other primitives that might be added can be fully defined by a finite number of points and since the position of any of these points can be described by line segments it should be possible to extend a parametric CAD system to cover all common primitives.

As an example consider adding circles. One common way of defining a circle is by its centre and radius. The centre position could be described by a parametric construction line from some known point, and the radius could be described by a parametric construction line from the centre. This would allow both the circle position and size to be parametrically altered. Another way of defining a circle is by any three

points on its circumference. Again the position of each of these points could be described by a parametric construction line (once more this allows both variable circle size and position). So by proving the feasibility of parametric CAD for lines only the feasibility for other primitives has also been largely proved. There appears to be no reason why a parametric CAD system able to perform all the functions suggested in the introduction cannot be designed. The only real difficulties arise when extra features, such as floating primitives, are required and such so such features would need careful further study.

This project has shown that parametric CAD is viable, useful and usable for line drawings and should be able to be extended to cover all common primitives used in conventional CAD packages.

Appendix A

Notation Used In Illustrations

+	startpoint of line
●	connect point (on startpoint or endpoint of line being connected)
○	floating line endpoint (on end of floating line)
◉	floating line endpoint that has been connected to another floating line and is not yet fully fixed
| or ||	indicates equal length line segments

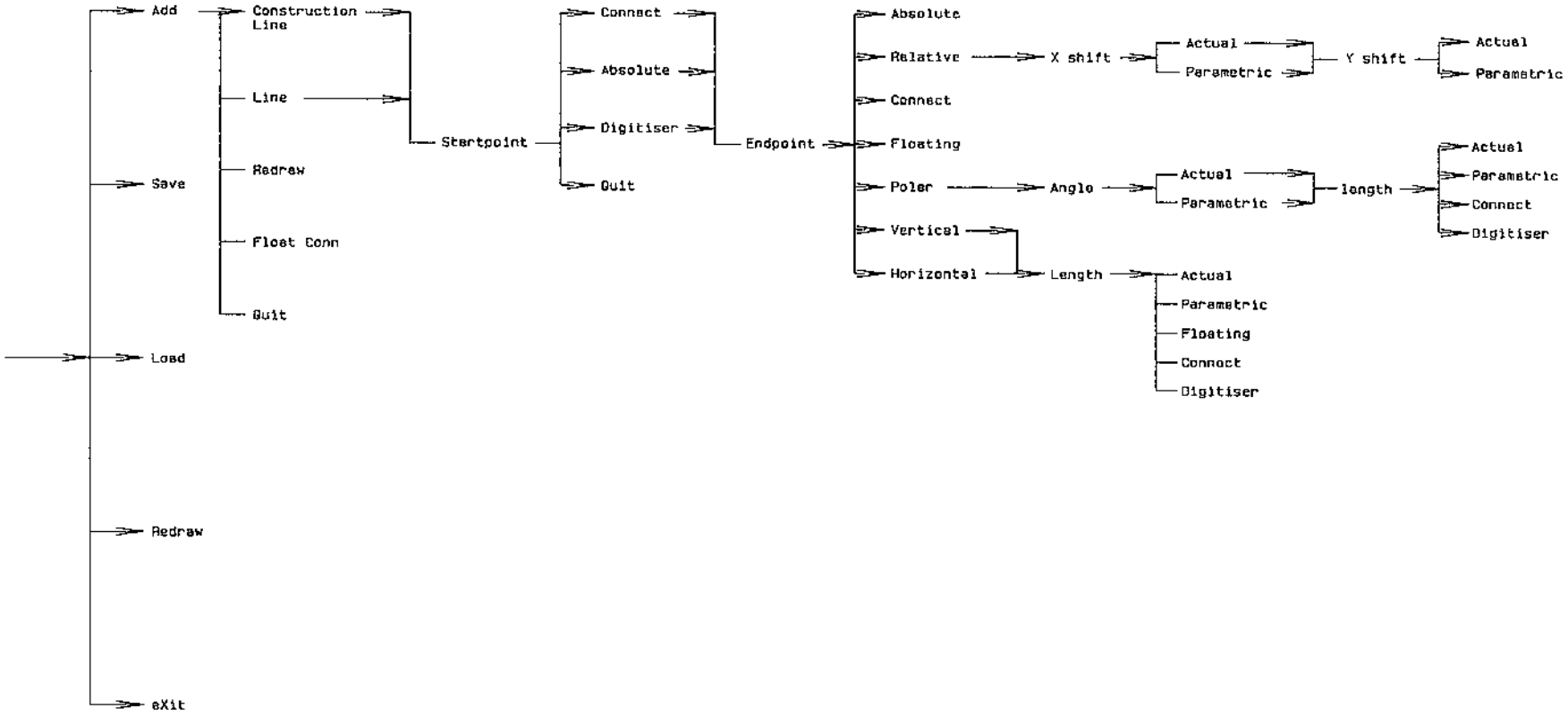
Points are indicated in capital letters (eg. A, B, C)

Lines are numbered (eg. 1, 2, 3)

Parameters are indicated in lower case letters (eg. a, b, c)

Appendix B

Paracad Menu Structure



BIBLIOGRAPHY

Bes. 1983 - Besant,C.B.
Computer-Aided Design And Manufacture
Wiley, 1983

CAD1 1984 - Teicholz,E. (Editor-in-Chief)
CAD/CAM Handbook
McGraw-Hill, 1984

CAD2 1985 - Barr,P.C. Krimper,R.L. Lazear,M.R. Stammen,C.
CAD Principles And Applications
Prentice-Hall, 1985

Enc. 1983 - Encarnacao,J. and Schlechtendahl,E.G.
Computer Aided Design
Springer-Verlab, 1983

Find. 1981 - Findlay,W. and Watt,D.A.
Pascal - An Introduction To Methodical
Programming
Pitman, 1982

Fol. 1984 - Foley,J.D. and Van Dam,A.
Fundamentals Of Interactive Computer Graphics
Addison-Wesley, 1984

Good. 1978 - Goodman,T. and Spence,R.
The Effect Of System Response Time On Interactive Computer Aided Problem Solving
Computer Graphics 12 (3) 1978, pp100-104

IBM.1 - IBM Corporation
Technical Reference Manual - Options And Adapters, Volume 3
IBM Corporation, 1984

Kro. 1982 - Krouse,J.K.
What Every Engineer Should Know About Computer-Aided Design And Computer Aided Engineering
Marcel Dekker, 1982

Kurt. - Kurta Corporation
Kurta Graphics Tablet User Manual
Kurta Corporation, 1984

Merm. 1980 - Mermet,J. (Editor)
CAD In Medium Sized And Small Industries
North Holland, 1980

Myer. 1985 - Myer,D. and Wohlers,T.
Realising The Potential Of Micro-Based CAD
Autofact '85 Conf. Proc. (Detroit) 1985

New. 1973 - Newman,W.M. and Sproull,R.F.

Principles Of Interactive Computer Graphics

McGraw-Hill, !973

Pav. 1982 - Pavlidis,T.

Graphics And Image Processing

Computer Science Press, 1982

Ryan 1985 - Ryan,D.L.

Computer-Aided Graphics And Design

Marcel Dekker, 1985

Turb. 1985 - Borland International

Turbo Pascal 3.0 Reference Manual

Borland International, 1985

Wohl. 1984 - Wohlers,T.

Potential Of Personal Computer CAD Systems

Autofact 6 Conf. Proc. (Anaheim) 1984,