Table of contents

Interrelationships between mind and computer: processes, images, symbols

Published in *Advanced Technologies in the Teaching of Mathematics and Science* (ed. David L. Ferguson), New York: Springer-Verlag, 385–413. 1993.

Interrelationships between mind and computer: processes, images, symbols

David Tall[1]

Mathematics Education Research Centre
University of Warwick
COVENTRY CV4 7AL
U.K.

Introduction

In my work using the computer in mathematics education I have often taken the image of mankind as a tool-using species, enhancing the limitations of human capabilities by designing and using tools to make up for those deficiencies. Thus the individual succeeds in society not just because of her or his individual skills but through a combination of personal qualities operating in tandem with the technology. In education the principles should be the same – using the combination of learner, teacher and computer so that each maximises its part in improving the learning process.

The computer is a tool that provides algorithmic capabilities to carry out complex processes which, if carried out by the individual, might lead to extra cognitive load too difficult to bear, or take such an inordinately long time that the relationships involved are more difficult for the individual to encompass. Traditional learning in mathematics usually involves the student first learning certain processes, such as the algorithms of arithmetic, or the manipulation of algebraic symbols, or solution processes for differential equations, and then routinizing them so that the processes can be suppressed to a lower level of consciousness. It is then possible to move on to the next stage in which the routinized algorithms are chunked to give new mathematical objects that are more easily manipulable. The new mathematical objects are given symbols to be manipulated at a higher level of mathematization.

This is all very well in theory. However, in practice, many students do not perform the process of encapsulation of processes as objects and are not able to give meaning either to the objects or to the symbols which are supposed to represent them. They are thus faced either with the meaningless manipulation of symbols or burdened with the greater cognitive strain of coordinating the original processes.

[1]The author wishes to thank Norman Blackett, Eddie Gray, Guershon Harel, Michael Thomas and Bernard Winkelmann in the formation of ideas expressed in this article.

With a computer tool capable of carrying out some of these processes, a new form of learning becomes feasable in which the individual can concentrate on constructing mental relationships that are important for conceptualization, whilst the computer carries out the routine algorithms. It is the task of the educator to provide appropriate environments to focus on selected mathematical concepts or processes whilst suppressing routine algorithms carried out by the computer, to enable the learner to make selected mathematical constructions. I term this the *principle of selective construction*. The student may focus actively on desired constructions whilst others are suppressed to a lower level of consciousness. On other occasions it may be appropriate to focus on different processes, for example to look at the specifics of the algorithms used within the computer. This in turn allows for different possible sequences of learning, on some occasions focussing on understanding the computer algorithms (at an appropriate level) before using the software, sometimes in parallel, and at others using the software to establish conceptual linkages to set the computer algorithms into context later on.

The research and development carried out at the Warwick University Mathematics Education Research Centre has involved the observation of conceptual difficulties in students, and the design and testing of computer environments to enable selected processes to be demonstrated by teachers and explored by students. A number of research investigations have been carried out, all of which show significant improvements in conceptual understanding in such diverse areas as algebra, trigonometry, straight line graphs, and the calculus.

In this article I shall focus on the broader issues involved in this kind of learning, illustrating them with respect to specific examples from the Warwick research and development.

The Burden of Failure

It is a well-known phenomenon that there is a divergence of performance in the learning of mathematics – those who succeed seem to find it easier and easier, whilst those who fail soon find it almost impossible. Studying the difficulties of slow-learners in arithmetic Gray (to appear) has shown that more able proceed from the “counting on” algorithm for addition to develop “known facts” then begin to use “derived facts” to develop new knowledge (e.g. knowing 4+4=8 to derive 4+5=9 or 24+4=28). Thus they develop a feed-back loop in which their ability to derive facts feeds back to give new known facts. Soon they that they do not have to memorize as many new facts because they can be derived when necessary. Thus the relational processes developed by the more able make arithmetic *easier* for them, enhancing their success.

Meanwhile the slow-learners cling to the tried and tested process of “counting on”. They will calculate a sum such as 8+5 by counting on to get 13, but then, if asked “what makes 13?” are likely to have forgotten the initial inputs by the time they have completed the task. Thus the process of addition is not encapsulated to give the concept of “known facts”, and any odd facts that they do remember (often involving doubling such as 5+5=10) are not easily used for deriving new facts. The relational structure of arithmetic is missing and the weaker student is faced with a greater burden than his more able counterpart, leading to inevitable failure.

Analogous phenomena occur with the passage from arithmetic to algebra. The child who views the arithmetic symbols 3+6 as a *process* to produce an answer may find difficulty in coping with the symbol 3+6x which, when viewed as a process, cannot be carried out until the value of x is known. In addition, such a child will see the symbols 3+6x and 3(1+2x) as representing different processes and faces an obstacle in attempting to view them as equivalent expressions.

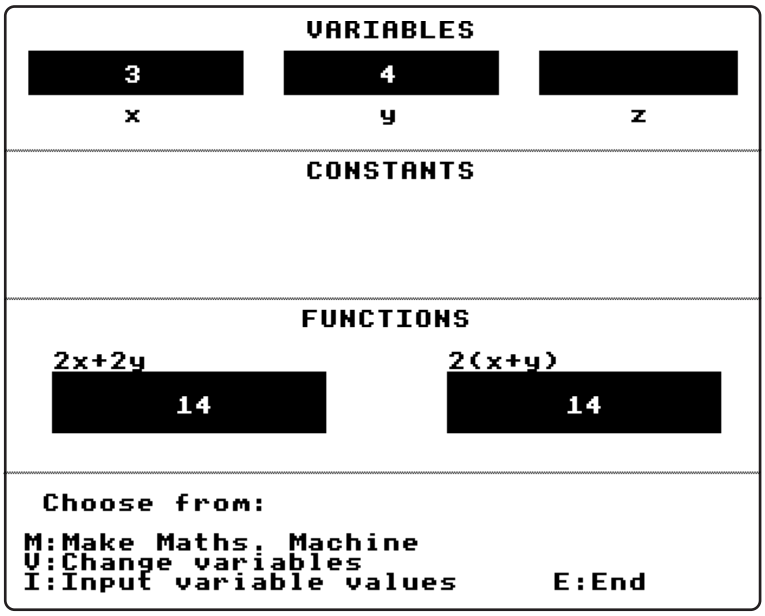
Faced with such difficulties, traditional teaching tends to emphasise the correct implementation of the algebraic processes, attempting to give the children the necessary manipulative ability. However, without meaning, this soon degenerates into learning the “trick of the week” – “do multiplication before addition”, “calculate expressions in brackets first”, “collect together like terms”, “of means multiply”, “do the same thing to both sides”, “turn upside down and multiply” etc, etc. Thus the child is reduced to the meaningless manipulation of symbols, with a large number of disconnected tricks to use, which increases the cognitive burden on a mind already sorely stretched.

The Principle of Selective Construction

Using the computer a new possibility arises. In order to focus on new ideas and to suppress the processes that may cause the cognitive burden, it may be possible to get the computer to carry out the processes which are not desired to be the focus of attention. The educator may provide the learner with an environment in which the learner can focus on selected constructions, whilst other constructions which are not to be the focus of attention are performed by the computer. This I term the *principle of selective construction*.

For instance, to see algebraic symbolism as giving a product, instead of focussing on the actual process of calculating that product, one could use programming in BASIC to print out the values of, say, 2+6*X and 3*(1+2*X) for various given values of X. Every time the BASIC instruction to print these expressions is given, the computer carries out the process and the child may simply focus on the fact that the results are always the same.

On the other hand, pupils with experience of certain kinds of calculator will find that 2+3x5 gives 25 (two add three is five and then five times five is twenty five) and may expect 2+3*X



```
VARIABLES
3            4            
x            y            z

CONSTANTS

FUNCTIONS
2x+2y                  2(x+y)
14                     14

Choose from:

M:Make Maths. Machine
V:Change variables
I:Input variable values      E:End
```

figure 1 : The Algebraic Maths Machine

to give 25 when X=5. But it does not. Thus it is good to have examples which are not equivalent: such as 2+3*X is not the same as 5*X, but (2+3)*X *is* the same as 5*X.

Thomas (1988) introduced algebraic symbolism through programming (in BASIC) and the use of software in which the computer carried out the process to produce a product. Not only did he use BASIC programming, he also designed a piece of software (the *algebraic maths machine*) which allows the use of standard algebraic symbolism (such as implicit multiplication and superscripts as powers) for algebraic expressions. The machine allows individual letters to stand as labels for input numbers and two expressions involving these letters can be evaluated and compared (figure 1).

Now the child can be selectively focussed on the sameness of the results of calculations using standard algebraic expressions, rather than the difference of the processes to carry out the calculations. In addition, cases where equivalence might be expected but where it fails (such as 2+3x, 5x) may also be investigated.

In a separate activity the children took part in a game which involved playing the part of the computer in performing the calculations, so that, at a different time, they could concentrate on (a model of) the inner workings of the computer.

This involved a *cardboard maths machine*, which consists of just two large sheets of cardboard and some smaller cards marked with letters and numbers. One piece of cardboard acted as a screen and programming instructions were placed on the screen and then carried out by members of a group of pupils. The other piece of cardboard – placed a short distance away –

had a number of rectangles marked on it which represented computer stores. Each could be labelled by a letter placed above the store and each labelled store could have a number placed. Thus to carry out the instructions

```
A=1
B=A+3
PRINT B+2
```

involves marking a store with the label A and placing the number 1 inside, then labelling another store B, looking inside the store A to find the value 1, adding 3 to get 4 and placing it in the store B, and finally looking in the store B, noting the number 4 inside, adding 2 and placing the number 6 back on the sheet of cardboard representing the screen (figure 2).

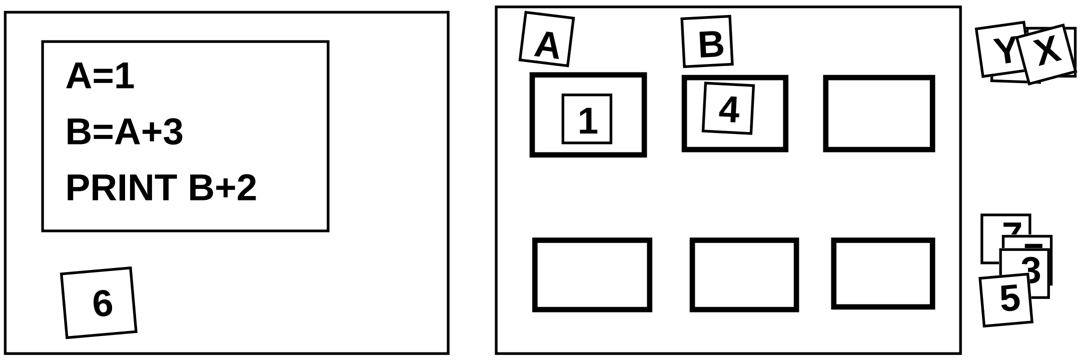


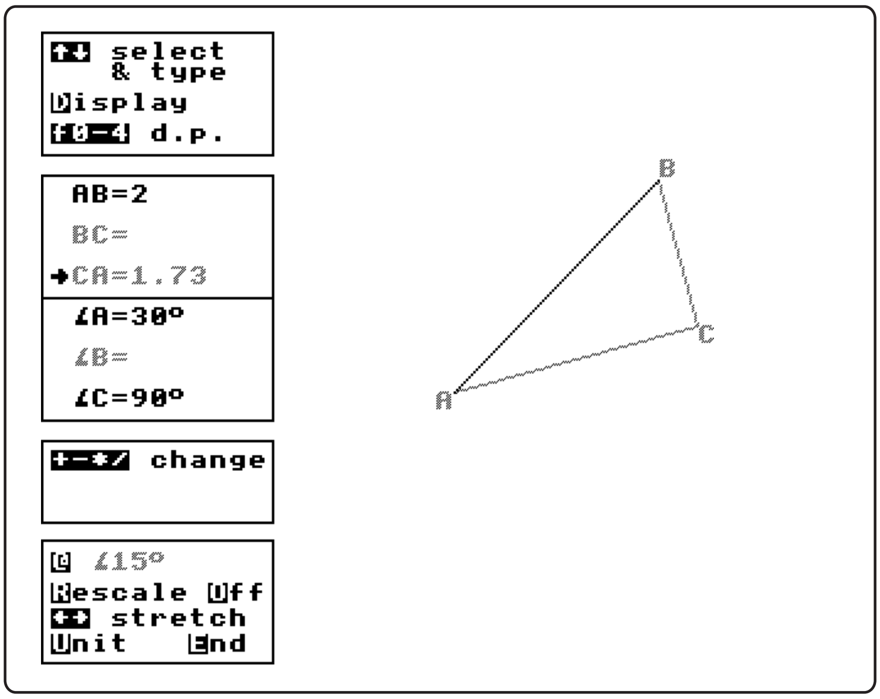
figure 2: The cardboard maths machine

Working with the cardboard maths machine clearly represents the *process* aspect of evaluating algebraic expressions, just as the algebraic maths machine represents the *product* of computing their numerical values.

Thus, at different stages, different aspects of the process were selectively given an appropriate focus in order to produce a more versatile form of learning. The results of empirical studies shows significant improvement overcoming cognitive obstacles and in acquiring a more versatile insight into the meaning of the algebraic symbolism (Thomas & Tall 1986,1988).

Blackett (1990) studied similar difficulties in the introduction of trigonometric functions. Once again the pupils are faced with cognitive difficulties – handling the ratio concept (itself the product of a process of comparison) and its application in trigonometry through similar triangles and proportion (an equivalence of ratios – requiring a further level of encapsulation of process). Added to these are the problems of versatile thinking which requires the linking of geometrical concepts to numerical concepts.

As in the case of algebra, a traditional approach is often to give the students practice in the routine procedures: calculating the trigonometric ratios using a specified rule (in Britain the



```
AB=2
BC=
CA=1.73
∠A=30°
∠B=
∠C=90°
∠15°
```

figure 3: the screen display of the trigonometric software

mnemonic SOHCATOA for Sine is Opposite over Hypotenuse, Cosine is Adjacent over Hypotenuse, Tangent is Opposite over Adjacent). Again, although many students could learn to cope initially with the rote algorithms, they often had little flexibility when it came to responding to questions a little different from those that they had practised.

Blackett was convinced that part of the problem involved in the use of inaccurate sketches to represent the initial concepts, thus playing down the value of visualization. By using the computer to carry out the algorithms to calculate and draw a right angled triangle from any combination of sides and angles sufficient to define the triangle, he was able to design software that responded to the user's input by drawing an accurate picture (figure 3).

The software is relatively unsophisticated and was programmed in less than two days. It allows the user to specify such information as is necessary to define the right angled triangle (such as another angle and a side, or two sides, in addition to the right angle) and then display any other desired information about the remaining lengths or angles. Various other facilities are available, including the possibility to add, subtract, multiply or divide any length or angle by a given quantity, thus allowing exploration of what happens when sides or angles are changed. To avoid the common misconception that the shorter sides are usually horizontal and vertical, the triangle may be turned through any angle. To assist visualization of similarity, it is possible to turn on or off an auto-scale facility to rescale the triangle to fit in the picture when lengths are changed.

Using the software, Blackett again found significant improvements in more versatile understanding of the concepts. A delayed post test showed the experimental students increasing in their ability to respond to both standard problems and those of a more versatile nature. Details will be given later in the article.

Generic Organizers

In Tall (1986a) I described a *generic organizer* as

> a microworld which enables the learner to manipulate examples (and preferably also non-examples) of a specific mathematical concept or a related system of concepts. The term “generic” means that the learner’s attention is directed at certain aspects of the examples which embody the more abstract concept.

When I first made this definition I noted that there were many examples of concrete manipulatives, such as Dienes’ blocks, Cuisenaire rods, and so on, which fulfilled the stated conditions. However, these are *passive* in the sense that the user must act on them and interpret the results of those actions. Generic organizers as programmed on a computer are *cybernetic* in that they carry out algorithms in response to user action. It is this cybernetic property that gives generic organizers on a computer their special qualities, allowing the user to experiment, predict and test constructed conjectures.

The algebraic maths machine and trigonometric software are both examples of (cybernetic) generic organizers, the first allowing the student to explore examples and non-examples of equivalence of algebraic formulae, the second allowing the user to draw accurate triangles from specific information and to explore the relationships between numerical and graphical information. The cardboard mathematics machine is a passive generic organizer.

The origins of the theory of the use of generic organizers in learning originated in the development of *Graphic Calculus* (Tall 1986, Tall *et al* 1990) as an approach to the foundations of calculus which built on the implicit idea that a differentiable function is “locally straight”, rather than the explicit limit concept, which is known to cause students great cognitive difficulties. Thus the Graphic Calculus consists of a number of programs that act as generic organizers, focussing the learner on various fundamental ideas of the calculus. First, a *Magnify* program allows the learner to explore the possibility of magnifying graphs of their choice to develop the idea that many graphs are “locally straight”, which means that, highly magnified, small portions look straight. This means that the concept of gradient of such a curved graph now has a meaning: magnify it and look . A second piece of software allows the user to explore the gradient of a line through two points on the graph to see what happens as the points get close together. It also allows the user to build up a picture of the changing gradient through two points a fixed distance c apart. As c gets small, the gradient graph stabilizes and the learner can conjecture the possible formula for the gradient, linking this to the symbolic process of differentiation and gaining versatile insight into the concept of derivative as the gradient of the graph.

Further generic organizers deal with anti-differentiation (knowing the gradient, to find the graph), area calculations, both as a numerical and visual process and also as a growing area function, and differential equations: first order, second order and simultaneous first order. Simultaneous equations involving, say, dx/dt and dy/dt are viewed in three dimensional (t,x,y) space as specifying a tangent direction (dt,dx,dy) and hence seen as a generalization of the single first order case. Second order differential equations involving d^2x/dt^2 and dx/dt are translated into simultaneous first order equations in the usual way by putting $y=dx/dt$, thus obtaining equations in dy/dt $(=d^2x/dt^2)$ and dx/dt. Thus higher order (simultaneous) differential equations can be seen as growing out of the universal example of the first order differential equation which poses the problem of reversing the notion of local straightness.

A more recently developed generic organizer for the calculus is the *Solution Sketcher,* designed for the School Mathematics Project (Tall 1989b) to enable students to *enact* a physical process of solving a differential equation. This program draws a short line segment whose gradient is given by the differential equation (figure 4). It may be moved around using the cursor keys or a mouse. By touching the space bar, or clicking the mouse, a copy of the line segment is deposited on the picture, and by sticking such segments end to end, an approximate solution curve can be drawn which, by construction, has a gradient which satisfies the differential equation (at least at the centre of each short line segment).

Hidden within the program is the algorithm to compute the gradient of the line segment and to draw it on the screen. However, by using the program, it is possible to suppress the internal construction and concentrate on the link between a point in the plane, the gradient of a solution through the point, and the way in which the solution curve is built up to satisfy the equation. Thus, by a process of selective construction using the computer, if desired, the student can focus on the essential nature of the solution of a differential equation *before* carrying out the solution process.

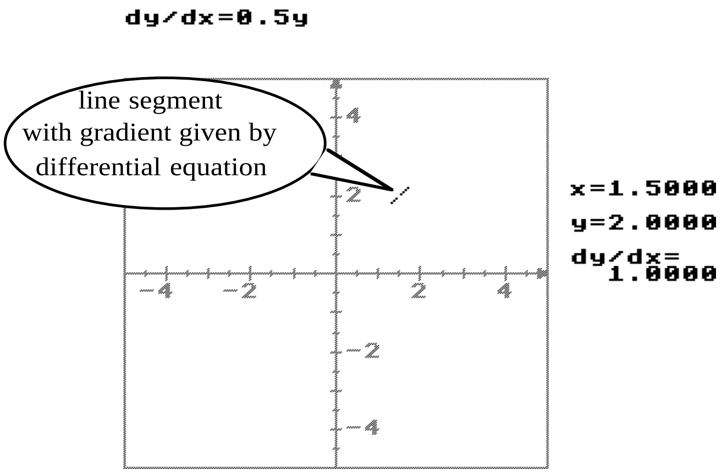


figure 4: The Solution Sketcher

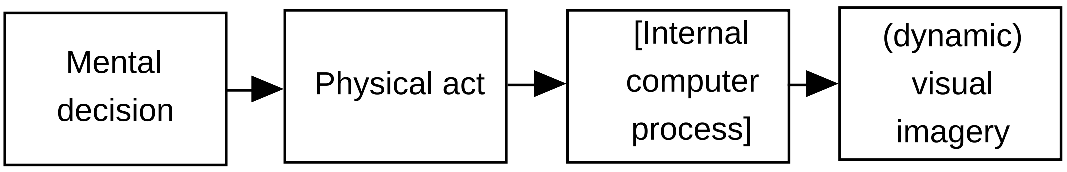
There are options in the program which allow an array of line segments to be drawn to show the global tendencies of different solutions, and automatic solution drawing through any selected point. The step-length of the line segment may be changed to gain an insight into how good the numerical approximation is. By choosing larger step-lengths, the errors in calculation will be seen, but by choosing smaller step-lengths the approximations will be seen to stabilize to a unique theoretical solution.

Using these facilities the student is able to gain a rich mental image of the notion of the solution of a differential equation: that a first order differential equation has *many* solutions, but that through any starting point there is a *unique* solution which everywhere has the gradient specified by the differential equation. This mental image can now be used to give advanced organisation for investigating the numerical process underlying the solution or seeking a symbolic solution to the equation. It can inform the student that, when attempting to reverse the symbolic process of differentiation to find a symbolic function which satisfies the equation, then solutions of a certain kind – and no others – should be expected. Thus, by using the principle of selective construction to change the order of attack on the concept of solving a differential equation a greater network of mental imagery can be built to enhance meaning.

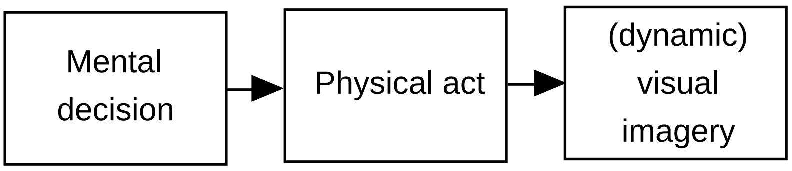
It may be seen that this principle of selective instruction is implicit in the use of software in virtually all of the other papers in this collection, be it the *Newton* software to explore motion, *Stella* to focus on model-building, software to explore *feedback systems*, or the *intellectual mirror* software of Judah Schwartz.

The human-computer interface

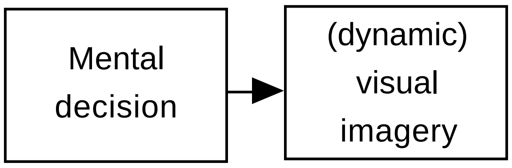
The interface between the human operator and the computer, whilst of lesser importance than the overall theory of selective construction using generic organizers, is of great importance in its implementation. The software used operates in a mode which requires the user to take a mental decision, to transform this into a physical act to make an input to the computer, which is then algorithmically processed and output as a visual image on the screen.



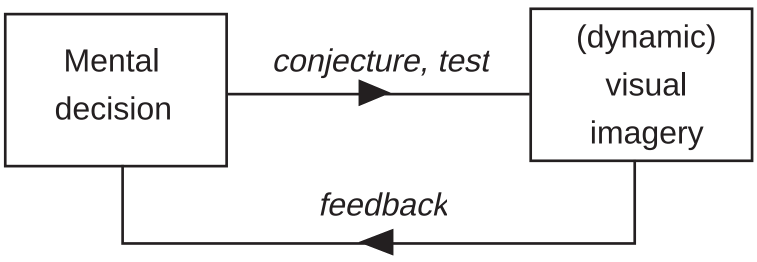
The most important parts of this schema are the mental processes and the interpretation of the visual imagery. The internal computer process is, at this point, being mentally suppressed to concentrate on the relationship between mental decision and visual image:



The interposition of a physical act to carry out the mental decision is one which should, as far as possible, support the educational objectives and, at the very least, not be positively harmful. If this physical act becomes routinized, so that it involves little conscious effort, then the system reduces to a mental decision followed by the production of visual imagery.



This can then become a part of a feed-back system:



In order that the student may concentrate on the important features of the computer representation of the mathematics, the interface must be as unobtrusive as possible. In one sense, familiarity with the interface will help routinize its use and suppress its conscious intrusion – be it a QWERTY keyboard with its initially difficult learning curve or the more intuitive use of a mouse to point and select. Some familiar with a keyboard often prefer the use of key-strokes to the use of a mouse, because these have become automatic and require no conscious thought. However, here they will be aided by general conventions to make the links between mental act and physical act automatic, for instance by ensuring that the same key-strokes work uniformly across many pieces of software.

If a keyboard is used, then it helps if the keys touched in some way represent the required decision. In most early forms of software this involved touching a key representing the initial letter of the required command (or a prominent letter later in the word). But this has some draw-backs in the choice of names of menu items to have different letters.

In other cases it might be possible to select keys which have some other relationship with the mental act. For example, the *Function Analyser,* written for the School Mathematics 16-19 Project, is designed, among other things, to study the translation of graphs of functions. The interface problem (using a computer that may only have standard keyboard input) was to design a way of communicating with the computer that allowed the user to instruct how the graph should be shifted or stretched.

The solution was to have a "transform" option that allowed the user to specify a string of one or more moves (using the cursor key to specify the direction), so that ↑2 means "move up 2" (as in figure 5), →3 would mean "move right 3", and so on. A slightly more abstruse combination of cursor and $\boxed{\text{SHIFT}}$ key produces stretches, so that →$\boxed{\text{SHIFT}}$ produces the icon ↔, which means multiply the *x*-coordinate by the given factor, so that ↔2 means stretch by a factor 2, and ↔ -1 means stretch by a factor -1 (which is a reflection). The inverse operation, to divide the *x*-coordinate by the given factor is obtained by ←$\boxed{\text{SHIFT}}$. The combinations ↑$\boxed{\text{SHIFT}}$ and ↓$\boxed{\text{SHIFT}}$ correspondingly multiply and divide the *y*-coordinate by the given factor. It is even possible to chain these commands in a string such as ↑2→3↔ -1, which means shift the graph up 2, then right 3, then reflect the resulting graph in the *y*-axis. The conventions using the cursors, whilst not universally applicable in other software, are reinforced in the Function Analyser in another context by being used to shift a zoom window around the screen: cursors left, right, up, down to move the zoom window and the corresponding $\boxed{\text{SHIFT}}$ -cursor combination to stretch or squeeze the cursor window horizontally or vertically.

The graph-transforming routines enable the user to investigate effects of shifts and stretches on a graph *y*=*f*(*x*), for instance to see that *y*=*f*(*x*) shifted one unit right (with iconic representation *y*=*f*(*x*) →1) is not the same as *y*=*f*(*x*+1), but is *y*=*f*(*x*-1)... Such exploration, in which the

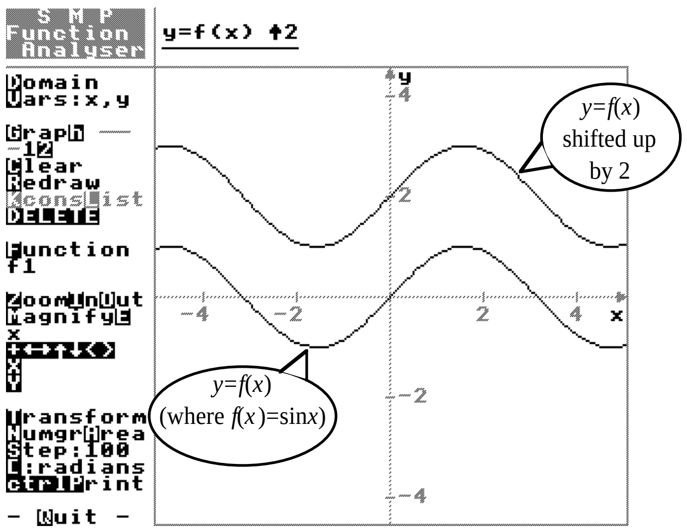


figure 5: The Function Analyser used to translate the graph of a function

computer carries out the instructions by using graph-drawing algorithms, enables the student to gain some sense of what might or might not be true, but focus must be placed on the algebraic theory and the reasons why this is so. However, it should be emphasized that the weakness of the software as a generic organizer is that it gives no assistance as to *why* $y=f(x)\rightarrow 1$ is the same as $y=f(x-1)$. (Perhaps a facility to move the axes whilst fixing the graph might help?...)

Such an iconic method of interface is minimally more helpful than standard key-presses. However, a more intuitive interface can sometimes provide an enaction of facets of the desired process in a way which is more supportive to the learning. For instance, one could envisage a mouse interface that enabled the user to drag the graph around and to update a representation of the shifts being caused, or even to specify an axis, then use the mouse to select and stretch the graph away or towards the axis. Alternatively one might wish to fix the graph and drag the axes.

The *Solution Sketcher* (figure 4, above) has an enactive computer interface designed to aid the learning. The student may sense the illusion of moving a line segment around the screen, constrained by the differential equation. By placing such line segments end to end, the further illusion is created that the user is building the solution by her or his own action.

With a graphic user interface involving a mouse or tracker ball to give the illusion of pointing and dragging on the screen there are many more enactive possibilities. A new graph plotter being developed in association the Warwick Mathematics Research Centre allows a straight line to be specified in many flexible ways and dragged around the screen. For example, one point may be specified by clicking the mouse and dragged out to give a line whose equation is displayed and updated as the mouse is moved. Or the gradient may be specified and then as the mouse is clicked and dragged around, the line is drawn through the point with given gradient and its equation updated.

Bivariate data may be plotted, then a possible line of best fit may be drawn and dragged around to see how various measures of fit change as the line is changed. Data points may be added or deleted to see the effects of outlying points on the goodness of fit. Thus the user has the illusion of interfacing directly with the data and seeing the effects of enacting various changes.

Such an enactive interface revealing links between objects being dragged onscreen and their continuously updated position coordinates already exists in graphic packages such as MacDraw on the Macintosh.

The need to focus on the internal processes

At some stage in the process, it is often important for the individual to change focus and to think about the internal processes within the computer. There are very many different ways in which this may be done, including:

(i) enactive

(ii) procedural

(iii) symbolic.

Often it is possible to *enact* a version of the algorithm, possibly using equipment which acts as a passive generic organizer, as in the case of the cardboard maths machine mentioned earlier (figure 2, above).

In the trigonometric work, the enaction involved constructing right-angled triangles using ruler and protractor. The advantage of the trigonometric software lies in the ability to manipulate it – changing side-lengths or angles, or changing the scale or orientation – and getting an immediate and accurate visual feedback.

In the SMP 16-19 syllabus a piece of equipment called the *Gradient Measurer* was designed to introduce the calculus (figure 6). This consists of a plastic transparent circle marked with a diameter and fixed within another piece of transparent plastic on which is marked a vertical ruler in units, one unit horizontally away from the centre. A student can place the gradient measurer over a point on the graph, rotate the disc until the marked diameter is visually in the direction of the graph at that point, then read off the gradient. Thus she or he can move the

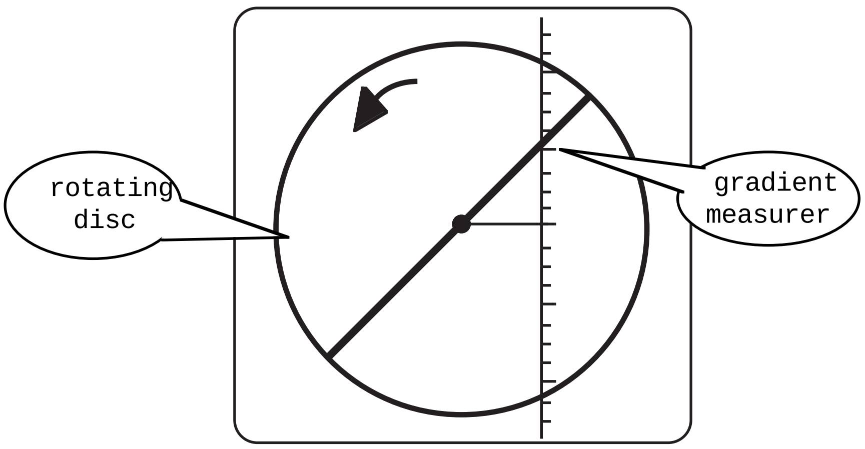


figure 6 : a physical tool for measuring gradients

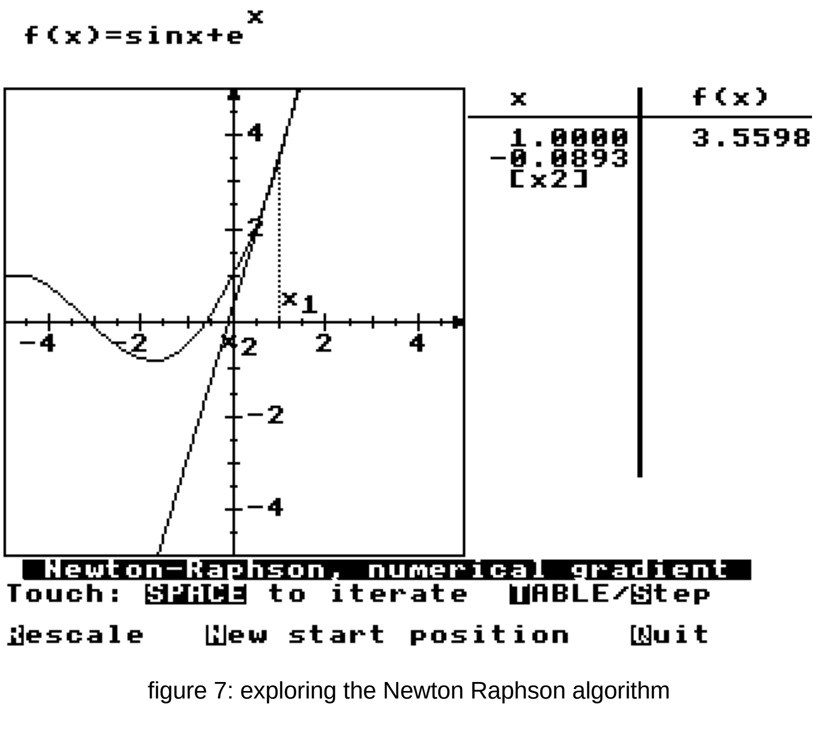
measurer along the graph and enact the changing gradient, as well as obtaining approximate numerical results.

The gradient measurer is a passive generic organizer which the student may use to focus on the concept of the gradient of a graph. This must be carried out through the individual's own actions – turning the dial to move the diameter to approximate a tangent, reading off the gradient, recording its approximate value, then plotting a sketch of the gradient of the graph. Once the principles are understood, the student may then use cybernetic organizers to give greater power in focussing on the gradient concept. First a graph plotter is employed to magnify small portions of the graph to establish the idea of local straightness of differentiable functions. Then a gradient plotter (e.g. Tall 1986b) is used to give a reasonably accurate graph of the gradient function of a given function, for the student to investigate its properties, including the initial guessing of a possible formula (for instance the gradient of sin*x* (in radian measure) is cos*x*, but the gradient of cos*x* is *minus* sin*x*). This is linked to the calculation of the symbolic gradient in simply cases by calculating $\frac{f(x+h)-f(x)}{h}$ in simple cases such as $f(x)=x^2$ to relate the various aspects of the gradient concept.

Thus it can be seen that the framework in which the generic organizer is used may be fairly flexible, allowing students to build up the concepts gently in ways which relate to their growing experience, rather than being given a "simplified" version of a "formal" approach starting from the limit concept. The limit concept is *implicit* within the generic organizers (using the selective construction principle) and need not be a conscious focus of attention at the outset. The magnification program implicitly involves the beginnings of a limit process by focussing on a small portion of the curve which, for a differentiable function, looks locally straight. This means that, provided *h* is small, the value of $\frac{f(x+h)-f(x)}{h}$ stabilizes. No one in their right mind would put *h*=0 to calculate the gradient in such a circumstance, so one of the serious obstacles facing the naïve student in a traditional mathematics course is less likely to arise.

In the SMP 16-19 curriculum the algorithm used in the gradient drawing program is studied in an enactive way *before* the software is being used. Elsewhere in the syllabus, generic organizers are also used to study the broad implications of an algorithm before the algorithm is studied in detail. For instance a *Newton Raphson Program* (Tall 1989b) is used to obtain visual insight into the idea of using a tangential approximation to solve an equation *before* the Newton Raphson iteration is considered symbolically (figure 7). The tangential approximation is determined numerically (which is a familiar concept to the students concerned). The program has an auto-zoom feature which allows the user to zoom in to see how quickly the graph approximates to a straight line, thus giving visual support to the power of the method. Investigations are encouraged using different starting points to gain insight into how it is that if the starting point is not near a fairly straight part of the graph then the iteration process may end up at a different root. Only when the students have some idea of the strengths and weaknesses of the process do they study the symbolism of the iteration, and carry it out in a programming environment or a spreadsheet.

In the SMP 16-19 syllabus the same holds true for the use of the Solution Sketcher. Because the students already have a visual concept of local straightness, solving a differential equation may be seen as reversing the visual gradient process: given the gradient, find the graph. It is therefore natural for the students to enact this process using the Solution Sketcher before studying the details of numerical or symbolic solution processes.



```
x        | f(x)
1.0000   | 3.5598
-0.0893  |
[x2]     |
```

```
Newton-Raphson, numerical gradient
Touch: SPACE to iterate   TABLE/Step
Rescale   New start position   Quit
```

figure 7: exploring the Newton Raphson algorithm

In practice it is not considered important whether the investigations preceded or followed the introduction of the algorithm. What matters is that, with the software either method is possible.

Procedural insight into the computer algorithms

When I first wrote *Graphic Calculus I* (Tall 1986b), I naïvely thought that the software could be used as a precursor of symbolic calculus, without the need to understand precisely what was going on within the software. I was encouraged by empirical research which showed impressive improvements in versatile thinking about the visual aspects. However, there are *features*[2] in any graph-plotting program and vagaries of floating point arithmetic, which mean that what happens onscreen is not what is expected, and it became evident that some knowledge of the internal algorithm is essential.

There are certainly times when one just uses a computer tool without asking what is going on inside. Using a hi-fi system, or an electric washing machine or a microwave hardly require an understanding of the algorithms that control the management software. If such domestic appliances go wrong, they are often repaired simply by replacing the complete electronic unit. Thus, provided one has the external knowledge to carry out what one wishes to do and a method of gaining help when something goes wrong, there are many times in society when one does not need to know of the internal workings of the machine. Even with a computer most people only understand certain layers of sophistication (external use of software packages, but not programming, or programming in a high level language, but not in machine language, or in machine language, but not at the electronic level, and so on.)

When one is using the computer to carry out an implicit algorithm as part of an educational process, there are at least two occasions in which some kind of knowledge of the internal workings is essential. One is when part of the educational process is to understand the internal algorithm, and the other is when the internal algorithm has features which might mislead the user.

For instance, if the software is designed to illustrate a known numerical algorithm – such as the Newton Raphson method or the method of solution of an equation $x=f(x)$ by iteration – then it is evident that the focus of the work is to understand and use the algorithm, with the generic organizer being used to test its behaviour in a variety of situations. On the other hand, software relying on numerical methods invariably contains features that might go seriously wrong in exceptional circumstances, and here a little insight is absolutely necessary.

[2] A "feature" is a polite name for a known bug!

In the initial stages of the calculus my aim had been to give the student some visual insight into the processes of calculating gradients, areas and solutions of differential equations, as a precursor to studying the symbolic algorithms of differentiation and integration. The numerical algorithm for calculating a gradient, by calculating the function given by $\frac{f(x+h)-f(x)}{h}$ for fixed h and variable x has some features in common with calculating the symbolic limit of $\frac{f(x+h)-f(x)}{h}$ as $h\rightarrow 0$. Both involve calculating a value of an expression, the first numerically, the second algebraically, but there the similarity ends. The symbolic limit considers a few standard cases, such as $f(x)=x^n$, and, in each case, simplifies the expression to obtain a very specific formula for the derivative (or gradient function). The algorithms of formal differentiation for combining such functions as sums, products, composites etc, though distantly related to the numerical gradient, bear little family resemblance. Hence, insight into the algorithm for the numerical derivative may give little insight into the procedures of formal differentiation.

However, the numerical algorithms allow a fairly good picture to be drawn in many cases and give the student a visual insight into *why* the formulae might be true (the derivative of cos$x$ is the graph of sin$x$ upside down, so it must be *minus* sin$x$). And in these days in Britain where there seems to be less fluency in the population in the use of algebra, such informal feedback is proving most helpful. But the features of the software, for instance the catastrophic inaccuracies of computer arithmetic in calculating ratios of small numbers, make some kind of understanding of the numerical algorithms essential. This is why, in both *Graphic Calculus II* (Area) and *III* (Differential Equations), numerical programming algorithms were specified. Subsequently these are becoming part of new syllabuses to be studied as important processes in their own right.

In Tall & Winkelmann (1988) we described three different kinds of insight:

External, analogue, specific

External insight occurs when the user has no idea what is going on inside the computer, but has knowledge which allows him or her to check that the results are sensible. For instance, the software may be a symbolic manipulator which computes an integral by an unknown internal method but the student may use knowledge of differentiation to check that it is correct.

Analogue insight occurs when the user has an idea of type of algorithm in use, for instance, knowing that a root of an equation is being computed by the Newton Raphson rule, but is unaware of precisely how this implemented.

Specific insight is when the user is fully aware of how the software is programmed (though this, in practice, remains only partial for, even if the user knows how a high-level language works, the implementation within the hardware is likely to include features that are not understood).

Specific insight into computer software is rarely possible or even desirable for the majority of computer users, but it is helpful for the student to have at least external insight or, preferably, analogue insight.

Analogue insight can be implemented by investigating the algorithm by programming either in a high level language or a pseudo-code that mimics such a language. In the UK the Mathematical Association has for several years had a committee deliberating the impact of computers on the mathematical curriculum. Initially the committee published a book of short (BASIC) programs to encourage teachers to get students to explore simple algorithms (MA 1985). Initial drafts of their report contained many programs in BASIC and Logo, but as the redrafts proceed and available software becomes more sophisticated, the notion of programming is being broadened to include the programming of spreadsheets and other high-level software.

The Symbolic Stage

So far we have mentioned the *processes* of mathematics that the student may carry out and eventually routinize, which may sometimes be given to the computer to carry out more efficiently as the student concentrates on other aspects. We have also mentioned the *images* produced by the computer that are intended to help the student form appropriate mental images of their own. But we have yet to concentrate on the *symbols* which were mentioned in the title of the article.Symbols are of many kinds, including enactive (body language), visual, verbal, literal, and so on. But here we are concerned more with the mathematical literal symbols which the student must write down and which may be manipulated at a higher level of mathematization for, in the end, it is the use of symbols which makes mathematics easier for mathematicians.

After many years of working with generic organizers, it is clear that there are some which relate more directly to symbolization than others. For instance, the *algebraic maths machine* is directed at building up meaning for algebraic expressions. There is empirical evidence that it leads to a higher level of conceptual insight into the manipulation of expressions, inequalities and the solution of linear equations (Thomas 1988).

However, many of the other organizers mentioned in this article, although *using* algebraic *notation*, do not themselves *manipulate* the symbols. It is therefore asking too much to expect them to lead, *without further activity*, to a higher level of symbolic manipulative ability. Tall (1986a) shows that the use of generic organizers in the calculus led to a higher level of ability to handle graphical concepts but that there was no significant change in the ability to cope with formal differentiation. The positive view is that the graphical treatment did not lower the manipulative ability. Indeed, the evidence from Thomas's work is that if the conceptual foundation is laid, then less work is needed to reach a given level of routine manipulation.

This is consistent with Heid's research in calculus where it was shown that students who use the computer for conceptual work and only carry out routine manipulation for a short time at the end of the course were better at higher level conceptual problems than control students following a traditional course, and not significantly different in ability with routine manipulation (Heid 1988).

In Harel & Tall (to appear), we came to the conclusion that the formal level of mathematics, involving formal definitions and deductions requires a new level of construction of the abstract concept from the definition. This involves a very difficult transition in which formal objects must be constructed whose attributes must follow solely from the definition. Thus a concept such as a limit, or a continuous function, or the derivative as a formal limit, all involve a difficult reconstruction which we believe can be built on the experience of generic organizers, but requires a new phase of constructive activity.

What are the concept images available in the mind of the student after using a generic organizer? They are the product of the experience with the organizer: an awareness of the behaviour of examples and non-examples of the concept(s) on which the focus is placed. Thus a student who has used a magnifier program to see that certain graphs are locally straight and others may have corners, or be so wrinkled that they are nowhere locally straight, has a rich collection of examples and non-examples of a differentiable function. But a new level of constructivity is necessary – first to isolate the essential characteristics that form the basis of a definition, and then to reconstruct other characteristics of the formal object that can be deduced from this definition. Thus a generic organiser moves on the *generic* (general example) level rather than the *abstract* (formal definition) level. I believe this to be a feature (known bug?) of many current computer based activities. It would be interesting for others who work with the computer in education to analyse the cognitive outcome of their work to see if it promotes generic or abstract thought.

This observation is not to be seen as a weakness of the approach, only a specification of the limits of this stage in development. Certainly the rich collection of examples and non-examples is likely to form a better cognitive foundation for the formal theory than conceptual imagery based only on routine processes. It also suggests new possibilities in constructive growth using the computer, from *physical action* to *mathematical process* via *generic concept* to *formal concept*, each building on the previous phase, but each requiring an explicit cognitive reconstruction.

Empirical Testing

The ideas in this article have been subjected to various levels of empirical testing. The developments for the SMP 16-19 have been implementations of a new curriculum in the time-honoured British way: try it out, if it works, leave it, and if it doesn’t, fix it. Here the desire is to implement a new mathematics course for 16-19 year olds to address the problem that too few students in Britain study mathematics beyond the age of 16 and that a new approach needs to be designed that is more accessible to a wider range of abilities. To achieve this the pupil is required to be an active participant in the learning process and the use of computer software (and graphic calculators) is an important element. The course has met with approval from the participants and the results of the first examinations (taken in recent weeks) show a level of achievement higher than expected. More detailed study is needed.

Other generic organisers, particularly those for calculus, algebra and trigonometry, have been subjected to a closer in-depth study.

The initial work on the calculus was subjected to a controlled experiment in which the experimenter worked with one group of students using computers, two other groups followed a similar computer course and four groups followed a traditional course. In the experimental groups there was a marked increase in student participation in the mathematics generated by the introduction of the computer and the ability of students to improve on visual skills (such as sketching derivatives of functions given in graphical form) was significantly improved. There was no significant difference in carrying out routine algorithms of the calculus (Tall 1986a).

The work on algebra and trigonometry was carried out in controlled experiments in two adjoining halls of a comprehensive school where pupils had been set in ability so that the corresponding groups in the two schools had comparable results on previously applied tests. One school was given the computer treatment using the software, the other used the “tried and tested” methods evolved and agreed by the teachers over the years.

Thomas (1988) found that initially the pupils practising routine skills in algebra were marginally better than the experimental pupils at these skills, whilst the experimental pupils were better at problems requiring more versatile thinking. However, some sixth months later, following a brief recapitulation of routine skills, the experimental pupils now performed at a statistically significant higher level on both routine and more versatile questions.

Blackett (1990) has shown remarkable differences on the versatile understanding of trigonometry using his software. He studied four experimental and four control classes in two parallel halls which began their studies with almost exactly the same level of performance in each of the corresponding groups. Figure 7 shows the mean marks attained by each group on a school exam (given at the end of the previous year), a pre-test on trigonometric questions, and two post-tests, one immediately following the treatment, the other after a delay of eight weeks. Each of the latter are divided into the two parts denoted by S (for standard questions, which involved straightforward use of the trigonometric formulae) and V (for more versatile questions). It will be seen that the experimental students perform as well or better on standard questions and outscore the control students on versatile questions. There continues to be an improvement in the performances of the experimental students over the control students from the first post test to the second. Those cases where the mean of the experimental score is greater than that of the control score at a level $p<0.05$ are denoted by *sig*.

An observation of great interest is that the girls in this experiment performed differently from the boys. The girls started off marginally lower on scores than the boys in most groups, and in the control groups their performance did not increase as much as the boys, but in the experimental groups it overtook the boys, stretching away by the second post-test to a significant difference. Details of this development will be given elsewhere (Blackett & Tall, in preparation).

Conclusion

After several years of development and testing of the ideas proposed in this article, we have

	Exam	**Pre-test**	**Post-test 1**			**Post-test 2**		
			S	**V**	**Total**	**S**	**V**	**Total**
E1	61	37	81	78	79	89	83	86
C1	70	38	80	47	63	86	50	68
			–	*sig.*	*sig.*	–	*sig.*	*sig.*
E2	47	18	45	56	51	57	66	62
C2	48	20	46	38	42	39	41	40
			–	*sig.*	*sig.*	*sig.*	*sig.*	*sig.*
E3	47	20	47	52	50	57	66	62
C3	49	21	31	28	29	24	23	23
			sig.	*sig.*	*sig.*	*sig.*	*sig.*	*sig.*
E4	39	*not given*	22	48	35	22	37	29
C4	39	*not given*	17	5	11	*not given*		

Table 6: significance of results of post-tests by group and type of question

consistent evidence in the improvement of student’s performances in more versatile tasks and performances in more routine tasks have either not changed or have been improved as a result of the more versatile insight gained. This improvement in skills occurs particularly at the generic level in handling applications of the concepts to specific examples.

We have some knowledge of the way in which children learn mathematics and the cognitive conflict that can occur when confronted with new knowledge. Now we have new technology which can be configured to shed new light onto the knowledge so that the individual, using the technology is faced with a different conceptual task. The mind of the individual, *coupled with the complementary powers of the computer* offers a completely new conceptual framework for education.

References

Blackett N. 1990: *Developing understanding of trigonometry in boys and girls using a computer to link numerical and visual representations*, Ph. D. thesis, University of Warwick.

Blackett N. & Tall D.O. (in preparation): “Improving understanding of trigonometry in girls using computer software linking numerical and visual representations”.

Gray E. 1990: “An analysis of diverging approaches to simple arithmetic: preferences and its consequences” (submitted for publication).

Harel G. & Tall D. O. 1990: “The generic, the abstract and the general in advanced mathematical thinking”, (in preparation).

Heid M.K. 1988: “Resequencing Skills and Concepts in Applied Calculus using the Computer as a Tool”, *Journal for Research in Mathematics Education*, 19 1, 3-25.

Mathematical Association 1985: *132 Short Programs for the Mathematics Classroom*, [book & disc of computer programs] Stanley Thornes.

Tall D. O. 1986a: *Building and Testing a Cognitive Approach to the Calculus Using Interactive Computer Graphics*, Ph.D. thesis, University of Warwick.

Tall D. O. 1986b: *Graphic Calculus I, II, III*, (for BBC compatible computers), Glentop Press, London.

Tall D. O. 1989a: “Concept Images, Generic Organisers, Computers and Curriculum Change”, *For the Learning of Mathematics*, 9, 3.

Tall D. O. 1989b: *Real Functions & Graphs: SMP 16-19* (for BBC compatible computers), Rivendell Software, prior to publication by Cambridge University Press.

Tall D.O., Blokland P. & Kok D. 1990: *A Graphic Approach to the Calculus*, Sunburst, Pleasantville, NY. (for IBM compatible computers). (Also published as *Graphix* in German by CoMet Verlag, Duisburg).

Tall D. O. & Winkelmann B.,1988: “Hidden algorithms in the drawing of discontinuous functions”, *Bulletin of the I.M.A.* 24 111-115.

Thomas M.O.J. 1988: *A Conceptual Approach to the Early Learning of Algebra Using a Computer*, unpublished Ph.D. thesis, University of Warwick

Thomas M.O.J. and Tall D.O. 1986: “The Value of the Computer in Learning Algebra Concepts.”, *Proceedings of the 10th Conference of PME*, London.

Thomas M.O.J. and Tall D.O. 1988: “Longer-Term Conceptual Benefits From Using A Computer in Algebra Teaching”, *Proceedings of the 12th Conference of PME*, Veszprem, Hungary.

AUTHOR’S DATABASE ENTRY SHEET

TITLE OF THE NATO MEETING/ NATO ASI SERIES VOLUME:

ADVANCED TECHNOLOGIES IN THE TEACHING OF MATHEMATICS AND SCIENCE

NAME OF THE DIRECTOR:

DAVID L. FERGUSON YEAR: **1990**

NAME OF THE EDITOR:

DAVID L. FERGUSON

TITLE OF THE PAPER: **INTERRELATIONSHIPS BETWEEN MIND AND COMPUTER: PROCESSES, IMAGES, SYMBOLS**

AUTHOR OF THE PAPER: **DAVID TALL**

AUTHOR’S AFFILIATION: **SCIENCE EDUCATION DEPARTMENT, UNIVERSITY OF WARWICK, U.K.**

KEYWORDS:

Mathematics education, cognitive psychology, learning theory, computers, calculus, trigonometry, algebra, gender.

ABSTRACT:

Empirical research in the learning of mathematics using software developed at Warwick University is consistent with a theory of *selective construction of mathematical concepts* during learning. Traditionally, students must first routinize certain algorithms before being able to consider them as entities for mental manipulation. Software may be designed to carry out algorithmic processes, allowing the learner to focus on selected concepts and the relationships between them, permitting a new flexibility in the learning sequence. The article considers the relationship between processes carried out by the computer, images generated by the computer and in the mind of the individual, and the resultant relationship with mathematical symbols and their mental manipulation.

Biographical Notes

David Tall received his M.A. and D.Phil in Mathematics from the University of Oxford in 1967 and his Ph.D. in Education from the University of Warwick in 1986. He has published several text books on mathematics, six packs of computer software, including *Graphic Calculus*, over a hundred articles on mathematics and mathematics education, and is editor of the British student journal *Mathematics Review*. His main research interests are in cognitive development in the learning of advanced mathematical concepts and the use of the computer to aid learning. He has held visiting appointments in Germany, Australia, USA and Canada, and is Reader in Mathematics at the University of Warwick, U.K.

Subject Index