Direct Manipulation in Virtual Reality

Steve Bryson, NASA Ames Research

1. Introduction

1.1. Direct Manipulation in Virtual Reality for Scientific Visualization

Virtual Reality interfaces offer several advantages for scientific visualization such as the ability to perceive three-dimensional data structures in a natural way [1][2][7][11][14]. The focus of this chapter is *direct manipulation,* the ability for a user in virtual reality to control objects in the virtual environment in a direct and natural way, much as objects are manipulated in the real world. Direct manipulation provides many advantages for the exploration of complex, multi-dimensional data sets, by allowing the investigator the ability to intuitively explore the data environment.

Because direct manipulation is essentially a control interface, it is better suited for the exploration and analysis of a data set than for the publishing or communication of features found in that data set. Thus direct manipulation is most relevant to the analysis of complex data that fills a volume of three-dimensional space, such as a fluid flow data set. Direct manipulation allows the intuitive exploration of that data, which facilitates the discovery of data features that would be difficult to find using more conventional visualization methods. Using a direct manipulation interface in virtual reality, an investigator can, for example, move a "data probe" about in space, watching the results and getting a sense of how the data varies within its spatial volume.

Throughout this chapter, in order to focus the discussion we will use the example of a data probe of a vector field in three-dimensional space that emits streamlines of that vector field. The user is allowed to move the data probe anywhere in three-dimensional space, and in response to that movement several operations must occur:

- **Collision Detection:** the system must identify that the user has "picked up" or "moved" the data probe.
- **Data Access:** for a given spatial position of the data probe, the system must locate the data (vector data in our example) for that location and access that data, as well as all data involved in the visualization computation.
- **Visualization Computation:** the geometry of the visualization (the streamline in our example) must be computed.
- **Graphical Rendering:** the entire graphical environment must be re-rendered from the viewpoint of the user's current head position.

In order for the user to have a sense that the object is moving with the user's hand position, the above process must happen quickly with low latency. Additional issues include the design of the data probes by the designer of the visualization environment and the possibility that the virtual environment may be implemented on a distributed system.

In this chapter we will explore the design and implementation of direct manipulation interfaces useful for scientific visualization. The steps described above combined with the low-latency human factors requirements place demands that result in several design challenges. After providing a simple model of the visualization process will develop implementation strategies tailored for scientific visualization in virtual environments, including issues of run-time software architectures, distribution, and control of time flow. These implementation strategies will be driven by consideration of the human factors of interaction.

1.2. The Data Analysis Pipeline

In order to describe the special issues that arise in the implementation of a direct-manipulation based scientific visualization system in virtual reality, we require a conceptual model of a scientific visualization system. There are many ways to conceptualize scientific visualization, and we make no claim to present the most complete or optimal conceptualization. We have, however, found the following model very informative when considering implementation issues.

We consider the scientific visualization process as a pipeline, which in its most generic form starts with the data to be visualized. From this data visualization primitives are extracted. These primitives may consist of vertices in a polygonal representation, text for a numerical display, or a bitmap resulting from, for example, a direct volume representation. Primitive extraction typically involves many queries for data values. The extracted primitives are then rendered to a display. This pipeline allows user control of all functions, from data selection through primitive extraction to rendering. We show this pipeline in figure 1.

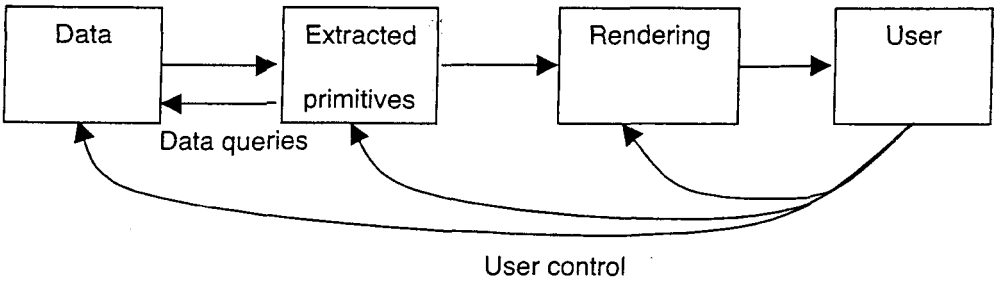

Figure 1: The Data Analysis Pipeline

Let's examine the operation of this pipeline in our example of streamlines of a vector field. Given a starting point of a streamline, data (the vectors at that point) are accessed by the streamline algorithm. The vector value is then added (sometimes with a complex high-accuracy algorithm) to the starting point, creating a line primitive. This process is iterated to build up a (typically curved) line with many vertices. These vertices are the streamline's extracted geometrical representation. These vertices are then rendered in the visualization scene. The extraction of these primitives may involve significant computation even though the data may exist as a pre-computed file. Computations like those in this example will turn out to be a significant issue in the implementation of scientific visualization in virtual environments.

1.3. Advantages of Direct Manipulation in a Virtual Environment

Direct manipulation in a virtual environment offers several advantages for many classes of scientific visualization. Three-dimensional interaction techniques common in virtual environments provide natural ways to control visualization selection and control in 3D. In addition our experience has shown that one of

the greatest advantages of scientific visualization in virtual environments is the inherent "near-real-time" responsiveness required by a head-tracked direct-manipulation based virtual environment. This responsiveness allows rapid queries of data in regions of interest. Maintaining this responsiveness in a scientific visualization environment is the most challenging aspect of such an application and will be one of the primary foci of this chapter. When designed well, the combination of three-dimensional display, three-dimensional interaction and rapid response creates an intuitive environment for exploration and demonstration.

Figure 2 shows the Virtual Windtunnel [1][5], an example of scientific visualization in virtual which makes extensive use of direct manipulation. The Virtual Windtunnel is used to investigate the results of simulations in Computational Fluid Dynamics. This examples exhibits the use of multiple visualization extracts in a single environment, all of which are interactive via visualization widgets.

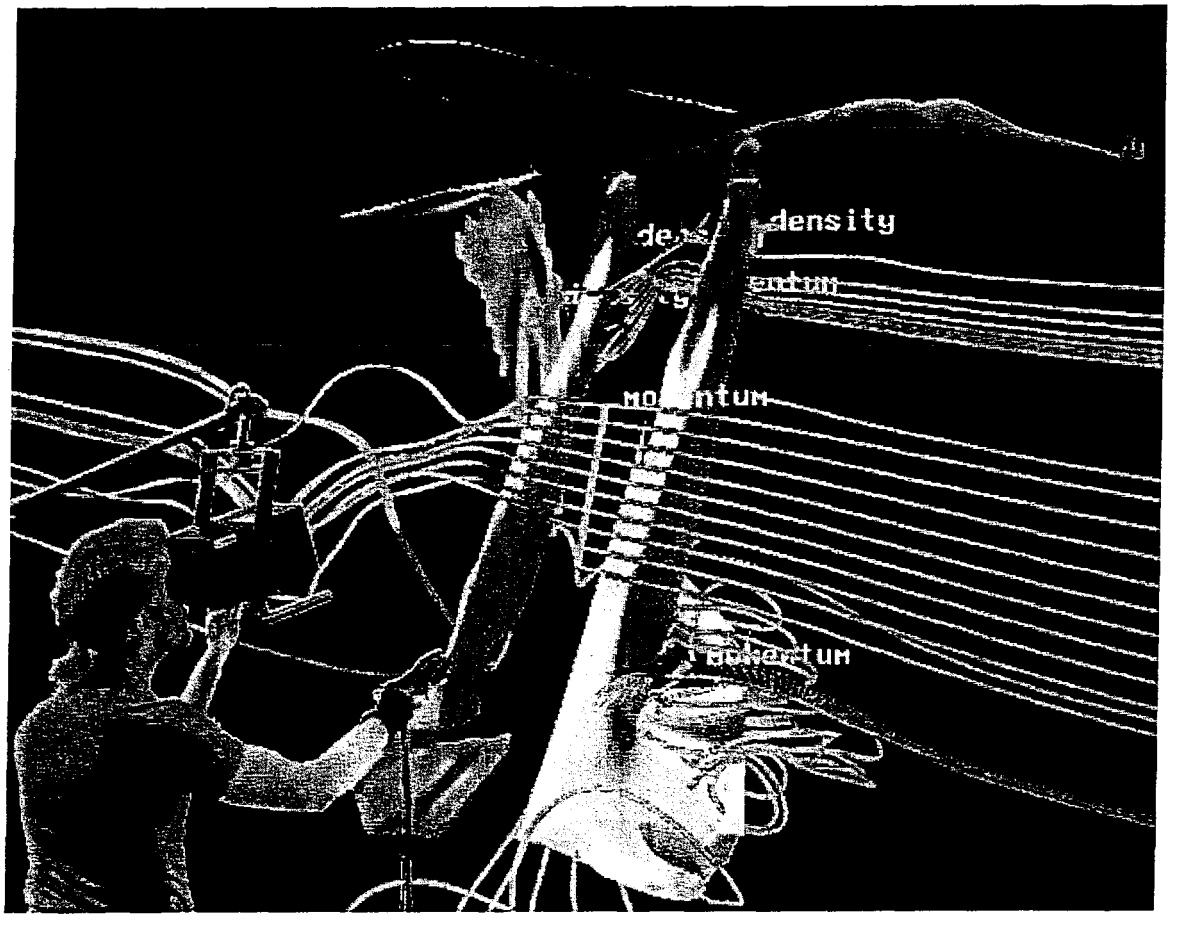

Figure 2: The Virtual Windtunnel, a virtual environment for the visualization of results of simulations arising in Computational Fluid Dynamics. This example shows a variety of visualization widgets with visualizations including streamlines, local isosurfaces and local cutting planes.

1.4. How Scientific Visualization Differs from Other VR Applications

The design and development of a scientific visualization application within a virtual environment is different from most virtual reality application development. Scientific visualization environments are often abstract and involve large amounts of data access and computation in response to a query. For time-varying data, different senses of time arise, in which the data may evolve more slowly or even backwards relative to user time. Such differences between scientific visualization environments and more conventional VR applications can be generalized into the following areas:

- **Greater flexibility in graphical representation:** The inherently abstract nature of information implies opportunities for simpler, faster graphics, such as representing a streamline as a simple polyline. Conventional applications, such as training or entertainment, are typically more concerned with realistic graphical environments, and so may have less flexibility in the selection of graphical representation. Of course this is not a universally applicable rule, as some representations of information such as direct volume rendering can be very graphically intensive.
- **A large amount of numerical computation may be required:** While scientific visualization typically addresses pre-existing data, visualization extracts may themselves require considerable computation. Streamlines or isosurfaces in the visualization of continuous vector and scalar fields are well-known examples that require large amounts of computation. As more sophisticated data analysis techniques are used in virtual environments more computational demands can be expected.
- **A large amount of data access may be required:** Visualization extracts require access to data, and some extracts require more data access than others. Complete isosurfaces, for example, require traversing an entire data set (at a particular timestep for time-varying data) for the computation of the data set. Time-varying data sets can be extremely large, requiring access to hundreds of gigabytes of data in a single analysis session, albeit a single timestep at a time.

- **There may be various senses of time:** As discussed below in section 3.1, several senses of time can arise in a scientific visualization system, particularly when addressing time-varying data sets. While some of these senses of time correspond to conventional time in other VE applications, completely new ways of thinking of time arise from the fact that a user may wish to manipulate time flow in a scientific visualization system.

These differences between a scientific visualization virtual environment and other applications have major impacts on the design of the virtual environment system. These impacts are the focus of this chapter.

2. Direct Manipulation Basics

2.1. What is Manipulated: Visualization Widgets

The phrase "direct manipulation in virtual reality" refers to the ability of the user to pick up a virtual object much as the user would pick up an object in the real world. This type of manipulation is in contrast with "indirect manipulation", where an object in the environment responds to controls generated by manipulating some other object. A common example of indirect manipulation is a conventional graphical user interface (GUI) where buttons or sliders control some aspect of a visualization. While we focus on direct manipulation in this chapter, the strategies we describe to deliver fast responsiveness is beneficial to indirect manipulation as well.

A scientific visualization environment often does not mimic the real world, so there is some freedom to choose which objects should be manipulated and which objects are not. In some cases it is not clear what direct manipulation of a visualization means. In our example, what should it mean to directly manipulate a streamline? While it conceptually makes sense to manipulate a streamline by grabbing it at any point, this can be very difficult if the data is time-varying so the streamline is changing shape, possibly quite rapidly and dramatically.

Experience has shown that while in some cases it makes sense to grab an object, in many cases it is easier for the user to grab tools which control a visualization rather than the visualization itself. These tools are

spatially co-located with the visualization in some sense, so the user has the feeling of directly manipulating the visualization. We call these tools **visualization widgets**. This approach of directly manipulating visualization widgets rather than the visualizations themselves has several advantages:

- **Unified interface:** the user has to learn the interactive behavior of only a small set of widgets, rather than having to figure out how each type of visualization is manipulated.
- **Affordance:** a well-designed widget will have a natural interpretation, facilitating the user's knowledge of what the widget does.
- **Visualization grouping:** it is often desirable to manipulate groups of visualizations. Such a group may consist of several streamlines in a row, or may be heterogeneous, combining streamlines, cutting planes and isosurfaces all controlled by one widget.
- **Unchanging spatial position:** in a time-varying visualization environment the visualization widgets need not move with the data, which makes them easier for the user to pick up.

Of course the use of visualization widgets does not preclude the ability to manipulate visualizations directly, but in our experience caution is advised. Visualization environments can become quite rich with many different objects in the environment. If many of these objects respond to user grab actions then grabbing an object by mistake can become a significant issue. Restricting manipulation to a relatively small number of data widgets helps prevent this situation.

In order for the spatial location (and possibly orientation) of a visualization widget to control a visualization, that visualization must use spatial information in its specification. In our streamline example this specification is natural: the widget controls the location of some point on the streamline. In our description of the streamline computation above that point would be the start of the streamline, but by integrating backwards as well as forwards that point could be anywhere on the streamline. We will call the spatial location used to specify a visualization the **seed point** of that visualization.

Some visualizations, however, are not typically specified by a spatial location. An example of a non-spatially defined visualization is a traditional isosurface of a scalar field, which is specified by a scalar field value. The natural way to specify such a field value is through indirect manipulation via a GUI. One way

to convert such a visualization to a spatially specified *local isosurface* is given in section 5.3. This is an example of the kind of creativity that may be required in using direct manipulation to control a visualization.

A visualization system need not use virtual reality to benefit from direct manipulation. Conventional mouse input can be used to directly manipulate objects in a visualization window, and nearly all of the considerations in this chapter apply.

2.2. Human Factors Requirements

The act of reaching out and picking up an object is natural in the real world, but when the responsiveness of our hand is degraded by *inaccuracy* (it doesn't go where we tell it), *latency* (it doesn't go when we tell it) or our feedback has a low *update rate* (we see our hand in snapshots) the act of picking up an object can become difficult. We can summarize this situation as follows:

- **Accurate tracking, fast response** to user commands and **smooth feedback** are required for a direct manipulation interface to succeed in giving the user a sense of directly "picking up and moving" objects in the virtual environment.

In the presence of fast, smooth feedback, a certain amount of tracking inaccuracy is tolerable. Fast response and smooth feedback are, however, critical to the user's experience of directly manipulating objects in the virtual environment. In a scientific visualization environment this places performance requirements on the data access, computation and graphical rendering triggered by the manipulation of a visualization.

How fast the graphics and interaction response must be turns out to be both application and domain dependent. A virtual environment for training for real-world manual tasks such as a flight simulator requires response times of less than 1/30 of a second in order to mimic the response times of real world objects. Information visualization, however, does not typically require fidelity to real-world time scales, so the performance requirements are driven by the human factors of manual control [12].

The human factors issues that turn out to be important for scientific visualization in virtual environments are the following:

- **Graphics Update Rate:** How fast must the graphical display update rate (graphical animation rate) be to preserve a sense of object presence, the sense that the virtual object has a position in three-dimensional space independent of the user? By graphical update rate we mean the rate at which frames are drawn to the graphics frame buffer(s), not the display device's refresh rate.
- **Interaction Responsiveness:** How quickly must objects in the environment respond to user actions in order to maintain a sense of presence and direct manipulation?
- **Data Display Responsiveness:** How fast must interactive data display devices, such as a data probe, update to give the user a sense of exploring the data?

While these considerations are related, in a virtual environment they are distinct. The fast graphics update rate is required by the virtual reality requirement of rendering the graphical scene from the user's current head position and orientation. Interaction responsiveness measures how well the interactive objects move with the user's hand, and is a function of the graphical update rate, the latency of the input devices that measure the user's actions, and any computation that the interactive objects require to respond. This latter computation is trivial for visualization widgets. Data display responsiveness measures how quickly the visualizations respond to the user's manipulation. The responsiveness of the interactive objects and the visualizations they control need not be the same, for example the user can position a widget and subsequently watch the visualizations appear in response.

The relationships and differences between these time scales are subtle. Graphics update rate will limit the interaction and data display responsiveness because the interactive displays cannot be presented faster than the graphics update rate. Update rate and responsiveness, however, are very different kinds of measures: update rate is measured in frames/second, while responsiveness is the **latency**, measured in seconds: the time interval between a user action and when the system's response is displayed. This latency is determined by all processes triggered by the user's action, from reading the user tracking devices through processing the user's commands through possible subsequent computation to the display of the result.

Experience has shown that the limits on these time scales are:

- **The graphics update rate must be greater than 10 frames/second.** While faster update rates are desirable, 10 frames/second is sufficient to maintain a sense of object presence even though the discrete frames of the display are easily perceptible. Slower update rates result in a failure of the sense of object presence, compromising the enhanced three-dimensional perception advantages of a virtual environment.
- **Interaction responsiveness must be less than 0.1 seconds.** While lower latencies and faster responsiveness is desirable, a latency of 0.1 seconds is fast enough to give the user a good sense of control of objects in the virtual environment. Longer latencies typically cause the user to experience unacceptable difficulty in selecting and manipulating objects in three-dimensional space.
- **Data display responsiveness must be less than about 1/3 of a second.** While faster responsiveness is desirable, a data display latency of 1/3 of a second maintains a sense of "exploring" the environment, though the user may use slow movements to adapt to this relatively long latency. Longer latencies in data display require such slow movements on the part of the user as to lose usability.

The graphics update rate and the interaction responsiveness requirements are similar: one graphics frame of latency is allowed for to maintain good responsiveness for user interaction. The data display responsiveness requirement, however, is less restrictive. The difference in latency requirements between interactivity and data displays is due to the fact that user interaction (selecting, acquiring and moving objects) is a manual task driven by the human factors of manual control, while observing data display during movement is an intellectual task, in which the user observes what happens as a data probe is moved through space.

Because the interaction and data display responsiveness requirements are different, the primary design strategy of a direct manipulation system is to make the graphics and interaction processes independent of the visualization computation processes. In such a system the widgets can be made to update with 0.1 second latency even if the visualizations they control have a latency of 1/3 of a second.

A simple crosshair in three-dimensional space, with an associated streamline of a vector field, is an example of such a tool. In this example, the user can "pick up and move" the crosshair, which will be very responsive (within the limits of the graphical update rate) due to its simple graphical nature. When this crosshair is moved it will trigger the computation of a streamline at its current location. If the process computing the streamline runs asynchronously from the process handling user interaction and graphical display, interaction responsiveness will be only slightly impacted by the computation of the streamline (assuming a pre-emptive multi-tasking operating system). This example shows that the difference in time scales between the interaction responsiveness and data display responsiveness has strong implications for the run-time architecture of the scientific visualization system. These implications for the overall system architecture are discussed in section 3.2.

Interactive time-varying environments potentially present a difficult challenge in meeting the above requirements: all non-precomputed visualization extracts, not just the ones most recently manipulated, must be computed whenever the timestep changes. Furthermore, when the timestep changes, all computations must take place before any of the extracts can be displayed so that extracts from different data timesteps are not displayed at the same time. Experience has shown that it is acceptable for the data timestep to change quite slowly, so long as the 10 frames/second graphical update rate and the 1/3 second data responsiveness requirements are met.

2.3. Input Device Characteristics

Direct manipulation typically uses two pieces of information: the position and orientation of the user's hand and some kind of user command indicating what action to perform at that location. In virtual reality systems this information is provided by a three-dimensional tracking device which delivers the hand position and (usually) orientation as well as a way to sense the user's command. The position and orientation data is often subject to noise, inaccuracies in position/orientation, and latency, which contribute (along with system latencies and frame rates) to degrading the user's ability to directly manipulate objects in the virtual environment. Care must therefore be taken to use the highest quality tracking device

available within practical constraints. Latency is minimized by always using the most recent tracking data, typically delivered by a high-frequency polling process.

Position tracking data is typically defined as a three-dimensional vector giving the user's hand position in some pre-defined coordinate system. The orientation data can take several forms. Commercial trackers typically return three orientation angles (roll, pitch and yaw or Euler angles), a 3x3 rotation matrix or a quaternion [13]. A matrix or quaternion representation is preferable to orientation angles because orientation angles are unable to describe some orientations due to singularities (so called *gimbal lock*). These singularities are not present in a matrix representation or quaternions. In any case the user's orientation and position data should be converted into a 4x4 graphics transformation matrix for use. This graphics transformation matrix should have the same form as transformation matrices used by the graphics system's matrix stack. The user data can then be used directly to transform any graphics objects being manipulated by the user.

Once the user's hand position and orientation is available as a graphics transformation matrix it is simple to transform an object grabbed by the user so that the object appears to be rigidly attached to the user's hand. Define M_H as the graphics transformation giving user's hand tracking data in the current time frame, and M'_H as the graphics transformation giving user's hand tracking data in the previous time frame. Similarly, let M'_O be the object's transformation matrix (from world coordinates) in the previous time frame. Our task is to compute the object's transformation matrix for the current frame M_O. Following a method of Warren Robinett, we have $M_O = M_H (M'_H)^{-1} M'_O$. This product is recomputed in each frame.

User commands are typically given via a device integrated with the hand position tracker. Such a command device may be simple buttons or the result of gesture recognition via an instrumented glove that measures the bend of the user's fingers. In any case we assume that the device output is converted into a discrete command state such as "grab on" or "point on". When no command states are active we have a neutral state.

Force-feedback devices are available which provide both user hand tracking data as well as user feedback by restricting the movement of the tracking device via a mechanically exerted force. Such devices can be very powerful in direct manipulation, as they can give the experience of moving an object among other objects. Such a capability has shown great promise in some areas of scientific visualization, *e.g.*, molecular modeling. Force feedback is an example of the many possibilities in interface devices that are available [15].

2.4. Collision Detection and User Commands

Collision detection in the context of direct manipulation refers to the ability to recognize when the user is able to "pick up" an object. In the real world we manipulate objects through a complicated interplay of muscle interaction, friction between complex surfaces and gravity. Duplicating this interplay in a virtual environment is a highly non-trivial task. While some virtual reality applications, such as task training, may benefit from mimicking real-world interaction in detail, scientific visualization applications can usually take a much simpler approach.

The simplest approach is to have a small number of active "grab points" contained in an interactive widget. These points become active when the user's hand position is within a pre-defined distance. Some kind of feedback is given to the user to indicate when the user's hand is close enough to a grab point to activate it. Then if the user commands, for example, a grab state then the grab point of the widget becomes grabbed and the widget reacts appropriately. When using this paradigm of interaction it is helpful to provide the user with unambiguous feedback as to the hand position. For example, representing the user's hand as a three-dimensional crosshair gives a better indication of the user's hand than showing an abstract hand shape.

More complex approaches can include using the user's hand orientation information as part of the widget control, or using a more complex geometry-based collision detection method.

2.5. Widget Design

The appropriate design of visualization widgets is critical to the success of a virtual reality-based scientific visualization system. Unfortunately space does not allow a complete review of this subject, so we will describe a simple approach that has proven effective in scientific visualization. For further examples see [8][10].

Inspired by the discussion at the end of section 2.2, we discuss the design of simple widgets that can control a variety of visualizations. These widgets are defined by some geometry defining the appearance of the widget, a set of defined grab points by which the widget may be manipulated, and a set of visualization seed points that are used to specify the visualizations controlled by the widget. Note that there can be several visualizations specified by the same seed point.

We give three examples of simple visualization widgets, which differ in their spatial dimension.

- Point widget (0-dimensional, see Figure 3): Geometrical representation: a three-dimensional crosshair. Grab point: a single point located at the center of the crosshair. Visualization seed point: the center of the crosshair.
- Line widget (1-dimensional, see Figure 4): Geometrical representation: line in three-dimensional space. Grab points: one at each end which move that end only (allowing control over the orientation and length of the line) and one at the line's center which, when grabbed, rigidly moves the entire line. Visualization seed points: equally distributed along the length of the line.
- Plane widget (2-dimensional, see Figure 5): Geometrical representation: a plane in three-dimensional space. Grab points: one at each corner which moves that corner only; one in the center of each edge which moves that edge holding the opposite edge fixed; one in the center of the plane which moves the plane rigidly. Visualization seed points: either a single seed point in the center of the plane (appropriate for local cutting plane visualizations, possibly restricted to the interior of the plane widget) or equidistributed on the plane, *e.g.* as an $n \times n$ array.

These widgets have a common design metaphor of simple geometry with grab points in the "obvious" places, and the visualization seed points have a natural association with the widget geometry. In all cases

the grab points provide the same graphical feedback to the user. These examples can be extended or generalized in obvious ways.

The visualization widgets and their associated visualizations will have properties that are typically controlled via a conventional GUI interface. For immersive virtual environments this GUI will usually be embedded in the three dimensional space of the visualization.

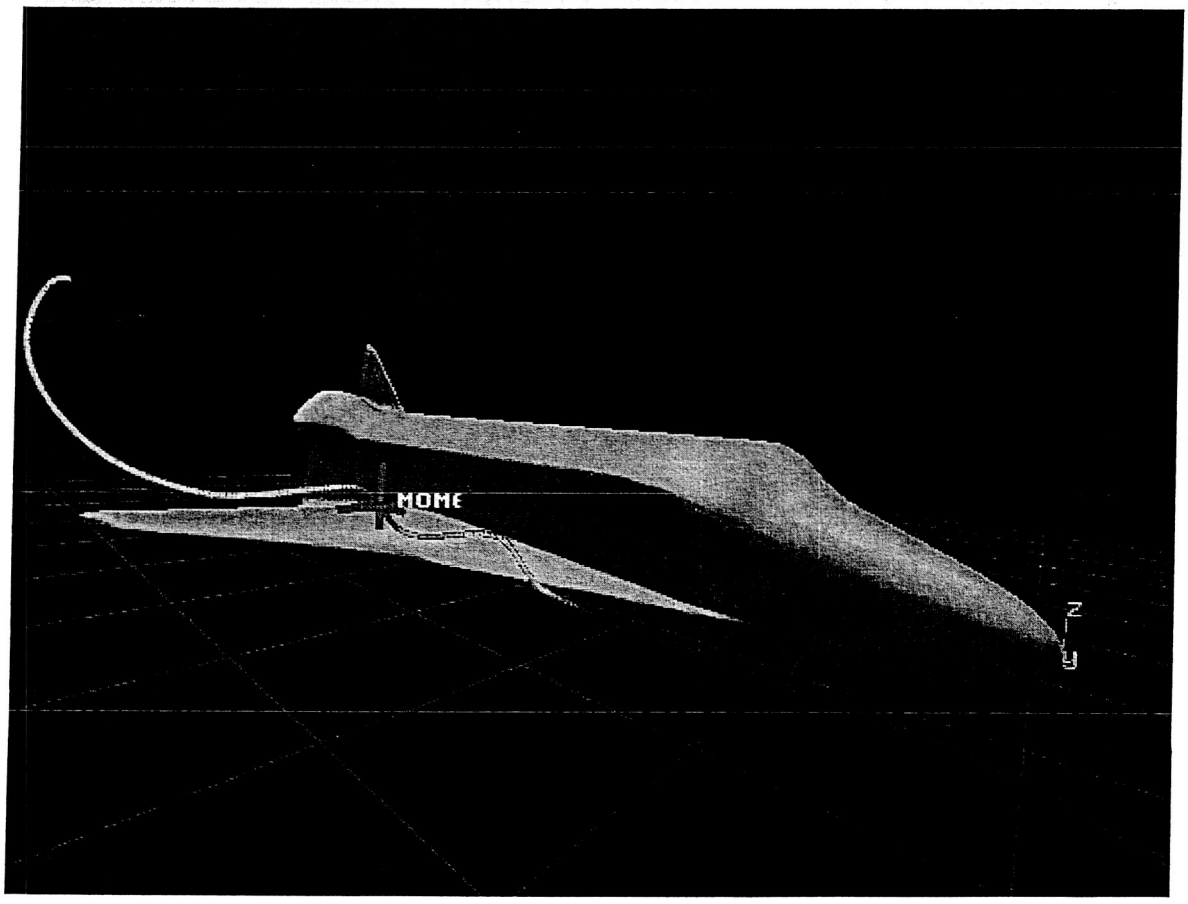

Figure 3: A point widget emitting a single streamline

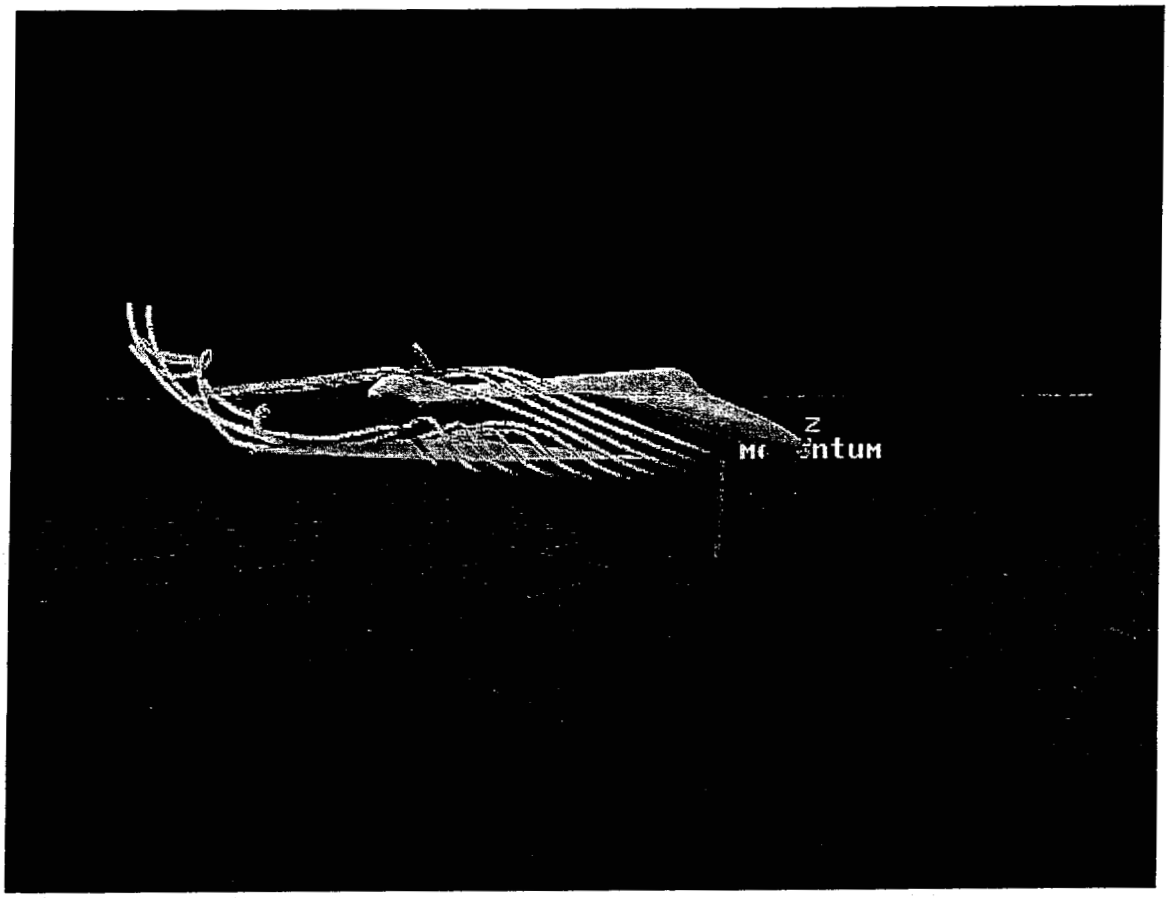

Figure 4: A line visualization widget emitting a collection of streamlines.

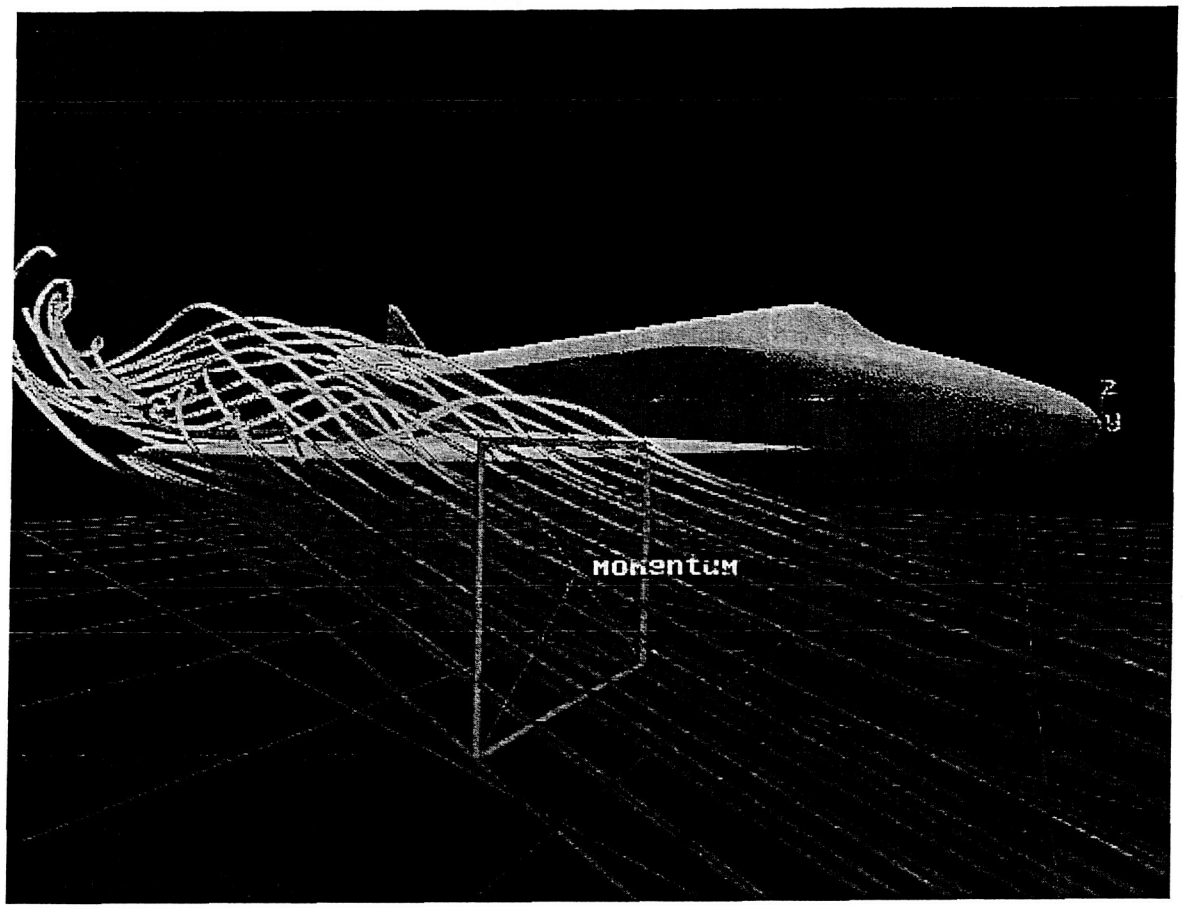

Figure 5: A plane visualization widget emitting many streamlines.

3. System Architecture Issues

There are several issues that arise in the design and implementation of virtual environments for information visualization. In this section we examine some of these issues in detail. First, however, we must classify the types of interaction that may occur, which in turn depends on the time flow of the data.

3.1. Classification of interaction and time flow

There are two design questions that must be answered before considering an appropriate implementation of a scientific visualization application in virtual reality:

- **Is the user allowed to interactively query the data at run time, generating new visualization extracts?** If so, the system will likely have user-driven data accesses and computation to support extraction of new visualization geometry.
- **Is the data time-varying?** If so, there will be at least two senses of time in the virtual environment: user time and data time. The flow of data time may be put under user control so that it may be slowed, stopped, reversed, or times randomly accessed.

These questions are independent, and both must be answered in order to determine which kind of implementation strategy will be appropriate. There are four combinations that arise from the answers to these two questions:

Non-interactive, non-time-varying data: This is the simplest scientific visualization environment, where the visualization geometry can be extracted ahead of time and displayed as static geometry in a head-tracked virtual environment. No data access or computation issues occur in this case. The user may be able to rotate or move the static geometry. The design issues that arise in this case involve only collision detection and possibly the design of widgets used to control the geometry.

Non-interactive, time-varying data: In this case the visualization extract geometry can be pre-computed as a time series which may be displayed as a three-dimensional animation in the virtual environment. The user may be given control over the flow of the animation to slow it down, stop it, or reverse direction. Such user-controlled time flow implies an interface which may include rate controls to determine the speed and direction of the data time or a timestep control for the random access of the extracted geometry at a particular data time. The issues that arise in this case are common to most virtual reality applications and so will not be considered further in this chapter.

Interactive, non-time-varying data: In this case the data does not change in time, but the user specifies the visualization extracts at runtime. In a virtual environment such extracts may be specified via a direct-manipulation interface where the user either specifies a point or manipulates an object in three-dimensional space. The visualization extract may require significant computation, which will have an impact on the

responsiveness of the system. This impact is discussed at length in section 3.2. When visualization extracts are do not change they are typically not recomputed.

Interactive, time-varying data: For this type of environment, the data itself is changing with time so any existing visualization extracts must be recomputed whenever the data time changes. This can result in a large amount of computation for each data timestep, the implications of which are discussed in section 3.2. In addition, the user may be given control over the flow of data time, allowing data time to run more quickly or slowly, or in reverse. The user may wish to stop time and explore a particular timestep. When time is stopped the system should act like an interactive, non-time-varying data environment, allowing visualization extracts to be computed in response to user commands.

3.2. System Architecture

The observations in the previous section imply that any implementation of an interactive scientific visualization virtual environment in which computation of visualization extracts takes place should contain at least two asynchronous processes: a *graphics and interaction process* and a *visualization extract computation process*. More generally, the graphics and interaction task may be performed by a group of processes we shall call the **interaction (process) group**, one or more processes for graphics display and possibly separate processes for reading and processing user tracking data. Similarly the visualization extraction task may be performed by several processes, called the **computation (process) group,** possibly operating on multiple processors in parallel. We choose these groupings because processes in the interaction group all have the same 10 frames/second/0.1 second latency requirements, while the computation group has the 1/3rd of a second latency requirement (see section 2.2). This process structure decouples display and computation, so that a slow computation does not slow down the display process, and the speed requirements of the display process do not limit the computation.

The interaction process group passes user commands to the computation process group, which triggers the computation of visualization extracts. These resulting extracts are passed back to the interaction process group for display. This basic architecture is outlined in figure 6.

In an interactive time-varying environment the optimal computation and synchronization of the visualization extracts produced by the computation process group is a delicate issue, the resolution of which can be application dependent. An example of such a resolution is described in [4][5].

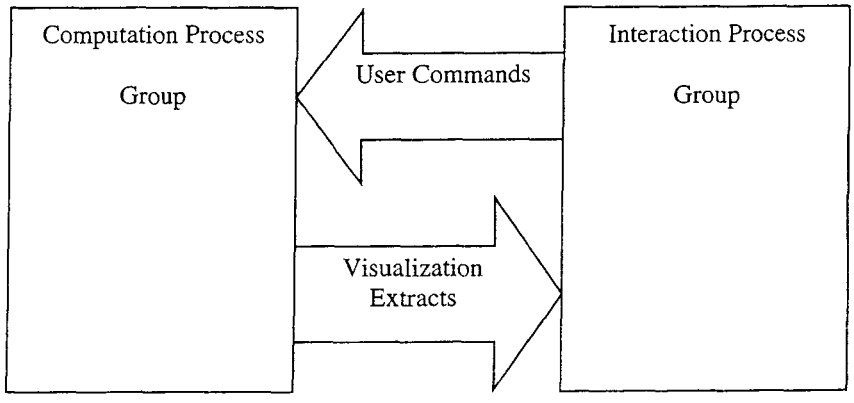

Figure 6: Runtime process architecture of an scientific visualization system for interactive and/or time-varying data

4. Distributed Implementation

Distributed data analysis can be highly desirable for performance, computational steering or collaborative purposes. The use of separate, asynchronous computation and interaction process groups communicating via buffers as described in section 3.2 facilitates a distributed implementation, where the process groups exist on separate, possibly remote machines communicating over a network.

4.1. Distribution Strategies

There are several strategies for the distribution of data analysis. These strategies are determined by selection of where to place which operations in the data analysis pipeline of figure 1. These strategies and their advantages and disadvantages:

- **Remote Data, Local Extraction and Rendering:** In this option the data exists on a remote system, and individual data values are obtained over the network as required by the local visualization extraction algorithm. This strategy has the advantage of architectural simplicity, with all visualization

activities taking place on the local system as if the data were local. This strategy has the disadvantage that it requires a network access each time data is required, which can be very time consuming. There are techniques to overcome this disadvantage, such as clever pre-fetching, where data is delivered in groupings which anticipate expected future queries. For many applications, however, it will not be possible to us this strategy and meet the performance requirements described in section 2.2.

- **Remote Data and Extraction, Local Rendering:** With this strategy visualization extraction occurs on a remote system, typically the same system that contains the data. In an interactive system the extraction computations are in response to user commands passed from the user's local system. The results of the extraction, typically geometrical descriptions of three-dimensional visualization objects, are transmitted to the user's local system for rendering. Architecturally, this strategy maps closely to the runtime architecture illustrated in Figure 6, with the computation process group on (one or more) remote machines and the display and interaction process on the local system. This strategy has the advantage that the extractions algorithms are "close to the data", so data access is not a bottleneck. It also has the advantage of local rendering, which allows head-tracking display for each participant as required in virtual environments. The disadvantages of this strategy include the fact that response to user commands requires a round trip over the network, and the requirement that the user's local system be capable of rendering the extract's geometry.
- **Remote Data and Extraction, Distributed Rendering:** This strategy is a variation of the above Remote Data and Extraction, Local Rendering strategy. In this case the rendering commands occur on the remote system and are passed as distributed rendering calls to the user's local system for actual rendering. A local client program is still required to read the user's trackers and process and send user commands. This strategy has advantages and disadvantages similar to the Remote Data and Extraction, Local Rendering strategy, except that the network round-trip time is now part of the graphics display loop. This may introduce unacceptable delays into head-tracking responsiveness.
- **Remote Data, Extraction and Rendering:** This strategy places all the operations of the data analysis pipeline on the remote system(s), with the final rendered frames returned to the user's local system over the network. This strategy has the advantage that very powerful remote systems can be used when the user has a very low-power local system. The disadvantage is that the rendered frames can be

large, for example for a 1024x1024 24-bit RGB-alpha display requires a 4 megabytes frame buffer, and two such frame buffers are required for stereo display. This implies an 80 megabyte transfer every second in our example to maintain the frame rate of 10 frames per second. This bandwidth requirement is beyond most available large-area networks. There are also serious issues of latency control in this strategy because the network time is part of the display responsiveness loop. There are, however, situations where the local system is incapable of the kinds of rendering desired and this strategy may be the only viable option. Direct volume rendering is an example when this strategy may provide the optimal choice.

The above strategies are not exclusive: one may have remote visualization extraction taking place on one remote system while the data resides on a third system.

4.2. Remote Collaboration Strategies

Once a system is distributed, the opportunity arises for remote collaboration, where two or more non-co-located users examine the same data together. Strategies for remote collaboration are related to, but different from distribution strategies. We briefly summarize the common remote collaboration strategies:

- **Distributed data:** This collaboration strategy places copies of the data to be examined on all participants' client systems. Collaboration is implemented by passing either user commands or computed visualization extract results among each of the participants' systems. The primary advantage of this strategy is that the software used is similar to stand-alone versions of the same systems. The main disadvantages include the difficulty of ensuring synchronization among the participants, and the requirement that each participant's system be capable of storing the data and computing the (at least locally generated) visualization extracts. Many distributed collaborative VE systems, such as military training systems, utilize the distributed data collaboration strategy.
- **Data Server:** This collaboration strategy builds upon the remote data distribution strategy. The data typically resides on a single remote system and is accessed by the participant's local system as needed. The visualization extracts are computed locally, and are communicated in the same manner as in the distributed data collaboration strategy above.

- **Visualization Extract Server:** This collaboration strategy builds upon the remote extraction distribution strategy, in which the visualization extracts are computed on a remote system, which is typically where the data being examined is stored. The extracts are sent to each participant's system for local rendering. The advantages of this strategy include:
  - As there is only one set of extracts associated with each set of data, synchronization is greatly simplified
  - Local rendering allows each participant to render the visualization extracts from a local point of view, as required for head-tracked displays
  - The extract server system can arbitrate conflicting user commands

  The visualization extract server collaboration strategy has the disadvantage of poor scaling to large numbers of users, though this problem will be alleviated when reliable multicast technologies become available. The other disadvantages of this strategy are the same as those for the remote extraction distribution strategy.
- **Scene Replication:** This collaboration strategy has a privileged user who's view is presented to the other participants. This collaboration strategy is similar to the remote rendering distribution strategy. This strategy has the same advantages and disadvantages as the remote data, extraction and rendering distribution strategy, with the added disadvantage that all participants will see the same view thereby precluding the use of head tracking for all participants.

5. Time-Critical Techniques

One of the prime requirements of virtual environments is responsiveness. In section 2.2 we discussed the performance requirements for various aspects of an scientific visualization application within a virtual environment. These requirements can be difficult to meet in light of the possibly complex graphics and extensive computation required for the computation of visualization extractions. One is often faced with a conflict between the requirements of a complete or accurate visualization and the requirements of responsiveness and fast graphical display rates. While accuracy is often critical in a scientific visualization environment, users often prefer fast response with a known degradation in accuracy for purposes of

exploration. When a phenomenon of interest is found in the more responsive but less accurate mode, the user can request that this phenomenon be recomputed and displayed more slowly with higher accuracy. The automatic resolution of the conflict between accuracy and responsiveness, finding the appropriate balance, is known as "time-critical design", the topic of this section.

5.1. The Time-Critical Philosophy

Time-critical design attempts to automate the process of finding a balance between required accuracy and responsiveness. This approach is very different from real-time programming, which guarantees a particular result in a specified time. Real-time programming typically operates in a fixed, highly constrained environment while time-critical programs are typically highly variable. This variability is particularly evident in an scientific visualization environment, where the data and extracts computed and displayed may vary widely within a single user session. Time-critical design does not guarantee a particular result, instead delivering the "best" result possible within a given time constraint. A successfully designed time-critical system will provide a graceful degradation of quality or accuracy as the time constraint becomes more difficult to meet.

Time critical design for a particular aspect of a program begins with defining a cost and benefit metric for the task to be completed. The task is then parameterized in a way that controls both costs and benefits. When the cost and benefit of a task is known as a function of the task's parameters before that task is performed, the appropriate choice of parameters is selected to maximize the benefit/cost ratio. There are often many tasks to be performed in an environment, and the benefit of a particular task can be a function of the state of that environment. The solution of this problem is often approached as a high-dimensional constrained optimization problem, maximizing the total benefit/cost ratio for the sum of the tasks to be performed given the constraint of the total time allowed for all tasks.

As we shall see below, however, it is often very difficult to know the benefit and cost of a task before that task is performed. In such situations, hints provided by the user or simple principles such as assuming equal benefit within a set of tasks are often used.

5.2. Time-Critical Graphics

Time-critical techniques were pioneered in computer graphics [9], where objects were drawn with higher or lower quality depending on such benefit metrics as position in the field of view and distance from the user. Such implementations often used multiple representations of an object at varying levels of detail. In scientific visualization, however, many visualization extracts are already in a minimal form such as streamlines defined as a set of points. There are opportunities for graphical simplification in scientific visualization, however. For example, It may be the case that many more primitive elements are used to define a surface than is necessary for its display. An isosurface may have regions that are close to flat but the algorithm used to derive the isosurface may create many surface elements in that flat region. Display of that surface would be faster if that flat region were represented by fewer surface elements. A surface may also be represented in simplified form until it became the focus of attention so that small variations from flatness may be important. Unfortunately, algorithms that identify such opportunities for surface simplification are computationally intensive and may therefore be unsuited to re-computation in every frame.

From similar considerations we conclude that unlike general computer graphics based on pre-computed polygonal models, the use of time-critical graphics in scientific visualization will be highly dependent on the domain-dependent specifics of the visualization extracts. It may be very difficult, for example, to assign a benefit to a particular extract, especially when that extract may extend to many regions of the user's view. While simple benefit metrics such as the location of the extract on the screen may be helpful, one should keep in mind that the user's head may be pointed in one direction while the user's eyes may be scanning the entire view. Such scanning is to be expected in a scientific visualization environment where the scene may contain many related, extended objects.

From these considerations few generalizations can be drawn about the use of time-critical graphics techniques in information visualization. Simple examples of time-critical graphics techniques that may be useful in scientific visualization include:

- Simplified representations, such as wireframe rather than polygonal rendering.

- Surfaces represented as two-dimensional arrays, where simplified versions of the surface may be obtained by rendering every *n* points in the array.
- Techniques that have been developed for time-critical direct volume rendering [16].

A more general approach to time-critical graphics is to use multi-resolution representations of the data or the resulting visualization extracts.

5.3. Time-Critical Computation

Computation of visualization extracts can provide several opportunities for time-critical design because such computation is often the most time-consuming aspect of a visualization system. As in the case of time-critical graphics, the specifics of time-critical computational design will be highly dependent on the nature of the extract computed. We can, however, make several general observations:

- Both the cost and benefit of a visualization extract can be very difficult to estimate *a priori* based on its specification and control parameters, especially since the extent of an extract (*e.g.* where a streamline will be visible) is difficult to predict based on its specification alone.
- The cost of an extract can be roughly defined as the time required for its computation. Experience has shown that this cost does not vary widely between successive computations.
- The cost of a visualization extract may be most easily controlled by parameterizing the extent or resolution of the computation. Techniques that begin their computation at a particular location in space, such as a streamline emanating from a point in space, lend themselves well to controlling their extent in space, which controls the time required by their computation. The cost of visualization techniques which rely on abstract or indirect specification, such as a global isosurface, is more effectively controlled by varying resolution.
- Other ways to control the cost of the visualization extract include choice of computational algorithm and error metrics for adaptive algorithms. These control parameters have a more discrete nature and may be set by the user or set automatically via specific trigger criteria. For examples see [4].

Given that the benefit of a visualization extract is hard to predict, one may treat all extracts as having equal benefit unless specified by the user. In combination with the observation that costs do not change dramatically in successive computations, this allows the following simple solution to the problem of choosing the control parameters. For simplicity we consider the situation in which all of the visualization extract's costs are controlled by limiting their extent. Here, each extract computation is assigned a time budget, and each extract's computation proceeds until its time budget is used up. Then the time taken to compute all extracts is compared to the overall time constraint. Each extract's time budget is divided by a scale factor determined by the total actual time divided by the total time constraint. This scale factor may have to take into account any parallel execution of the extract computations. . If the time required to compute all the extracts is greater than the time constraint, this will result in smaller visualization extracts which will take less time to compute. If the extracts become too small, a faster but less accurate computational algorithm may be chosen. If the time required to compute all extracts is smaller than the time constraint, the time budget of each extract is increased, resulting in larger visualization extracts. A similar approach may be used to choose the resolution with which visualization extracts may be computed.

It may be evident from the this discussion that the types of control parameters one may use in time critical design will be highly dependent on the nature of the extract and how it is computed. Clever approaches can significantly enhance the time-critical aspects of a visualization technique. As an example, consider isosurfaces of a three-dimensional scalar field: these isosurfaces are traditionally specified by selecting a value, resulting in a surface showing where that value is attained in the scalar field. Controlling the cost of a traditional isosurface by limiting the time of the computation will have unpredictable results: in the conventional marching cubes algorithm for computing isosurfaces the entire dataset is traversed. If the marching cubes algorithm is stopped before completion and before regions of the field where the isovalue is attained are traversed, no isosurface will appear at all. Controlling the cost of the marching cubes algorithm by controlling the resolution with which the dataset is traversed is possible, but this strategy does not provide fine control and may result in a significant degradation in quality. A different approach to isosurfaces, *local isosurfaces*, directly addresses this problem. Rather than choosing an isovalue, when defining a local isosurface the user chooses a point in the dataset, from which the isovalue is determined as

the value of the scalar field at that point. The isosurface is then computed (via a variation on the marching cubes algorithm) so that it eminates from that point in space, and is spatially local to the user's selection. The cost of a local isosurface in controlled by computing the isosurface until that isosurface's time budget has been used up. Two examples of local isosurfaes can be seen in figure 2.

6. Conclusions

Direct manipulation in scientific visualization provides the ability to explore complex environments in a natural and intuitive way. In order to implement an effective scientific visualization application in a virtual environment, however, issues of responsiveness and fast updates must be addressed. These issues may be resolved via use of appropriate system architectures, design based on human factors issues, appropriate time control for time-varying data, implementation of time-critical techniques whenever possible, and appropriate choices for distributed implementations. The details of how these solutions are implemented will be highly dependent on the target domain and the specifics of the visualization techniques used.

7. References

[1] Bryson, S. "Virtual Reality in Scientific Visualization", CACM 39 (5): 62-71 1996

[2] Bryson, S. and Levit, C., "The Virtual Wind Tunnel: An Environment for the Exploration of Three Dimensional Unsteady Flows", Proceedings of Visualization '91 San Diego, Ca, Oct. 1991, also Computer Graphics and Applications July 1992

[3] Bryson, S. and Gerald-Yamasaki, M., "The Distributed Virtual Wind Tunnel", Proceedings of Supercomputing '92 Minneapolis, Minn, Nov. 1992

[4] Bryson, S. and Johan, S., "Time Management, Simultaneity and Time-Critical Computation in Interactive Unsteady Visualization Environments" Visualization '96 San Francisco, CA.

[5] Bryson, S. Johan, S., and Schlecht, L., "An Extensible Interactive Framework for the Virtual Windtunnel" VRAIS '97 Albuquerque, NM.

[6] Conner, D. B. et al, "Three Dimensional Widgets", Computer Graphics (Proceedings of the 1992 Symposium on Interactive 3D Graphics), 25(2), ACM SIGGRAPH, March 1992, pp. 183-188.

[7] Cruz-Neira C., Leigh, J., Barnes, C., Cohen, S., Das, S., Englemann, R., Hudson, R., Papka, M., Siegel, L., Vasilakis, C., Sandin, D. J., and DeFanti, T. A., "Scientists in Wonderland: A Report on Visualization Applications in the CAVE Virtual Reality Environment", Proceedings of IEEE Symposium on Research Frontiers in Virtual Reality, Oct. 1993

[8] Forsberg A.S., Herndon K.P., and Zeleznik, R.C., "Aperture-Based Selection for Immersive Virtual Environments". Proceedings of the 1996 ACM Symposium on User Interface and Software Technology (UIST)

[9] Funkhouser, T. A., and Sequin, C. H., "Adaptive Display Algorithms for Interactive Frame Rates During Visualization of Complex Virtual Environments", Computer Graphics (SIGGRAPH `93)

[10] Herndon, K.P. and Meyer, T., "3D Widgets for Exploratory Scientific Visualization", Proceedings of the 1994 ACM Symposium on User Interface and Software Technology (UIST)

[11] Parker, S.G. and Johnson, C.R., "SCIRun: A Scientific Programming Environment for Computational Steering," Supercomputing `95, 1995

[12] Sheridan, T. B. and Ferrill, W. R., Man Machine Systems, MIT Press, Cambridge, Ma. 1974

[13] Shoemake, K, "Animating Rotations with Quaternion Curves", Computer Graphics V.19, N. 3, 1985

[14] Song, D. and Norman, M. L., "Cosmic Explorer: A Virtual Reality Environment for Exploring Cosmic Data", Proceedings of IEEE Symposium on Research Frontiers in Virtual Reality, Oct. 1993

[15] [Taylor et. al. 1993] Taylor, R. M., Robinett, W., Chi, V. L., Brooks Jr, F. P. and Wright, W., "The Nanomanipulator: A Virtual Reality Interface for a Scanning Tunnelling Microscope", Computer Graphics: Proceedings of SIGGRAPH '93, August 1993.

[16] Wan, M., Kaufman, A. E., and Bryson, S., "High Performance Presence-Accelerated Ray Casting", IEEE Visualization 1999