Automated Design of Virtual Worlds for Visualizing Multivariate Relations

Clifford Beshers Steven Feiner

Department of Computer Science
Columbia University
New York, NY 10027

Abstract

Interactive visualization systems provide a powerful means to explore complex data, especially when coupled with 3D interaction and display devices to produce virtual worlds. While designing a quality static 2D visualization is already a difficult task for most users, designing an interactive 3D one is even more challenging. To address this problem, we are developing AutoVisual, *a research system that designs interactive virtual worlds for visualizing and exploring multivariate relations of arbitrary arity.* AutoVisual *uses "worlds within worlds," an interactive visualization technique that exploits nested, heterogeneous coordinate systems to map multiple variables onto each of our spatial dimensions.* AutoVisual*'s designs are guided by user-specified* visualization tasks, *and by a catalog of design principles encoded using a rule-based language.*

1 Introduction

High-performance graphics hardware and software are beginning to become an affordable means to visualize the large amounts of complex data that are being produced in a variety of quantitative fields. There are significant design tasks involved in producing an effective visualization, however, that require expertise that practitioners in these disciplines rarely have. Designing a visualization not only requires skill in graphic design and data presentation, but if the visualization is to be an interactive one, in which the user can dynamically manipulate the data being represented, visualization design also entails knowing something about how to design user interfaces. Even if the designer has these skills, design can still take large amounts of effort and time.

As a case in point, consider our *n*-Vision visualization system and the "worlds within worlds" interaction technique that it uses [3, 4]. *n*-Vision provides its users with a 3D virtual world within which they can visualize and manipulate representations of multivariate relations. The "worlds within worlds" technique represents relations of an arbitrary number of variables by recursively nesting heterogeneous coordinate systems (*worlds*). An innermost coordinate system typically contains an encoding object (e.g., line graph, height field, or volume rendering) for which one axis of the coordinate system represents an output variable and the remaining axes represent input variables. To accommodate additional input variables, a coordinate system can be nested within an additional coordinate system, each of whose axes represents a new variable. The position of a distinguished point (e.g., the origin) of the inner coordinate system relative to the outer coordinate system within which it is embedded, fixes values for the variables associated with each of the outer coordinate system's axes. These values are used in evaluating the inner coordinate system's encoding object. Further variables are accommodated by recursively nesting the outer coordinate system within additional heterogeneous coordinate systems. Figure 1 shows an inner world containing a height field nested inside a single outer world.

n-Vision supports a rich set of interactive techniques for manipulating a visualization. We use a 3D stereo display (StereoGraphics CrystalEyes) and a 3D input device (VPL DataGlove) to create the feel of a virtual world within which the user can explore their data. Different hand gestures allow the user to scale and rotate worlds about their origins to view them better and to translate worlds to see the effects of changing the values of those variables represented by the coordinate system within which the translation occurs. Users can further examine their data by using *tools*, such as a "dipstick" that displays a precise readout of the value at a point in a height field. Figure 1 shows the dipstick in use.

While *n*-Vision was intended to be flexible, the drawback to its flexibility is that designing visualizations for it can be both *difficult* and *tedious*: *difficult* because there are many potential ways to visualize any relation; *tedious* because choosing among the many possibilities to create a specific visualization requires making a large number of

0-8186-2897-9/92 $03.00

decisions. These problems present themselves even in the design of static presentations of low-dimensional relations. Increasing the number of variables and adding interaction only compounds the problem. Designing an n-Vision visualization is actually a matter of designing an interactive virtual world that has both appearance and behavior. Interactivity is a particularly important issue since an interactive visualization, unlike a static one, need not present all needed information at once. The user can instead be provided with the ability to manipulate the world to obtain the information that is not currently being presented. The goal is to design a visualization world that makes the required interaction as straightforward as possible.

AutoVisual is a rule-based system that is being developed to explore how to automate the design of n-Vision's virtual worlds. The user specifies the *visualization task*, rather than a particular visualization. *AutoVisual* generates an interactive virtual world that is appropriate for the task. Removing the difficulty of visualization design is useful for beginning users, who do not yet understand the complexities, as well as for experienced users, who may understand the design process, but don't have the design expertise. Removing the tedium of design is a benefit for all users. The result is that visualizations can be constructed faster, which is especially important in applications where the relations being visualized and the needs of the user are constantly changing.

There are several examples of research systems that generate visualizations of relational data. Mackinlay's APT [8] designs static, 2D graphs of the results of database queries. This work has been extended by Roth and Mattis [9] to take into account additional ways to characterize the data being presented. Casner's BOZ[2] also refines the paradigm used in APT to design graphs appropriate for very specific visualization tasks. Some work has been done in generating more complex visualizations. Senay[10] is developing a system that will use APT's approach to produce 3D AVS [11] visualizations. Kochhar, Friedell, and LaPolla [7] use a cooperative approach to generating visualizations in which the system relies on the user to make difficult design decisions, providing a set of sample visualizations to choose from at each choice point.

The work we describe here differs from these other efforts in several ways. *AutoVisual* addresses the design of *interactive*, rather than static, visualizations, and takes into account how the user can manipulate individual components in its design. Its visualizations are also unique in addressing relations with arbitrary numbers of variables. In addition, *AutoVisual* attempts to account for the cost of user interaction time and rendering time in its rules.

2 Worlds within worlds

When building an automated design system, it is important to keep the representation of the domain as simple as possible. The worlds within worlds method, as described above, is a medley of outer worlds, inner worlds, and tools. In this section, we present a formalization that is both simple and homogeneous. We observe that worlds and tools have similar functions, allowing the user to selectively refine some existing graphical encoding of a relation.

Consider the similarity of the height field world and the dipstick tool in the visualization of Figure 1. Each is a graph that presents some subset of the relation encoded by its parent world. The particular relation subset is determined by the position of the graph's origin relative to its parent. The user can move each graph interactively throughout the space defined by its parent. Alhough there are obvious differences between the height field and the dipstick, such as the type and dimension of the encoding, they operate in basically the same manner. Thus, the most important characteristic of worlds within worlds is that it uses graphs as *interactors*.

We can now define an n-Vision virtual world as a hierarchy of *interactors*. Each interactor consists of four basic components: a set of *encoding spaces*, a set of *encoding objects*, a set of *selections*, and a *user interface*.

An *encoding space* is a region where the graph is rendered. In general, an encoding space may be of any dimension (up to three), arbitrarily positioned and oriented in the 3D world. However, many interactors have constrained placement; for example, the encoding space of the dipstick is a line constrained to be within the encoding space of the parent world and parallel to its vertical axis.

The set of *encoding objects* comprise the body of the graph (e.g., a height field). An interactor may have no encoding objects, as in the outer world of Figure 1. The potential set of encoding objects includes almost any known visualization technique.

A *selection* determines the relation encoded by the graph interactor. It consists of a geometric subset of the *encoding space* of the parent interactor. This geometric subset determines a subset of the relation encoded by the parent. The selection may be any axis-aligned rectangular volume of a dimension less than or equal to that of the parent interactor's encoding space. For example, the selection component of the height field interactor is a point (typically the inner world's origin) in the coordinate system of the outer world.

Rectangular subsets can be specified with upper and lower bounds for each variable. In addition, we allow for a precision to be specified. Given a multivalued relation

$$(y_1, \ldots, y_m) = R(x_1, \ldots, x_n),$$

we represent the restrictions on x_i with the notation:

$$(y_1, \ldots, y_m) = R(x_1, \ldots, x_i[v, l, u, p], \ldots, x_n),$$

where v is a default value, l and u are lower and upper bounds, and p is the precision. If any of p, l or u is omitted, a default is assumed. The value field may not be omitted, but may contain a *don't care* value ('–'), which is used to indicate a range. For example, $x[-, 0, 1]$ indicates the unit range of real numbers.

The *user interface* component of an interactor consists of bindings between user actions and properties of the interactor. The height field world has bindings for translation, scaling and orientation in 3-space. The dipstick only has bindings for translation, because its size and orientation are determined by its parent.

All examples in this paper describe interactors with only one selection, encoding space and encoding object. An example of an interactor with multiple selections is a comparison tool that redisplays several parent interactors in one space. Alternatively, an interactor with multiple encoding spaces may provide redundant encodings of the same relation using different encoding methods.

3 Visualization tasks

Visualizations are always created with some purpose: to illustrate an idea; to explore some unknown data; to find parameters that yield an interesting curve; to compare two functions. We call this the *visualization task* or *visualization goal*. The user interacts with *AutoVisual* by specifying visualization tasks to be performed on a set of relations. The overall task is taken as the conjunction of all these tasks. Thus, *AutoVisual* must ensure that there exists some visualization that satisfies each task specification.

We use a two-part task specification: *task selections* and *task operators*. *Task selections* are closely related to the *selection* component of an interactor, in that they specify subsets of a relation. *Task operators* are applied to task selections to refine the purpose of the visualization further. Currently, we allow operators for *exploration*, *directed search*, and *comparison*. These operators correspond roughly to Bertin's *summary*, *elementary*, and *comparative readings* respectively [1].[1]

Task selections

The user specifies the potential subsets of interest for a particular task through simple constraints on the variables of a relation. The syntax for task selections is similar to that used for *interactor selections*. The primary difference is that *task selections* allow *dynamic values*, not just constants. A dynamic value is simply a variable, specified as a symbol preceded by a question mark (e.g., $x[?c]$). A dynamic value indicates that the user needs interactive control of that field. For example, $x[2.5]$ specifies a specific value, and $x[-, ?l, ?u, 0.01]$ specifies a range with dynamic bounds.

The variable names are included in the notation so that the restrictions may be specified in any order. In addition, any omitted variable is assumed to have the default constraints.

Note that specifying a constant value does not prohibit *AutoVisual* from representing a variable with some interactor. It simply indicates that the variable has a low priority for the user in the context of a particular task. Similarly, specifying a range does not guarantee that *AutoVisual* will provide an axis encoding that range, because there may be no resources to do so if too many such ranges are specified.

Task operators

A *task operator* indicates what the user wishes to do with a task selection. We support *exploration*, *directed search*, and *comparison* operators.

Exploration has the form $explore(S)$, where S is a task selection. A user would typically choose exploration if the relation is unfamiliar. An appropriate visualization provides a *summary reading*, allowing the user to view the entire selection as quickly as possible.

Directed search has the form $search(S_{goal}, S_{context})$, where S_{goal} and $S_{context}$ are both task selections. $S_{goal} \cup S_{context}$ form the complete context, i.e., the set of variables that should be considered for binding to axes. If $S_{context}$ is omitted, the context is the entire relation. The task is to locate a particular subset of the relation, specified by the selection S_{goal}.

Comparison has the form $compare(S_1, \ldots, S_k)$, where $k \geq 2$. The task is to study the relationship between selections. The visualization must allow comparison to be done easily, ensuring that all selections are encoded in the same space, or perhaps juxtaposed spaces, and that all encodings are visible. The comparison task presents the most difficult design challenge of any of the tasks.

4 Automated design

Design as search

We follow the method used by Mackinlay in APT[8], formulating the synthesis of a visualization as a state space

[1] A summary reading involves viewing many values at once, while an elementary reading determines the value of a relation for a single set of input parameters. A comparative reading determines the relationship between different subsets of a relation.

search implemented with a rule-based language. Unlike APT, however, *AutoVisual* focuses on combining "ready-made" graph types using the worlds within worlds approach, rather than on designing the internals of each graph by composition.

Each state represents a possible interactive visualization world and is a directed acyclic graph of interactors. The state change operators simply add interactors or modify those already in the graph, producing the final visualization by incremental refinement. When necessary, backtracking is employed to undo a design path that will not be successful. The selection of operators is driven by a rule-base of visualization design principles, taking into account the task to be performed, the interactors available, and the current hardware.

Termination conditions are a necessary part of any search algorithm. *AutoVisual* uses two criteria to establish that it has found a satisfactory visualization: *potential expressiveness* and *potential effectiveness*. These are Mackinlay's *expressiveness* and *effectiveness* criteria, modified for interactive visualization. An *expressive* graph encodes all relevant information and only that information. An *effective* graph presents that information clearly. We say that an interactive visualization is *potentially expressive*, if it has the potential, under user control, to display all the information over time. Similarly, an interactive visualization is *potentially effective* if its use over time can present the information sufficiently clearly. (Note that no single static view need be sufficiently effective and expressive by itself.)

Finally, we note two issues that influence the potential effectiveness of a dynamic visualization. First, the design should limit the amount of interaction time that it takes to accomplish a particular visualization task. Second, the visualization must have low response time. High-speed interactivity is a key component of worlds within worlds. We have found that when using a 3D input device, slow frame rates make the visualization unpleasant to use.

Search strategy

Because the space of possible visualizations is exponential, some intelligent strategy must be used to prune the search. We use a greedy, incremental refinement approach. We divide the search into two major phases: defining the connectivity of interactors, and instantiating interactors. The first phase effectively specifies the composite type of the visualization, defining which type of interactors comprise each level and how they connect together. The second phase instantiates the visualization by creating interactors. We briefly outline each phase here. The following sections detail the criteria used for making decisions at each step.

Design of the visualization structure begins by defining the hierarchy of worlds, working outward from the innermost worlds. The assignment of variables to axes in each world is accomplished by first determining a priority among the relation variables. As each world is selected, it is assigned the variables of highest priority. Next, tools are selected that are useful for examining the innermost worlds, ensuring that elementary readings of all values are possible. Finally, tools are chosen that may be used to examine arbitrary parts of the hierarchy (e.g., a zoom tool).

Once the design is complete, the visualization must be instantiated by creating interactors and linking them together. *AutoVisual* decides how many instances of each interactor type to create, and adjusts attributes such as position, size and color. Instantiating a visualization need not imply actually instantiating many interactors. *AutoVisual* creates at least one of every type of selected interactor as a way of documenting the structure and capabilities of the world.

Task dependent design

The visualization task affects decisions made throughout the design process. This section describes how choices are made for the exploration and search tasks.

Given an exploration task, *AutoVisual* considers a visualization to be *potentially effective* if the user can survey the entire selection space quickly. If the selection space covers many variables, this implies a tradeoff of quality for quantity. Thus, *AutoVisual* attempts to present as many samples along as many axes as possible, emphasizing the number of values encoded, rather than the quality of encoding. Hence, when selecting the innermost world, priority is given to interactors that encode many variables. For example, a gray-scale point cloud has priority over a height field for the exploration task, because it encodes an additional variable. Furthermore, when instantiating a visualization designed for exploration, *AutoVisual* creates worlds at each level of the hierarchy, ensuring that all variables in the selection are sampled. This reduces the interaction time required to view the entire selection.

Given a search task $search(S_{goal}, S_{context})$, *AutoVisual* gives priority to encoding the goal selection. Variables in S_{goal} have higher priority than those found only in $S_{context}$, forcing the goal selection to be localized to the innermost axes. Interactors with less effective encoding techniques are restricted from consideration for use as the innermost world, because the user must be able to recognize the selection easily. If no interactor encodes the entire goal selection, priority is given to interactors that encode more variables. However, if more than one interactor can present the entire goal selection, the one with the most effective encoding is selected. In this case, the encoding of the selection goal is entirely within the innermost world.

Assigning priority to variables

Clearly, binding a variable to an axis in an outer world produces a very different effect from binding it to an inner world. Thus, the mapping of variables to axes may have a great effect on the quality of the visualization. When there are multiple inner worlds, it is easiest to perceive the effects of variables encoded by the innermost worlds. To follow the effects of a variable bound to an outer axis, the eye must skip between the inner worlds. The higher up in the hierarchy a variable is bound, the more difficult it is comprehend its effect on the value of the relation. Therefore, it is important to determine a priority among variables that determines a useful permutation.

The task may help determine an order, as described above. Another relevant factor that *AutoVisual* considers is the cardinality of a variable's domain. We hypothesize that domains with low cardinality should be given lower priority when selecting axes. If the domain set is small enough, it may be sampled completely by instantiating sub-worlds for each value.

Resources

The quality of a visualization design depends upon the the particular computing environment. We have identified four *resources* that affect the evaluation of whether a particular visualization is potentially expressive and potentially effective:

Interactor rendering time. If a dynamic visualization takes too long to render, interaction with it will not be effective. Some encoding techniques take too long to render on particular hardware under any circumstances. Others may be just fast enough to allow only a few interactor instances. Thus, a particular encoding technique may be useful in small visualizations, but not large ones.

Relation computation power. Even if an interactor renders quickly, the relation may be expensive to sample. Using interactors that sample fewer variables helps reduce the computation time. Reducing the sampling rate along each will improve performance but may cause aliasing. If the task is exploration, the preference is given to interactors encoding more variables at lower sampling rates. For search, the preference is for higher sampling rates.

Display area. The visualization design must ensure that interactors are *legible*. Legibility requires that the interactor be allocated enough screen space to ensure that the encoded values can be perceived. When instantiating worlds, *AutoVisual* ensures legibility by requiring a minimum size for an interactor.

Space in the virtual world. Interactors that are too large may overlap, disrupting each other's presentation. Thus, when increasing the size of multiple interactors instantiated at the same level to improve legibility, *AutoVisual* ensures that space is left between interactors.

Even if interactors do not overlap in 3-space, their projections on the viewplane may. If the closer interactor is opaque, it will obscure the other. If the closer interactor is sparsely populated with encoding objects, such as a point cloud, or is transparent, the other will be visible. However, the further interactor may provide a poor background for viewing the closer one.

Rather than attempting to calculate a configuration of worlds that ensures that visibility is always good, *AutoVisual* assumes that the user will adjust the view appropriately. Therefore, *AutoVisual* considers whether the selected interactors are *likely* to have visibility problems. We define the predicate $viewable(I_1, I_2, placement)$, where I_1 and I_2 are interactors and $placement$ is one of $\{equal, before, behind\}$. For example, $viewable(I_1, I_2, before)$ is true if I_1 is in front of I_2 and the user can perceive I_1 easily. The background of the scene is considered to be a special interactor $I_{background}$. Each interactor I must satisfy $viewable(I, I_{background}, before)$ at all times.

AutoVisual gives lower priority to interactors that conflict with the current set. Unfortunately, interactors of the same type are not always viewable against each other. For example, point clouds have this problem. In these cases, *AutoVisual* will include a zoom interactor that redisplays the interactor in a different space but with a good background.

5 Implementation

The version of *AutoVisual* described in this paper represents our first attempt at automated generation of multivariate visualizations. We avoided supporting the full complexity of *n*-Vision by restricting the class of possible virtual worlds. The encoding techniques available to *AutoVisual* are limited to a 1D dipstick, 2D line graph, a height field, and a 3D gray-scale point cloud. As well, only interactors with a single selection, encoding space and encoding object are used. The curent implementation does not yet put labels or tickmarks on the line graph tool's axes.

AutoVisual and *n*-Vision are implemented with C++ and *AutoVisual*'s rules are encoded in the CLIPS production system language [5], running on a Hewlett-Packard 9000 Series 380 workstation with a TurboSRX graphics accelerator.

6 Example designs

We now give some examples of automated design of virtual worlds applied to a simple financial application.

Given a relation and a visualization task, we step through the design process, demonstrating how the system applies the guidelines of visualization design discussed above in order to create visualizations suited to each task.

The relation of interest is the Black-Scholes formula for computing the value of a *European option* to buy foreign currency [6]. An *option* is a contract that gives the holder the right (but not the obligation) to buy something at a specific price (*striking price*) on a specified date (*maturity date*). In this case, foreign currency is being purchased. The Black-Scholes formula for foreign currency options has the following form:

$$opt(S, \sigma, t, K, i_d, i_f, M),$$

where S is the current (*spot*) trading price of the currency, σ is the standard deviation of S over the previous year, t is the time (date), K is the *striking price*, i_d and i_f are the risk-free interest rates in the domestic and foreign markets respectively, and M is the *maturity date*. M and t are in units of years and are normalized so that 0 represents the creation date of the contract. Each of these is specified with a default value and range.

Consider first the task of exploring the value of an option for fixed maturity:

$$explore(opt(S[-], \sigma[-], t[-, 0, c_M], i_f[-], i_d[-], K[-])).$$

(Note that the maturity M assumes its default value because it is omitted.) The design process begins by assigning priority to the variables of the selection. The interest rates have the lowest cardinality and hence lowest priority. All other parameters have similar cardinality, and are selected in the order presented. The point cloud interactor is selected for the innermost world because it encodes the most independent variables of any interactor available, and is thus suited to the task of exploration. The variables S, σ and t are bound to the axes of the point cloud world. The search now proceeds to outer worlds. Variables K, i_f and i_d are bound to the x, y and z axes of a 3D outer world, respectively. We do not describe how tools are selected, as this is discussed in detail for the following examples.

The design proceeds to the instantiation phase. Following the rules for exploration, multiple point cloud worlds are created, sampling along each axis in the outer world. The size of the worlds is initially set so that the point cloud is legible in its default configuration, and then increased until the space between worlds reaches the minimum acceptable value. This configuration of interactors is determined to have acceptable rendering time. The sampling rate of the point clouds is then adjusted so that the computation time is acceptable. The resulting world is shown in Figure 2.

Now consider two directed search tasks. The option variables can be divided into two categories: S, σ, i_d and i_f are characteristics of the market; K and M are features of the contract. Thus, two questions that are commonly asked are:

1. Given an option with a fixed K and M, under which market conditions will that option perform well?
2. Given an expected market, what values of K and M will yield high profits?

Consider question 1, the search for good market conditions:

$$search(opt(S[-], \sigma[-], t[-], i_f[-], i_d[-]), \\ opt(K[c_K])).$$

Here, the striking price K is explicitly specified to obtain a different value than the default. The priority of K decreases in this example. Thus, for this task the priority ordering is S, σ, t, i_f, i_d, K. The point cloud is again selected for the innermost world: the gray-scale encoding is considered suitable for the search task, no interactor encodes five independent variables, and the point cloud encodes more variables than any other available interactor.

Tools are then selected to examine the point cloud in more detail to allow elementary readings of values. *AutoVisual* considers all possible hierarchies of tools where each level decreases in dimension. For example, a point cloud may be probed using a height field, a line graph, or a dipstick. However, *AutoVisual* assumes that only line graphs and dipsticks provide satisfactory elementary readings. Thus, any tool tree with a height field at the root must have an additional level of interaction. *AutoVisual* gives priority to the shallowest trees of tools. Among these, the one with the root tool of highest dimension requires the least interaction, so it is selected. In this case, the result of the search is a tool tree containing only a line graph. The 1D selection space of the line graph is represented as a line parallel to the x-axis of the point cloud. The selection may be moved freely about the yz-plane of the point cloud.

Only one instance of each world is created when this design is instantiated. The single point cloud does not conflict with other worlds, so it is made much larger, improving its legibility. The result is shown in Figure 3.

In the search task of question 2, we include a context selection, assuming a fixed domestic interest rate and date of maturity:

$$search(opt(t[-, 0, c_M], K[-]), \\ opt(S[-], \sigma[-], i_f[-])).$$

The goal selection has many fewer variables than in the previous task, which causes a height field to be chosen for the innermost world. A height field is the only interactor that encodes exactly the number of variables in the goal;

the point cloud encodes more than are needed. Tool selection and instantiation proceed almost exactly as in the previous example, yielding the design in Figure 4. The only difference is that the graphical representation for the selection geometry of the line-graph tool has changed to a plane to indicate that there is no degree of freedom along the vertical axis of the height field.

7 Conclusions and future work

We have described *AutoVisual*, an experimental visualization design system that synthesizes interactive 3D virtual worlds customized for particular relations and visualization tasks. We provided an overview of *AutoVisual*'s architecture and its underlying approach, including:

- a general method for specifying complex visualizations by composing simple components;
- a simple syntax for specifying visualization tasks;
- a design algorithm based on search of a visualization space;
- a catalog of visualization design criteria that can be expressed as rules to guide the search.

We reviewed a set of examples generated by the system that indicate its current capabilities.

AutoVisual is an early prototype and a number of issues remain unaddressed. We have barely considered aspects of the relation during the design. For example, the smoothness of the relation affects whether a filled height field is an appropriate encoding technique. If a variable is a measure of something, such as mass or time, this information could be used by *AutoVisual* to make more informed decisions. We will be expanding *AutoVisual*'s rule-base to take some of these issues into account.

We anticipate that the incremental design algorithm will readily allow for incremental modification of visualizations. Our hope is to be able to respond to changes in visualization tasks by reusing or modifying an existing visualization, rather than creating a new one. The result would involve fewer context changes and be less jarring for the user.

One key benefit of automated design is the ability to create widely different virtual worlds quickly, making interactive experimentation possible. We intend to use this facility to explore the space of *n*-Vision visualizations more thoroughly.

Acknowledgments

This work is supported in part by the New York State Center for Advanced Technology under Contract NYSSTF-CAT(91)-5, the Center for Telecommunications Research under NSF Grant ECD-88-11111, and an equipment gift from the Hewlett-Packard Company. Earlier work on *n*-Vision was also supported by a gift from Citicorp.

References

[1] Jacques Bertin. *Semiology of Graphics*. The University of Wisconsin Press, 1983. Translated by William J. Berg.

[2] Stephen M. Casner. A task-analytic approach to the automated design of graphic presentations. *ACM Transactions on Graphics*, 10(2):111–151, April 1991.

[3] Steven Feiner and Clifford Beshers. Visualizing *n*-dimensional virtual worlds with *n*-vision. *Computer Graphics*, 24(2):37–38, March 1990.

[4] Steven Feiner and Clifford Beshers. Worlds within worlds: Metaphors for exploring *n*-dimensional virtual worlds. In *Proc. ACM SIGGRAPH Symposium on User Interface Software and Technology (UIST '90)*, pages 76–83, October 1990.

[5] Joseph C. Giarratano. CLIPS user's guide. Artificial Intelligence Section/NASA, Lyndon B. Johnson Space Center, 1988.

[6] John Hull. *Options, Futures, and Other Derivative Securities*. Prentice Hall, 1989.

[7] Sandeep Kochhar, Mark Friedell, and Mark LaPolla. Cooperative, computer-aided design of scientific visualizations. In *Proceedings IEEE Visualization '91*, pages 306–313. IEEE Computer Society Press, October 22–25, 1991.

[8] Jock Mackinlay. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics*, 5(2):110–141, April 1986.

[9] Steven Roth and Joe Mattis. Data characterization for intelligent graphics presentation. In *Proc. CHI '90*, pages 193–200. ACM Press, April 1–5, 1990.

[10] Hikmet Senay and Eve Ignatius. Compositional analysis and synthesis of scientific data visualization techniques. In N. M. Patrikalakis, editor, *Scientific Visualization of Physical Phenomena (Proc. CG International '91)*, pages 269–281. Springer-Verlag, Tokyo, 1991.

[11] Craig Upson, Thomas Faulhaber, Jr., David Kamins, David Laidlaw, David Schlegel, Jeffrey Vroom, Robert Gurwitz, and Andries van Dam. The Application Visualization System: A computational environment for scientific visualization. *IEEE Computer Graphics and Applications*, 9(4):30–42, July 1989.

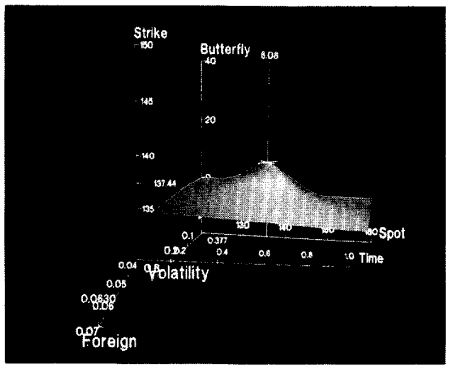

Figure 1: *n*-Vision virtual world with dipstick being used to examine a height field.

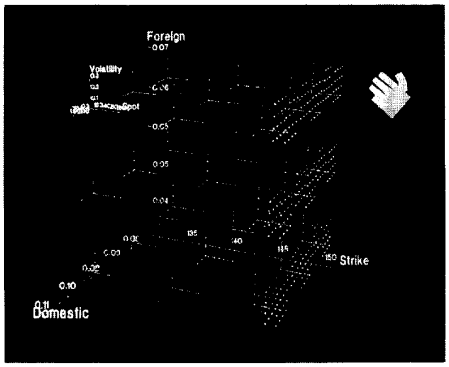

Figure 2: Visualization generated for exploration of an option.

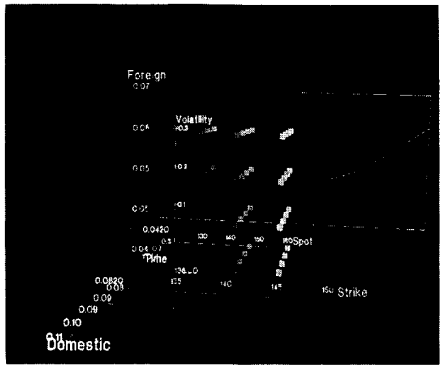

Figure 3: Visualization generated for the search for market conditions.

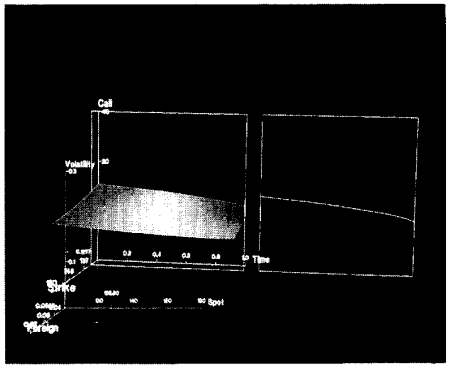

Figure 4: Visualization generated for the search for option parameters.

(See color plates, p. CP-31.)