DATA VISUALIZATION

HISTORY AND DEVELOPMENT

Use of computers to generate graphical representations of data has been around as long as there have been graphical displays. Among the earliest applications was the graphical depiction of the three-dimensional (3-D) structure of molecules in the early 1950s. To enhance the 3-D view, stereoscopic images were produced, which were viewed by either crossing one's eyes or using a viewer that separated left and right eye images. Here as later the impetus for applying graphics was the need to comprehend often complicated spatial structures and relationships. Since then the development of graphical representations of data has moved in step with the development of workstations (and lately the development of personal computers). The big impetus for data visualization was the appearance of interactive 3-D graphics workstations with pipelined geometry hardware and fast, rasterized image displays. Starting in the early 1980s such companies as Silicon Graphics, Stellar, and Pixar began to produce these workstations. At the same time the development of supercomputers like those from Cray and CDC, which were generally available to scientists and engineers (starting in 1985) through the national supercomputer centers, has meant the appearance of more complicated applications generating significantly larger amounts of data.

The culmination of data visualization activity was the seminal 1987 report "Visualization in Scientific Computing" (1) which defined the field and enumerated many of its goals. As a result of this report, visualization came to be used generally to denote the graphical representation of data, and the term *scientific visualization* was coined. Since then the field of visualization has grown rapidly. It now has its own conferences, symposia, and workshops including the IEEE Computer Society Visualization Conferences (2), sessions of the ACM SIGGRAPH conference series, and others. It also has its own journals, including the *IEEE Transactions in Visualization and Computer Graphics, The Visual Computer,* and special issues and reports in *IEEE Computer Graphics and Applications,* among others.

Now the spread and activity of data visualization is quite broad, and it overlaps with many other fields. These include of course computer graphics and such specific areas as volume visualization, medical visualization, flow visualization, and molecular graphics. Data visualization shares many techniques, applications, and modes of analysis with these areas. In addition data visualization has broadened considerably from its original focus on scientific, engineering, and other spatial data. One of its fastest growing areas is in information visualization (3), where the data may not have a spatial distribution at all. It may rather be the content and links of a World Wide Web structure or the products, activity, costs, and so on, in a large inventory database.

VISUALIZATION PROCESS

Richard Hamming has said, "The purpose of computing is insight, not numbers." For visualization, one might instead say, "The purpose of visualization is insight, not pictures." The point is that the end product of visualization is analysis and, ultimately, comprehension rather than a striking picture. In fact a crude picture is quite adequate if it serves the purpose. (It may even be better than a fancy picture if it lacks the artifacts due to rendering technique or coloring, which can obscure important detail in the clutter.) Foley gave a useful and very general definition of data visualization as *a binding or mapping of data to a representation that can be perceived* (4). This emphasizes the process of the mapping of data attributes or variables onto the graphical elements of the visualization. It also highlights the importance of matching the user's perceptual capabilities with the visualization. For example, one should map data to color in such a way that color contrast is good between adjacent features and that shading differences are noticeable even for color-blind people (5). Finally this definition does not restrict itself to visual perception; any sensory perception is valid including auditory, tactile, kinesthetic, and the like. The point is that the process of mapping, if not the final result, is similar for all these representations and that one could effectively use the whole sensory range in depicting data. Modern workstations and PCs are responding to this idea by developing a wider range of multimedia capabilities.

The process of mapping, graphical display, analysis, and interaction is sometimes called the *visualization pipeline* (6). The pipeline includes components for initial data representations, input of these data, filtering or sampling for visualization or analysis, application of visualization procedures, transforming of the data into geometric or image form, and display. The sections below will discuss all these aspects. The visualization pipeline represents both a dataflow and a visualization network. This concept has been implemented into very flexible and widely used tools called *dataflow visualization systems* (discussed later in the article). The components of the pipeline can be separated into process and data objects. The process function includes transformation, collection, sampling, and other operations on the data. The data objects store or prepare data for transfer between pipeline components. This modular functional structure for the pipeline has made it easier to build flexible, efficient visualization tools and also has made environments like the dataflow systems easier to use.

VISUALIZATION METHODS

It is useful to classify data by the number of components or dependent variables that they have for each data point. Thus,

if a dataset has one component per point, it is a scalar dataset; if it has N components per point, it is an N-dimensional vector dataset; if it has $N \times N$ components per point, it is an N-dimensional tensor dataset. If the dataset has multiple components per point but is not arranged like a vector or tensor, it is called *multivariate* data. In addition the dataset might have combinations of scalars, vectors, or tensors per data point; this is also multivariate data. Visualization techniques have been developed to display each of these types of data. [See (7) for many examples of these techniques.]

Commonly scalar, vector, and tensor data are considered in terms of spatial fields. Suppose that one is studying data from a global atmospheric simulation (8). The data elements might be defined at 3-D positions in the atmosphere (e.g., at some latitude, longitude, and altitude above sea level). If the dataset contains a temperature value at each of these spatial points, it would be a scalar field. If it contains 3-D wind vectors at each point, it would be a vector field. If it had a 3×3 stress tensor defined at each point, the components of which describe both pressure and shear forces, it would be a tensor field. Note, however, that data do not have to be arranged in spatial fields. Statistical data, such as are often attacked using information visualization, would have a number of dimensions equal to the number of statistical variables. Further each of the points might have no inherent spatial location. This would be the case, for example, for a dataset containing the height, weight, age, income level, education level, health characteristics, and so on, of a given population of individuals.

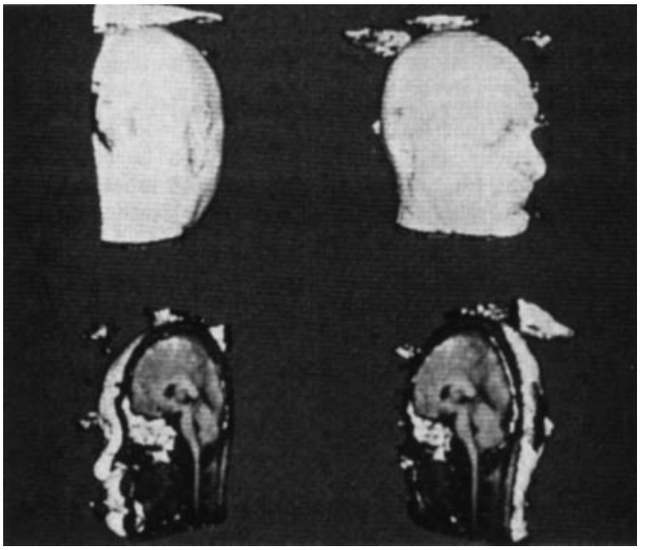


Figure 1. MRI images of a human head using different isovalues and cut planes.

Scalar Methods

Whether data are spatial or not, visualization involves the mapping of these data into 2-D or 3-D space. For 2-D scalar data, or a 2-D slice in 3-D space, this mapping might involve a simple color map. Here a range of the scalar variable is mapped onto a range of a color spectrum, often using simple linear interpolation. One might view two scalar variables simultaneously by using different color ranges (e.g., an orange-red range and a blue-green range) to depict the variables. It can be confusing to the viewer to use more than three color ranges to depict variables. Also different color ranges can bring out or obscure features in the data; they can even introduce visual artifacts such as discontinuities where none exist in continuous data. For a further discussion of the use of color in visualization, see Refs. 4,5.

The depiction of a 3-D scalar field $S = S(x, y, z)$ at a given value S_0 will be a 2-D surface. For a continuous field this will be a continuous surface, called an *isosurface.* (The 2-D analog is an isocontour.) For a given isovalue S_0 there might be multiple surfaces, some of which could be inside others. By changing the isovalue, one could get a depiction of the whole dataset. For example, if one had 3-D magnetic resonance imaging (MRI) data for a human head, one could choose an isovalue that shows the skull and then change the isovalue to show different regions in the brain. This is because the MRI intensity at each point depends on the depth and density at that point. See Fig. 1 for an example.

Volume visualization or volume rendering is often used to depict 3-D scalar fields. Here images are built up by casting rays from each pixel of the display through the data volume. The number and densities of data elements that the ray encounters are mapped to color and transparency values, which are then superposed to give a final color value for each pixel. The result is an overall rendering of the data volume where there are no surfaces explicitly displayed, and the shapes of colored regions represent patterns in the data.

The depiction of 3-D fields often poses difficulties due to the size of the data and the inherent problem of clearly seeing structure "inside" or behind other structure. One could have, for example, a visualization of the brain with outer lobe structure superimposed on inner folds. Or one could have a depiction of molecular orbital structure overlaying inner atomic arrangements and bonds. In both cases there are no definite boundaries or surfaces to depict. On modern graphics workstations, the alpha channel can be used to make outer layers semitransparent and thus reveal inner structure. This approach has the drawback that the shapes of semitransparent layers are hard to discern, as are their depths. More recently there have been methods using textured surfaces at key values of the field, with holes that reveal inner structure (9). The eye can easily translate a regular texture pattern into shape and depth information. See Fig. 2 for an example.

Methods to select parts of the data can be combined with interactivity to reduce dataset size and focus on important details. For example, one can use a cutting plane to define a 2-D slice of data that can then be depicted using color mapping or isocontours (6). The position and orientation of the plane could be interactively set. If the plane can be moved and rapidly updated, one can obtain a sense of the 3-D variations in the data and can also search for features. A 3-D analog of this capability is a 3-D box, moved through the data space and then sized and oriented by *direct manipulation* (where the user interacts directly with the 3-D objects in the scene, rather than indirectly through knobs, sliders, etc., attached to an interface). A generalization of all these capabilities is called a *magic lens* (10,11). Here the user controls an arbitrarily shaped region in 2-D or 3-D space, which changes the appearance of objects viewed through that region. The tool can act either as a lens through which the user looks or

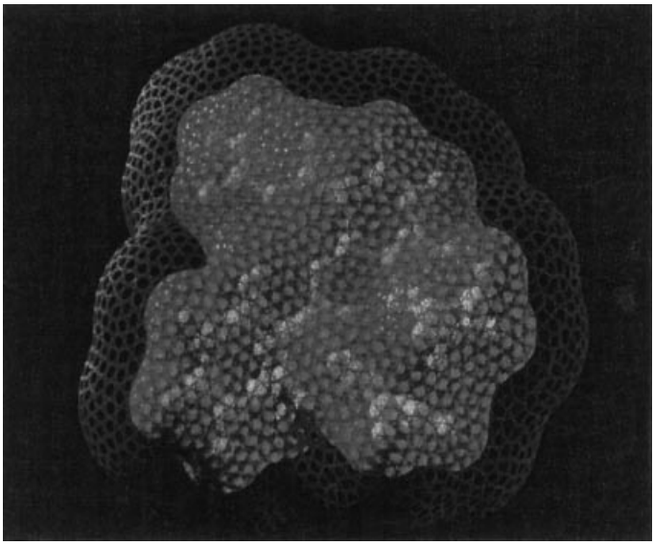


Figure 2. Textured surfaces revealing the structure of molecular orbitals at different values.

as a 3-D space, in either case controlled by the user. The magic lens might provide a magnified view of objects in its path, provide views of variables or annotations hidden otherwise, or offer a more detailed rendering of a variable (see Fig. 3). Magic lenses are applicable to information of all sorts, not just physical data; in the context of information visualization they are sometimes called *table lenses.*

Interactivity is an important component for all types of visualizations, since it allows individuals to efficiently use their eye-brain systems to bring out otherwise hidden relations or to quickly collect information on overall data structure and dynamics. If a scene can be rotated at 10 frames/s (sometimes a lower rate is adequate), the eye-brain system can use parallax to instantly clarify depth relations. Thus, for example, a viewer will not be able to obtain any depth information from a stationary 3-D scatter plot (where information is displayed just as points in 3-D space), but if she starts rotating it at 10 frames/s, the 3-D spatial structure will immediately jump out. In fact, a rocking of the scene by as little as 2° will produce the same effect (12). Another aspect of interactivity is animation, which is obviously effective in displaying the dynamic behavior of time-dependent data. Here, again, a rate of at least 10 frames/s is effective. One can also map almost any variable to time. For example, one might represent pressure changes in a thunderstorm simulation by color and then represent each step in the change of a simulation variable, such as the temperature gradient, with a separate time step. Among other things the resulting animation could show areas where there is a rapid buildup in pressure as the temperature gradient increases. Because of the complexity of the data being rendered and/or the level of the graphics computer doing the rendering, it is often not possible to achieve the requisite level of interactivity. Some of the dataflow visualization systems discussed later in this article have frame collection and playback tools to overcome this problem (e.g., the Sequencer module in the IBM DX). With these the user can collect several frames of an animation, rotation, or other set of interactions and then play them backward or forward at higher rates.

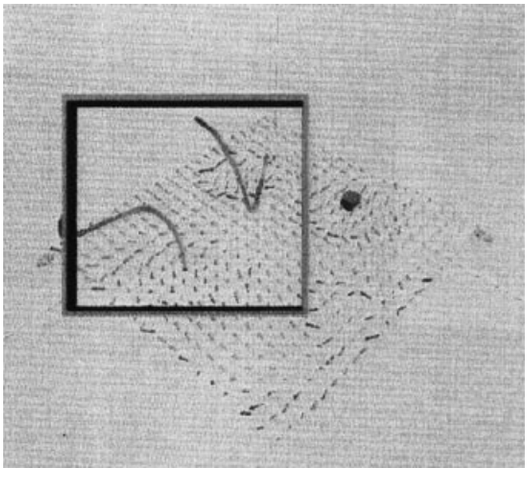


Figure 3. Magic lens revealing additional detail in a data field.

Vector Methods

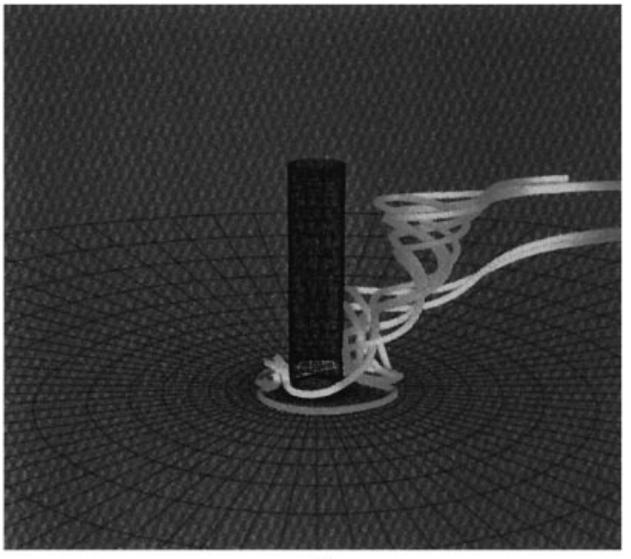
Vectors are N component objects in N-dimensional space; thus we need ways to present all N components at once. A simple way to do this for 2-D or 3-D vectors is to use directed lines, with or without arrows at the end. The problem for 3-D-directed lines is that it is hard to tell whether they are oriented into or out of the image plane. As discussed above, one can use interactivity to rotate the image and reveal this information. Alternatively, one could use a fully 3-D vector shape. In this case foreshortening, shading, and lighting combine to give additional information about orientation.

Vector fields are complicated to represent because they have N components at each data point throughout a 3-D space. Among common methods to present vector fields are *particle traces, streaklines,* and *streamlines* (13). Each of these cases involves the integration of the vector field with respect to time or a timelike parameter. Thus a particle trace would be drawn by integrating the vector field at successive time steps

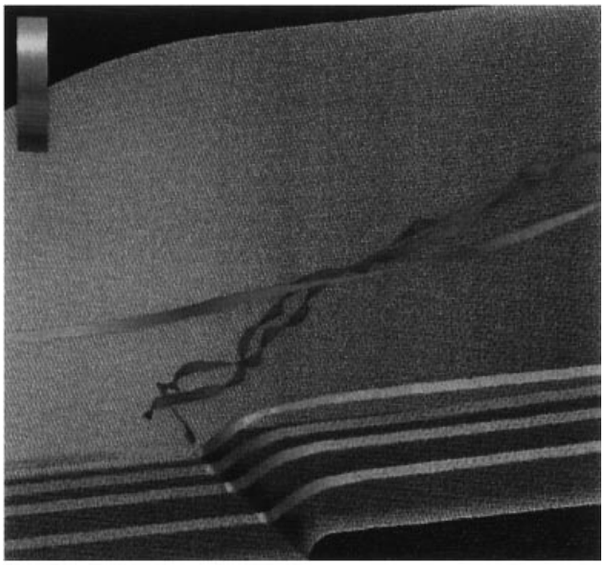
$$\boldsymbol{x}(t) = \int_t \boldsymbol{V}\, d\tau$$

Animation of the resulting trace provides a sampling of both the vector field direction and magnitude at each point the particle passes. Physically this is like inserting gas bubbles in the vector field and then following their motion. Often a probe is used to interactively insert lines of particles in different regions of the field. Streaklines are modifications of the particle trace where continuous lines are traced by the particle (as if they were trailing a tail of dye).

Streamlines are integrals of the vector field, but taken at one point in time. For static vector fields, streamlines provide trajectories that are identical to those for particle traces and streaklines. However, the trajectories differ for time-dependent fields. Streamlines have the property that at every point the vector field is tangent to the streamline. Thus they depict field direction but not magnitude, and one must use color, for



(a)



(b)

Figure 4. (a) Colored streamlines showing the structure and magnitude of flow. (b) Streamribbons showing flat and twisting regions of flow.

example, to show magnitude [see Fig. 4(a)]. In 3-D, streamlines can often have complicated trajectories; for example, they may twist. To bring out such behavior, *streamribbons* are often used. These objects are constructed by rendering two adjacent streamlines and then connecting them with a planar surface. This gives a clear representation of the twisting or vorticity of the vector field as long as the adjacent streamlines don't diverge too much [see Fig. 4(b)].

Tensor and Multivariate Methods

Multivariate datasets have several components, usually more than three per data point. For example, tensors in 3-D space have up to nine independent components. To depict this number of components simultaneously, analysts often turn to *glyphs* (sometimes also called 3-D icons). Glyphs are 2-D or 3-D objects whose graphical attributes (color, position, orientation, size, shape, etc.) are mapped to the variables at each point. They thus tend to be point objects, through higher-dimensional representations are also possible. The idea is to construct the glyph with mappings that are visually distinct so that one can see the mapped variables simultaneously. For example, we have in Fig. 5 a glyph representing several components of tensor and vector fields (14). The direction and length of the vector represent velocity, the twisting stripes on the vector represent rotation, the different colored rings at the base represent direction and magnitude of shear (by comparison to a reference ring) and field divergence (or convergence), and the half ellipsoid at the bottom represents acceleration. These combinations of color and different shapes permit the viewer to see each component distinctly. This "probe glyph" can be placed along streamlines, for example, to probe the structure of a tensor field.

The expressiveness of glyphs can be demonstrated in the following example. Here simulated flow of plastic in an injection mold is depicted (15). Important quantities are the pressure, temperature, and velocity of the flow. The plastic must flow at the correct rate and temperature so that it doesn't harden too quickly or too slowly, and it must fill all parts of the mold uniformly. Each glyph, placed according to a finite-element grid, is a 3-D object with the velocity field represented as a shape distortion of appropriate magnitude and direction along the glyph, and temperature and pressure represented by different color scales on the glyph and its base, respectively [see Fig. 6(a)]. When the injected plastic has hardened, there is no flow and the glyph assumes a round shape. Figure 6(b) displays one time step from the simulation (15). We see that it is quite apparent what regions of the injected plastic are hardened or flowing, the direction of the flow, and the accompanying pressure and temperature values. In addition time step sequences reveal that one can follow in detail the injection process, including the advance of the molten plastic and the effects of pressure and temperature.

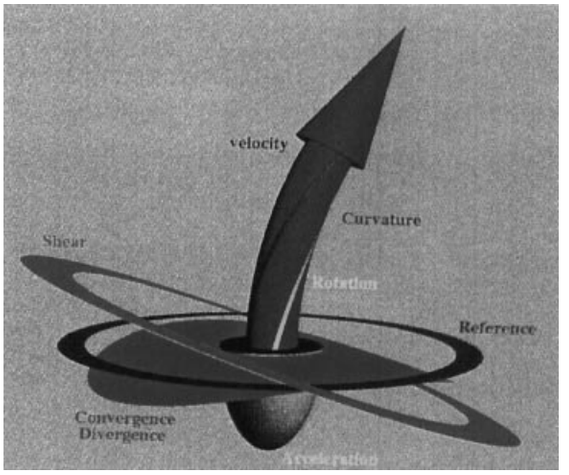


Figure 5. A glyph representation of multiple tensor and vector components.

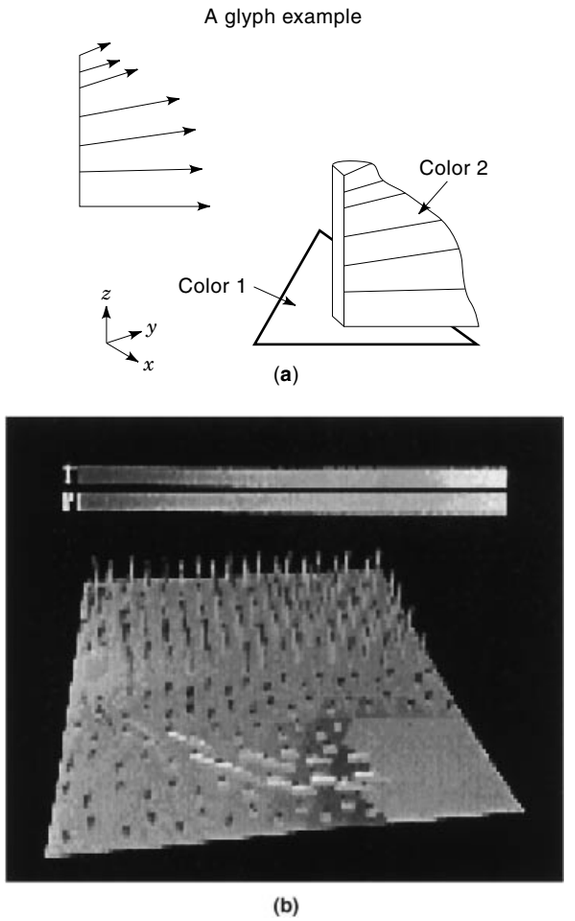


Figure 6. (a) Construction of glyphs for representation of details of injection molding. (b) A time step in the injection molding simulation using the glyphs in Fig. 6(a).

Instead of attempting to display several variables at once in a single view, one can display multiple views, each with different variables, simultaneously. For example one could display side-by-side frames (16) of the time steps from a simulation where one frame contains shear forces with respect to a 3-D object, one frame shows compressive forces, and another frame shows other variables such as temperature or velocity fields. If one then selects a region in one frame, the variables for the selected points are highlighted in the other frames. This interactive technique, called *brushing* (17), can significantly aid correlative analysis between the variables in the simultaneous views and is often used in statistical visualization. Other multivariate techniques are discussed later in this article.

DATAFLOW VISUALIZATION SYSTEMS

How can an individual who does not have experience in graphics, visualization, or even computer programming employ the methods discussed in this article for data analysis? To answer this question, a variety of visualization tools have been developed on the premise that users should need little other than knowledge of their data to employ them. These tools will be the focus of this section. However, there are also toolkits requiring some programming expertise in order to put their modules together into visualization programs. The advantage of these over the higher-level systems discussed next is that they give significantly greater flexibility in designing and optimizing visualization capabilities for specific applications and datasets. One of the most widely used is the *Visualization Toolkit* (VTK) (6), which offers a broad range of visualization tools (with source code) for 2-D and 3-D data including those for contouring, surface smoothing, cutting, slicing, decimation, triangulation, volume rendering, building Web-based visualizations, among others. VTK can be used on a broad variety of platforms including those running Windows 95, Windows NT, IRIX, Solaris, HP-UX, Digital UNIX, IBM AIX; it also has Java bindings. The tools and their data structures are constructed in an object-oriented fashion. The user should know C++ to most effectively use VTK. However, Ref. 6 is also a good source with detailed descriptions of visualization techniques (including examples and the complete toolkit modules on an accompanying compact disk).

The most successful, high-level data visualization tools have been *dataflow visualization systems.* Among the systems in wide use today are AVS (18), IBM Visualization Data Explorer (IBM DX) (19), SGI Iris Explorer (20), Khoros (21), and Vis-AD (22). These use *visual programming* methods whereby the user sets up a program by direct manipulation of graphical icons. The dataflow systems are based on *directed acyclic graphs* (DAGs) that map the flow of data from a source module to a sink module. The source module is usually a data reader, and the sink is usually a display module. A schematic for a simple dataflow map is shown in Fig. 7(a); and an actual interface with map for the IBM DX is shown in Fig. 7(b). As can be seen from the schematic, one can set up a simple visualization just by linking together appropriate modules and without any knowledge of the underlying program structure or data-handling characteristics. Each module may also have controls and even its own interface. In addition most of the dataflow systems permit the user to build her own interface (employing a simple graphical interface builder) that might control a collection of modules.

One can see that the dataflow approach is highly flexible and allows rapid building of visualization and analysis tools without programming knowledge. In most systems there is also the capability to build one's own customized modules in C or Fortran; the system then automatically generates appropriate wrappers for these so that they can be used just like other modules. This capability has led to large libraries of user-developed modules for systems such as AVS, IBM DX, and Iris Explorer. There are also newsgroups and Web pages devoted to these systems, and specific applications such as molecular chemistry or biomedical visualization. In addition most of the systems come with capability for distributed, networked operation. Thus one could run AVS, for example, on a large computational server and on a graphics workstation. Modules constructed in the same dataflow map could be on the different machines and still pass data. To take advantage of wide networking availability, dataflow systems such as IBM DX even provide Java applet front-ends with VRML controls and 3-D display.

Such high-level yet flexible and powerful systems are bound to have some drawbacks. One comes from the extensibility of the system and thus the proliferation of modules.

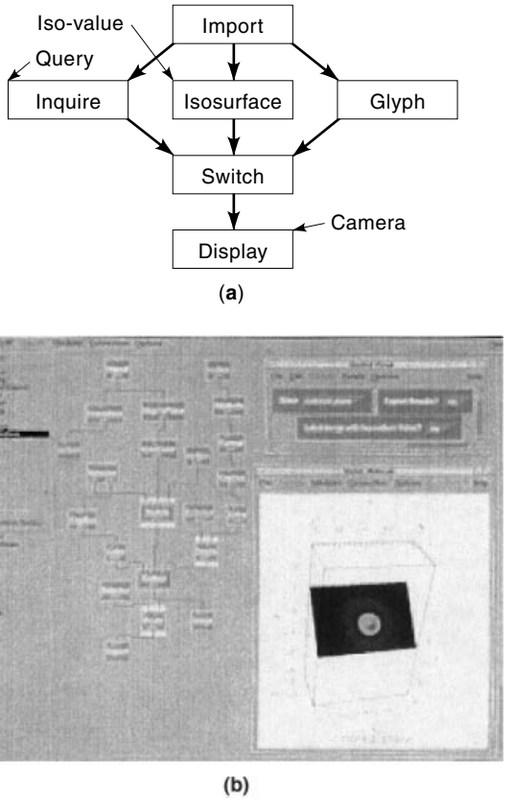


Figure 7. (a) A schematic of a simple dataflow map. (b) The actual interface with map for the IBM Visualization Data Explorer.

Some libraries now have a thousand or more modules. Such richness of capability means that there can easily be millions of ways to hook these units together in a map. The nonexpert user can quickly be overwhelmed. To help overcome this problem, improved organization and cataloguing of modules have been developed. In addition there is work on expert interfaces that suggest or refine visualization maps by asking the user leading questions about her needs. Another drawback is that high-level, modular systems cannot be as efficient as carefully integrated tools developed for specific applications. This is especially so in the data-passing part of the system and becomes a problem as datasets get larger. Most of the dataflow systems have developed improved capabilities, like caching and improved use of pointers, so that data do not need to be copied repeatedly. In addition there is work to handle data so large that they cannot be contained in memory or may even be distributed.

A final drawback that occurs quite frequently is that the user has trouble getting data read by the system and thus cannot even begin the visualization process. Typically application data will come in a form somewhat different than that accepted by the dataflow system readers. Even though the readers are often flexible in the formats they accept and the differences may be small, nonexpert users may still be confused. Indeed this problem may occur with any visualization tools. Much of this problem can be cleared up by simply describing the concepts used in graphical data representations, since the confusion is often just a matter of terminology. We thus give an overview of these concepts in the next section.

DATA REPRESENTATIONS

Data for visualization systems and for many applications are organized according to several attributes including dimensionality (number of independent variables), attributes or dependent variables (the data themselves at a given point, e.g., temperature or energy); type (byte, integer, float or real, complex, etc.), geometry (actual position and shape information for data objects), topology or mesh structure (general connectivity information that is invariant under certain operations); see Refs. 6,23,24. In addition there may be attributes associated with each variable, such as rank (scalar, vector, or tensor), or with groups of variables, such as aggregation (collections of independent variables or geometric structures treated as a single entity). One should distinguish between geometry and topology. The latter remains the same under geometric transformations such as rotation, translation, or scaling while the former does not. Thus the objects in Fig. 8 have the same topology, they are both quadrilaterals, even though they have different geometries (e.g., different orientations, scales, and angles). The ordering of the set of points convey the topological information. Thus in our example, there is an edge between 0 and 1, between 1 and 2, and so on.

There are a variety of topologies or mesh structures that appear not only in visualization but also in finite element simulations, computational fluid dynamics, and other applications that use meshes. Figure 9 shows some of these mesh structures, which are enumerated in the following list. Several of these structures are known by more than one name.

- *Rectilinear or Deformed Regular Grid.* A grid where the topology is regular and parallel to the global x, y, z coordinate system, but the geometry is only partially regular.
- *Structured Points or a Regular Grid.* Points arranged on a regular rectangular lattice or parallelepiped.
- *Unstructured or Scattered Points.* 2-D or 3-D scattered data with no connectivity and no topology.
- *Structured or Deformed Regular or Curvilinear Grid.* A grid where the topology is regular, but the geometry is irregular (could be defined by an implicit function).
- *Unstructured or Irregular Grid.* A grid where both topology and geometry are unstructured. Any cell type can be used in arbitrary combinations, so any dataset with connectivity information can be expressed as an unstructured grid. A *triangular irregular network* (TIN) is a type of unstructured grid, but more generally it can have a mix of cell types as shown in Fig. 9.

These mesh structures imply different data storage formats. However, they are all usually arranged as contiguous arrays

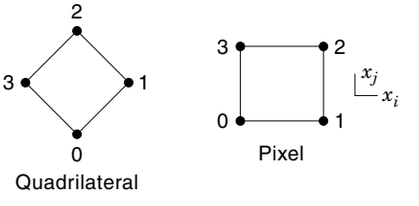


Figure 8. An example showing the difference between topology and geometry.

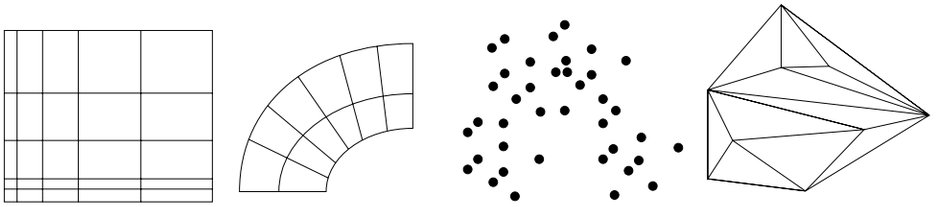


Figure 9. Examples of data mesh structures.

with a header that contains the total size of the array, its type, the number of attributes per data point, and so on. Following the conventions of C, the arrays are often 0-offset; that is, n data values would have IDs: 0, 1, 2, . . ., $n - 1$. The array structure for 3-D unstructured data, for example, might consist of an x, y, z position for each point followed by the attributes for that point. Often there are two arrays, one for positions and one for attributes. (The latter could also be multiple arrays, one for each attribute.) Unless one wanted to display such data as, say, colored points or glyphs, it would be necessary to derive a connectivity structure for these data. Thus, for example, one might resample the data on a regular grid and then apply an isosurface algorithm. The format for a regular grid would not need explicit position coordinates for each point. The header would contain the shape (number of cells in the x, y, and z directions), the data spacing increment in each direction, and the origin of the grid. The array would then contain a list of attributes, usually in increasing x, then y, then z. A rectilinear grid would be arranged like the regular grid (with implicit connectivity structure) but would have an additional position array (6).

INFORMATION VISUALIZATION

Information visualization is the application of 2-D and 3-D visualization techniques to information, whether these data are spatial or not. With this broad definition, scientific or engineering data visualization would also fall under the information visualization mantle. More typically information visualization is understood to deal with data that has one or more nonspatial components. Thus the inventories of a chain of stores, whether or not their geographical distribution is taken into account, would be a subject for information visualization. As would be expected, many of the techniques discussed in other sections of this article could also be used for information visualization. However, information visualization has also brought forth other techniques that are particularly useful for the types of data attacked in this field (3).

The research and interest in information visualization has grown quite fast due to the need to rapidly analyze and understand ever-growing information collections. This has engendered an ongoing symposium in the subject (24) as well as paper sessions in the IEEE Visualization conferences (2). In addition graphics workstations have become cheaper and more widely available, and PCs with 3-D graphics are now appearing. As a result interactive graphics capability is moving into offices and workplaces where the focus is on analyzing nonspatial information.

The information to be visualized can take the form of anything from spreadsheets to the text of novels. With this wide range it is useful to provide a taxonomy for the types of information and visualization techniques (25). Data can be divided into three categories: normal (N; are only equal or not equal to other values), ordered (O; obey an ordering relation), or quantitative (Q; one can do arithmetic on them). Visualization of these data are basically made from marks and their graphical properties. The types of marks are point, line, area, surface, and volume. The properties are color or size. All must be mapped in a 2-D or 3-D space for display (e.g., xy plane or $xyzt$—3-D space plus time). This general graphical taxonomy of course applies also to the visualization techniques described throughout this article. Just as with physical data, interactive techniques that control view, focus, or time are of importance in information visualization for exploration and revealing of detail.

Multidimensional plots are an information visualization technique that involves mapping nonspatial data onto point marks in the xy plane. The result is often a traditional scatterplot, frequently used in statistical visualization. One example of this technique is FilmFinder (26), where a scatterplot has been turned into an interactive, exploratory visualization by the use of sliders and buttons that control a filter function determining which films are shown on the scatterplot. The filter function controls a set of variables such as title (O), year (Q), rating (N), type (N), and others. As one moves the slider, the display is instantaneously updated so that one can move rapidly through a large amount of information. This interface, with its tight limits on update times, is called a *dynamic query interface* (27).

Another way of showing higher-dimensional data is *Worlds Within Worlds* (28). A series of nested coordinate systems is set up. Variables are mapped to the spatial dimensions in each coordinate system. Thus, if we had six variables, there would be two coordinate systems whose mappings might be variables a, b, c for the outer system and d, e, f for the inner one—namely we would have a function $f(a, b, c, d, e, f)$. If one positions the origin of the inner coordinate system at, say, a_0, b_0, c_0 with respect to the outer one, the function is $f(a_0, b_0, c_0, d, e, f)$. As we move the inner coordinate system around, we get other values for a, b, c. This is a way to explore a six-dimensional space. See Fig. 10 for an example. Worlds Within Worlds provides an overlapped coordinate space, which is a kind of *details-on-demand* approach. Overlapping is a useful visualization technique but must be used with care so that the user can perceive how the spatial dimensions are being used. Fast updates for any movement of coordinate systems are quite helpful here.

Yet another useful information visualization technique is the *information landscape,* where two independent variables are mapped to the XY coordinates of a surface; a dependent variable can then be used for the height coordinate, giving a height field representation. The themescapes visualization

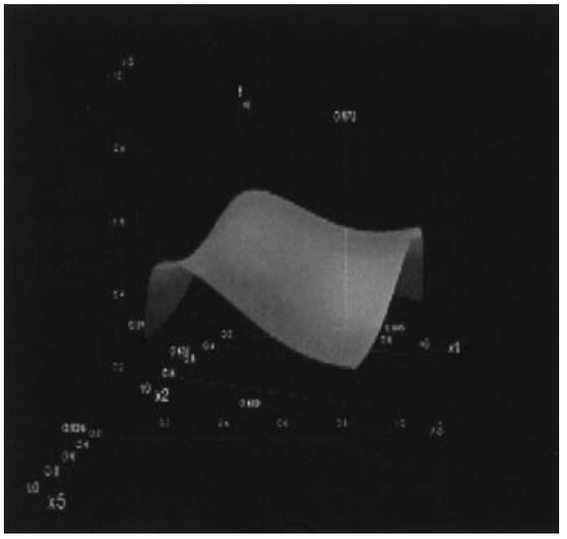


Figure 10. Representation of a multivariate function using the Worlds within Worlds overlapping coordinates system.

(29) is an information landscape example. The terrain visualization techniques outlined in the next section can be of use for handling large information landscapes. In addition there are *Cone Trees* (30) and other representations for showing large organizational structures. All the techniques outlined here are just a sampling of information visualization methods. For more details, see Refs. 3 and 24.

TERRAIN VISUALIZATION

The display of geospatial terrain, including elevation data, phototexture imagery or maps, and objects on the terrain, has enjoyed significant recent growth. One reason is the explosive growth of available digital terrain image and elevation data. The US Geologic Survey, for example, has on-line repositories of world data at 8 km resolution and US data at 1 km resolution. There are commercial sources with data at 100 or even 10 m resolution. And the future will really be big. Commercial satellites will be going up that will collect elevation and image data from anywhere at 1 m resolution (for a fee).

The input to the terrain visualization is usually a large digital terrain map (DTM), consisting of elevation data sampled on a regular or irregular grid, and corresponding texture data which are mapped onto the reconstructed terrain surface. The output is rendered images of the terrain surface, usually as part of a "flythrough" sequence that is often interactive. Terrain visualization is a difficult problem for applications requiring accurate images of large datasets at high frame rates because a complete visualization system must have components to manage disk paging of geometry and texture (because the datasets are too large to reside in memory), level-of-detail (LOD) selection for texture blocks, LOD for triangle geometry, culling to the view frustum (the volume containing the parts of a scene in the view of the user at a given moment), and triangle stripping (30,31). On current graphics hardware, the problem is to maintain dynamic, view-dependent triangle meshes, and texture maps that produce good images at the required frame rate.

In recent years one of the main research areas in terrain visualization is to develop multiresolution terrain representations that can be used to build adaptive triangle meshes for interactive, view-dependent rendering (32,33). View-dependence means rendering according to the user's viewpoint. Objects that project to small areas on the screen are rendered at lower detail. An adaptive terrain meshing algorithm is needed for this. Almost all existing algorithms are developed to rely upon a hierarchical model representation in which objects are described at multiple levels of detail and can be drawn with various rendering algorithms. The idea behind recent algorithms is to adjust image quality adaptively to maintain a uniform, user-specified target frame rate (32,33).

Among hierarchical representations, the quadtree is most often used for terrain (34). In one approach the quadtree representation is used to preprocess the terrain height field on a uniform grid. Vertices at each quadtree level are computed using an approximate least-squares fit to the level below. For each frame at run time, a priority queue drives quadtree refinement top-down from the root, thus allowing specified triangle counts to be achieved directly. The priority for a quadtree element is a heuristic involving view-independent (error in surface) and view-dependent (screen-area coverage) components aimed at minimizing the squared error in output image pixel intensities (35).

In another approach one chooses continuous triangle-bintree meshes, using a compact and efficient regular grid representation and employing a variable screen-space threshold to bound the maximum error of the projected image (32). A coarse level of simplification is performed to select discrete levels of detail for blocks of the surface mesh, followed by further simplification through repolygonalization in which individual mesh vertices are considered for removal. These steps compute and generate the appropriate level of detail dynamically in real-time, minimizing the number of rendered polygons and allowing for smooth changes in resolution across areas of surface. Reductions in detail of a factor of 100 or more are possible without noticeable loss in image quality, and one can fly in continuously from a global overview to a view at 1 M resolution or less, as shown in Fig. 11 (30). The

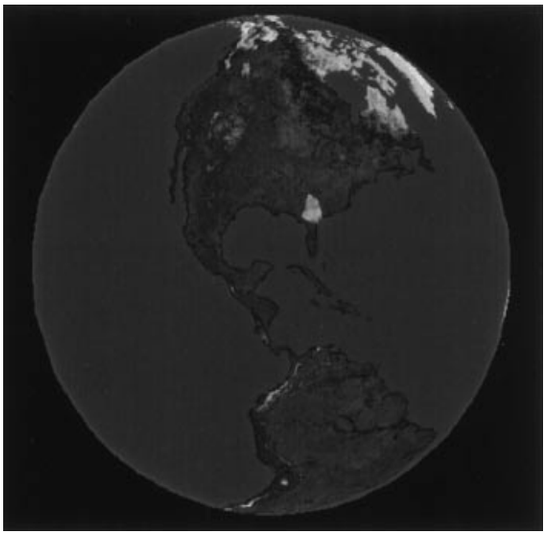


Figure 11. Global view of hierarchical data structure that one can navigate continuously to views at 1 m resolution or less.

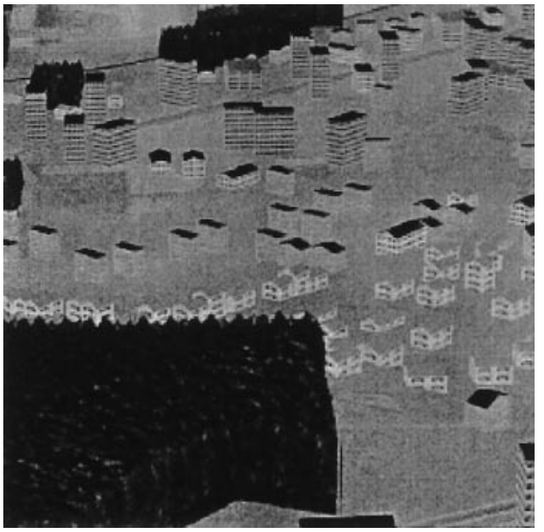


Figure 12. A flythrough of a cityscape with terrain, buildings, and roads.

regular grid can also be much more compact than the data representations of other methods, and the preprocessing stage can be significantly less time-consuming.

Yet another approach uses a hierarchical triangular-irregular-network (TIN) to represent the terrain mesh (33). This is in some sense the "optimal" triangulation for the mesh and thus requires fewer triangles than other methods for a given terrain. Two of the most comon methods are a base metric derived from the edge-collapse operations inherent in progressive meshes, which gives only a loose heuristic estimate of geometric or parametric screen-space distortions, and a metric that separates nondirectional and normal-direction errors to measure errors in approximating nonlinear texture coordinate mappings. The TIN data structure can also have "near/far" annotations for vertex morphing (geomorphing), along with queue-driven top-down refinement procedure for building the triangle mesh for a scene. The method applies to general base (coarsest-level) triangle meshes. The vertex morphing capabilities are powerful, since they permit further simplification of the terrain mesh without distracting "popping" of features as the algorithm switches from one LOD to another.

Once one has accurate displays of high resolution terrain, one wants to populate the landscape with buildings, trees, roads, moving vehicles, and myriad other objects. For example, one might want interactive flythroughs of cityscapes with accurate placement of buildings and roads, as shown in Fig. 12. To handle this detail, which is quite different from terrain detail, new methods have been developed. These include using generic textures and building types to model the cityscape, with landmark buildings rendered more accurately (36). Another method uses background images, something like the backdrop paintings that were used to fill out landscapes in older movies. These images are cut and warped to take into account the user's moving viewpoint (37). With this method new background images need be created only occasionally rather than for every frame.

Terrain visualization techniques that can handle fairly large amounts of high-resolution terrain are now appearing in some software packages, such as SGI's Performer. Also packages such as Multigen, used for the development of 3-D models, permit the development of multiresolution terrain datasets. In the future there will be packages that can address global terrain and provide the capability to handle very large datasets.

CURRENT AND FUTURE DIRECTIONS

Terrain is just one of the data applications where immense data size must be handled at interactive rates. Other fields also face this need. In the area of scientific visualization, the input datasets are often very large, such as in computational fluid dynamics (CFD). To address these very large datasets, out-of-core visualization techniques are being developed (38).

In addition there is continuing work on *visual steering* of computations (39,40). Visual steering denotes two-way communication through direct manipulation of graphical representations of data to bring about user involvement with the calculations as they occur. The typical approach to visualization, involving postprocessing data in static files, does not work here. One must be able to insert on-the-fly parameter changes, even over 4-D regions of the simulation and see results dynamically updated. This iterative push-pull between control and response greatly improves analysis, locates subtle errors, and contributes to deeper understanding of the simulated processes. Large-scale simulations are now so complex that often the contributing physical processes are not well understood, even by the experts who run them; studies using controlled simulations allow focus on the detailed process mechanisms and thus illuminate their workings.

The landscape is rapidly changing for computer graphics and visualization. What was once only available on high-end workstations is now appearing on desktop systems and even PCs. Certainly PC graphics has arrived with substantive capability and real visualization applications (41). In the coming years workstations or personal systems will be defined by their capabilities and focus, not by their operating systems. As a result a sort of grand unification is taking place so that, for the casual consumer, the line between UNIX and the latest versions of Windows will be blurred to the point of irrelevance. Graphics applications and customers will soon be much more numerous than they are now, and the traditional graphics and visualization markets will become a niche. This does not mean necessarily that traditional vendors of graphics hardware and software will struggle, but rather that all segments of the market will grow. However, new consumer-targeted applications will grow by far the fastest. Since PC home office, business, and Web products are already here with their huge markets, we can expect to see an integration of 3-D graphics with these tools. If we are lucky, we will see completely new tools as well, such as interactive visual browsers that permit one to quickly explore vast collections of files (a PC may soon be able to hold tens of thousands), knowledge bases, multimedia stores, and deep Web structures.

BIBLIOGRAPHY

1. B. H. McCormick, T. A. DeFanti, and M. Brown, Visualization in scientific computing, *Comput. Graph.,* **21**: 1987.

2. See *IEEE Visualization Conf. Proc.* (starting in 1990; e.g., *IEEE Visualization '95*) for a broad cross-section of research papers and case studies dealing with data visualization.
3. S. K. Card, J. Mackinlay, and B. Schneiderman, *Readings in Information Visualization,* San Francisco: Morgan Kaufmann, 1998.
4. J. D. Foley and W. Ribarsky, Next generation data visualization tools, in L. Rosenblum et al. (eds.), *Scientific Visualization: Advances and Challenges,* London: Academic Press, 1994, pp. 103–127.
5. H. Levkowitz and G. Herman, GLHS: A generalized lightness, hue, and saturation color model, *CVGIP: Graph. Models Image Process.,* **55**: 271–285, 1993.
6. W. Schroeder, K. Martin, and W. Lorenson, *The Visualization Toolkit,* 2nd ed., Upper Saddle River, NJ: Prentice-Hall, 1998.
7. P. Keller and M. Keller, *Visual Cues,* Los Alamitos, CA: IEEE Computer Society Press, 1992.
8. L. Treinish, G. Nielson, and D. Bergeron, Visualization of stratospheric ozone depletion and the polar vortex, *Proc. Visual. '93,* 1993, pp. 391–396.
9. P. Rheingans, Opacity-modulating triangular textures for irregular surfaces, *Proc. Visual. '96,* 1996, pp. 219–225.
10. M. C. Stone et al., The movable filter as a user interface tool, *Proc. CHI '94,* 1994, pp. 306–312.
11. M. M. Loughlin and J. F. Hughes, An annotation system for 3-D fluid flow visualization, *Proc. IEEE Visual. '94,* 1994, pp. 273–279.
12. E. J. Farrell, Visual interpretation of complex data, *IBM Syst. J.,* **26**: 174–200, 1987.
13. J. Helman and L. Hesselink, Representation and display of vector field topology in fluid flow data sets, *Visualization in Scientific Computing,* Los Alamitos, CA: IEEE Computer Society Press, 1990, pp. 61–73.
14. W. C. Leeuw and J. J. van Wijk, A probe for local flow field visualization, *Proc. Visual. '93,* San Jose, 1993, pp. 39–45.
15. R. Ellson and D. Cox, Visualization of injection molding, *Simulation,* **51**: 184–188, 1988.
16. P. C. Chen, Climate and weather simulations and data visualization using a supercomputer, workstations and microcomputers, *Proc. SPIE,* **2656**: 254–264, 1996.
17. A. Buja et al., Interactive data visualization using focusing and linking, *Proc. Visual. '91,* San Diego, 1991, pp. 156–163.
18. C. Upson et al., The application visualization system: A computational environment for scientific visualization, *IEEE Comput. Graph. Appl.,* **9** (4): 30–42, 1989.
19. G. Abram and L. Treinish, An extended data-flow architecture for data analysis and visualization, *Proc. Visual. '95,* 1995, pp. 263–270.
20. Silicon Graphics Computer Systems, *Iris Explorer User's Guide,* Document 007-1369030 (1993).
21. K. Konstantinides and J. R. Rasure, The Khoros software development environment for image and signal processing, *IEEE Trans. Image Process.,* **3**: 243–252, 1994.
22. W. L. Hibbard et al., The VIS-AD data model: Integrating metadata and polymorphic display with a scientific programming language, Proc. Workshop on Database Issues for Data Visualization, *IEEE Visual. '93,* 1994, pp. 37–68.
23. R. B. Haber, B. Lucas, and N. Collins, A data model for scientific visualization with provisions for regular and irregular grids, *Proc. Visual. '91,* 1991, pp. 298–305.
24. See the *IEEE Inf. Visualization Symp. Proc.* (starting in 1995; e.g. *IEEE InfoVis '95*).
25. S. K. Card and J. Mackinlay, The structure of the information visualization design space, *Proc. 1997 IEEE Symp. Inf. Visual.,* 1997, pp. 92–99.
26. C. Ahlberg and B. Shneiderman, Visual information seeking: Tight coupling of dynamic query filters with Starfield displays, *Proc. CHI'94 Conf. Human Factors Comput. Syst.,* 1994, pp. 313–317.
27. E. Tanin, R. Beigel, and B. Schneiderman, Research report: Design and evaluation of incremental structures and algorithms for dynamic query interfaces, *Proc. 1997 IEEE Symp. Inf. Visual.,* 1997, pp. 81–86.
28. C. Beshers and S. Feiner, Autovisual: Rule-based design of interactive multivariate visualizations, *IEEE Comput. Graph. Appl.,* **13**: 41–49, 1993.
29. J. A. Wise et al., Visualizing the non-visual: Spatial analysis and interaction with information from text documents, *Proc. 1995 IEEE Symp. Inf. Visual.,* 1995, pp. 51–58.
30. P. Lindstrom et al., An integrated global GIS and visual simulation system, Report GIT-GVU-97-07, *Trans. Visual. Comput. Graph.,* submitted.
31. J. S. Falby et al., NPSNET: Hierarchical data structures for real-time three-dimensional visuali simulation, *Comput. Graph.,* **17**: 65–69, 1993.
32. P. Lindstrom et al., Real-time continuous level of detail rendering of height fields, *Proc. SIGGRAPH'97,* 1996, pp. 109–118.
33. H. Hoppe et al., Mesh optimization, *Proc. ACM SIGGRAPH '93 Conf. Comput. Graph.,* 1993, pp. 19–26.
34. H. Samet, The quadtree and related hierarchical data structures, *ACM Comput. Surveys,* **16**: 187–260, 1984.
35. M. C. Miller, *Multiscale Compression of Digital Terrain Data to Meet Real Time Rendering Rate Constraints,* PhD thesis, Univ. California, Davis, 1995.
36. M. Suter and N. Nuesch, Automated generation of visual simulation databases using remote sensing and GIS, *Proc. IEEE Visual. Conf.,* 1995, pp. 86–93.
37. F. Sillion, G. Drettakis, and B. Bodelet, Efficient impostor manipulation for real-time visualization of urban scenery, *Comput. Graph. Forum '97,* **16**: C207–C218, 1997.
38. M. Cox and D. Ellsworth, Application-controlled demand paging for out-of-core visualization, *Proc. IEEE Visual. Conf.,* 1997, pp. 235–244.
39. J. D. Mulder and J. J. van Wijk, 3-D computational steering with parameterized geometric objects, *Proc. IEEE Visual. '95,* 1995, pp. 304–311.
40. Song Zou et al., Collaboration and visual steering of simulations, *Proc. SPIE Conf. Visual Data Exploration Anal. IV,* 1997, pp. 274–285.
41. W. Ribarsky, The times they are a-changing: PC graphics moves in, *IEEE Comput. Graph. Appl.,* **18**: 20–25, 1998.

WILLIAM RIBARSKY
TIAN YUE JIANG
Georgia Institute of Technology