The Immersive Visualization Probe for Exploring *n*-Dimensional Spaces

James R. Miller and Estela A. Gavosto
University of Kansas

A new tool uses two types of dimensional navigation to display continuous 4D subsets of *n*-dimensional data. Thanks to the tool's embedded coordinate systems, researchers can better understand a model's underlying physical or mathematical process.

An *n*-dimensional space, where *n* is larger than 3, naturally describes many interesting phenomena studied by scientists and mathematicians. Exploiting visualization to understand these systems' structure is an active area of research. In this article, we describe a new method for visualizing data structure or models defined in higher-dimensional spaces. This technique is suitable for applications in which a scalar function, defined mathematically or procedurally, depends on *n* variables or parameters. Scientists must understand this function's structure, or procedural model, thereby gaining insight into the underlying physical or mathematical system. The function's values essentially describe a set of points in *n*-dimensional space. To visualize these sets, we fix all but three of the parameters and then sample the resulting 4D set (the three parameters and the function's resulting value) on several discrete grids located on planes in the 3D space. We compose the image by using color to represent the fourth dimension (the function's value) at discrete locations on these grids. Interactive control over the way the parameters are fixed results in a highly dynamic system that researchers can easily use to explore the *n*-dimensional space's structure.

An alternative approach would be to fix all but two parameters and then display a simple surface in 3D. However, for the data sets of interest to us, fixing all but three works better for two main reasons. First, the surfaces we encounter are by no means simple. They tend to be highly irregular—even chaotic; hence, some sort of discrete sampling is necessary anyway. Second, the model definitions are continuous throughout the *n*-dimensional space (and hence continuous throughout all lower-dimensional subsets). A 3D surface representation doesn't convey this situation nearly as well as our room, which can give the impression of a solid block of material carved out of the surrounding area.

The current prototype of our immersive visualization probe (IVP) uses two dynamically linked types of navigational control. First, the user manipulates visual representations of a set of embedded coordinate systems to define how to fix the values of the $n - 4$ variables. (This navigational tool's design was inspired by the *n*-Vision system that Feiner and Beshers described.[1,2]) Once the user establishes values for the variables, we employ a set of discrete sampling grids to view the resulting 4D data subset. Hence, with the second type of control, we give the user the illusion of being inside a room, where the walls define the locations of the discrete grids. By studying the wall displays, the user can determine potentially interesting directions to explore in the space and then move the room (and hence the discrete sampling grids) in that direction.

An important design constraint was the need to be able to explore these multidimensional data sets in real time. It's important to let users control model parameters and other data selection operations with an interactive device, while the resulting visualization changes continuously in response. This goal was a primary consideration when designing the type of visualization as well as the interactive viewing and data set traversal operations supported.

This work demonstrates that the IVP probe is effective for displaying continuous 4D subsets of *n*-dimensional data, shows how users can interactively adjust the hyperplanes to slice down through several dimensions, and illustrates how the combination of an embedded coordinate system and the resulting 4D slice representation contributes to an understanding of the underlying physical or mathematical process involved. For both computational and visual reasons, we chose sampling on a set of planes rather than a volumetric display for the final 4D slice. Computation times for volumetric displays were orders of magnitude beyond what was needed to achieve interactive exploration, and the resulting displays were difficult to comprehend.

Published by the IEEE Computer Society

We use two different systems to illustrate the use of our method. First, we study the filled-in Julia set in two complex variables. We can view this set as a 9D object that defines an escape rate as a function of eight parameters. Next, we visualize the stability of orbital trajectories of satellites launched between two planetary objects. We use a simple model, essentially a 10D object in which the satellite's trajectory fits into one of several categories. The assignment is based on the values of nine parameters describing such quantities as the satellite's initial position and velocity and the planetary objects' masses. We model the former using an explicit closed-form iterated function system, and we describe the latter using a numerical simulation.

The only operation that the IVP requires internally to support a specific model or data set is the ability to compute a functional value for a point in an n-dimensional space. The IVP engine then maps this value to a color through a user-defined function mapping to encode the fourth dimension. The "Related Work" sidebar compares our approach to other recent approaches.

System architecture and operation

We developed the IVP in C++ using OpenGL. We initially developed a single-threaded prototype version displaying only the IVP's interior on a Windows-based PC. Our current production IVP is multithreaded and employs the embedded coordinate-system display and manipulation mechanisms. This version runs an SGI workstation, and it is the version we used to produce the images in this article. In addition to conventional workstations in the lab, the system was particularly effective in the Collaborative Visualization Room (CVR) at the University of Kansas. This room houses a (25×6)-foot, wall-sized display, driven by three overhead projectors; and a six-processor SGI Origin 2000 server, with three pipes and InfiniteReality2 graphics. Not surprisingly, this environment provided the most effective sense of immersion.

We divided the work of generating the wall textures across an arbitrary number of threads. Running in the CVR on the Origin server, we typically allocated one thread per wall and achieved the ability to move coordinate systems, watching the texture on the wall change continuously (although we had to move a coordinate system somewhat slowly for the Origin to keep up).

We used two different strategies to improve interactivity. While dragging a selected coordinate system, we generated low-resolution wall textures so that the response was relatively fast, thereby letting us move more rapidly through the space. When an input manipulation ended, we regenerated the wall textures at the higher resolution.

Related Work

Viewing n-dimensional data requires performing a series of slices or projections to arrive at a 3D set that is viewable through traditional schemes. Taking successive slices involves fixing the values of corresponding independent variables. A more information-rich 3D display is possible by preserving a visual representation of the slices performed. Feiner and Beshers first introduced the notion of using embedded coordinate systems for this purpose.[1,2] In their n-Vision system, they place coordinate system β at some point (a, b, c) in coordinate system α. This placement signifies that the three independent variables corresponding to the axes in α are fixed at a, b, and c. If the resulting data set's dimension is no greater than 3, it can be directly displayed. Otherwise, the nesting of coordinate systems can be repeated—for example, coordinate system γ is placed at (d, e, f) in β, and so forth—until such a displayable data set is obtained. The visual display of the various nested coordinate systems then illustrates which independent variables were fixed and the values to which they were set to arrive at the final displayed slice. The first stage of our visualization scheme stems from this approach.

Another interesting scheme for visualizing high-dimensional data and the effects of various fixed independent variables employs the display of hierarchically nested axes.[3] This scheme samples an independent variable's successive values across a fixed range. For each sampled value, the scheme analogously samples another independent variable. The scheme scales down the axis displays and stacks them to make the relationship between the independent variables visually apparent. This approach seems best suited for functions having fairly smooth derivatives. For models and data sets, such as ours, that are not smooth, the approach described in the previous paragraph seems better.

After fixing sufficient independent variables to obtain a data set that is displayable using three spatial dimensions and color, we need a visualization tool to make this data set viewable. In our applications, this set has a complex structure, and is in fact frequently a fractal or an object with a chaotic boundary. There has been considerable research regarding the display of such objects.

Norton introduced the notion of *boundary tracking* for fractal surfaces.[4] The approach rests on the ability to classify points as being inside or outside of a fractal; the decision depends on an escape-time characterization similar to that described in our Henon map example in the main text. Norton defines a regular 3D grid and classifies the points in the grid. A point is a *boundary point* if it is in the interior of the fractal and has at least one neighbor in the grid that is outside the fractal. His algorithm then shades each point using a z-buffer algorithm from the viewpoint of an assumed light source, and then renders the points directly into the frame buffer using again a z-buffer algorithm, this time from the viewpoint of an assumed observer. The boundary-tracking process begins with one or more *seed points* determined to be on the boundary. This method uses grid adjacency information to traverse the space, looking for adjacent boundary points. To ensure that the algorithm eventually terminates, it's necessary to keep track of all boundary points previously discovered, leading to large storage requirements for reasonably sized grids.

Researchers have examined variations to this approach.

continued on p. 78

continued from p. 77

For example, Wyvill et al. generate a polygonal approximation instead of simply projecting points.[5] This makes conceptually easier the computation of surface normal approximations and avoids the need to project individual points onto several pixels as in Norton's method. Zahlten employs a *chain-of-cubes* algorithm while generating a surface polygonal approximation.[6] An important contribution of this approach is that it requires less storage than Norton's boundary-tracking algorithm.

Others have used ray tracing to render fractals. In any ray-tracing algorithm, the ray-surface intersection operation is the performance bottleneck. Using so-called unbounding volumes,[7] Hart et al. describe an efficient ray-fractal intersection algorithm that produces high-quality images fairly quickly. Because ray tracing produces a frame buffer image, storage requirements are negligible in their approach, thereby overcoming a major problem with algorithms based on boundary tracking. We've experimented with ray tracing and other visualization approaches for the Henon map as well.[8,9]

We can roughly characterize the tradeoffs in the two major rendering schemes as follows. Boundary-based algorithms create an approximate boundary at a given resolution and after a nontrivial (and noninteractive) consumption of time and storage resources. Users can then rotate and view this boundary representation at interactive rates on conventional graphics workstations. Thus, although users can dynamically rotate the resulting fractal image, the major difficulties are storage requirements and the inability to see additional detail when zooming.

By contrast, ray-tracing algorithms use negligible storage and generate detail adequate for a given field of view. However, they require considerable computation (even one like that described by Hart et al.,[7] which is reasonably fast by ray-tracing standards); hence, they aren't suitable for the real-time explorations we hope to facilitate.

We're attempting to bridge the detail/interactivity chasm by providing high resolution in certain areas and low resolution in others, and by trying to optimize the generation to get as close to interactive rates as possible for the entire process. The problem is that fractal boundaries have infinite detail, and any approximation by geometric primitives whose projection covers more than one pixel risks making the image appear smoother than the surface actually is. Such inappropriate smoothness can also stem from the use of normal vectors in lighting models if the computation and use of the normal vectors is either too simplistic or involves artificial Gouraud-like shading interpolation. We don't attempt to shade the function surfaces themselves, but rather just the planes on which the sampling grids are defined.

But, clearly, it isn't possible to render the entire surface at pixel resolution and achieve real-time rendering speeds, let alone analyze every point in a volumetric region at a comparable pixel-sized resolution. Our approach is based on the premise that we can render at high resolutions in some areas and provide low-resolution visual cues in between the areas of high resolution. By structuring the placement of high-resolution data into a familiar 3D shape, we can produce visualizations that provide an alternative way to understand complex model structure.

References

1. S. Feiner and C. Beshers, "Worlds within Worlds: Metaphors for Exploring *n*-Dimensional Virtual Worlds," *Proc. ACM Symp. User Interface and Software Technology* (UIST 90), ACM Press, 1990, pp. 76-83.
2. S. Feiner and C. Beshers, "Visualizing *n*-Dimensional Virtual Worlds with *n*-Vision," *Computer Graphics* (*Proc. Siggraph*, ACM Press), vol. 24, no. 2, Mar. 1990, pp. 37-38.
3. T. Mihalisin, J. Timlin, and J. Schwegler, "Visualizing Multivariate Functions, Data, and Distributions," *IEEE Computer Graphics and Applications*, vol. 11, no. 3, May 1991, pp. 28-35.
4. A. Norton, "Generation and Display of Geometric Fractals in 3-D," *Computer Graphics* (*Proc. Siggraph*, ACM Press), vol. 16, no. 3, 1982, pp. 61-67.
5. G. Wyvill, C. McPheeters, and B. Wyvill, "Data Structures for Soft Objects," *Visual Computer*, vol. 2, no. 4, 1986, pp. 227-234.
6. C. Zahlten, "Boundary Tracking of Complicated Surfaces with Applications to 3-D Julia Sets," *Fractal Geometry and Computer Graphics*, J.L. Encarnação et al., eds., Springer-Verlag, 1992, pp. 103-110.
7. J.C. Hart, D.J. Sandin, and L.H. Kauffman, "Ray Tracing Deterministic Fractals," *Computer Graphics* (*Proc. Siggraph*, ACM Press), vol. 23, no. 3, 1989, pp. 289-296.
8. A. Johnson, *Ray Tracing the Complex Henon Map*, tech. report, Dept. of Mathematics, Univ. of Kansas, 1998.
9. L. Cao, *Coexisting Attracting Basins in Complex Holomorphic Dynamics*, tech. report, Dept. of Mathematics, Univ. of Kansas, 1999.

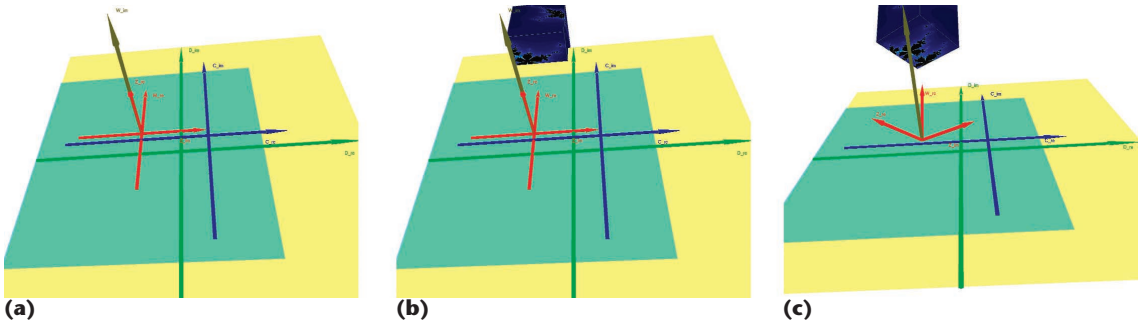
We also developed a distributed version of the system that uses another SGI Origin 2000 with 64 processors. The main visualization runs in the CVR and uses sockets and the common object request broker architecture (Corba) to communicate with a computation server running on the remote Origin 2000. When running on this server, the program is highly multithreaded and can feed the display engine in the CVR at far higher frame rates.[3]

Using the IVP-based system

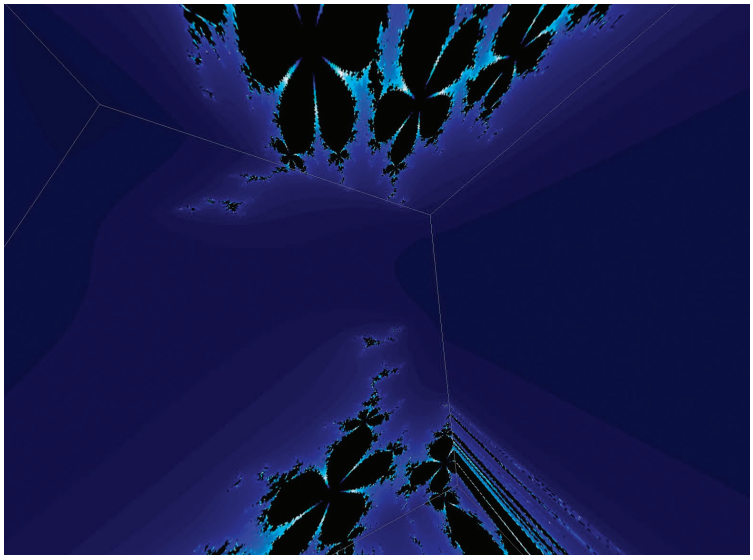
Navigating and visualizing *n*-dimensional data sets using our IVP-based system involves three interrelated steps:

1. specifying and visualizing the $n - 4$ slices taken through the original *n*-dimensional data set to obtain a displayable 4D subset,
2. visualizing the 4D subset obtained, and
3. interactively exploring the *n*-dimensional set through subsequent interactive adjustments of the slicing planes.

Our primary focus here is to illustrate how to view 4D data sets with complex structures using our IVP probe and auxiliary display tools (step 2), and how to transform Feiner and Beshers' nested-coordinate-system visualization technique into an interactive exploratory tool to



1 (a) Nested coordinate-system display. (b) Visualizing the 3D slice. (c) Rotating the 3D slice.



2 View inside the IVP.

allow real-time generation of these 4D subsets of an n-dimensional data set (step 3).

Initially specifying and displaying a 4D subset

Several parameters determine the characteristics of the systems we're studying. Viewing these parameters as positions on the axes of an n-dimensional coordinate frame means specifying a value for each determines a slicing plane perpendicular to the axis at the given point. As we employ the concept of Feiner and Beshers' nested coordinate systems to visualize the exact sequence of slicing operations performed, we can use each nested coordinate system to fix one, two, or three parameters. Specifically, the position at which the ith coordinate system ($i \geq 1$) goes inside the $(i-1)$th coordinate system illustrates the one, two, or three parameter values fixed by that placement, depending on the dimensionality of the $(i-1)$th coordinate system.

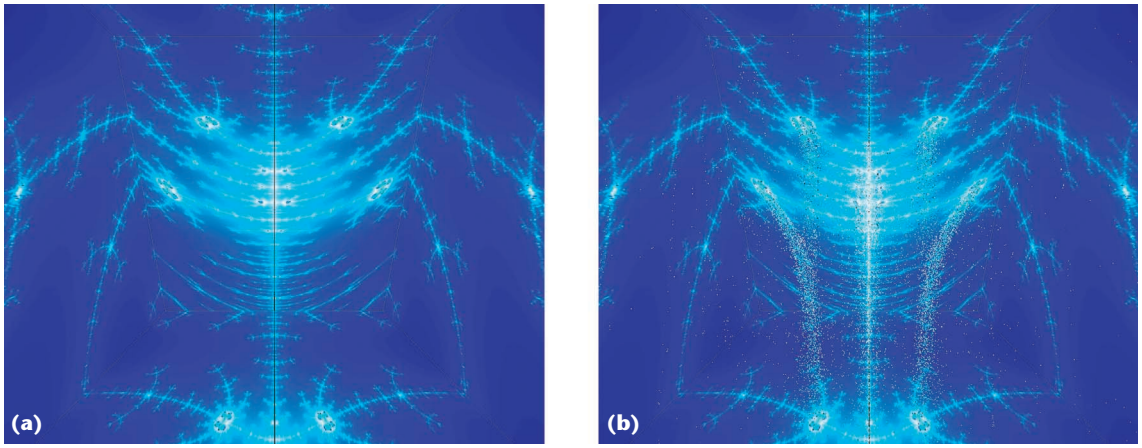
In our first example (visualizing a 9D fractal), it's convenient to use a coordinate system to fix each of two independent parameter pairs. A third nested system then fixes a fifth parameter, giving the 4D slice for display. In Figure 1a, the green coordinate system is the base (0th) system, inside of which we place the blue axis system. The two coordinates corresponding to the position of the blue origin with respect to the green system fix the first two parameters. The position of the single brown axis with respect to the blue axes fixes another pair of parameters. Finally, the position of the red coordinate system along the brown axis fixes a fifth parameter, giving a 4D slice, a portion of which is visible in Figure 1b. Imagine the cube in Figure 1b as carved out of solid fractal material defined continuously throughout the space described by the final red coordinate system.

Our system employs a simple configuration file to specify how parameters are mapped to axes, initial values for $n-4$ parameters, and the subset of the final coordinate system to be displayed (for example, the blue textured cube of Figure 1b). A general CoordinateSystem object, defined as an affine point and three linearly independent vectors, represents each nested coordinate system. The system assigns a CoordinateSystemRenderer object to each CoordinateSystem object to maintain information such as which axes are in use and hence are to be drawn. Allowing each nested coordinate system to fix one, two, or three independent variables lets the user ensure that the display of the nested coordinate systems—and thus the presentation of the navigational tools discussed later—matches well with the specifics of the data set under study.

The orientations of the various coordinate systems relative to one another are irrelevant. The only significant fact is that the origin of each remains fixed at the proper point in the containing coordinate system. For all but the final 4D slice to be displayed, this fact isn't relevant. However, it's useful for interactively rotating the 4D slice to view it from all sides. The IVP interaction tools permit either rotating (and zooming) the entire set of nested coordinate systems as a unit or, as illustrated in Figure 1c, rotating just the 4D slice while keeping the display of the other systems fixed.

Displaying the 4D subset using the IVP

We originally imagined the IVP as a room with glass walls where we could fly through a 4D space, much like being in a glass-bottom boat observing fish and other underwater life (see Figure 2).[4] As we fly through the space, we can shrink or expand the room. As the room moves or changes size and shape, the system computes, at high resolution, 2D slices of our original n-dimensional data set corresponding to the walls, ceiling, and



3 View inside the IVP: (a) with no point cloud; (b) with a point cloud.

floor and places them as texture maps onto the corresponding polygons in real time.

Of course, the model definitions are continuous throughout this space; hence, an alternative display technique would be to use a volumetric rendering strategy. However, doing so would completely sacrifice interactivity for obtaining sufficient detail to adequately convey the structure. Our goal from the start was to develop a tool that allowed interactive exploration of high-dimensional spaces, so we realized we needed an alternative. After some experimentation, we discovered that an effective technique was to display very high resolution in some areas (on polygons corresponding to our room's walls) and nothing in between. This approach let us interactively adjust the slicing parameters—as well as the room's size, shape, and position—and see the results immediately.

Although displaying high-resolution data on the walls conveys complex structure well, visualizing how structure on one wall transforms into structure on adjacent walls can be difficult. We discovered a relatively fast way to address this issue by generating and displaying low-resolution point clouds inside the room. This approach partly affected our ability to keep displays current during parameter changes, but the additional data often significantly improved the clarity of the data set's structure in the room's interior. The resulting displays frequently clarified how major features on one wall connect to features on other walls. A fairly small number of such points often suffices.

Figure 3a illustrates the interior of a room without this point cloud; Figure 3b shows the same room containing a cloud of points. Notice how the point cloud reveals how the structure represented by the four major white spots on the far wall connect to that represented by the two white spots on the floor. The effect resembles a waterfall. Exactly how (or even if) these pieces connect is unclear without the benefit of this point cloud.

Interactively exploring a data set

The primary means of interactive exploration is to interactively drag the embedded coordinate systems of Figure 1. As the user clicks and drags a coordinate system, the selected system (and all others nested inside it) move. Because the dragged system's position with respect to its parent system fixes one, two, or three parameters, the IVP continuously updates these parameter values and regenerates the resulting images on the walls (as well as any other auxiliary display, such as point clouds).

Some of the more complex data sets for which we've used the IVP require so much computation that keeping up with a dragged coordinate system isn't always possible. We adopted two approaches to address this problem. The first was to let the user specify how hard the system should try to keep images current. We adopted a three-state image regeneration model for this purpose. In decreasing order of CPU intensity, these three states are as follows:

- Users can require that the 4D slice remain continuously current as they drag the coordinate system. The system spawns several threads (at least one per wall) and assigns each thread a specific area of a wall to recompute. The threads run continuously, updating the display of the slice as often as possible. The system periodically synchronizes the threads to simultaneously update the display of all walls.
- Users can tell the system to update the 4D slice's display at the end of each coordinate-system manipulation. In this case, the system continuously updates the coordinate systems' display and any simple related displays as the user drags the coordinate system, but the system doesn't recompute the 4D slice itself until the manipulation ends (for example, until the user releases a mouse button).
- Users can tell the system not to update the 4D slice until they specifically request it. This mode is most useful when users wish to change a series of parameter settings and the intervening displays aren't of particular interest.

Our second approach involves sending the computation to a larger server and then retrieving the remotely computed texture data. This approach uses Corba to communicate between our display system and the remote supercomputer. We discussed additional specifics of this technique in the "System architecture and operation" section. By tapping the availability of faster processors on the remote server, we can provide

far higher frame rates for interactive coordinate-system manipulations.

The user can also interactively vary the parameters by directly entering them in response to system prompts, although of course this is not as demanding computationally. This approach is not exciting from a graphics and computational perspective, but it does prove useful for quickly reaching specific areas of the data set known to be of interest.

Visualizing fractals in higher dimensions

Our first example illustrates the use of our system in the study of fractals and their parameters. Specifically, we consider the so-called *filled-in Julia set* in two complex dimensions. The basic Julia set is a set of complex numbers that depends on one complex parameter. Visualizing this set has led to amazing images and many new theoretical developments. The filled-in Julia set is defined by quadratic polynomial $P_c(z) = z^2 + c$, where z and c are complex numbers. It is the set of all points in the complex plane such that the n-fold application of function $P_c(z)$ to an initial point, z, doesn't go to infinity. Specifically, if the initial point is z_0, then $z_1 = P_c(z_0) = {z_0}^2 + c$, and $z_n = P_c(z_{n-1})$. The initial point z_0 is in the filled-in Julia set if z_n doesn't become arbitrarily large when n is arbitrarily large. We can generate images of this set as simple raster images in which each pixel corresponds to a point z and is assigned a color according to how fast the corresponding z_n approaches infinity. A common way of visualizing this mapping and its dependency on the parameters is with side-by-side windows—one representing parameter space c, another representing the resulting raster image. Because users can dynamically adjust the values of the parameters displayed in the one window, they can then watch as the other window displays the corresponding Julia set.

Hence, such a visualization approach lets us study three important related concepts:

- the filled-in Julia set's geometry,
- the geometry of the sets in the parameter space, and
- the relationship between the parameters and the dynamic space represented by the filled-in Julia set.

We can generalize the Julia set to higher dimensions in two basic ways. One is by considering the set to be defined on the field of the quaternions,[5] and the other is through the Henon map.[6] The latter appears in the study of dynamic systems and chaos theory. It's one of the simplest maps that allows modeling of complicated chaotic behavior, and researchers have used it extensively in experimental simulations.[7,8] In addition, studying the properties of the complex Henon map has recently attained important theoretical interest.[9,10]

The Henon map is defined by the ordered pair $(P_c(z) + dw, dz)$, where z, w, c, and d are complex numbers. If d is 0, this becomes $(P_c(z), 0)$, the simple initial example just described. We can view the filled-in Julia set as a 9D data set. Specifically, the functional mapping depends on two complex constants (c and d) and two complex spatial variables (z and w), for a total of eight real variables. (Each of c, d, z, and w has both a real and an imaginary component.) We then compute an escape rate, m, based on these eight values as a measure of how fast an initial point escapes to infinity under this mapping. We can generate this generalized filled-in Julia set similarly to the way we described the Julia set earlier. Our starting point is now a pair of complex numbers, (z_0, w_0). The n-fold application of P_c then yields (z_n, w_n). The original pair of complex numbers, (z_0, w_0), is in the filled-in Julia set if (z_n, w_n) doesn't grow arbitrarily large when n is large.

Initially specifying and displaying a 4D subset for the Julia set

To visualize the Julia set in two complex variables, we first need to fix the two complex parameters (four real coordinates). Because case $d = (0, 0)$ corresponds to the one-variable filled-in Julia set, we've found it most useful to specify its value first to allow easy comparisons with $d = (0, 0)$. Therefore, we use a 2D coordinate system (the green axes of Figure 1) to fix the real and imaginary components of complex parameter d. (Placing the origin of the blue coordinate system at a specific location within the green one specifies the value of d.) Once we've determined d in this manner, we use the now-fixed blue coordinate system to fix the value of complex parameter c by placing the brown axis at a specific location inside of it. Fixing c and d determines a specific Julia set mapping. Those values are then constant throughout the application of the map.

Fixing the values of c and d reduces our original problem from nine to five dimensions. We use the third 1D coordinate system (the brown axis of Figure 1) to fix one more parameter by placing the origin of the red coordinate system of Figure 1 along the brown axis. (We generally choose the imaginary component, w_{im}, of w to be fixed in this manner.) This leaves us with a 4D data set, which we can view using color on the walls of the IVP.

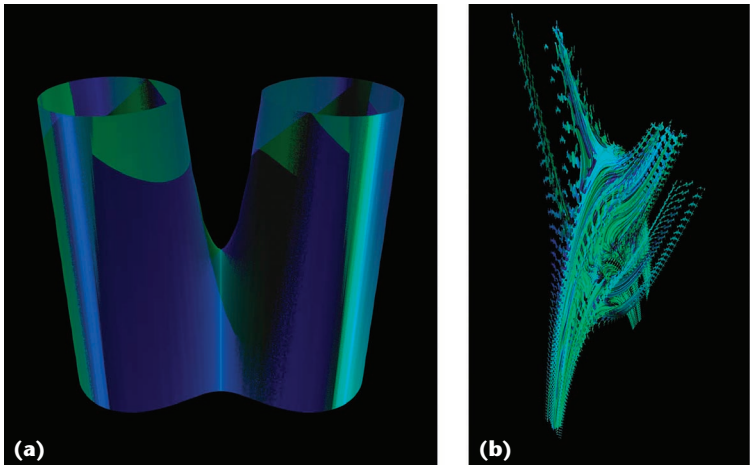
Figure 1 illustrates the coordinate-system placements corresponding to $d = (0.41017, 0.267917)$, $c = (-1.02003, 0.123774)$, and $w_{im} = 0.0247274$. The green axes and the yellow plane represent the d coordinate system; the blue axes and the cyan plane represent the c coordinate system. The single brown axis represents the coordinate system used to fix w_{im}. Finally, the red axes themselves represent the coordinate system of the data's 3D slice in which color encodes the fourth dimension.

Displaying the 4D subset using the IVP

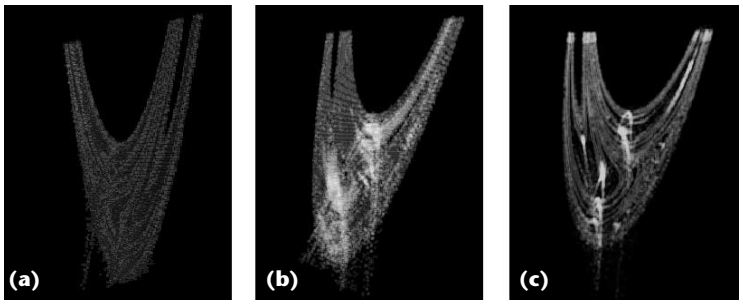
The IVP displays the three remaining coordinates for ranges of the variables, using a 3D ramp in which color denotes the escape rate. Particularly useful is the location of the z-plane in the horizontal position. This z-plane 2D slice corresponds to the one-variable Julia set, which is now even clearer because the IVP permits easy visualization of it in the context of its higher-dimensional structure. That is, the third dimension and the immersive display convey how the higher dimensions relate to the 2D slice's structure.

Interactively exploring the Julia set

In addition to seeing how the higher-dimensional structure relates to the familiar 2D Julia set, we can



4 (a) Smooth object. (b) Object with a chaotic boundary.



5 Visualizing a set using volume rendering and transparency: (a) 128,731 points; (b) 224,618 points; and (c) 463,442 points.

interactively explore how the filled-in Julia set changes with parameter changes. In particular, starting with $d = (0, 0)$ and moving in different directions in the d-plane, we can see how the geometric characteristics of higher-dimensional sets appear. For $d = (0, 0)$, the filled-in Julia set is an infinite cylinder with a wall with different shapes and a section that corresponds to the one-variable set. As soon as we change the value of d, the set becomes an infinite tree with infinite branches, as Figure 4a shows. This tree trunk's size decreases as d gets farther from (0, 0) until it loses all the tree structure, and the set becomes intricate, as Figure 4b shows.

Other visualization approaches

We can use other approaches to visualize complex sets such as the filled-in Julia set. We've experimented with some of these alternatives, but because of their limitations, we decided not to use them. Here we briefly review our experiences and the difficulties we encountered.

Ray tracing. We used a locally modified version of the Persistence of Vision Ray Tracer (POV-Ray) to visualize this data set.[11] For some values of parameters c and d, obtaining useful high-quality pictures (as in Figure 4a) was relatively simple. The filled-in Julia set's boundary appeared to be well behaved, so a ray-traced image with its use of surface normal vector approximations didn't produce an artificially smooth image. However, for most c and d values, ray tracing wasn't very useful. Because the boundary of most data sets is quite chaotic, ray-traced images such as those in Figure 4b, while perhaps more visually appealing, don't reliably present the fractal's structure. First, the sampling inherent in ray tracing is far less reliable for such objects. Second, using approximated normal vectors to support the lighting model can actually hinder attempts to understand the object's structure by making it appear far smoother than it actually is.

Volume rendering and blended transparency. We also experimented with visualizing the set using volume rendering and blended transparency in IRIS Explorer.[12] We found transparency to be a useful visualization technique, especially for presenting detailed structure in static images. However, we encountered problems while investigating the tradeoffs between generating lots of points (thus achieving better visualizations) and maintaining interactive responses to dynamic rotations. Specifically, when we kept the data set sufficiently small so that interactive 3D viewing manipulations were reasonably responsive, there was no longer sufficient detail to visualize the data set's structure. Table 1 presents a coarse quantification of this problem. Along with Figure 5, it illustrates the tradeoffs we observed, ranging from good interactive response but poor visualization to poor interactive response but good visualization. Our goal in this work was to develop a technique that would provide good interactive response and high resolution imagery, at least in select regions.

Immersive parameter space paradigm

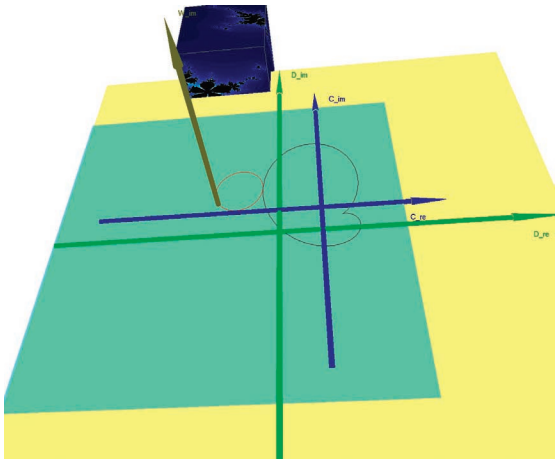
While the IVP itself is quite general in terms of how to define and use coordinate systems, it's sometimes useful to overlay additional information specific to the system under study. Our implementation lets users add such additional problem-specific information to the otherwise problem-independent coordinate-system display. Here we describe how we've used this functionality in the filled-in Julia set example.

In the study of Julia sets, it's essential to understand the system's dependence on the fixed complex parameters (c and d). A dynamic system can be stable for some parameter values but chaotic for others. The interesting set to study in parameter space is the one analogous to the Mandelbrot set for a single complex variable. We don't include this set in our model because it's computationally intensive in two complex dimensions. Instead, we include two curves (C1 and C2) that define the main regions in parameter space. Fornaess and Gavosto provide technical definitions and derivations of the closed-form expressions for these curves.[13]

The idea is that when c is in the interior of one of these boundary curves, the dynamics are fairly stable. By contrast, when c is on or close to the boundary curves, the

Table 1. Visualization quality and interactive response based on the number of points observed for visualization using volume rendering and transparency.

Figure	Number of Points	Visualization Quality	Interactive Response
5a	128,731	Poor	Good
5b	224,618	Fair	Adequate
5c	463,442	Good	Poor

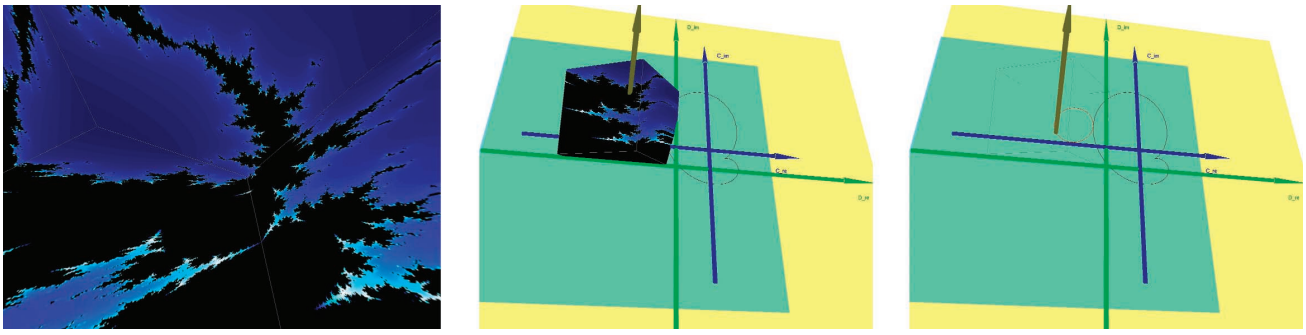


6 Boundary curves for the slice depicted in Figures 1 and 2.

dynamics are far less stable. Figure 6 displays these curves for $d = (0.41017, 0.267917)$. (We suppressed the display of the 3D slice coordinate system for clarity.) We produced the image in Figure 6 using the same parameter settings illustrated in Figure 1: $c = (-1.02003, 0.123774)$. The object in Figure 7 is an example with c fixed on one of the curves. The object in Figure 8 has c close to, but just outside, both boundary curves.

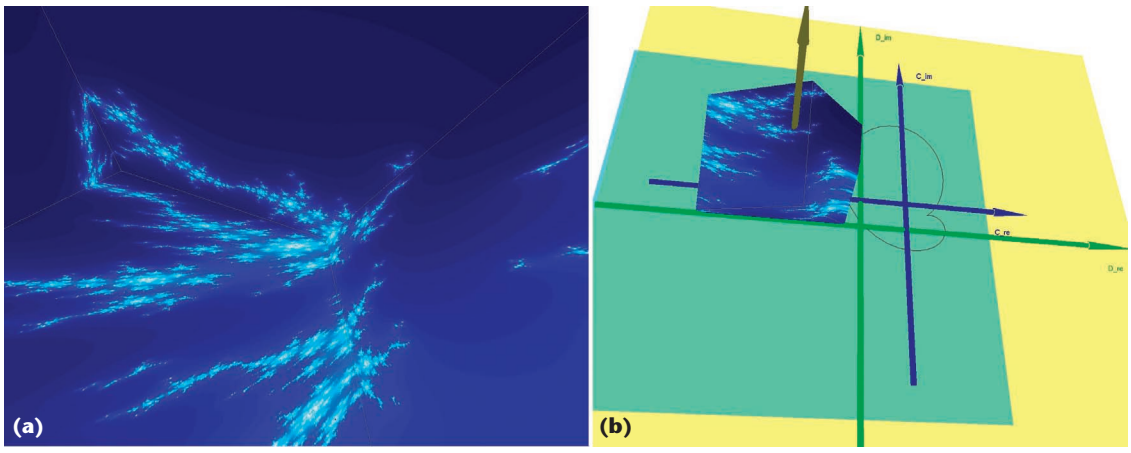
Visualizing the stability of satellite orbits

Our second example involves the study of paths that satellites follow when launched between two neighboring planetary bodies. The original motivation for this research was to study the paths that satellites would take when launched somewhere between the earth and the moon.[14] Such satellites would travel in reasonable orbits for a while and deliver useful information. Eventually, however, their trajectories would become more

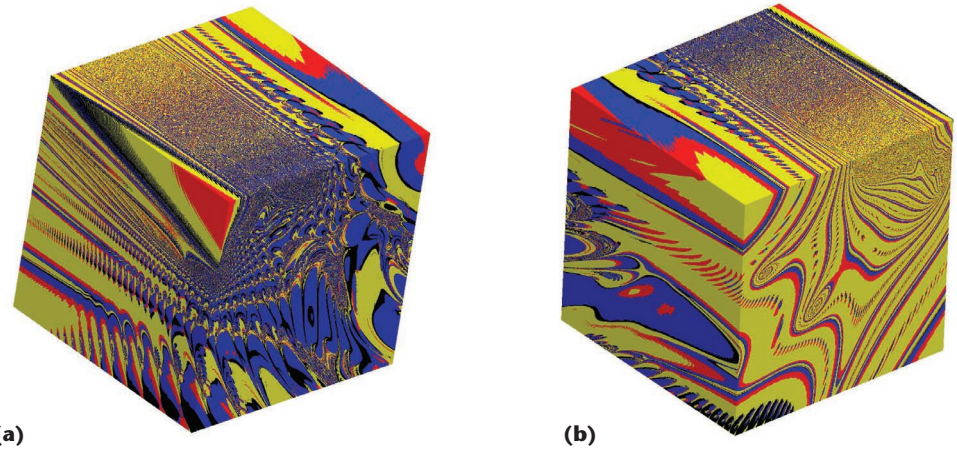


(a) (b) (c)

7 (a) View inside the IVP for c on the boundary curves in Figure 6. (b) Coordinate-system view. (c) Same as (b), but with the wall displays off.



8 (a) View inside the IVP for a value of c just outside the boundary curve. (b) Coordinate-system view.



(a) (b)

9 (a) Four-color display of the satellite motion model. (b) Same as (a), but from the other side of the cube slice.

erratic, and they would escape the gravitational attraction of the two bodies and fly away. Moreover, depending on such quantities as their starting position and their velocity, the paths taken before escaping could be quite erratic.

Some of the important goals of Andrade's work were to illustrate the inherent chaotic behavior of such a satellite system and to provide guidance for deciding where and how to launch multiple satellites between the earth and the moon that wouldn't collide with one another. The particular model we illustrate corresponds to a classical three-body restricted problem, in which one mass (the earth) is far heavier than the other (the moon).[14] Andrade's model performed the computations in the vicinity of one of the unstable equilibrium points. The parameters assigned to the coordinate-system axis controls are the

- masses of the two planetary bodies,
- coordinates of the satellite's initial position,
- satellite's initial velocity,
- angular frequency, and
- time interval considered.

We use one of Andrade's algorithms, which simulates a satellite's path. The current version tracks the path in the plane determined by the two bodies and the satellite. When the satellite eventually escapes, the algorithm records the quadrant into which it escapes. We use the three spatial dimensions to encode the satellite's initial x and y location and the frequency. As in the filled-in Julia set example, the IVP once again uses color to encode the fourth dimension—in this case, the quadrant into which the satellite eventually escapes. The nearby starting points in Figure 9 have different colors, forming a complicated pattern. Satellites launched close to one another eventually escape into different quadrants. The IVP uses red, yellow, blue, and black to encode the quadrants into which the satellite escapes. This simulation and visualization illustrates that the satellite's basins of attraction are fractals, and it highlights the inherent uncertainty about the final destination of a satellite launched at a particular location with a given initial velocity. (A basin of attraction is the set of points that, under the mapping of the current model, will map to a particular point called an attractor. In simple systems, such basins would be relatively large homogeneous areas. In cases such as this, the basins are complex fractals.) An additional useful related visualization would be to color our space according to how long it takes the satellite to escape from a region around the starting point.

Figure 9 illustrates this satellite system's inherent chaotic behavior. Indeed, if we zoomed in on various areas, including those exhibiting solid color in Figure 9, we would see more fractal boundaries emerging. These images let us visualize, and thus better understand, this known chaotic behavior.

Future work

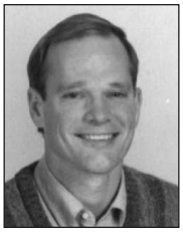
Our goal in designing the IVP was to develop a tool capable of real-time exploration of multidimensional data sets. Both the Henon map example and the satellite motion example revealed the elaborate structure that users could effectively visualize on the IVP's walls. We plan to direct part of our future work toward increasing system performance. That is, we wish to further optimize distributed processing using highly parallel multiprocessor systems to effectively eliminate delays during interactive parameter adjustment. We also hope to build more interactive configuration tools so that users can more readily assign parameters to axes as well as modify those assignments during interactive sessions. Finally, we'd like to develop tools that, within certain well-defined limits, will let a user define a new system to the IVP interactively without having to write C++ code. ■

Acknowledgments

Miller performed his portion of this work in DesignLab, a multidisciplinary research laboratory at the University of Kansas funded in part by National Science Foundation grant CDA-94-01021. Gavosto's portion of the work was partially supported by a General Research Fund grant from the University of Kansas and by NSF grants 9970576 and 0112375. John Sheu wrote the original Windows prototype for the visualization of the 3D slice. Joe Shannonhouse developed the distributed Corba-based server. We thank Victor Andrade for sharing his computer model of satellite motion with us.

References

1. S. Feiner and C. Beshers, "Worlds within Worlds: Metaphors for Exploring *n*-Dimensional Virtual Worlds," *Proc. ACM Symp. User Interface and Software Technology* (UIST 90), ACM Press, 1990, pp. 76-83.
2. S. Feiner and C. Beshers, "Visualizing *n*-Dimensional Virtual Worlds with *n*-Vision," *Computer Graphics* (*Proc. Siggraph*, ACM Press), vol. 24, no. 2, Mar. 1990, pp. 37-38.
3. J.G. Shannonhouse, *A Framework for Building High Performance Computing Environments*, master's thesis, Dept. of Electrical Engineering and Computer Science, Univ. of Kansas, 2002.
4. E.A. Gavosto, J.R. Miller, and J. Sheu, "Immersive 4D Visualization of Complex Dynamics," *Proc. Workshop New Paradigms in Information Visualization and Manipulation* (NPIV 98), ACM Press, 2000, pp. 62-64.
5. K. Shoemake, "Animating Rotation with Quaternion Curves," *Computer Graphics* (*Proc. Siggraph*, ACM Press), vol. 19, no. 3, 1985, pp. 245-254.
6. M. Henon, "A Two Dimensional Mapping with a Strange Attractor," *Comm. in Mathematical Physics*, vol. 50, no. 1, 1976, pp. 69-77.
7. Y.F. Gong et al., "Recovering Strange Attractors from Noisy Interspike Intervals of Neuronal Firings," *Physics Letters* A, vol. 258, nos. 4-6, 1999, pp. 253-262.
8. A. Tufaile and J.C. Sartorelli, "Hénon-like Attractor in Air Bubble Formation," *Physics Letters* A, vol. 275, no. 3, 2000, pp. 211-217.
9. E. Bedford and J. Smillie, "Polynomial Diffeomorphisms of C2: Currents, Equilibrium Measure and Hyperbolicity," *Inventiones Mathematicae*, vol. 103, no. 1, 1991, pp. 69-99.
10. J.E. Fornaess and N. Sibony, "Complex Henon Mappings in C2 and Fatou Bieberbach Domains," *Duke Mathematical J.*, vol. 65, no. 2, 1992, pp. 345-380.
11. A. Johnson, *Ray Tracing the Complex Henon Map*, tech. report, Dept. of Mathematics, Univ. of Kansas, 1998.
12. L. Cao, *Coexisting Attracting Basins in Complex Holomorphic Dynamics*, tech. report, Dept. of Mathematics, Univ. of Kansas, 1999.
13. J.E. Fornaess and E. Gavosto, *The Mandelbrot Set for the Henon Map in the Complex Domain*, submitted for publication.
14. V. Andrade, *Chaotic Transients in Satellite Motion*, master's thesis, Dept. of Physics and Astronomy, Univ. of Kansas, 2001.



James R. Miller *is an associate professor of computer science at the University of Kansas. His research interests include graphics, visualization, geometric modeling, the use of computers and technology in education, and programming languages. Miller received a BS in computer science from Iowa State University and an MS and a PhD in computer science from Purdue University.*

Estela A. Gavosto *is associate chair of the Department of Mathematics at the University of Kansas. Her research interests include the study of several complex variables, dynamic systems, and visualization. Gavosto received a PhD in mathematics at Washington University, St. Louis.*

Readers may contact James R. Miller at the Dept. of Computer Science, Univ. of Kansas, Lawrence, KS 66045, jrmiller@ku.edu.

For more information on this or any other computing topic, please visit our Digital Library at http://www.computer.org/publications/dlib.