# Visualizing Complex Functions with The Geometer's Sketchpad

Nicholas Jackiw
KCP Technologies, Inc.
njackiw@keypress.com

To appear October 2003 in: Triandafillidis, T. and Hatzikiriakou, K. Eds. *Proceedings of the 6th International Conference on Technology in Mathematics Teaching*. 291-299 (Volos, Greece: University of Thessaly.)

## Abstract

This paper reviews the possibilities for modeling complex numbers in dynamic geometry software environments, and suggests that as such tools evolve, despite not being specifically adapted to complex analysis, their growing capabilities to support a user's conceptual refocusing on "arbitrary visual mathematics" makes them a compelling tool to bring to the introduction to, and analysis of, functions in the complex plane. The particular software used throughout is *The Geometer's Sketchpad version 4*.

## Dynamic Geometry and Complex Numbers

The greatest single leap forward in the history and development of the complex numbers comes some 250 years after Cardano first proposes them, in the near-simultaneous realizations of Argand, Gauss, and Wessel that a complex number can be interpreted as, and represented by, a geometric point in two dimensions. This simple correspondence introduces not only the ability to visualize the complex numbers—or, now, the complex plane—in a simple and straightward sense, but to bring geometric techniques to the interpretation and analysis of the mathematics of that complex plane. With this deft movement, the connotation of the imagination in "imaginary numbers" shifts quickly from imagining their gross mathematical absurdity to imagining their great mathematical potential.

Over the past decade, interactive geometry software environments such as *The Geometer's Sketchpad* (Jackiw, 1991) and *Cabri Geometry II* (Laborde & Bellemain, 1992) have rapidly come to play large roles in contemporary planar geometric visualization and exploration, both within school and undergraduate education, but also—though to a lesser degree—within mathematical research. This paper's purpose is to explore the potential for using these software tools to investigate and model complex numbers and their operations, and to visualize functions defined on the complex plane.

Of course, the present work is not the first Dynamic Geometry perspective on complex numbers. One of the activities to which software tools like Sketchpad were first applied in the early 1990's was geometric modeling, and thus the geometric interpretation of complex values was quickly explored by early enthusiasts of the tools. (Lee Dickey with Cabri, and Susan Addington and David Dennis with Sketchpad, are among the early pioneers of such investigations.) Since the fundamental workspace of Sketchpad or Cabri is (an electronic simulation of) the unbounded geometric plane, these programs at their debut in some sense *already* offered effective models of the complex numbers.

What the tools lack at their most basic level, however, are pre-defined operations on complex values—complex arithmetic, in other words, and perhaps complex integration and differentiation. But within the constructionist paradigm of dynamic geometry software, these were fit operations to be defined by an inquisitive user, who might then not only build models of complex expressions, but models of the operators used within those expressions.

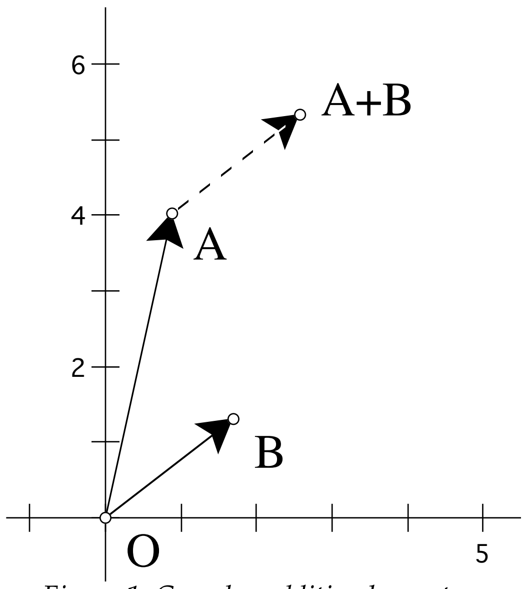


*Figure 1. Complex addition by vectors*

Thus—for example—the model of complex addition visualized in Figure 1, in which *A*, *B*, and *A+B* are complex numbers, and the dashed vector represents the image of *B* translated by vector *A*. Vector addition produces the sum *A+B*. If the horizontal axis is declared to be real and the vertical axis imaginary, we have computed the complex sum of *A* and *B*.

Though in the illustration at left, *A* and *B* obviously have specific locations and thus fixed complex values, within dynamic geometry software points can be dragged to new locations. Thus dragging *A* and *B* generalizes the constructed operation to the addition of *any* pair of complex numbers.

While such constructions lead to effective demonstrations of the dynamic behavior of basic operations, the act of creating them is not in all cases necessarily revealing to a student first grappling with the ideas of complex arithmetic. The emphasis on such activity is more on the Euclidean operations that must be performed to accurately model a—presumably given—definition of the complex operation, rather than on that operation itself. Such constructions may be a useful exercise in modeling, but they focus more on the model than on the modeled domain.

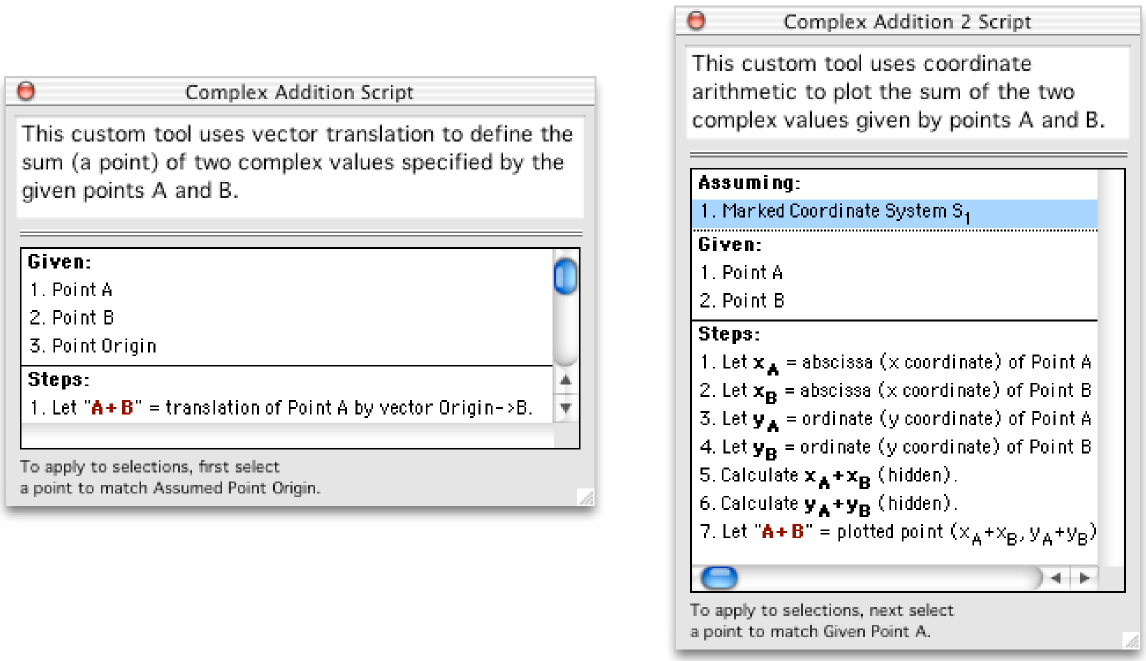
## Building Microworlds for Complex Numbers

Fortunately, these laborious constructions can be abstracted into new fundamental tools within the software. Beyond geometric operations and mathematical definitions, dynamic geometry software has, since its first conception, offered sophisticated facilities for virtualizing composite constructions into new conceptual units, that can then be applied atomically as "extended primitives" of the software. This idea is found in Sketchpad scripts and custom tools, or in Cabri macros. It reflects a deeper tradition stemming from earlier didactic geometry technologies, including the "Repeat" mechanism of *The Geometric Supposer* software of the mid-1980's, and of course the idea of subroutines in Logo and other programming languages. Among other things, these technologies permit a teacher or curriculum developer to provide a pre-made dynamic geometry environment that has been extended to include tools for a particular topic or area of investigation.[1] We use them here to develop basic microworlds for complex arithmetic and function visualization.

In version 4 of Sketchpad, such extensions take the form of *custom tools*, devices that define new objects (much as a compass or straightedge defines new objects) and that appear—after they have been created—in the software's basic toolbox along with its

[1]See Jackiw, 1997, for more on custom microworld design within dynamic geometry environments.

fundamental tools (such as the electronic compass and straightedge). The most straightforward way to define such a custom tool is by example: having built an appropriate construction within the software—such as Figure 1—a user can identify the portion (or totality) of the construction to be repeated, and then to **Create A New Tool**. Sketchpad responds by abstracting the operation away from its particular geometric (or coordinatized) form, and then summarizing it as a symbolic procedure. This procedure describes the function of a new tool added to the toolbox. Custom tools may be simple or sophisticated, and may draw on geometric constructions, planar transformations, or arithmetical or algebraic derivations to define the objects they produce. Figure 2 shows the procedural view of two equivalent custom tools that provide definitions of complex addition, one based on geometric translation, the other based on coordinate arithmetic.



Complex Addition Script

This custom tool uses vector translation to define the sum (a point) of two complex values specified by the given points A and B.

**Given:**
1. Point A
2. Point B
3. Point Origin

**Steps:**
1. Let "A+B" = translation of Point A by vector Origin->B.

To apply to selections, first select a point to match Assumed Point Origin.

Complex Addition 2 Script

This custom tool uses coordinate arithmetic to plot the sum of the two complex values given by points A and B.

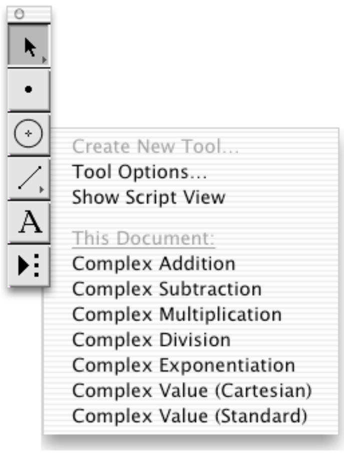
**Assuming:**
1. Marked Coordinate System $S_1$

**Given:**
1. Point A
2. Point B

**Steps:**
1. Let $x_A$ = abscissa (x coordinate) of Point A
2. Let $x_B$ = abscissa (x coordinate) of Point B
3. Let $y_A$ = ordinate (y coordinate) of Point A
4. Let $y_B$ = ordinate (y coordinate) of Point B
5. Calculate $x_A+x_B$ (hidden).
6. Calculate $y_A+y_B$ (hidden).
7. Let "A+B" = plotted point $(x_A+x_B, y_A+y_B)$

To apply to selections, next select a point to match Given Point A.

*Figure 2. Two addition tools, with separate derivations but equivalent results*



While such verbal descriptions of a custom tool's procedure can be inspected, or edited, by users, once the tool has been defined, there is no requirement that the verbal procedure ever be referenced again. Instead, users can choose the tools by name from the main toolbox, and wield them directly in the sketch. (In the case of the tools in Figure 2, clicking a "Complex Addition" tool on any two points will produce a third point, their sum.)

Figure 3 shows a toolbox of basic complex arithmetic tools, as well as two notational tools.[2] (Clicking the Cartesian or Standard value tools on a geometric point produces a textual notation of its complex value, in one of the two notations illustrated in Figure 4.)

[2]The toolbox illustrated here is contained in one of the example documents that come with the default installation of Sketchpad. See samples/sketches/advanced/complex.gsp.

of the two notations illustrated in Figure 4.)

*Figure 3. Custom toolbox for complex arithmetic*

The benefit of having a coherent toolbox of related tools present at the start of a student investigation or activity, rather than only as the object or endpoint of an activity, is dual. First, clearly, the focus and content of that activity can be on the behavior or properties of the phenomena being modeled by those tools, rather than on the mathematics of the modeling act. Second, since these tools encapsulate mathematical constructions that are at times themselves sophisticated, students combining them can work at a much higher level of mathematical sophistication without requiring a correspondingly much higher level of technical sophistication (with the software) or conceptual sophistication (with the entire network of relationships from which the higher-level construction's behavior emerges from the combined effect of the behavior of each of its components' procedural subconstructions).

As students work, they can extend the toolbox with additional tools that they themselves find useful—and that, presumably, they themselves create. This process may repeat itself at several layers of abstraction over the course of an activity or an exploration, as students apply existing tools to define new ones at progressive levels of sophistication or abstraction.

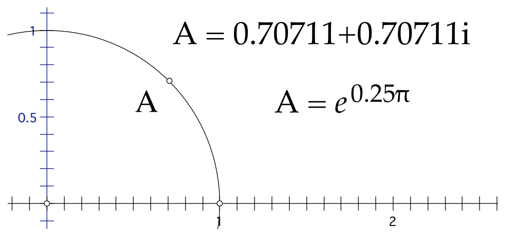


*Figure 4. A complex value (point A, left) described in Cartesian (upper) and Standard (lower) notation*

## Graphing Complex Curves As Loci

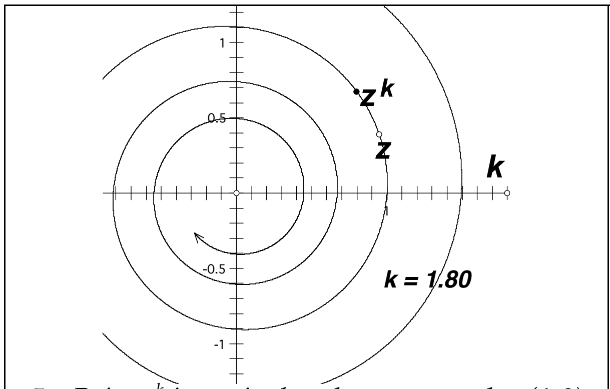
Once we have access to a toolbox, we can begin modeling in earnest. Though the toolbox only gives us single operations that are defined on points (for example: addition, defined to map two addend points to a third sum point), two general operations allow us to extend these productively:

<u>Concatenation of Operations.</u> By using different tools sequentially, we can build more detailed expressions out of primitive operations; or (the reverse) we can decompose tangled expressions into simple sequences of primitives. This is straightforward "order-of-operations"-type thinking. To construct $(2z-A)^2/B$ from points $z$, $A$, and $B$, we might first use the addition tool to double $z$, then the subtraction tool (for $2z-A$), then a multiplication—$(2z-A)^2$—and finally a division (by $B$). The tools of the Complex Arithmetic toolbox label their produced points with an appropriate symbolic expression stated in terms of the labels of the input points (as in Figure 1). These labels—combined with judicious hiding of intermediate results when they are no longer necessary—help track the arithmetic interpretation of the growing total expression, as we develop its parts with individual tools.

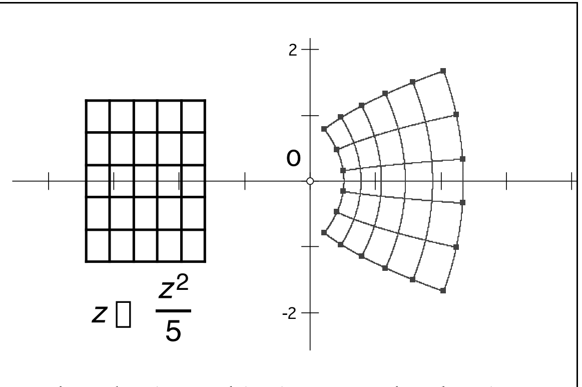
<u>Graphing by Locus Construction</u>. While so far we have worked only with points, the **Locus** command allows us to visualize—or "image"—any planar curve or curves under a complex transformation, by constructing the locus of a transformed point as a function of the motion of its pre-image point traveling along the pre-image curve. This is a straightforward application of the locus interpretation of graphing used on the Cartesian plane to first conceptualize graphs of real functions: if a given pre-image point $x$ defines, through its position on the horizontal axis, some image point $(x, f(x))$, then the locus of the image point as the pre-image travels the horizontal axis is the total graph $y = f(x)$. Here, if a given pre-image point $z$ defines an image $z'$ under some transformation, then

the locus of z′ as *z* travels some curve *s* is the image *s′* of that curve. With this technique, Sketchpad's Locus command can construct the image of segments, rays, lines, circles, polygon perimeters, arcs and their segments and sectors, and finally, function plots and other loci.

These two operations combine to provide a general-purpose complex graphing facility, as demonstrated in figure 5. In the left illustration, two draggable points—*z* and *k*—define a third point $z^k$. The spiral graph—of *z* raised to all real powers—is the locus of $z^k$ as *k* travels the real axis. Dragging point *k* in the completed diagram simply slides $z^k$ along the spiral; dragging point *z* (which is free in the plane) expands and contracts the spiral's curvature. This example, as well as that of Figure 5b, also demonstrates the conventional solution to what might be called "the problem of complex graphing." When graphing [illegible] functions on a Cartesian coordinate system, the one-dimensional output range—*f*(*x*)—is projected perpendicular to the one-dimensional input domain *x*, creating a two-dimensional graph. But with complex functions, both input and output values are already two-dimensional (real and imaginary), and thus *four* dimensions are required to visualize the result. The approach commonly taken for textbook illustrations is to graph only some subset of the input plane, and to superimpose its output on the same coordinate system. Usually these examples adopt the language of transformations (a *transformed image* is superimposed on its own *pre-image*), and a frequent choice for pre-image is some sort of checkerboard or lattice, since its abundant interior right angles allow one to quickly inspect conformality. In Sketchpad, the pre-image lattice in Figure 5b is constructed with Euclidean segments. Complex arithmetic tools produce a point $z'=z^2/5$ given some independent point *z*. (These points are no longer visible in the illustration.) The image lattice is then constructed curved segment by curved segment, where each curved segment is the locus of point *z′* as point *z* travels the corresponding straight segment in the pre-image lattice.



*5a. Point $z^k$ is z raised to the present value (1.8) of k; the spiral depicts z raised to all real powers*



*5b. A lattice and its image under the given transformation*

It is only at this point—that is, only once these conceptual and technological tools are established—that applying a dynamic geometry environment to complex functions becomes uniquely interesting. Other mathematics packages can compute complex values and graph complex functions with less conceptual overhead than Sketchpad, but they lack dynamic geometry's signature ability to experiment with a configuration interactively by dragging its components. Dragging and—to a lesser degree—animation allow one not only to visualize these systems across change or parametric evolution, but permit one to following Needham's (1997) seminal work in bringing a kinematic, Newton-like approach of "geometric calculus" to complex analysis and visualization.

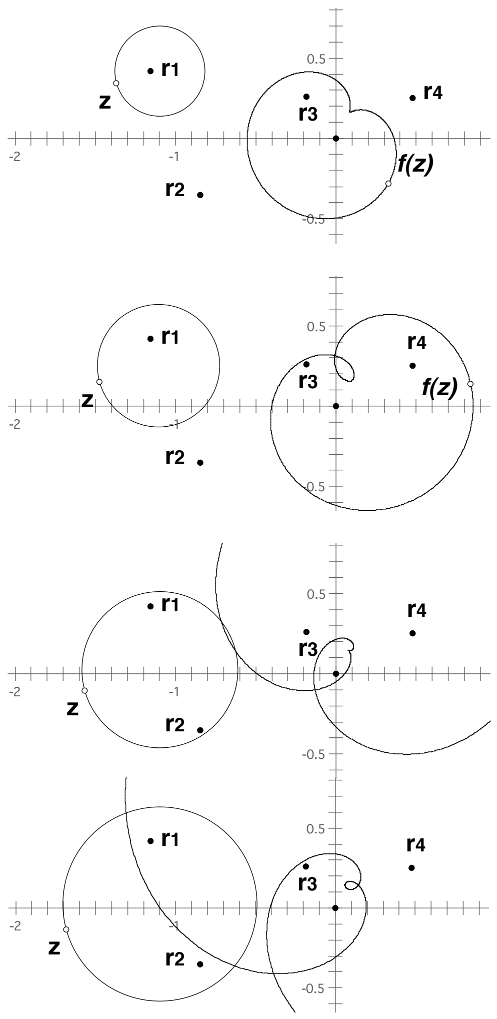


*Figure 6. Winding Around Roots*

Figure 6 suggests some of the visual dynamics that become accessible in the dragging environment. Each picture shows a circle being dragged across the plane, and the image of that circle under the function

$$f(z) = (z - r_1)\ (z - r_2)\ (z - r_3)\ (z - r_4)$$

As usual, the image was constructed by first calculating—using the complex arithmetic tools—the function's value $f(z)$ for one independent point $z$, and then constructing the locus of $f(z)$ as $z$ defined (or traveled around) the pre-image circle.

Over the four frames (read from top to bottom), the user drags and expands the circle to encircle roots of the function. In the first frame, only $r_1$ is "captured." Correspondingly, the circle's image winds around the origin, since the origin (zero) is the image of the root. But as the circle grows (frame 2) to capture a second root ($r_2$ in frame 3), an inner loop of the image snakes out to encircle the origin a second time. In the final frame, the circle expands toward $r_3$, and we see a new loop forming to wind a third time around the origin. The winding number of the image determines the number of roots (counted in their algebraic multiplicity) encircled by the pre-image.

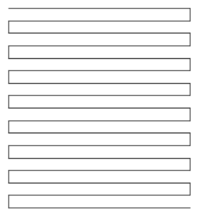
As is usually the case with descriptions of dynamic geometry, actually dragging these configurations is much more compelling than observing static pictures!

## Graphing Planar Transformations As Surface Plots

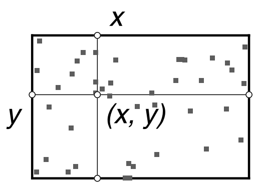
Despite the appeal of dragging, Figure 6 reveals one of the shortcomings of the conventional solution to the "problem of complex graphing." When both pre-image (the circle) and image (the origin-winding quasi-cardioid) are super-imposed on the same coordinate system, legibility suffers. In more involved visualizations, it becomes increasingly difficult to distinguish input from output, "cause" from "effect." More generally, the conventional approach struggles between opposing imperatives. On the one hand, we want our image and pre-image to be as "large" as possible, since we are interested in gaining a sense of the effect of our complex transformations across the entire plane, not just on a single point or point locality. On the other hand, we are interested in keeping our image and pre-image small, to avoid overlap. Physically isolating the image into a separate visualization from the pre-image (a separate illustration of the plane) is one approach, but replaces the immediacy of a unified visualization with a more abstract representation requiring additional interpretation to reconcile conceptually. Thus we are encouraged to move beyond the conventional solution, and to explore techniques for visualizing the effect of a complex transformation *over* the entire plane, *on* the entire plane.

To proceed in this direction, we need first establish techniques for plotting surfaces, and since we are in a dynamic geometry environment, we must derive these from mathematical principles and primitives rather than look to built-in software commands. To plot a surface ideally, one needs first to be able to visit every point on the plane. Practically, we seek to visit some regular subset of points in a planar region. Once we can visit such points, we can evaluate a condition or function there, to determine the value of the surface at or above that planar point. But traditional loci or function plots in *Sketchpad* and loci in *Cabri* are one-dimensional curves, rather than two-dimensional areas. So this basic requirement—a tour of some planar region—is sufficiently challenging from the perspective of compass-and-straightedge constructions, or even from the much richer set of primitives that characterize a dynamic geometry environment, that it's worth struggling with, with students. Workable solutions draw on a variety of mathematical approaches and governing metaphors:

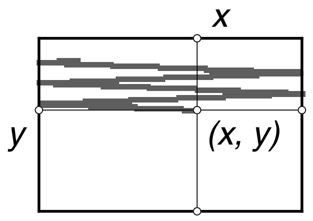
**Space Filling Curves**. Can a one-dimensional curve be wrapped in such a way that it fills a two-dimensional area? If so, sampling points on the curve—that is, taking a locus of a point traveling that curve—is equivalent to sampling a planar region. As well as wrapping configurations, one can imagine torn-and-reassembled configurations, as when one tears a (near) one-dimensional roll of wallpaper into strips that, reassembled side-by-side, cover a two-dimensional wall.



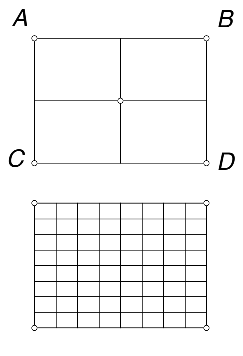
**Random Motion.** At right, *x* and *y* are points on their respective segments; *(x, y)* is their projection. Randomly varying *x* and *y* along their segments causes *(x, y)* to sample randomly the rectangular region.



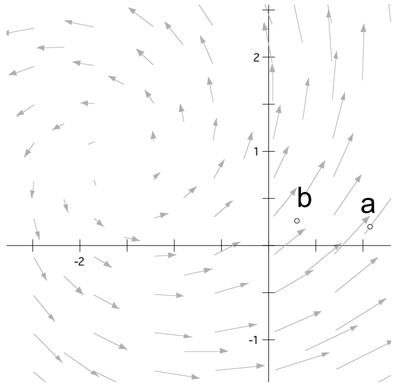
**Synchronized Motions**. If *x* and *y* tour their segments at different rates, their projection scans the rectangle in much the way a cathode ray "scans" the phosphor screen of a conventional television monitor. Different speeds can be achieved with Sketchpad's animation, or the velocity of each particle can be constructed as a separate parameterization of time.



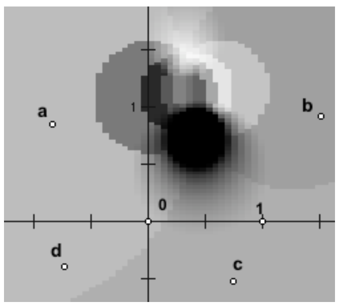
**Iterative Decomposition**. In this approach, the quadrilateral ABCD is decomposed (via Sketchpad's **Iterate** command) into its four corner quadrilaterals. Repeating this decomposition recursively a few times generates progressively smaller-meshed "screens" covering the original quadrilateral area (with 4, 16, 64, 256, 1024… interior samples). The advantage of this approach is its variability of resolution—one can work at low resolution when building a draft model for speed and convenience, and then increase the depth of iteration to improve the sampling detail of a final visualization.



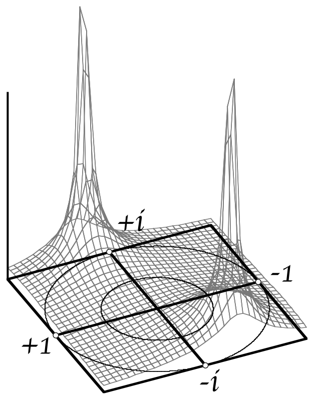
Once a technique has been established for sampling points on the plane—that is, for visiting a region of complex values—surface plots of complex functions or transformations can be built by evaluating the function at each sampled point, and characterizing the surface at that point by the resulting value. The particular geometric interpretation of that value can vary richly, as the visualizations of Figure 8 attempt to suggest. In 8a, an arrow connects each sampled point *z* with its image *f*(*z*)—quite literally, $z \mapsto f(z)$. 8b adopts a planar coloring scheme proposed by Frank Farris (1998) in his MAA review of Needham (1997). In this approach, a standard colorization is given to the complex plane, in which the hue of some point *z* corresponds to its argument (its angle with respect to the positive real axis) and the shade value or tonal density ranges with *z*'s modulus (distance from the origin), from white at the origin to black at infinity. Given this or any other distinctive coloration of the plane, a Farris plot of the complex function *f* involves recoloring the plane so that each point *z* has the color assigned to point *f*(*z*) in the canonical coloration. These vivid plots readily reveal algebraic multiplicities through multiply-spiraling hue configurations, and the roots and poles of *f* appear as pure white and pure black locations. Finally, 8c depicts perhaps the most literal interpretation of "surface plotting." Here the height of the visual surface corresponds to the modulus of the given function. These modular surfaces reach down to touch the plane at their roots, and explode upwards toward infinity at their poles.



*a. Vector field of $f(z) = az+b$*



*b. Farris plot of the general Möbius transformation $z \mapsto (az+b)/(cz+d)$*



*c. Modular surface of $f(z) = a/(b+z^2)$*

*Figure 8. Visualizations based on planar sampling*

## Conclusion

The possibilities for visualizing complex functions and results in complex analysis with dynamic geometry software appear numerous. What environments such as Sketchpad lack in native facility for working with complex numeric data types and visualization capabilities, they compensate for with interesting didactic trajectories through the geometric interpretation of complex numbers; the mechanics of user-constructed graphs; and the potential for interactive and direct manipulation of the resulting, dynamically-generalized visualizations.

## References

Farris, Frank. (1998) *Complex Colorations*. MAA Online: http://www.maa.org/pubs/amm_complements/complex.html

Jackiw, Nicholas (1991, 2001). *The Geometer's Sketchpad* computer software. Emeryville: Key Curriculum Press.

Jackiw, Nicholas (1997). "Drawing Worlds: Scripted Exploration Environments in The Geometer's Sketchpad" in *Geometry Turned On!: Dynamic Software in Learning,* *Teaching, and Research*, eds. James R. King and Doris Schattschneider (Washington, D.C.: The Mathematical Association of America): 179-184.

Laborde, J.-M. and Bellemain, F. (1992). *Cabri Geometry II* computer software. LSD2-IMAG Grenoble and Texas Instruments.

Needham, Tristan (1997). *Visual Complex Analysis.* Oxford: Oxford University Press.