

# Designing Interactive Mathematics



**June Lester**  
**jalester@cecm.sfu.ca**  
**Centre for Experimental and**  
**Constructive Mathematics**  
**Simon Fraser University**

**Abstract.** The use of interactivity in computer-based mathematics education has been growing explosively in recent years, fueled mainly by the internet, but also by the production of educational CDs and in-classroom software packages. Unfortunately, the development of design principles for onscreen mathematical interactivity is still in its infancy, and examples of ineffective interaction or completely gratuitous "eye-candy" abound.

There is a need for a "grammar of interactivity" as a tool for onscreen communication in much the same sense that the grammar of a language is a necessary tool for verbal communication. Some of the required principles can be adapted from basic principles of CHI (computer human interaction) or even from webpage design ideas. However, effective mathematical communication also requires a careful examination of the intrinsic mathematical nature of onscreen objects and how users interact with them: how should a curve behave when dragged, for example, or how should an equation transform when the equals sign is clicked?

In this paper, I'll try to identify and clarify some of the relevant issues for the design of onscreen mathematical interactivity, and propose some preliminary principles for its grammar.

**1. Introduction.** I should emphasize from the start that what I present here comes not from any scientific study, but from a preliminary and somewhat idiosyncratic exploration of what makes onscreen mathematical interactivity work - or not. The impetus was frustration: while much well-intentioned onscreen interactivity clearly fails to accomplish what it set out to, it's often very difficult to see just *why* it fails. I've been scanning various sources - CHI/GUI design, graphical design, instructional design, my own head - for potential rules to adapt to mathematical interactivity. The result is this paper.

The section which follows discusses some basic issues related to mathematical design; the next two examine specific sample principles. The last section briefly lists other potential principles and how they might be applied.

**2. The behaviour of onscreen objects.** In the concrete-to-abstract continuum of objects in the universe, onscreen objects lie somewhere in the middle: though lacking physical reality, they nevertheless

- possess a visual form
- move, reshape or otherwise change form
- interact with each other or with external “users”

(unlike more complete abstractions like “truth” or “set”).



This position in the middle has an important implication for computer-based mathematics: while physical objects become more abstract when modeled onscreen (e.g. science simulations), mathematical objects, already inherently abstract, become more concrete. Dynamic geometry software ([Cinderella](#), [Cabri](#), [Geometer's Sketchpad](#)) is currently the strongest illustration of this tendency: static, abstract geometrical objects become dynamic and even “tangible” onscreen. Some CASs (computer algebra systems) are also headed this way: [LiveMath](#) users, for example, do substitutions by dragging one equation on top of another.

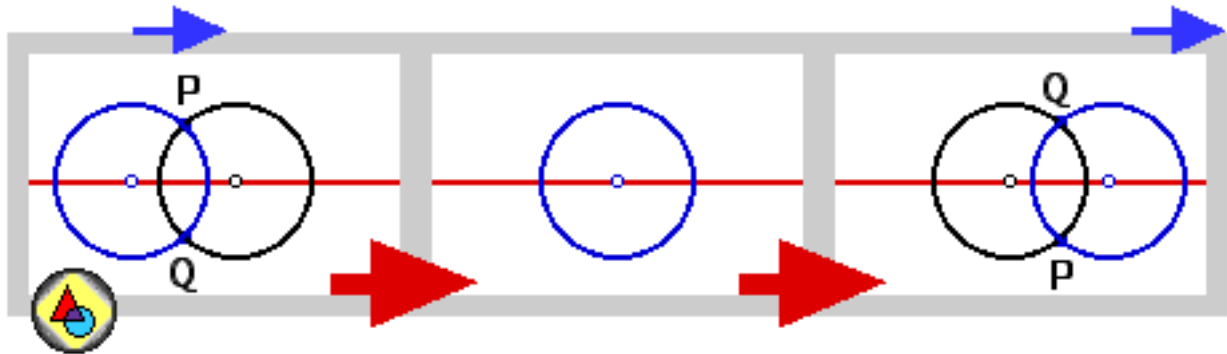
For mathematical design, this has two consequences:

a) having more concrete mathematical objects allows object design principles from engineering, manufacturing, etc. to be used (e.g. as in [[Norman](#)])

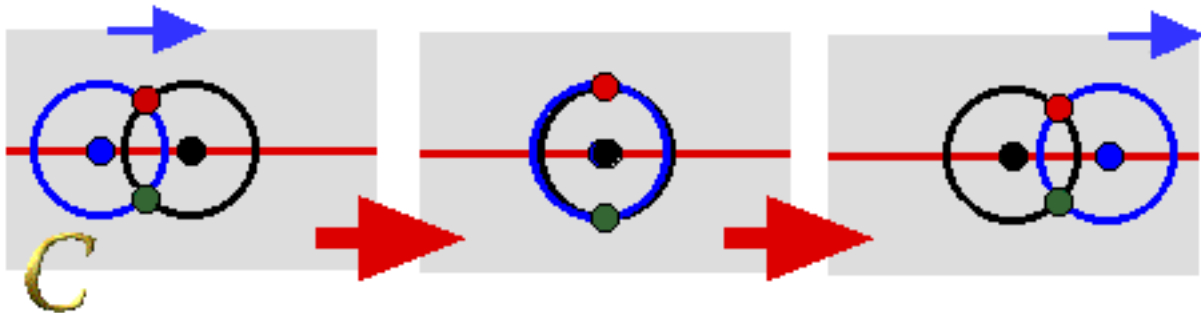
b) while real world objects have pre-ordained physical behaviours that can be modeled directly onscreen, mathematical abstractions do not; appropriate onscreen behaviours must be either determined from basic mathematical principles and conventions, or invented.

The latter consequence can make determining appropriate onscreen mathematical behaviours sometimes difficult. Let's look at how Geometer's Sketchpad and Cinderella treat circle intersections. Suppose a moving circle passes another of equal radius with its centre constrained to a line through the centre of the latter. What should happen to their points of intersection as the moving circle passes by?

In Sketchpad, the two points exchange positions after the circles coincide - a somewhat counterintuitive and undesirable result: continuous motion should produce continuous consequences.



In Cinderella, the points don't exchange positions:




A more "natural" behaviour - or is it? Suppose the moving circle's centre passes only very near the fixed circle's centre (so the circles never exactly coincide). The intersection points are then seen (in either program) to do a rapid flip around the stationary circle as the other passes by. So if small perturbations in position should cause only small changes in behaviour, perhaps the flipping behaviour is indeed more appropriate? In fact, either behaviour is arguably better than the other; we have no "physical" universe to arbitrate the question.

**3. Natural mappings.** Sometimes appropriate onscreen mathematical behaviours can be derived from natural mappings - "physical analogies or cultural standards" [Norman] which relate "controls and their movements" to the real world. So, for example, push a window control up to raise the window and down to lower it. The design principle: exploit natural mappings wherever possible; otherwise standardize. In the context of onscreen mathematics, this may be expanded to mean

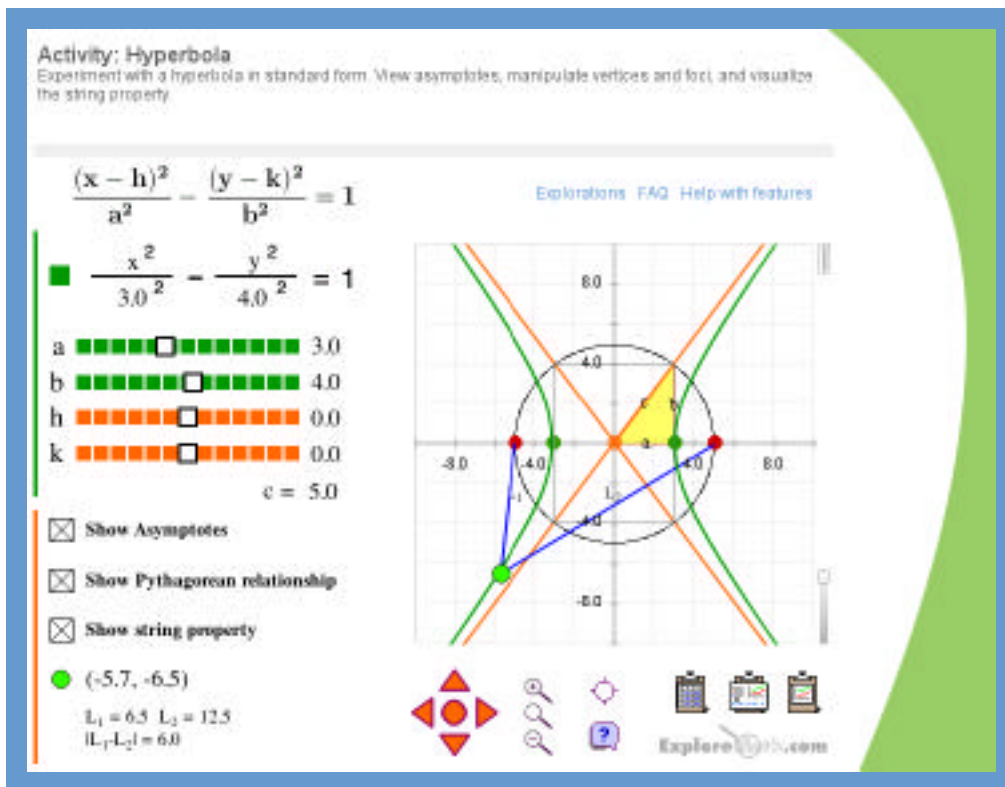
**If possible, model the behaviours of onscreen mathematical objects by observing and analysing their "natural" mathematical properties; otherwise, invent and standardize behaviours.**

A first criterion is of course that the natural mapping represent the mathematics concerned *correctly*. This shouldn't need saying, but does: I clearly remember an educational television animation which "solved" the equation  $3x - 2 = 2x + 5$  by moving an  $x$  to the opposite side:

$$3x - 2x = 2x + 5$$


Though incorrect, this example does illustrate one very basic math-to-screen mapping: mathematically transforming an expression by moving bits of it around. LiveMath implements this mapping extensively - and correctly: dragging the  $x$  across the  $=$  sign in  $3x - 2 = 2x + 5$  gives the intelligent result  $\frac{3x-7}{x} = 2$ .

To see how an analysis of mathematical behaviour might map mathematics to interactive design, we look at an interactive hyperbola from the [Explore-Math website](#). The page is visually attractive and well laid out:

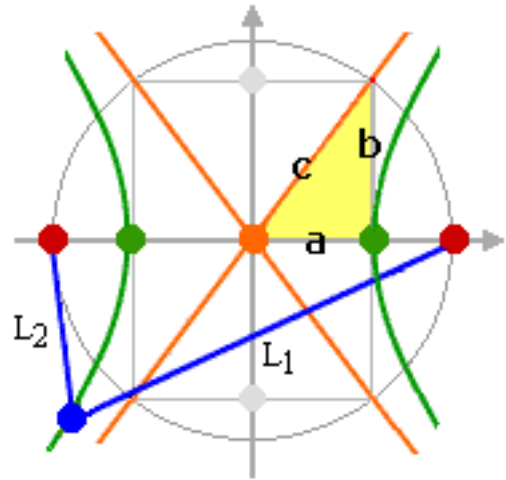


Below the graph are standardized tools for panning, zooming, saving, etc.. Check boxes (lower left) control the visibility of the asymptotes, the "string", etc. The sliders above them control parameters defining the position and shape of the hyperbola; dragging the marked points of the graph also changes these parameters.

How then does dragging actually change the hyperbola? Dragging the **centre** of the hyperbola or adjusting the **h** and **k sliders** translates the hyperbola itself exactly as expected. But careful analysis reveals a problem with the dragging interaction between parameters and shape.

The shape of the hyperbola is naturally determined in either of two ways:

- if we are looking at the *equation* of the hyperbola, by the pair of parameters **{a, b}** (the lengths of the semi-axis and conjugate semi-axis)
- if we are looking more at the *geometry* of the hyperbola, by the pair of parameters **{c, L}** (the focal distance and the **string number**  $L = |L_1 - L_2|$ ).



The parameters of each pair are independent, so changing one parameter of a pair shouldn't affect the other. That is almost what happens:

- ✓ changing either **a** or **b** by *dragging its slider* leaves the other fixed and changes **c** and **L** as appropriate
- ✓ dragging a **focus** changes **c** but not **L**. (Unfortunately, **L** cannot be changed directly; dragging the point on the hyperbola just moves it around the curve.)
- ✗ changing **a** by *dragging a vertex* changes **b** but not **c** (the slider for **b** is seen to move disconcertingly on its own).

This last behaviour is both inconsistent and inappropriate; perhaps it was decided that, since **b** can't be varied by dragging directly on a point, it should become dependent on the other parameters. A possible remedy: remove this dependence and allow dragging the conjugate foci (the ghost points in the diagram) to vary **b**.

We may abstract a basic principle here.

**Principle of Parametric Independence: If two or more parameters controlling the behaviour of an onscreen object are mathematically independent, then changing one of them, either directly or by dragging on the object, should have no effect on the other parameters or on the aspects of the object they control.**

In light of the previous section, rules derived from natural mappings may be difficult to obtain in general. The fallback: standardize - choose appropriate but essentially arbitrary behavioural conventions and stick to them. This is

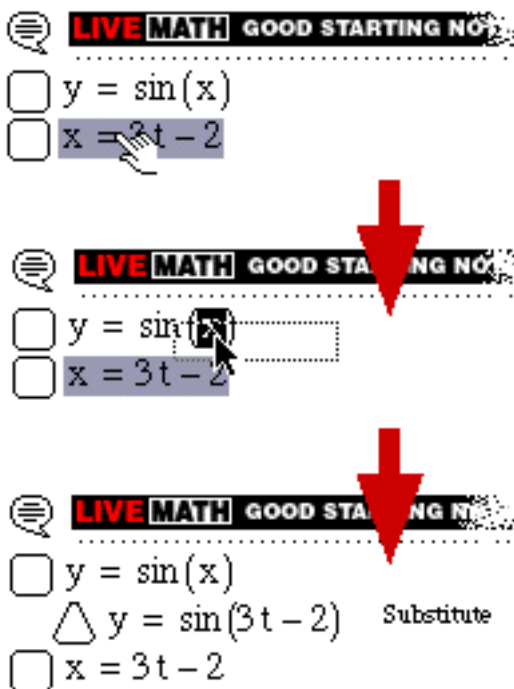
itself unknown and possibly dangerous territory: hic sunt dracones. For relatively elementary mathematics, it may be possible to develop universal conventions: click on a  $\oplus$  sign to expand the sum, for example, or click on the word "Theorem" to go to its proof. But in higher mathematics, where symbols are often "overloaded" (many meanings for one symbol, depending on context), it may be impossible. There, it may be more practical simply to define interactive conventions at the beginning of the work much as we now define notation; something like "in this paper, clicking on any blue operator will commute the operands on either side of it" or "all tree diagrams on these pages have the property that dragging any node off the screen deletes it and all dependent nodes".

**4. Visible behaviours.** Another basic design principle, adapted from CHI:

**Make interactive behaviours visible, at all stages of the interaction:**

- pre-interaction: show the **affordances** of an onscreen object (the possible interactions it affords), i.e. show what the viewer can do with it.
- co-interaction: provide **feedback**, i.e. show something happening as the user interacts with the object.
- post-interaction: show the resulting **state** of the object, i.e. show how the user changed it and whether/how it can be undone or restarted.

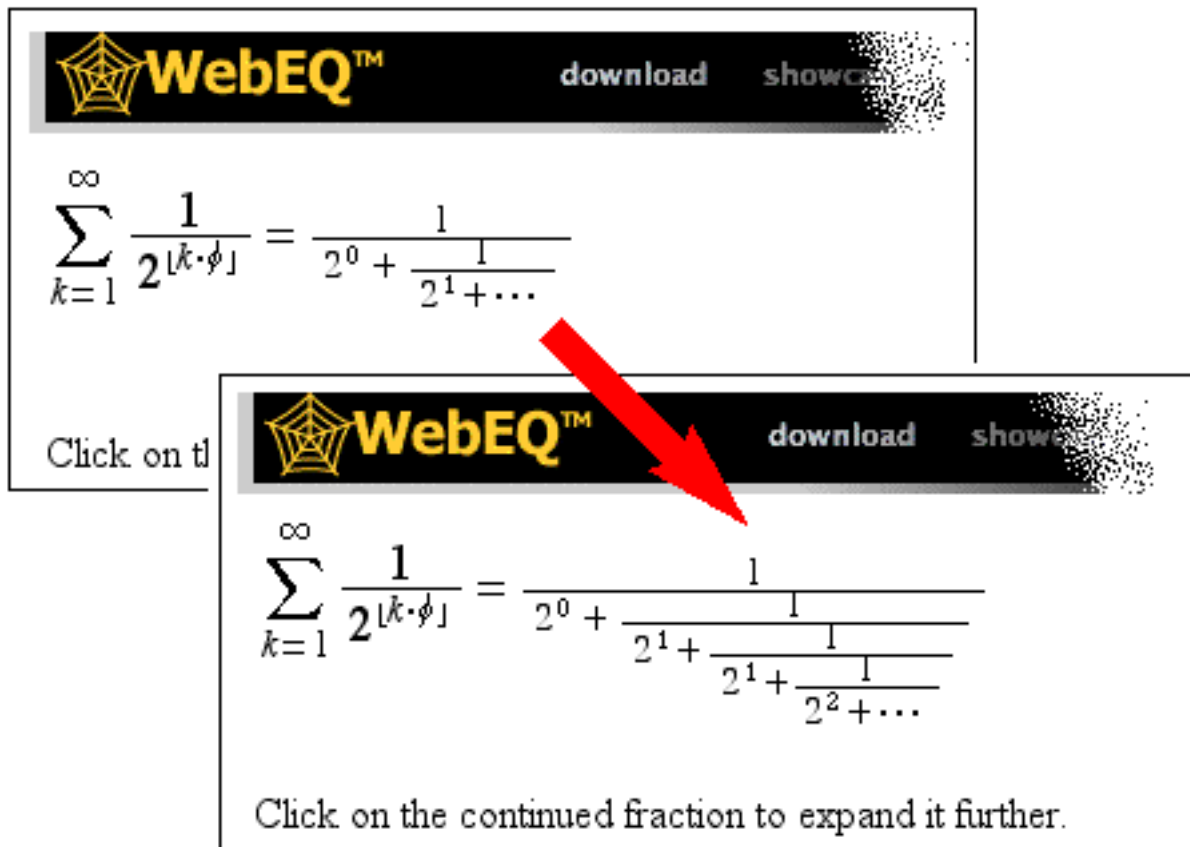
Look at how LiveMath does this with substitutions. A LiveMath substitution is made by dragging one equation onto the other.



The equation to be dragged is first selected (and thus highlighted). Then

- while the cursor is over the selected equation, holding the command/ control key gives a pointing hand icon, indicating that the equation **can be dragged**
- **as the equation is dragged**, a box appears around it and potential destinations are highlighted when close enough
- **after the equation is dropped**, a new equation appears along with the word "substitute", to indicate how it was created.

Another example, from [WebEQ](#), shows how invisible affordances can make effective interaction difficult. Though we are instructed to click on the continued fraction to expand it, most clicks have no effect. Trial and error eventually hits the right target: the ellipsis ... . The fraction expands by two levels, and by two more when the next ellipsis is clicked. It *reverts* two levels when the third ellipsis is clicked, and - more trial and error - whenever the last + sign at either stage is clicked.



The technology used here is clearly very useful; the website has other interesting demonstrations. Visible affordances would greatly improve this particular one - a blue ellipsis or a green + sign to hint that something happens when they are clicked, for example. (There is also a question of the appropriate mathematical behaviour here: why does the fraction expand by *two* levels at each click?)

**5. Additional design principles.** There are potentially many of these. What follows is a somewhat eclectic list from various sources (including me) with some comments as to how they could be adapted to onscreen mathematical design.

- **KISS - keep it strictly simple.** Simplify details of calculations, proofs, navigation, explanations, visual appearances, etc. Simplify and clarify basic mathematical structures and relationships *before* designing how they interact.

- **Keep the interactivity focussed.** Keep interactions minimal, restricted to the important mathematics. Use constraints to keep viewers on track and off tangents, and don't present too many choices. Use "lockouts" to prevent omission of essential steps.

- **Adapt basic CHI principles to mathematical interactions.** For example, numerical information essential to understanding or decision making should be immediately available onscreen and not several clicks away or (worse) to be recalled from memory. Or, for numerical inputs that change during interactions, allow point and select input in preference to typed input (e.g. design slider inputs for parameters).

- **Use appropriate onscreen graphical design.** For example, the KILL rule (keep it large and legible) - watch out for subscript sizes, etc. Fade the unimportant parts of diagrams (e.g. graph grids) into the background. Use "white space" liberally. And so on - the visual interface to onscreen mathematics is as important as the interactive one.

- **Adapt instructional design principles to onscreen presentations.** For example, avoid spatio-temporal language (e.g. "from the previous lemma, ...") in hyperlinked materials (since you don't necessarily know where your reader is coming from). Distance education ID should be particularly relevant here, since it deals with non-face-to-face instruction. An example: provide "reader-stoppers" - natural stop-and-think points - at appropriate places in the material [[Rowntree](#)].

- **Allow the viewer to act directly on onscreen objects** by clicking or dragging rather than (only) through buttons, sliders, or other controls. Objects which can be manipulated directly are easier to understand and feel more natural: dragging one equation onto another to substitute (LiveMath) is much more intuitive than typing a command.

- **Don't use animation as a substitute for interaction.** It is more instructive if a viewer produces changes himself by dragging, clicking or otherwise actively controlling them than if he passively watches an animation of those changes (e.g. a graph whose shape changes as a parameter is varied). For similar reasons, any animations that *are* used should be stoppable and steppable.



- **Use “hypertelevision” sparingly.** Don’t rely solely on interaction/navigation of the form “click here to go to the next step”, which can be passive and unengaging (like using a television remote control to click through channels). If possible, replace it by something more meaningful, such as “click on an equilateral triangle to continue” in a lesson on triangles, which also checks understanding. On the other hand, hyper-television may be appropriate for content which is inherently stepped (e.g. a proof) or may be useful to help “chunk” material appropriately.

- **Allow for multiple use and reuse.** This is particularly important when designing “learning objects” (small chunks of learning material meant for use in a larger context). Suggestions:

- keep lesson materials separate from the interactive mathematical objects (one object might be usable in several lessons)
- make the mathematical object configurable (so it can be used to teach different ideas, e.g. a grapher with multiple input possibilities)
- consider the possible contexts of use in the design.

.....

**References and Notes.** All links checked and correct as of July 31, 2000.

## Software/Webpages

- Dynamic geometry software allows onscreen construction of geometric objects and interaction, principally by dragging. Three examples:
  - *Cinderella* <[www.cinderella.de](http://www.cinderella.de)>
  - *Geometer's Sketchpad* <[www.keypress.com/Pages/Prod\\_Sketchpad.html](http://www.keypress.com/Pages/Prod_Sketchpad.html)>
  - *Cabri* <[www.cabri.net/](http://www.cabri.net/)>.

All three programs allow pages to be saved/converted into applets for webpages.

- *ExploreMath* <[www.exploremath.com](http://www.exploremath.com)> is a website of shockwave-driven activities for school-level mathematics. The interactive hyperbola, at <[www.exploremath.com/activities/Activity\\_page.cfm?ActivityID=5](http://www.exploremath.com/activities/Activity_page.cfm?ActivityID=5)>, is part of the section on conic sections; note that the interactive ellipse in that section behaves correctly. The companion site, *ExploreScience* <[www.explorescience.com](http://www.explorescience.com)>, has similarly designed science activities.

- *LiveMath* <[www.livemath.com](http://www.livemath.com)> is a teaching-oriented CAS published by WebPrimitives <[www.webprimitives.com](http://www.webprimitives.com)>. A LiveMath plugin allows LiveMath notebooks in web pages. LiveMath versions for Macintosh and Windows currently exist; a Linux version is in the works. In previous lives, LiveMath was *Theorist* and then *MathView*.

- *WebEQ* <[www.webeq.com](http://www.webeq.com)> is a programming interface for interactive mathematical text. The example cited, entitled Ramanujan Identities, is at <[www.webeq.com/showcase/tour/ramanujan.html](http://www.webeq.com/showcase/tour/ramanujan.html)>; other examples can be found in this "tour" section. WebEQ has recently been acquired by Design Science, Inc. <[www.mathtype.com](http://www.mathtype.com)>, makers of *MathType*.

## Books

- Julie Jacobs & William Mueller, *Design Principles for Interactive Texts*. An online book discussing graphical, psychological and evaluation principles. <[www.math.duke.edu/education/ccp/resources/write/design/index.html](http://www.math.duke.edu/education/ccp/resources/write/design/index.html)>
- Donald A. Norman, *The Design of Everyday Things*, Basic Books, 1988. ISBN 0-385-26774-6. A very readable book on general design principles.
- Derek Rowntree, *Teaching Through Self-instruction*. Kogan Page, London, 1986. ISBN 1-85091-148-7. Discussion and examples of "reader-stoppers" appear on page 93 and many others; see the index.
- John Sweller, *Instructional Design in Technical Areas*. ACER Press, Camberwell, Australia, 1999. ISBN 0-86431-312-8. A potential source of further onscreen mathematical design principles based on cognitive load theory.