Introduction

The chapters in this book are divided into sections, each dealing with the representation and solution of a single geometrical problem. The sections are usually headed with a diagram to make reference easier. As far as possible uniform conventions have been observed throughout diagrams, algebra, and code. The nomenclature adopted is a compromise between uniformity and the symbols already in wide use for certain equations. It is as follows:

A, B, C, and D	Coefficients of implicit equations of lines and planes
F, G, and H	Coefficients of parametric lines
I	Subscript of a known radius
J, K, L, M, and N	Labels of points
P and Q	Vectors
R	Radius and distance in general
S and T	Parameters
U, V, and W	Second set of coordinate values
X, Y, and Z	Coordinates in space
α, β, γ, and θ	Angles (in radians)

In the text and diagrams only points appear as capital letters. In the coding examples capitals are used throughout to avoid problems for readers whose computer facilities do not support lower case

letters. Symbols are never used for different purposes in upper and lower case.

Subscripts are widely employed in a number of different ways. They may define more than one set of constants, as in two different straight lines:

$$a_1 x + b_1 y + c_1 = 0$$

$$a_2 x + b_2 y + c_2 = 0$$

They may specify values corresponding to points, so that, for example, the coordinates of point J would be (x_J, y_J), or they may indicate parameter values; x_0 might be the value of x where a parameter, t, say, was zero. Generally, subscripted variables are those known in advance, and variables without subscripts are to be calculated. Primed variables, such as x', are avoided as far as possible, but are sometimes used where numerical subscripts would be inappropriate.

In drawing the diagrams the aim has been to strike a balance between consistency and an excessive use of special symbols, line types, and tones. Three line types (dashed, thin, and thick) show lines of increasing interest, and dashed lines are also used where only distance, rather than an actual line, is to be indicated. Three intensities of tone are used to define areas, and to indicate surfaces and solids. Points given as part of a problem are labelled with the letters J to N, as mentioned above, while unknown points are labelled (x, y), with numerical subscripts on x and y when there is more than one value for each. Lines are unlabelled if they are given, unless there is more than one, when they are numbered. Lines to be calculated are labelled with their equation: ax + by + c = 0. Planes are labelled similarly. After the first diagram in a section, which usually shows a simple case of the geometry to be discussed, the conventions are often relaxed, especially in complicated diagrams enumerating all possible cases of a problem. This informality in labelling also extends to non-circular curves and three-dimensional figures, with clarity as the main objective throughout.

The selection of a programming language in which to provide examples of coding involved some deliberation. The authors finally decided to use FORTRAN 77. They did this in an attempt to recognise the obvious virtues of structured languages, whilst acknowledging that many geometric applications are tied to a vast amount of existing software, such as graphics packages, written in FORTRAN. They hope that there is enough structure in FORTRAN 77 to satisfy ALGOL 68 and C programmers. In any case there are no curly brackets or semi-colons to baffle the BASIC enthusiast. The authors have tried to use FORTRAN 77 straightforwardly, too. All the algebra has been written out in the code, without attempting to use arrays, subroutines, or functions to mimic the vector and matrix operations available implicitly in other languages, such as APL.

The code is distinguished from the rest of the text and algebra by a different typeface. Comments have not been included in FORTRAN standard form, but in italics to the right of lines of code, to save space. The number of comments has been decided by the fact that code is preceded by a diagram and explanation. The reader would often be well advised to lay his code out more sparsely

with more comments, directed, of course, at his particular problem. Most of the examples of code are assumed to be part of a larger program section, not subroutines in their own right. In these cases there are no SUBROUTINE, FUNCTION, RETURN, or END statements. Any arrays used are declared in a disconnected section above the executable statements.

If a condition, most often an error, occurs such that all the code should not be executed, a special italic comment line is inserted, starting with a row of dots. This indicates what condition has occurred, and should be replaced in an actual program by code to report the error in a WRITE statement or to set a condition flag. As a final deviation from the FORTRAN 77 standard, continuation lines are started with the ampersand (&) character, which is a little more readable than the FORTRAN standard system when column numbers are not shown.

Single variables from the text are translated directly into the code. Subscripts simply form the second letter of the variable name, so that x_J becomes XJ, for example. Angles are written out as ALPHA, BETA, GAMMA and THETA, and are never subscripted. When additional variables are introduced, these are assembled as far as possible from the components of the relevant algebraic expression. Thus $x_L - x_K$ becomes XLK, for instance. When two different terms would have the same name using this convention, or the FORTRAN six letter name limit is exhausted, other mnemonics are chosen. Some names, such as DENOM for the bottom line of a fraction, and ROOT for the result of a square root operation, are used consistently throughout. Also suffices, such as SQ for a squared term, or INV for a reciprocal, are generally employed, again where the name length restriction will allow. Thus lines such as

```
XJSQ = XJ*XJ
XJINV = 1.0/XJ
```

are common. Multiplication by a reciprocal is often used when many terms must be divided by a single term. This is done because the division operation is commonly the slowest on a computer, but the best balance will depend on the reader's own system.

If there is any chance at all of the denominator in a division operation being so near to zero that the result of the division exceeds the largest real number with which the computer can cope, then the value of the denominator must be checked before the division commences. In this book this is achieved by comparing the absolute value of the denominator with a notional accuracy value, which is always called ACCY. This value should be selected with regard for the likely size of numbers that will be encountered in a program. ACCY should be set so as to avoid rejecting good data, while also considering the characteristics of the computer being used. For instance, an accuracy parameter of 10^{-6} might be set in a data statement:

```
DATA ACCY /1.0E-6/
```

This is a good general purpose value for systems where real number calculations are done in a floating point format giving an average resolution of 6 or 7 decimal places, and the data are values not too far from 1 (say 0.01 to 100.0). In extreme cases a single accuracy parameter may not be

applicable throughout a program, especially if it is used for other purposes; for instance to determine the distance between points below which they can be considered to be coincident. Careful thought is needed to decide what accuracy values actually refer to, particularly when values to be compared are dimensionally different, such as distance and squared distance. These are problems of numerical analysis, and the reader is referred to Weeg and Reed for a fuller treatment.

Where other books are referred to in the text only the author's name is given. There is a list of references after Chapter Nine, where the reader may find the full titles of the books, together with a few words about each one.