W(h)ither Calculus?

David Tall

In the dying years of the twentieth century, calculus stands at the crossroads, so we are told. But is it about to stride purposefully in new directions, or quietly fade away and die? After three hundred years as the major focus of mathematics, the arrival of the computer threatens to thrust the calculus from centre stage. The forces pressurising this change are potent and complex. They come with bewildering speed, causing a heightened sense of excitement for a few participants at the forefront and a mixture of anxiety or indifference for many who can only stand and wait for the outcome. There may be experts who claim to know what this outcome will be, but the truth is that a huge paradigmatic change is in progress and only a fool, or a prophet, (or both) would claim to see far into the future. Burkhardt [2] pinpoints the dilemma:

> 'One other unusual factor makes curriculum development involving advanced technology more difficult than usual. It is the mismatch of time-scales between technical change (one year) and curriculum change (ten years).'

The factors influencing change in the calculus are mostly of recent vintage. For example, the arrival of the high resolution graphics on the Apple computer in the U.S.A. was followed by the development of a number of packages, such as ARBPLOT [1] for the study of calculus topics.

Symbolic manipulators, requiring much more memory for the programs, arrived quite separately on mainframes which often lacked graphical capabilities. The fateful year 1984 first saw a full page advertisement for the computer algebra system *MACSYMA* (in the *American Mathematical Monthly*) that

> '... can simplify, factor or expand expressions, solve equations analytically or numerically, differentiate, compute definite and indefinite integrals, expand functions in Taylor or Laurent series, compute Laplace transforms ...'

In effect, such a system can carry out all the routine manipulations of the calculus on which many students rely to accumulate most of their marks in calculus examinations.

In America, where 400,000 students study calculus at college or university every year, there is a growing realisation that the needs of many are changing. For them calculus is a peculiar mixture of the algorithms of differentiation and integration found in the British sixth-form, brewed up with more theoretical aspects of mathematical analysis taught in British universities. Writing in 1984, E.E. Moise commented:

Published in *Mathematics Teaching*, **117** 50–54 (1987).

> ‘For the overwhelming majority of students, the calculus is not a body of knowledge, but a repertoire of imitative behaviour patterns’.

A number of American professors[7] argued for a significant decrease in this traditional diet and a corresponding increase in what they term ‘discrete mathematics’ suitable for applications in computer science.

In Britain the cold wind of change is only just beginning to blow its icy blast on the sixth-form pure mathematics syllabus. But the sky is red in the morning... Schools are beginning to realise the full impact of the changes at 16+ and their resultant effects on the teaching of calculus at A-level. As this happens two opposing factors are coming into play that are peculiar to the British system. On the one hand, the broader span of the new G.C.S.E. contains less of the material that is currently considered as essential prerequisites for the calculus (for example there will be less stress on algebraic manipulation). This will make it difficult to reach the same level of performance at A-level, producing a pressure for reduction in content, accompanied by some desire to increase investigational elements through the introduction of coursework. On the other hand, the agreed ‘common core’ shared by A-level Examination Boards will, in practice, militate against major changes.

But change will come, because the pressure of the computer in every walk of life will be so great that it will not be denied. Many of the benefits of the computer are very positive, if it is used in an appropriate way. In this context, ‘appropriate’ is a simple term to define: we must retain control over the things that we humans do well and pass to the computer those things that it does better.

Suitably programmed, the computer is very good at totally reliable numerical and symbolic calculations, together with providing almost instant moving graphical displays. It is certainly much better at these things than we mere mortals. The obvious move to make is to develop a curriculum where these computer facilities are used to their fullest advantage, without losing sight of the human need to have a curriculum which gives insight into the processes being used.

In each of the three major areas: *numerical*, *graphical* and *symbolic*, there is a broad spectrum of activity possible, from the use of prepared software to individual programming by the students. As time passes, this spectrum will become less clearly defined. Software will become more open-ended, allowing the user to develop individual modifications and additions, meanwhile languages will become more subtle and provide new facilities that are closer to the powerful software being developed today. At the same time the distinction between the numerical, the graphical and the

symbolic will also become more diffuse as multi-representational software is developed to link the three areas together.

Numerical Methods

Numerical methods have been with us longer than dynamic graphics or symbolic manipulation, but their full impact has yet to be realized in the A-level curriculum. For many years, they have been an option, usually at the end of the course, to solve problems numerically that are either difficult, or impossible, to solve symbolically. For instance, there may be no formula for the solution of a complicated polynomial equation, such as

$$x^9 - px^5 - 2 = 0, \qquad (1)$$

but it is certainly amenable to a numerical solution.

The Newton-Raphson method for the solution of an equation $f(x)=0$ starts with a guess for the root, say x_1, then repeatedly replaces x_n by

$$x_{n+1} = x_n - f(x_n)/f'(x_n).$$

If the graph is reasonably straight near a root and x_1 is close to it, the sequence x_1, x_2, ... quickly homes in on it.

The Newton-Raphson method in this form requires the knowledge of the calculus (to calculate $f'(x)$) before it can be used. But its purpose is purely to estimate a numerical approximation to a root. An obvious approach is to perform the calculation using a numerical approximation to the gradient instead of the derivative. The numerical formula:

$$g(x) = \frac{f(x+h)-f(x)}{h}$$

will give a close approximation to the gradient for an appropriate small value of h (say h=0.0001) and then the iterative formula:

$$x_{n+1} = x_n - f(x_n)/g(x_n)$$

can be used to home in on a solution.

Before the computer, this calculation might be quite forbidding to carry out. Now a language like BBC BASIC will do all the hard work using formulae such as:

```
1000 DEF FNf(x)=x^9-PI*x^5-SQR(2)

2000 DEF FNg(x)=(FNf(x+h)-FNf(x))/h

3000 DEF FNnext(x)=x-FNf(x)/FNg(x)
```

With *h* defined as a small number, for any value of *x* the command

```
PRINT FNnext(x)
```

will print the next approximation in the sequence.

New features, already available on more advanced computers, will soon make this power even more user-friendly. Figure 1 shows a screen display of the above program as I wrote it on a Macintosh computer using "True BASIC" (a new version of the language from the original inventors of BASIC, Kemeny and Kurz[5]). There are 'windows' on the screen that can be moved around at will. In one the program has been typed, beginning with the function definitions just mentioned, prompting for an input of the starting value of *x*, defining a small value of h, then entering a 'DO LOOP' to let *s*=next(*x*), printing the value of s, checking to see if *s*=*x*, and if it does, exiting the loop, otherwise letting *x*=*s* and going round the loop again. When *x* does equal *s*, control passes to the statement after the loop, printing the solution, and ending the program.

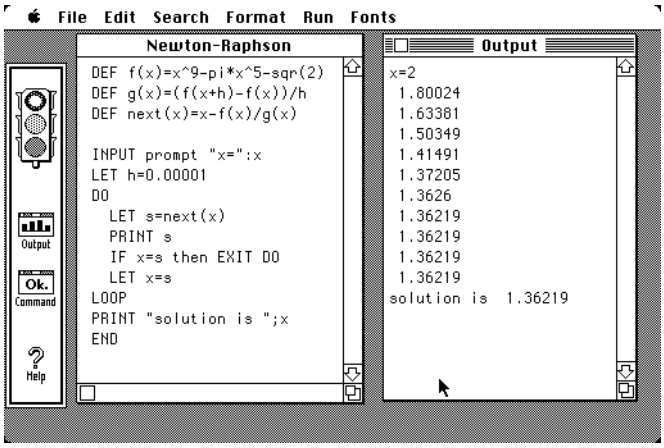

Figure 1 : The Newton-Raphson algorithm in TRUE BASIC

As well as a usual keyboard, the Macintosh computer has a 'mouse' (a box with a ball inside to be moved around on the desk) controlling the position of an onscreen pointer. To run the program, the pointer was moved to the "go light" on the traffic light symbol at the side and the mouse control button clicked. The 'Output' window shows the output of the program, requesting the value of *x* before printing the iterations in the loop until the value of *x* is constant.

This power with numerical calculations will surely have a profound effect on the curriculum. Numerical solutions of equations could be studied meaningfully *before* the calculus, not as an optional extra to cope with awkward cases afterwards. This may be done *now*, with pupils using personal calculators, whilst a single computer in the room could be used for more insightful class discussion and investigations by pupils on a rota system. As the price of technology falls, massive computer power will be placed in the hands of individual pupils, so it is wise for teachers to experience these facilities in preparation for future trends.

Spread Sheets

Another way of using the power of the computer is through commercially available spreadsheets. These can be most powerful if the pupil is in control, but they are less easy to understand when used by a teacher for class demonstration. The onscreen layout is very condensed and each entry position on the screen has three different meanings: each store can have a label attached to it to name it, it can have a numerical value displayed onscreen and it can have an optional formula to be used in calculating its numerical value.

To demonstrate these three different aspects I designed what I term the Algebraic Calculator[10], which has three separate columns, the left for the label, the middle for the numerical value and the right for the optional formula. Figure 2 shows the algebraic calculator being used to calculate a root of the equation solved earlier using True BASIC. The entries are typed in using the cursor keys to move round from one position to another. The first row contains the value of the function *f*, calculated using the formula x^9-px^5-sqr2, the second row contains the value of the derivative *d* using the formula 9x^8-5px^4, and the third row calculates the value of *x* using the formula *x*–*f*/*d*. Each time the SPACE BAR on the keyboard is touched, the whole display is re-calculated, so that successive presses iterate the Newton-Raphson formula.

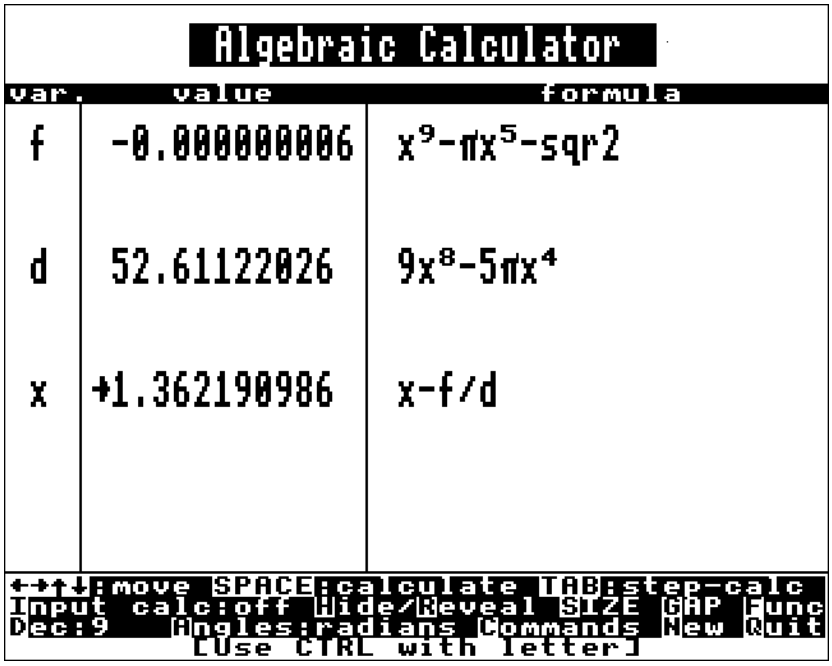

Figure 3 : The Algebraic Calculator

There are some that think that spread-sheets will produce powerful new ways of doing mathematics. My own observation is that it works well for certain types of calculation, but it may require considerable mental gymnastics to work out precisely where to put the variables in the display (e.g. the variable *x* in figure 3 is input in the bottom row, to be re-calculated after the variables *f* and *d*). Programming languages with subtle looping controls such as "repeat this calculation until the given condition is satisfied" are far more flexible than current spreadsheets, though future developments may modify this observation. Once again it is a case of 'letting the computer do those things that it does well', selecting the appropriate software for the given problem.

Dynamic Graphical Displays

Numbers by themselves are rarely as informative as graphical representation of the data. For instance, finding all solutions of equation (1) would be so much more straightforward if a graph of the function were drawn.

Early calculus software concentrated on trying to represent standard theory visually on the computer. For example, ARBPLOT[1] begins its work on differentiation with a program on 'limits' with a graphical version of 'if I specify epsilon, can you specify delta, so that, when *x* differs from *a* by less than delta, the value of f(*x*) differs from a specific number *A* (the limit) by less than epsilon?' Other software carried out the mathematics and showed the results as pictures.

My own work, described in earlier articles in this series, has concentrated on seeking a new approach to the subject that is more meaningful for the learner. After a great deal of investigative research [11], it proved possible to develop what I term a cognitive approach to the subject. First one may see, by using a program such as *Magnify* (in *Graphic Calculus I* [12]), that small portions of many graphs under high magnification look almost straight. I see this 'local straightness' as the fundamental idea in the calculus. It can be used graphically before formal calculus to determine the gradient of a curved graph. All one does is highly magnify a bit of the graph and the (approximate) gradient of the graph is the gradient of this straight line.

This gives a new approach which does not rely on the 'intuitive notion' of a tangent as a "straight line which touches a curve at one point only". We could define a practical tangent at a point $x=a$ on a curve $y=\mathrm{f}(x)$ to be a straight line through two very close points $(a,\mathrm{f}(a))$, $(a+h,\mathrm{f}(a+h))$ on the curve. This is an operative definition, in the sense that it can be used to calculate good approximations to the tangent which are quite adequate for pictures, whereas the 'intuitive definition' is of limited value in drawing.

A program (such as GRADIENT in Graphic Calculus I) which slides a practical tangent along the curve and simultaneously plots the value of the gradient as a new graph then displays the dynamic notion of the gradient graph. Again, these ideas can be introduced before the symbolic differentiation if one so desires.

Likewise, dynamically moving programs such as AREA and SIMPSON (in Graphic Calculus II) can be used to visualize how the area under a graph may be calculated dynamically as a growing area function, quite separate from the symbolic manipulation involved in integration.

The combination of numerical solutions of equations, numerical tangents and gradients, and numerical calculations of area could easily become part of a numerical pre-calculus course, parts of which could be studied before, or parallel with, the corresponding calculus notions. The numerical methods option in the Oxford A-level mathematics has been modified with such an approach in mind. When the symbolic methods of differentiation and integration are introduced, they can then be linked to a cognitive understanding which of the underlying dynamic processes.

Symbolic Manipulation

Recent developed software to carry out the symbolic manipulation of differentiation and integration is of a different order of complexity from the kind of software discussed so far. Numerical and graphical routines can be programmed in languages such as BASIC

which fit into the small memories of computers such as the BBC. Manipulators require languages that can cope recursively with strings of symbols and tend to need far greater memory space, although rudimentary manipulators can be written for small computers using languages like Lisp, Logo or Prolog.

The rules for differentiation are of two kinds:

- derivatives of specific functions such as constants, sin, cos, etc,
- general rules for composites of functions in pairs, such as the derivative of the sum f+g, or the composite fog, in terms of the derivatives of f and g.

These rules are quite formal, without any reference to limiting processes. They simply operate on strings of symbols that, for the moment, can be divested of their meaning as functions.

To write a symbolic differentiation procedure in Logo one may use what I shall term 'symbolic functions', which are

(a) either constants or strings of letters and digits such as 3, -7, sin, cos, ln, f, g, u1, u2, etc
(b) or triples in the form [u o v] where u, v are again symbolic functions and o is one of the symbols +, -, *, /, o, ^.

Thus the following are symbolic functions:

sin, cos, [sin / cos], [f / [ln + g1]], etc.

In this notation the derivative of cos may be written as

[0 - sin]

and the derivative of tan as

[1 / [cos ^ 2]].

The formal differentiation procedure D is then defined to operate on a 'symbolic function' f. If f is a word, D outputs the derivative if it is known, or the symbol f'. If f is not a word, it must be a triple so the general rules of differentiation may be used to reduce the problem recursively.

For instance, D [sin + cos] would be broken down to [D sin + D cos], which is [cos + [0 – sin]].

The manner in which this is done requires a little experience with the nitty gritty of Logo and the precise details are left as an interesting challenge for Logo programmers.

Such a procedure D, though handling differentiation quite nicely, soon gives results that begin to look quite complicated. For example:

```
SHOW D [x ^ 7]

    [7 * [x ^ 6]]

SHOW D D [f + g]

    [f " + g "]

SHOW D [sin / cos]

    [[[cos * cos] - [sin * [0 - sin]]] / [cos ^ 2]].
```

This leads on to the human need to simplify expressions, possibly performing a translation from standard mathematical notation into ‘symbolic functions’, applying the procedure D, then translating the result back into standard mathematical notation.

It is not a simple task, especially as standard’ notation is not quite as well regulated as it might be. It also opens up the thorny question as to what constitutes ‘appropriate mathematical notation’. The answer might be different for humans and computers, for humans have a limited ability to grasp long strings of symbols which computers handle with ease. Conversely, humans can interpret shorthand symbolism that they may find difficult to translate into computer terms. It may be easier to program a computer using a formula such as

```
[[[a * [x ^ 2]] + [b * x]] + c]
```

whilst preferring

ax^2+bx+c

for ordinary discourse.

Despite these notational difficulties, it is a revelation for pupils with a knowledge of Logo to explore the differentiation procedure, if only to see that it is a well-defined algorithm.

Symbol Manipulators

Symbol manipulators are now becoming more widely available. The first such software was written on mainframe computers because it require vast memory and speed. Subsequent generations, such as *muMATH* [7] works on more generally available micros including the IBM. Its successor *Derive* is more a general symbolic facility with a better user interface, whilst now there are several fully fledged symbolic manipulators which work on faster IBM compatibles and Macintosh computers with larger memory capacities, including *MAPLE* and *Mathematica*.

All manipulators can carry out formal differentiation using fairly normal notation, most being able to cope with implicit multiplication, though requiring idiosyncratic notation, such as the usual computer arrow ^ for powers. But some are less good with integration, as they essentially attempt to try out the various routines available, such as integration by substitution or by parts, and may not be well enough programmed to use subtle substitutions. In extreme cases it is possible for such manipulators to differentiate a formula f(x) to get the derivative d(x) but be unable to recognize the formula d(x) as one that can be integrated.

This state of affairs has been altered by an algorithm of Risch (1969) which can systematically decide if a function has an 'elementary integral'. (Technically an elementary function is one built up from constants and the integration variable using arithmetic operations, exponentials, logarithms and the solutions of algebraic equations.) Further details and references are described by Davenport [3].

Some symbolic manipulators now incorporate the Risch algorithm to *guarantee* finding an elementary integral function when one exists.

It is still early days in the use of these manipulators in education. Several of my friends who are applied research mathematicians cite examples where they have been able to use the symbol manipulator to carry out a great deal of routine work in a short time. One wrote privately to me explaining how he used one to calculate a large number of partial derivatives for a problem in general relativity:

> 'It took about 1/2 an hour to write out the functions and about 1 hr 30 to 1 hr 45 minutes to perform the differentiations and simplifications on the computer. I estimate that working about 3 hrs a day, including rechecking the calculations would have taken about a month.'

At an undergraduate level Lane et al. [6] quote a number of examples of student investigations using the symbolic manipulator *MAPLE* (produced at Waterloo University, Ontario) including an investigation into the function

(x^2-4)/(x^2-1)

by getting the system to find the zeros of the numerator and denominator, to calculate the derivative to find maxima and minima, and so on. In Britain we would expect A-level students to crack this problem by hand. But the implication here is that once one understands the ideas behind simple problems, it may be possible to use the symbolic manipulator to solve more complicated examples.

Experiences using symbolic manipulators with students have been mixed. Lane's experience is a positive one: he claims that symbolic manipulators can be used with profit for exploration of ideas and problem-solving. Elsewhere there is both enthusiasm and a growing awareness of possible pitfalls in using symbolic manipulators in the mathematical curriculum. For instance, a number of universities in Canada have students working with symbolic systems to solve problems where the fundamental ideas should be straightforward, but involving computation of derivatives or integrals which are more difficult. The idea is to set up an effective partnership between the student doing the mathematical thinking and the computer doing the routine calculations. Though there was a measure of success, some students found themselves with very little they could accomplish, once the routine parts of the mathematics were taken away from them.

Those who get the most out of symbolic manipulators are often experts with a good understanding of the calculus before they start. To enable students to use symbolic manipulators effectively seems to require an imaginative programme of tasks based on a prior understanding of the fundamental processes.

Multiple Representations

The current buzz-words in American mathematical education are ‘multiple-representation systems’. The idea here is to produce software that can furnish several different ways of viewing a problem simultaneously, enabling the student to see the links between them. *Graphic Calculus* is a multiple representation system, in that it includes simultaneous numerical and graphical elements, but it currently lacks symbolic differentation. Some new programs written on computers with larger memories now include simultaneous numerical, graphical and symbolic facilities. For instance, John Kemeny's *Calculus*[4] for the Macintosh or I.B.M. computer includes a differentiation algorithm and provides facilities to draw graphs and their first and subsequent derivatives, together with the display of tables of numerical values. Figure 3 shows a sequence of operations in which the formula sin(x) is entered, its graph plotted, the

derivative formula calculated (but not displayed) and the derivative graph plotted. The derivative may be revealed by selecting the formula option, but it is also possible to conjecture what it may be and superimpose the graph of the conjecture before revealing the actual derivative. Thus the program may be used for investigating derivatives, though it lacks the facilities of Graphic Calculus to demonstrate the dynamic growth of the global gradient function.

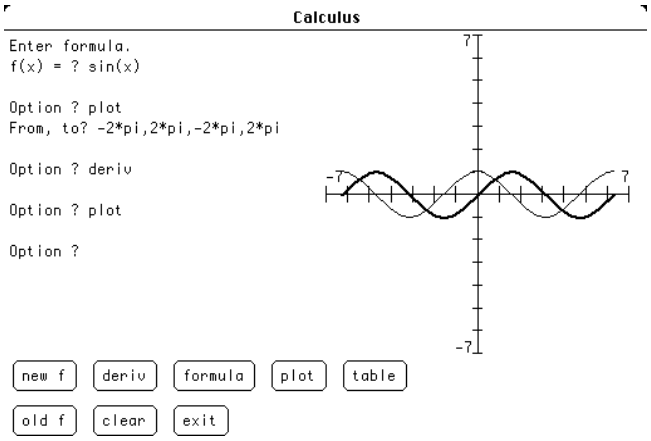

Figure 3 : The graph of the (symbolic) derivative of sin(*x*)

Deep Structure and Surface Structure

Looking at the calculus from numerical, graphical and symbolic viewpoints exposes the major difference between the deep structure of limiting processes and notions of rate of change and cumulative growth on the one hand, and the surface structure of symbolic manipulation to obtain formal derivatives and integrals on the other. In the current calculus curriculum most students mainly learn the surface structure of symbolic differentiation.

It may not be absolutely clear where the calculus will go in the next few years, particularly what new topics in computer studies may need to be introduced to replace current content. But it is hard to believe that the fundamental ideas of change and growth represented most clearly by graphical techniques will be supplanted, even if the rise of numerical methods and the streamlining of symbolic manipulators changes the balance of the curriculum.

Symbolic manipulators are, in one way, a red herring. They are certainly valuable in complicated cases where computation by hand becomes oppressive, but in the real world there are many problems in integration and differential equations which have no elementary solutions for which symbolic manipulation is useless.

Even though it may not be clear where the current change in paradigm will lead, it is a sensible course for teachers and pupils to embrace computer methods to get to know what they can do and what their limitations are. In the present A-level system the deep structure of the calculus is well served by an exploration of the graphical approach and a more imaginative use of numerical techniques.

References

1. A R Brown & M Harris : ARBPLOT (for the Apple II computer) , Harper & Row, 1982.

2. H Burkhardt: 'Computer aware curricula: ideas and realization', in A G Howson & J P Kahane (eds) *The influence of computers and informatics on mathematics and its teaching*, CUP, 1986

3. J H Davenport, 'Mathematics of computer algebra systems', in A G Howson & J P Kahane (eds) *The influence of computers and informatics on mathematics and its teaching*, CUP, 1986.

4. J E Kemeny, *Calculus* (for the Macintosh and IBM computers), True Basic Inc, (39 South Main St, Hanover NH 03755 USA), 1986.

5. J E Kemeny & T E Kurz: *True BASIC*, True BASIC Inc, Hanover NH, 1986.

6. K D Lane, A Ollongren & D Stoutemyer: 'Computer based Symbolic Mathematics for Discovery', in *The Influence of Computers and Informatics on Mathematics and its Teaching* (ed. Howson A.G. & Kahane J.-P.). Cambridge Univ. Press, pp 133-146, 1986.

7. A Ralston & G S Young: *The Future of College Mathematics*, Springer-Verlag, 1983.

8. D Stoutemeyer *et al* : *MuMath*, The Soft Warehouse, Honolulu, Hawaii 96822, 1985.

9. D Stoutemeyer *et al: Derive*, The Soft Warehouse, Honolulu, Hawaii 96822, 1988.

10. D O Tall, The Algebraic Calculator, *Secondary Mathematics with Micros - A Resource Pack*, Mathematical Association.

11. D O Tall, *Building and testing a cognitive approach to the calculus using interactive computer graphics*. Ph.D. thesis, University of Warwick, 1986.

12. D OTall, *Graphic Calculus I, II, III*, Glentop Publishers, London, 1986.