Lifting the Curtain: The Evolution of The Geometer's Sketchpad[1]

Daniel Scher

It starts as a thought experiment. You imagine an arbitrary quadrilateral with its four midpoints connected to form another quadrilateral nested inside. You wonder whether this inner quadrilateral possesses any special properties, and if these properties continue to hold when the vertices of the outer quadrilateral assume different locations.

Almost instinctively, you move to your computer and launch a dynamic geometry software program such as The Geometer's Sketchpad (Jackiw, 1995) or Cabri Geometry (Texas Instruments, 1994). With a few clicks and movements of your mouse, the construction is complete: there on your screen is a model of the quadrilaterals that can be freely manipulated, just as in your thought experiment. Some dragging of the outer quadrilateral's vertices convinces you there is a behavior worth noticing; the inner quadrilateral appears to remain a parallelogram.

The scenario described here is by now a common one. In an age when we become surprised if something on our computer screen *doesn't* move, it might seem natural, even obvious, that the static figures from Euclidean geometry should give way to dynamic ones. But "obvious" would be an inappropriate word to describe the foundations of dynamic geometry. Despite the similarity of ruler-and-compass geometry to its software counterpart, the crafting of Sketchpad and Cabri was not a straightforward matter of transporting Euclid's axioms to the computer. Jean-Marie Laborde, one of four Cabri authors, explains that some departures from Euclidean axioms were inevitable:

> The general principle was to make the distance [between Cabri and Euclid] as small as possible, but at the very beginning I was not aware that it would remain finally at some distance...People weren't happy at all [with the] expression 'Cabri geometry.' But we decided nevertheless to introduce that concept to make definite the point that what comes from the screen is not Euclidean geometry, it's not projective geometry...It has to be different.

Nicholas Jackiw, the designer and programmer of Sketchpad, echoes these comments in a November 1994 posting to the online Swarthmore Geometry Forum:

> 'Why hasn't anyone done this before?' is the most common initial reaction to seeing something like Sketchpad. But then...one realizes something strange is going on behind the curtain—something that may seem intuitive, but which is by no means obvious, and by no means predetermined by the geometry and mathematics we understood before the advent of these programs. (http://forum.swarthmore.edu/epigone/geom.software-dynamic/3/smoke-0711941827100001@olmo.swarthmore.edu)

This article aims to peer "behind the curtain" to see how educational software ideas that are clever and powerful come into being. Largely, it is an historical account of one particular implementation of dynamic geometry, The Geometer's Sketchpad, assembled from early design documents and interviews with its creators. But to spotlight the places where design choices could have been made differently—there is no single model of what it means for geometry to be "dynamic"—references are also included to Cabri and the Geometry Inventor (Logal Software, 1994).

The Early History

Sketchpad began as an outgrowth of the Visual Geometry Project (VGP) at Swarthmore College, directed by Eugene Klotz and Doris Schattschneider. In the mid 1980's, the project proposed the development of a videotape series that would focus on three-dimensional geometry. Klotz says that as an "article of blind faith" he included interactive computer programs in the pro-

Daniel Scher is a doctoral student in mathematics education in the Department of Teaching and Learning at New York University. His homepage, http://members.xoom.com/dpscher, *contains a Java Sketchpad-based curriculum in conic sections. His e-mail address is* dscher@mac.com

posal description, as he felt video was a transient medium that would one day be replaced by the computer.

For programming, Klotz turned to Nicholas Jackiw, a student he advised in Jackiw's freshman year of college. Jackiw's interests included both English and computer science (he began programming at the age of nine) but not, surprisingly, mathematics. Indeed he avoided mathematics courses entirely in college.

Jackiw's first work for the VGP was a software program called Cavalieri. Jackiw describes the program as a foreshadowing of the dynamic element of Sketchpad. It allowed users to take a shaping tool—essentially a virtual bulldozer—and "plow" into a figure from the left or right, reconfiguring its shape dynamically (see Figure 1).

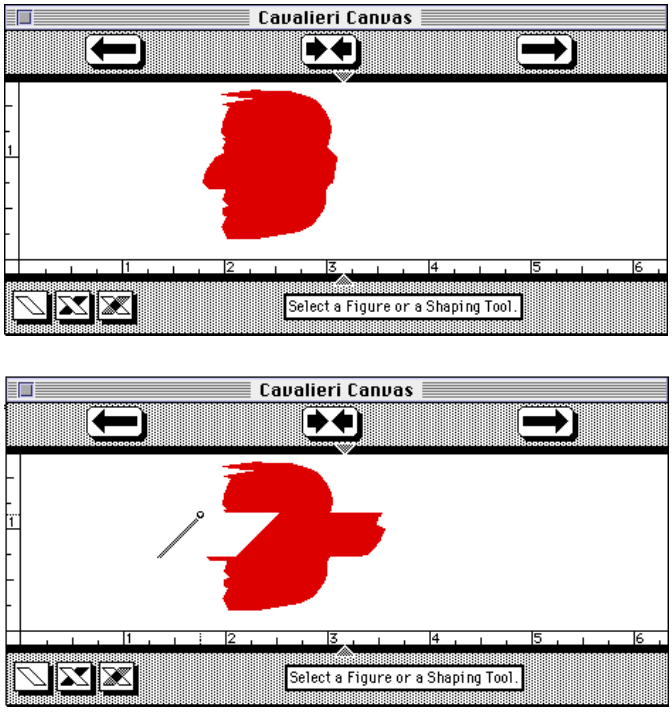

Figure 1. Using Cavalieri to "bulldoze" a face.

The original aim of the VGP was to produce computer programs to accompany each videotape of a three-dimensional geometry concept. But given the time it took to complete just Cavalieri and the difficulty of programming three-dimensional models, Klotz and Schattschneider decided to shift their focus and create a single, two-dimensional graphics program.

One of the earliest vector graphics programs to showcase the graphical capabilities of the computer was Ivan Sutherland's "Sketchpad." A hand-held light pen allowed the user to draw and manipulate points, line segments, and arcs on a cathode ray tube monitor (Franklin Institute Online, 2000). Klotz also wanted to mine the graphics potential of the computer, and in honor of Sutherland's work, named their new program-to-be, "The Geometer's Sketchpad."

The Dragging Story

The ability to "drag" objects and manipulate them dynamically is perhaps the most defining feature of Sketchpad and Cabri. Yet from a historical perspective, the ideas of dragging and motion did not arise in a vacuum. Colette Laborde (1994) writes,

> The idea of movement in geometry is not new—the Greek geometers devised various instruments to describe mechanically defined curves—but the use of movement was nonetheless 'prohibited in strict geometric reasoning' for reasons that were more metaphysical than scientific. The 17th century marked a break with Greek tradition, and the use of movement to establish a geometric property or carry out a geometric construction became explicit. One can find numerous examples starting then....
>
> This idea was first expressed in school geometry by the replacement of the geometry of Euclid's *Elements* by the geometry of transformations (which continues to be the only kind of geometry taught in some countries)—quite some time, one must point out, after the characterization of geometry as the study of the invariants of transformation groups, and also quite some years after a daring proposition made in France by Meray (Nouveaux éléments de géométrie, first edition 1874)...Meray's idea was to teach geometry through movement: translational movement allowed for the introduction of the notion of parallelism; rotational movement led to perpendicularity. (pp. 61-62, French original)

Writing in 1945, Syer describes the ability of film to create "continuous" geometric images. His advocacy of the moving picture reads much as a modern-day justification for dynamic geometry:

> In addition to true-life demonstrations of solid geometry, it would be interesting to make greater use of the peculiar advantage of moving pictures over ordinary models. In plane geometry films, we used figures that changed shapes, position, and color without distracting pauses or outside aid. This

> continuous and swift succession of illustrations is fast enough to keep up with a spoken description, or even as fast as the thought processes that are developing the idea. Thus no time is lost erasing pictures from the blackboard, changing lantern slides, or holding up illustrations, because the illustrations and thought move simultaneously. (p. 344)

Given the historical antecedents of dynamic geometry's dragging feature, it is easy to picture these as factors pivotal in its creation. The actual development of dragging, however, tells otherwise.

Sketchpad

Early design plans for Sketchpad did not include a dragging component. Klotz envisioned the program as a way for students to draw accurate, static figures from Euclidean geometry:

> Basic motor skills were keeping [students] from being able to draw. I thought we needed to have something that allowed people to make the basic constructions. So to me, [Sketchpad] was a drawing tool. You'd make a geometric drawing that was precise and accurate, and scroll over the page to see what was going on.

Drawing and selecting objects was to occur via the mouse or by issuing menu commands. Many of the proposed menu items and selection methods would be unfamiliar to today's user of Sketchpad. Table 1 lists several of them.

Table 1
Drawing and selection options considered for Sketchpad

Drawing Options	Selection Options
draw point by typing coordinates	select point by giving coordinates
draw line segment by giving a point, direction, and distance	select circle by giving name (its label)
draw circle by giving the coordinates of three non-collinear points	select figure by listing part names

These options were sufficiently different from the standard Macintosh user interface to cause Jackiw to wonder whether there could be a better alternative. With the Macintosh, one could select objects and drag them directly with a mouse. Jackiw wanted to extend this notion to Sketchpad. His interest and experience in programming video games for the Macintosh helped inform his sense of what a computer geometry program could be:

> It's the video game aspect that gives me my sense of interactivity when dealing with geometry...Looking at the input devices of video games is a tremendously educational experience. In the old days, you had games with very interesting controls that were highly specific...[The video game] Tempest had a marvelous input device...The types of games they would write to suit this bizarre and unique device were always interesting experiments in what does this hand motion transport you to in your imagination. I wanted to have a good feel in all of my games.

The mouse of a Macintosh was not an ideal input device for games, but it was suitable for manipulable environments where objects could be dragged. The illustration program MacDraw, in particular, contained the rudimentary features of dynamic geometry, as one could draw and move a segment with the mouse and change its length. When Jackiw took the basic premise of MacDraw and applied it (with considerable reworking) to Euclidean geometry, Klotz found the results striking:

> I remember how shocked I was when I first saw it. [Jackiw] had played with a Macintosh long enough to know that you should be able to drag the vertex or a side [of a triangle] and protrude the figure. I was flabbergasted. I mean, he made the connection, and I didn't.

Once Jackiw had decided to make Sketchpad "dynamic," he did not want any of its elements—even something as seemingly innocuous as labels—to suggest a static state. He says,

> I resisted labeling for a long time because I wanted the user to be engaged in the world of graphics. I didn't want things that made [the program] seem like it was representational, which labels seemed to me to do. They turned it into an illustration, whereas I wanted it to be a world.

Cabri

As with Sketchpad, the early motivations behind the dragging feature of Cabri were not tied to mathematical pedagogy. Jean-Marie Laborde maintains there was no curricular goal, learning theory, or educational research agenda that sparked the original idea.

> Int: What did you imagine was going to happen with this world [of Cabri Geometry]? Were you thinking of it as pedagogy, or...
>
> J-ML: No, at the very beginning it was just for fun, to have such a tool in geometry...For myself, it was just thinking back to those figures from [high school or university] geometry and thinking it would be nice to [explore them] in the dragging mode, just 'for fun,' in quotation marks.

While "drag mode" was an agreed upon aspect of Cabri, its status was not immediately clear. For Laborde, it was central—he had a strong interest in the "desk" metaphor and direct manipulation concept[2] as a way to replace particular diagrams with an infinite number of diagrams—but others were not initially convinced that it should be the default behavior. The first implementation of Cabri did not grant it freely; one had to access the menu to drag a point. The elevation of dragging to default status followed experience with the new tool: as people played with the early version, they nearly always wanted to drag points dynamically.

As a postscript, it is worth noting that the dragging features of Cabri and Sketchpad were both well underway before either knew of the other. Klotz recalls his first encounter with Cabri:

> We had just that Fall got into our dragging bit, and were very proud of what we had. We thought, God, people are going to really love this. But [Cabri] had scooped us, and we had scooped them. It was one of these, you know, just amazing things where...maybe you can sort out the exact moment, maybe there was a passing meteor, or something.[3]

Menu Items: Many or Few?

For better or worse, constructing a parallelogram or building a circle through three given points are both multi-step procedures with a ruler and compass. Transporting geometry to a computer presents other possibilities. One could imagine a tool that instantly generates a parallelogram on the screen. The shape of the parallelogram could then be altered, but the actual work of building the parallel sides would already have been done. Likewise, one could provide a tool that encapsulates the entire circle construction into the single step of selecting three points. In fact, these options appear in at least one dynamic geometry program, The Geometry Inventor (Logal Software, 1994).

While one could conceivably provide a user with a host of tools to automate the most specific of constructions, it was not clear to the Sketchpad designers that more tools meant a better program. Says Klotz,

> I sort of wore two hats; one of them was minimalist Euclid and the other was trying to make things user convenient. Depending on what hat I wore, I said well, let them construct the damn thing, and you learn something from that. On the other hand, I realized, even then, you've got to listen to user interests and so forth.

Jackiw faced this issue from both a programmer's and a learner's perspective. Providing menu options to automate an assortment of geometric construction tasks meant that Sketchpad would become a hefty program. All totaled, Jackiw faced a daunting wish list of near 400 menu items—an amount, he joked, that would require 21 volumes of documentation. In a memo to the Sketchpad project staff, he explains,

> I and the other students in my geometry class would far rather have sat down with graph paper, a pencil, and a protractor and meticulously done by hand just about anything that could reasonably be assigned to us than to have read the instruction manual for a program that does everything listed here...I don't think that having 40 items in a tool palette and another 60 in menus is either logical or intuitive.

By advocating for a leaner Sketchpad, Jackiw aimed to trim the construction commands down to their "atoms" and eliminate the majority of automated constructions that could be accomplished by more primitive techniques. This is not to say, however, that a student would need to rebuild a square from scratch each time one was needed. Jackiw viewed the scripting feature of Sketchpad as a way for students to start from the "atoms" and gradually build their own collection of reusable, multi-step constructions. He writes in a May 1995 posting to the Geometry Forum,

> In my ideal vision, students author their own tools, which forces them to confront and think through the geometry implicit in a construction before making the result available to them in the future, as a magic recipe for achieving their larger construction goals. (http://forum.swarthmore.edu/epigone/geom.software-dynamic/17/njackiw-2305951359370001@198.95.207.164)

Ultimately, the scripting feature in versions 1 and 2 of Sketchpad did not receive heavy use. To construct a square using a script, the user needed to match two points on the screen with the two "givens" of the script—a multi-step procedure. Version 3 of Sketchpad streamlined the process by allowing the user to create a script tool button. With this new design, clicking two points on the screen automatically constructed a square with the points as neighboring vertices.

Sketchpad as a Visual Spreadsheet

As Jackiw refined his model of how geometric objects would respond to dragging, he developed a deeper sense of Sketchpad's underlying structure. He comments,

> The birth of dynamic geometry as a concept, as opposed to just an artifact of trying to be like the Mac, consisted of two sorts of parallel understandings of what Sketchpad would be. One was the notion that it was a spreadsheet, but a spreadsheet that worked with graphics.
>
> In the traditional spreadsheet paradigm, visualization—graphing or plotting the data set—comes as an optional postscript to an often tedious exercise in data generation. Could Sketchpad dispense with the externalization of numeric data, and allow users to work at all times directly with a visual model...?

Sketchpad, as a visual spreadsheet, would share some of the characteristics of a traditional numerical spreadsheet. With an Excel spreadsheet, a user might indicate that cell Z = cell Y + 1. Any changes then made to the value of cell Y would affect the value of cell Z. Such dependencies exist in Sketchpad, too, only in graphical form. For an arbitrary ΔABC, dragging vertex A affects the location of AB and AC.

This spreadsheet analogy worked well for Jackiw, but it was not as robust as he would have liked. His other desire for Sketchpad was that it allow for *reversibility*. In the formula given above (cell Z = cell Y + 1), a user cannot input the value of cell Z and ask the spreadsheet to work backwards to calculate cell Y. Put another way, imagine a user has entered a formula that calculates profit in terms of two variables: number of widgets sold and price per widget. The spreadsheet operates in only one direction—given values of the two variables, it calculates profit. Jackiw imagined what would happen if the program had a reversibility feature built in:

> With reversibility, having given it that model, you can say, 'How do I make a million bucks?' And it would run the whole model in reverse and say, 'Ah, you need to sell this many widgets at this price.'

Of course, the solution might not be unique. There could be more than one combination of widget quantity and cost that produces the desired profit.

Geometric Reversibility

How does this notion of reversibility apply to a geometric setting? Jackiw offers an acoustics example to explain. Imagine you are in a room with two stereo speakers placed on the floor. Somewhere in the room is a location where the music reaching your ears will provide the optimum listening experience—what's commonly known as the "sweet spot." While the music plays, you move the speakers by trial and error, attempting to get the sweet spot to coincide with the location of your easy chair.

By contrast, suppose you're resting comfortably on the easy chair and have no intention of moving about to fiddle with the speakers. If every location of the speakers defines a sweet spot, then why not reverse that logic and say that every location of the sweet spot defines a (possibly non-unique) placement of the stereo speakers. So relax in your chair—from a logical standpoint, the speakers should be able to relocate themselves!

Shifting now to a Sketchpad environment, Jackiw describes the following investigation: Using Sketchpad 2.0 or higher, draw a circle with center A and place three points, C, D, and E, at random locations on the circle. Connect the points to form ΔCDE, and then draw medians connecting the triangle's three vertices to the midpoints of their opposite sides. The medians meet at the triangle's centroid, point I, as shown in Figure 2. By

experimenting with this construction, what can you say about Δ*CDE* when points *I* and *A* coincide?

One approach to this question is to vary the locations of *C*, *D*, and *E* in an attempt to get *I* to move onto *A*. This method frames points *C*, *D*, and *E* as independent variables whose placements on the circle determine the location of the dependent centroid *I*. Drawing on the stereo example, points *C*, *D*, and *E* can be imagined as the speakers we move by trial and error to reposition the sweet spot *I* to land at our easy chair, point *A*.

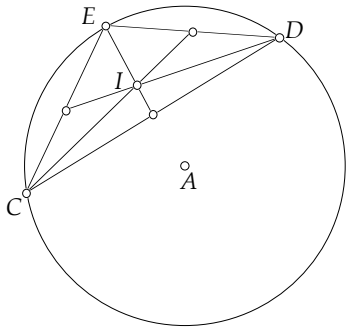

Figure 2. A circle with inscribed Δ*CDE* and triangle centroid *I*.

Jackiw explains that *I*, in fact, can assume the role of the independent variable. Since we wish to know what happens when *I* overlaps *A*, drag *I* onto *A* and watch Δ*CDE* reconfigure itself to become equilateral (see Figure 3). In doing so, we reverse the notion that the location of Δ*CDE* must determine the location of *I*, and instead examine how *I*'s location can determine Δ*CDE*.

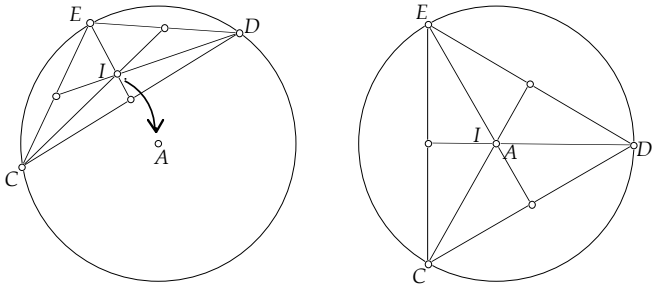

Figure 3. Dragging *I* onto *A* causes Δ*CDE* to become equilateral.

Since the vertices in Figure 3 cannot mimic *I*'s precise path as it is dragged from one location to another, programming decisions needed to be made about how the points would move. Notably, for a given location of point *I*, the placements of *C*, *D*, and *E* on the circle are not uniquely determined. Drag *I* onto point *A*, release the mouse, and note the locations of *C*, *D*, and *E*. Now reach for the mouse again, drag *I* away from *A*, and then drag it back. Most likely, the locations of *C*, *D*, and *E* will have changed.

A Democracy of Points and Segments

A user who draws an arbitrary polygon *ABCDE* with Sketchpad can construct it in several ways. Vertex *A* might get drawn first, or it could very well be the last point placed on the screen. In this simple example, the order of the construction does not confer any hierarchy or special behavior on the vertices. The user expects to be able to drag any vertex of the polygon and deform the figure into different shapes. Jackiw refers to this behavior as the "democracy" of vertices in a polygon.

While Jackiw found this democracy desirable, it was not simple to achieve. One way to reflect the equal nature of the triangle's vertices was to represent them internally as a looping cyclic graph with no beginning and end. But in such a model, programming the constraints of the triangle meant representing them algebraically as a system of simultaneous linear equations. The time it took for the computer to solve the equations would not have allowed the program to be interactive.

As another option, Jackiw considered using an acyclic graph. Figure 4 shows a triangle *ABC* with point *D* on side *B C*, along with its acyclic graph representation on the right. In this model, each geometric object in the triangle—its vertices, its segments, and the point on its segment—becomes a node of the graph. Arrows flow from node to node, indicating the chronology of the construction and its dependencies. Vertices *A* and *B*, for example, define segment *k*, and thus their nodes come together with an arrow pointing toward node *k*.

An acyclic graph did have its problems: Jackiw wanted a motion on segment *l* to percolate to *A* and *C*, but the directionality of the arrows did not allow it. Dispensing with the arrows altogether was also troublesome, as an action on a node would then propagate throughout the graph without a clear idea of where to end.

After thinking through these issues, Jackiw devised a variation on the acyclic graph that gave him the

desired behavior. He viewed the graph as a tree whose roots and dependencies could change:

> Imagine this computational tree was composed out of thread and objects. If I picked up *C*, everything would hang from it, and I could make *C* become essentially the new root of the tree. And that's the algorithm Sketchpad uses, which is very different than simply saying, 'Here's the graph implied by the chronology of how the user put things together, and I'm always going to follow the arrows in the same direction he did.'

Sketchpad does not use this algorithm in every situation, but the notion of being able to move any component of a figure is a defining feature of the program.

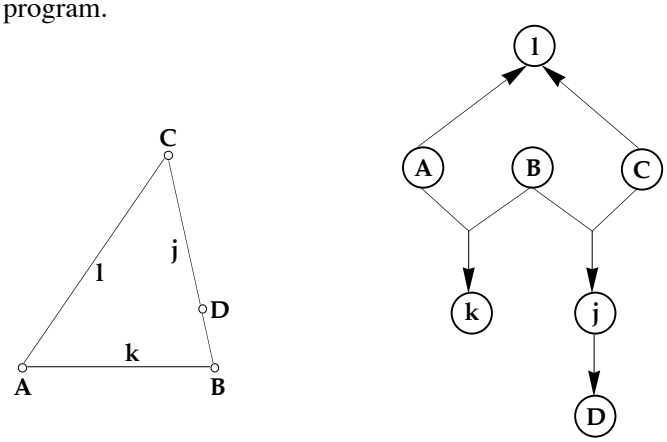

Figure 4. Δ*ABC* with its acyclic graph representation.

Conclusion

This overview of Sketchpad's evolution, from the origins of dragging to the nature of its menu items, suggests there was very little about the program that was inevitable from the start. The underlying mathematics of geometry and algebra certainly guided its development, but did not dictate how features would operate, nor even which features to include. Jackiw emphasizes this point when characterizing the overall nature of the program:

> The totality of [my] decisions forms, to my mind, more of an aesthetic entity than a mathematical one. We should not lose sight of that. Part of the reason students respond so well to these environments has nothing to do with the [software's] mathematics; it has to do with the functional, balanced appeal of their industrial design.

Because Sketchpad's design features invite exploration and play, users sense their own role in shaping and crafting their understanding of mathematics. Ultimately, they come to see mathematics less as a collection of rules and procedures and more as an ongoing human endeavor.

References

Franklin Institute Online (2000). *Dr. Ivan E. Sutherland*. [On-line]. Available: http://sln.fi.edu/tfi/exhibits/sutherland.html.

Jackiw, N. (1995). *The Geometer's Sketchpad*. [Computer software]. Emeryville, CA: Key Curriculum Press.

Laborde, C. (1994). Enseigner la géométrie: Permanences et révolutions. In C. Gaulin, B. Hodgson, D. Wheeler, & J. Egsgard (Eds.), *Procedings of the 7th International Congress on Mathematical Education* (pp. 47-75). Les Presses de L'Université Laval, Sainte-Foy, PQ, Canada.

Logal Software (1994). *The Geometry Inventor*. [Computer software]. Cambridge, MA.

Syer, H. W. (1945). Making and using motion pictures for the teaching of mathematics. In W. D. Reeve (Ed.), *Multi-sensory aids in the teaching of mathematics* (pp. 325-345). New York, NY: Bureau of Publications, Teachers College.

Texas Instruments (1994). *Cabri Geometry II*. [Computer software]. Dallas, TX.

Notes

[1]This work was supported by the National Science Foundation, grants RED-9453864 and MDR-9252952, and the National Center for Research in Mathematical Sciences Education. Thanks to Nicholas Jackiw, Eugene Klotz, and Jean-Marie Laborde for the interview sessions that contributed to the information in this article. Special thanks to E. Paul Goldenberg who provided invaluable suggestions throughout the writing process. The opinions expressed here are not necessarily those shared by the NSF.

[2] Researched at Xerox PARC and implemented ("for the rest of us") on the Apple Macintosh.

[3] Shortly after the two projects learned of each other, the programmers began collaborating electronically, testing each other's code, offering interface design feedback, and reporting bugs and their potential fixes.