Three-Dimensional User Interfaces for Scientific Visualization

Research Grant No. NAG 2-830

STATUS REPORT (January 1994 through January 1995)

Brown University Computer Graphics Group
Dept. of Computer Science
PO Box 1910, Brown University
Providence, RI 02912

Principal Investigator: Andries van Dam

January 20, 1995

Abstract

This document presents the current status of work being done by the Brown University Graphics Group under NASA grant NAG 2-830. By the end of the second year of support, we have implemented an application framework in which we can visualize computational fluid dynamics (CFD) datasets and develop and experiment with novel three-dimensional user interfaces. This UI technology allows users to interactively place visualization probes in a dataset and modify some of their parameters. We have also implemented a time-critical scheduling system which strives to maintain a constant frame-rate regardless of the number of visualization techniques.

In the past year, we have published parts of this research at two conferences, the research annotation system at Visualization'94, and the 3D user interface at UIST'94. The real-time scheduling system has been submitted to the SIGGRAPH'95 conference. Copies of these documents are included with this report.

1 Project Description

The main goal of this project is to develop novel and productive user interface techniques for creating and managing visualizations of computational fluid dynamics (CFD) datasets. The existing commercial applications for CFD visualization generally provide many visualization techniques but lack easy-to-use interfaces. Because both these datasets and the techniques used to visualize them are inherently three-dimensional, our strategy has been to apply our knowledge of 3D user interfaces to this new domain.

In the first year of this grant, we implemented an environment in which we could begin to explore new interaction techniques specifically tailored for scientific visualization. This environment was written within our comprehensive 3D rapid prototyping system (UGA [11]).

More recent accomplishments include:

- Support for curvilinear grids
- A range of general positioning techniques for probes in a dataset
- 3D interfaces for controlling common visualization parameters
- A system for graphically annotating fluid flows
- Novel visualization techniques
- A time-critical scheduling algorithm

2 Current Status

We have developed a number of prototype 3D widgets for our scientific visualization environment. These fall into two main categories:

- positioning tools
- generalized probes

Our repertoire of visualization techniques currently includes:

- scalar and vector probes (with numerical, colored or tuft display)
- streamlines and particle paths
- isosurfaces
- "smoke rings"
- "flux balls"

We have combined these various tools and visualization techniques into a single coherent system for exploring CFD datasets. Descriptions of our user interfaces are given in the following sections. To date, we have tested our interfaces on relatively simple (less than 500,000 points) steady-flow datasets on both regular and curvilinear computation grids. The majority of our work has been done on a curvilinear dataset of airflow velocity (speed and direction) past the body of the Space Shuttle.

Much of the research described here was published in a technical note at the 1994 Symposium on User Interface Software and Technology [3].

2.1 Positioning Tools

The repertoire of positioning techniques discussed here were designed for use with conventional desktop hardware: a 2D mouse and CRT. Thus, these techniques aim to overcome many of the difficulties which result from using 2D devices for 3D interaction tasks. It remains to be seen how useful these techniques will be in more immersive environments with higher degree-of-freedom input and output devices.

2.1.1 Interactive Shadows

The default positioning technique in our system is direct-manipulation screen-aligned translation (objects are moved in the plane parallel to the screen plane). However, since this is a 2D mouse-based technique, the user must change the viewpoint to move objects in other planes. To move objects in three-space without changing the viewpoint, we have added "interactive shadow" widgets (Figure 1) to this environment. These shadow objects are generated for every 3D object, provide a valuable depth cue, and can be displayed on any axis-aligned plane. Further discussion of this tool is in [4].

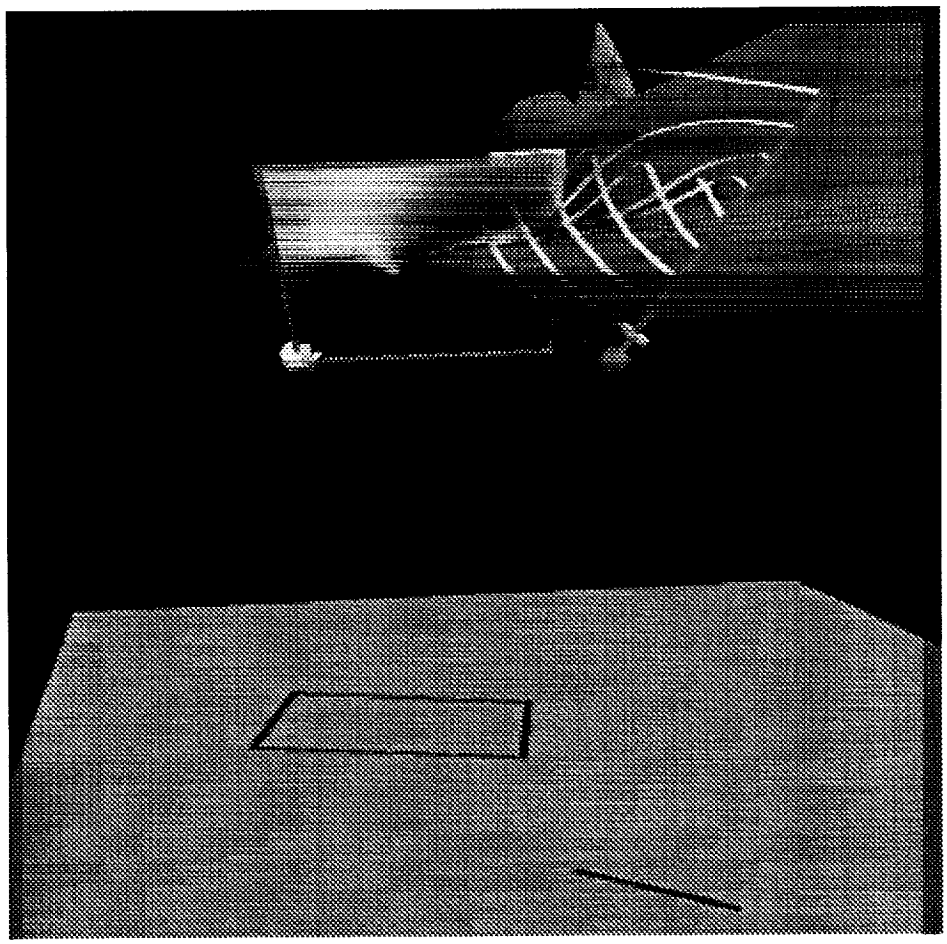


Figure 1: *Interactive shadows.* The shadow widgets for a cutting plane and a rake are projected onto the floor plane. The position of either probe can be modified by dragging the probe itself or its shadow. Manipulating a probe translates it in a plane parallel to the film plane of the camera viewing the scene; manipulating its shadow widget moves both the shadow and the probe in a plane parallel to the shadow plane. Note that the shadows provide a useful depth cue.

2.1.2 Object Handles

Some geometry, such as the Shuttle's fuselage, can easily obscure the shadow widgets projected onto a wall and render them unusable. To address this problem, we have implemented another technique for moving objects in 3D, called "object handles" (Figure 2). With this technique, we attach three new objects (in our case, simple line segments) to the 3D object and align them with the principal axes of world coordinate system. Dragging one of these handles translates the 3D object along the axis defined by that line. These widgets offer much of the same functionality as the "interactive shadows", but provide no depth cues. Their main advantage over shadows is that they are always attached to the 3D object – if the 3D object is visible, then so is the positioning widget.

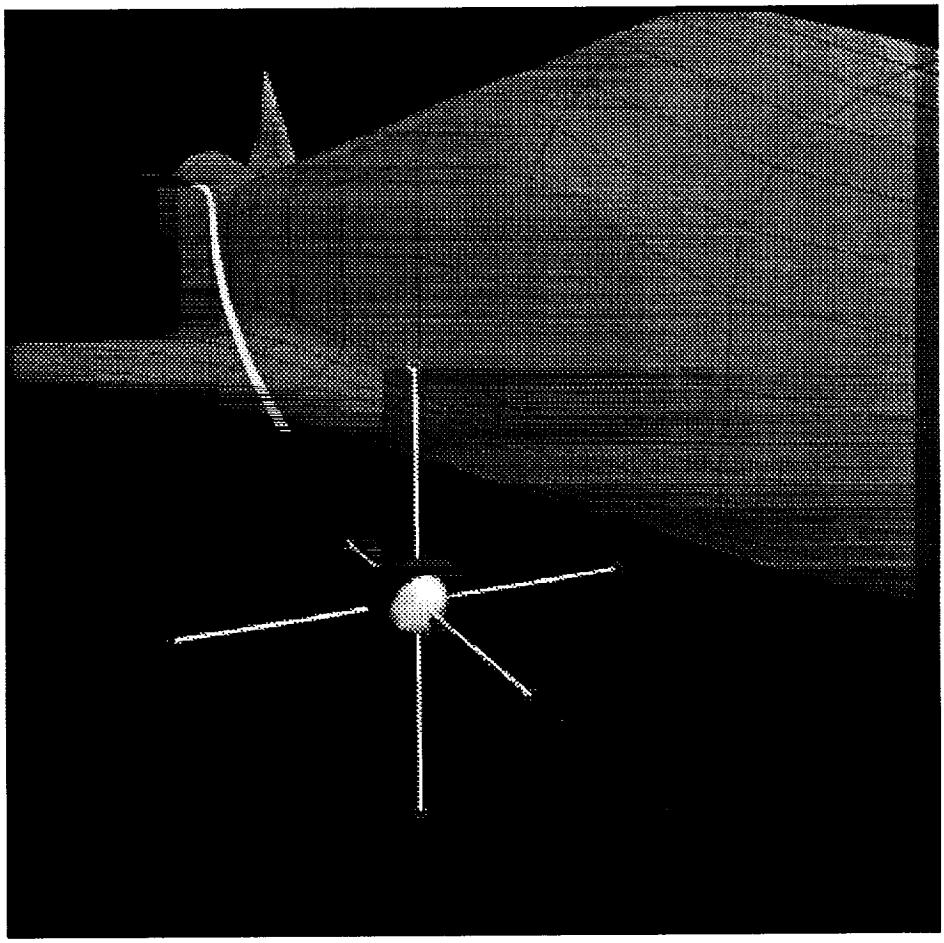
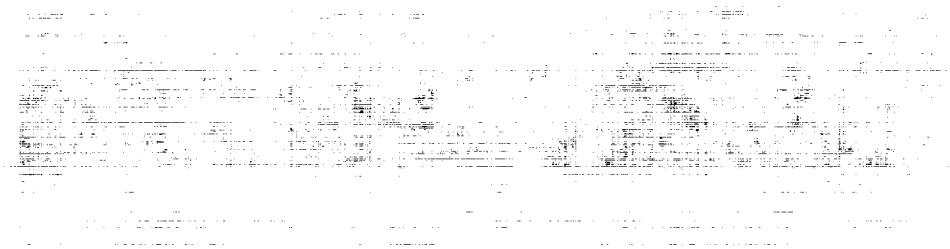


Figure 2: *Object handles.* The object handles widget is attached to a probe. The purple line represents the current translation axis.

These widgets also provide visual feedback to user actions in the form of projection lines and ghosting. A purple projection line, drawn when a user drags one of the three handles, indicates the widget's restricted degrees of freedom. Also, a ghosted copy of the handles widget is drawn in the original location to indicate the distance that the widget has been moved.



2.1.3 Grid-Aligned Handles

Both the shadow and object handle widgets use axes in the Cartesian coordinate space to help position objects more easily in 3D. We have also designed similar techniques which constrain the movement of a probe to features in the underlying computation grid. We have implemented a version of our object handles, called "grid-aligned handles" (Figure 3), which allow constrained translation along lines in the computational grid. With this technique, it is straightforward to move objects along the curved surfaces of a CFD object such as the leading edge of the Shuttle's wing.

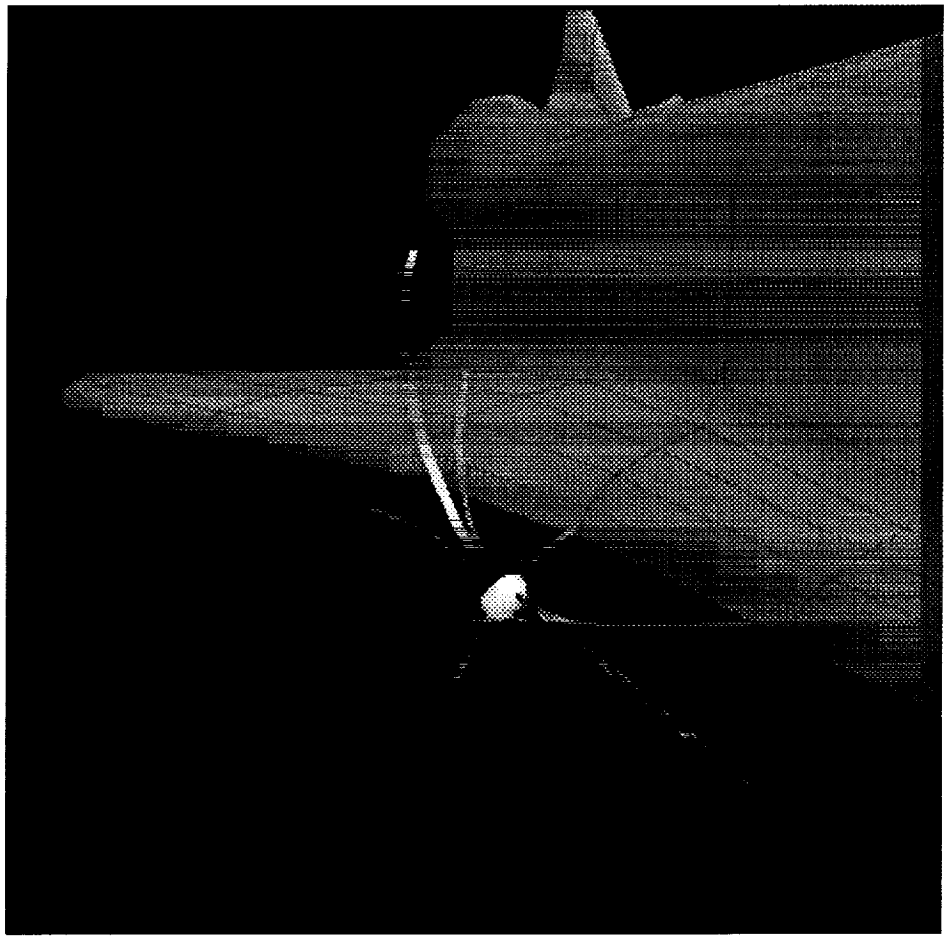


Figure 3: *Grid-aligned object handles.* The grid-aligned object handles widget is attached to a probe.

The grid-aligned handles work by tracing out lines in the computation grid from the point in the grid closest to a given sample point. An added benefit of these widgets is that they provide a visual representation of the local structure of the grid in the area surrounding the sample point. Users may exploit this information to gain a better understanding of the dataset and the behavior of visualization techniques.

Apart from its slightly different visual representation, this widget behaves identically to the object handles.

2.1.4 Data-Space Handles

We have also developed some interaction techniques based on the actual data being visualized. For example, the vector probe widget (Figure 4) consists of a grey spherical sample point, an arrow representing the direction of the vector field at that point, and a disk representing the plane perpendicular to the vector. By dragging the arrow component, the sample point can be moved along the streamline formed by the flow through that point. The disk is used to move the sample point perpendicular to the flow, allowing the user to explore nearby streamlines in the flow field.

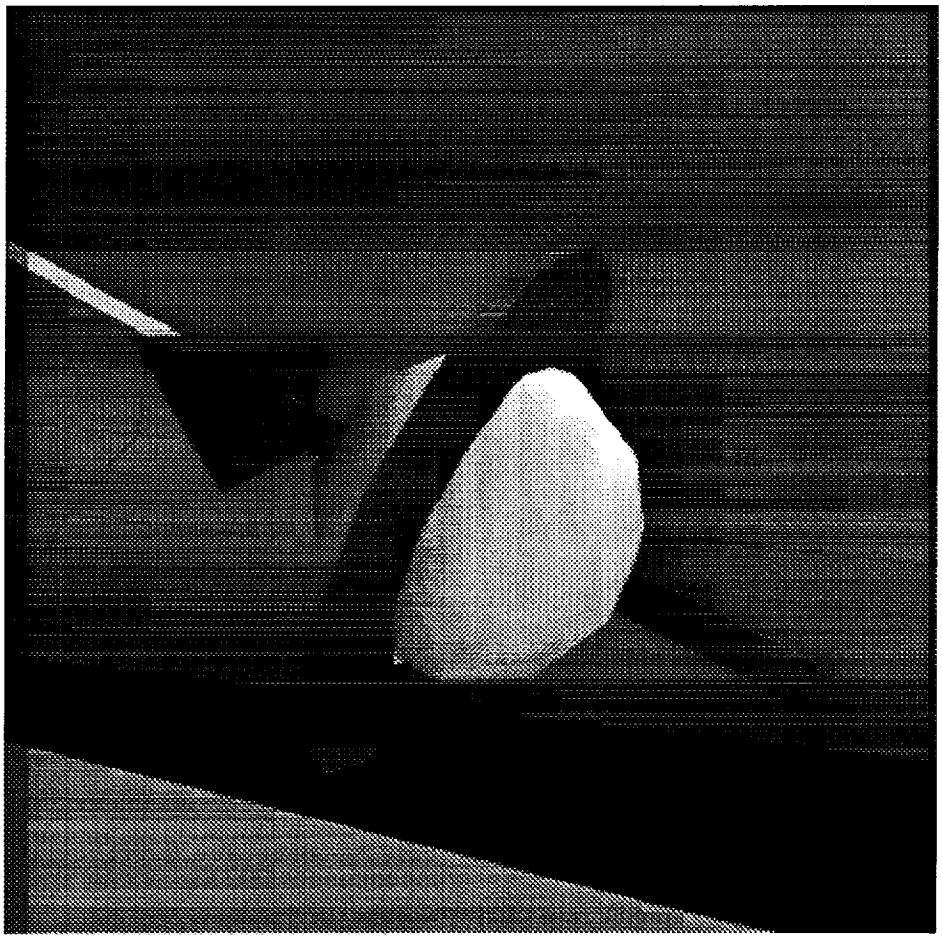


Figure 4: *Data-space handles.* The data-space handles on a point probe widget include a blue arrow and a red disk.

When we use this same general probe widget to visualize scalar data, the vector component displays the gradient of a scalar field. Pulling the vector changes the value of the isosurface that passes through the sample point; translating the red disk moves the sample point along the isosurface. Note that this last technique does not change the level of the isosurface, just the initial seed point from which the isosurface is computed. Since we are using an interactive isosurface algorithm [7], moving the probe in this way allows us to explore different regions of a single isosurface.

2.1.5 Direct-Manipulation Visualization Techniques

We have also explored other direct manipulation techniques using "grab-anywhere"

streamlines and rakes. While our previous streamlines required that users drag a probe through the dataset, a direct-manipulation streamline can be grabbed anywhere along its length. When a user clicks on some point along the streamline, the system moves the 3D probe widget (and thus the sample point) to that new point, and updates the streamline by integrating both forward and backward from the new sample point. With this technique, users are able to manipulate a streamline very precisely in particular regions of interest that might otherwise have been difficult to reach. For instance, if an interesting feature is observed at the very end of a given streamline, a user can simply click on the streamline right at the feature and move it around. With the older method, the user would have tried to move the probe at the origin of the streamline only to realize that small motions there caused very large changes in the streamline near its end.

We also applied this same technique to translate rakes of streamlines. In addition, rakes can be rotated and scaled. In our system, we use a separate mouse button to "twist" the rake about the streamline selected. This has the effect of keeping the picked streamline constant but modifying the neighboring streamlines.

2.2 Generalized Probes

Most of the visualization techniques we have implemented in our system are generated by sampling single points in a dataset, calculating scalar or vector values, and displaying some visual representation of the data. The positioning techniques described above are designed to help scientists quickly place these sample points in a dataset, but we also need methods for controlling collections of sample points as single groups. To address this need, we developed the notion of a generalized probe which can manifest itself as a zero-, one-, two- or three-dimensional widget. This generalized probe widget (Figure 5) is used to define a variety of visualization techniques including streamlines, rakes, hedgehogs, cutting-planes and isosurfaces.

2.2.1 Zero-dimensional probes

The zero-dimensional widget is a simple probe that samples a single point in the dataset. From this point, we can choose to generate one of five visualization techniques: a number, color, tuft, advected particle, or isosurface. Multiple visualization techniques can be generated simultaneously from a single sample point (though we have not yet devised a good user interface for controlling this functionality). Users may then use any of the positioning techniques described above to place this widget in the dataset. The direct-manipulation, "grab-anywhere" interaction technique described in the previous section applies only to the advected particle visualization technique.

2.2.2 One-dimensional rakes

The one-dimensional widget is essentially the same as a rake tool commonly used in real wind tunnels. This widget produces a set of sample points at regular intervals in Cartesian space along a line; it can be translated and rotated freely and can be scaled along a single axis by translating the red ball at one end. Additionally, we supply a resolution handle, the orange disk, to change the distance between sample points. Again, any of the visualization techniques described above can be generated from this set of sample points; advected particles produce the familiar rake of streamlines, the color technique produces a colored bar, and the isosurface produces an "onion" – multiple isosurfaces at different levels of the dataset.

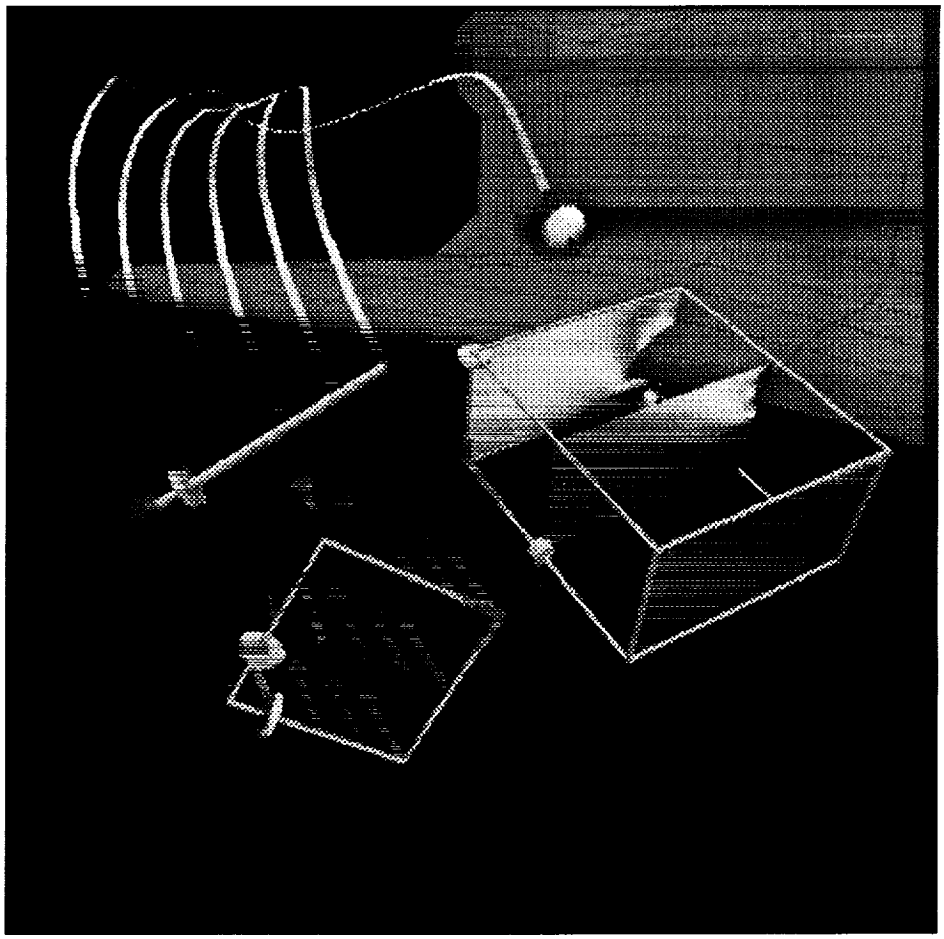


Figure 5: *Probes.* Counterclockwise from left, the point probe with a streamline, 1D probe with streamlines, 2D probe with tufts and 3D probe with color. Visualization techniques are colored by velocity of the vector field from blue (slow), through green, to red (fast).

2.2.3 Two-dimensional planes

The two-dimensional widget samples a set of points arranged in a regular planar grid (similar to the one-dimensional widget). This widget can be translated and rotated freely, and can be scaled independently in two dimensions, much like a 2D window in a desktop-style GUI. Also, the resolution of this widget can be changed independently in these two dimensions. Note that we maintain continuity between the different probes by using the same visual language for these handles.

Using the color technique with this widget produces a cutting plane; similarly, the tufts produce a hedgehog. This widget can be confusing when the advected particle technique is chosen, especially if the sample points are very close together in both dimensions: the visual effect is something like a volume of streamlines, and is not very intelligible. However, if we reduce the number of sample points in one dimension, say down to three, we effectively produce a widget which controls a set of three rakes as group. In this configuration, we have a useful tool once again.

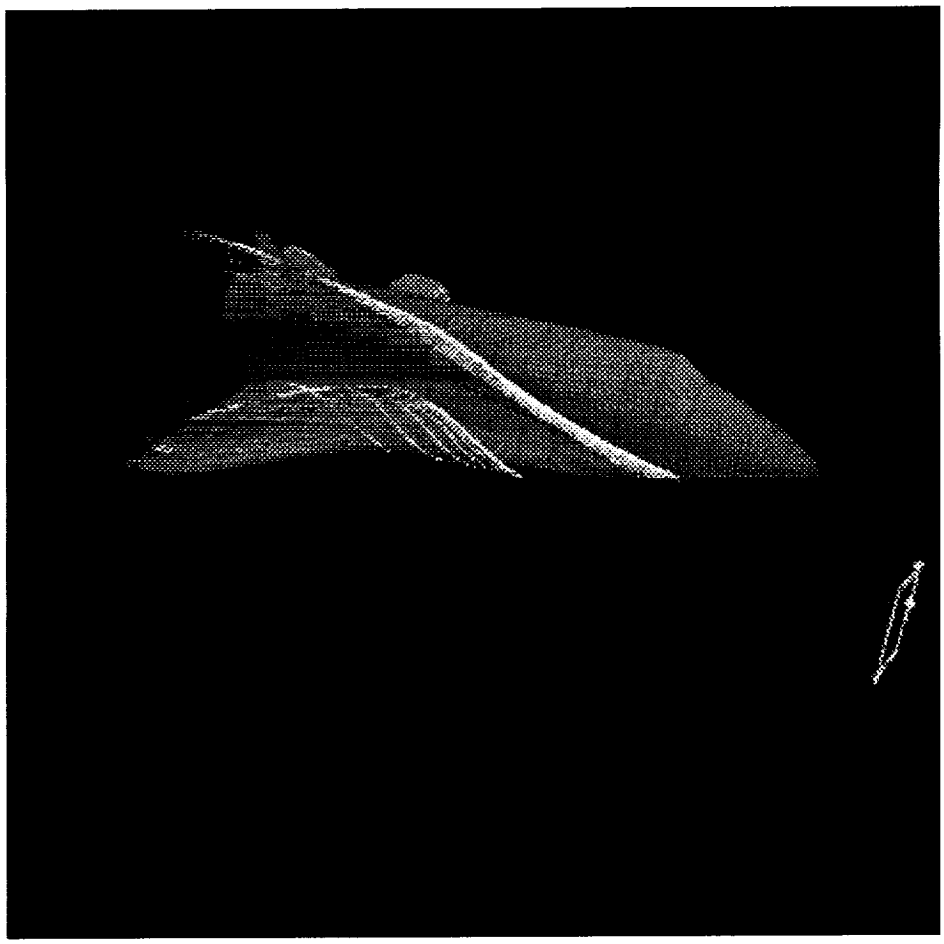


Figure 6: *2D probe with streamlines.* Properly configured, the two-dimensional probe acts like a collection of one-dimensional probes which can be controlled in unison.

2.2.4 Three-dimensional volumes

Finally, the three-dimensional widget generates a volume of sample points. It can be scaled in three dimensions and has resolution sliders for each dimension as well. Like the two-dimensional probe, this widget can produce very confusing visualizations if not parameterized correctly. However, by choosing the color technique and adjusting the resolution sliders so that there are lots of sample points in two dimensions and very few in the third, we can produce a set of cutting planes that can be moved around as a unit.

In our current system, we provide a set of 2D buttons outside of the 3D view for changing the probes from one dimension to another. When a probe changes, it fades from one representation to the next, thus maintaining visual continuity. Another set of buttons changes the visualization technique generated at each sample point. When multiple widgets are being displayed simultaneously, the 2D buttons only affect the last widget used. In this way, we can have many widgets on the screen, each with a different dimension and producing a different visualization technique. We plan to migrate our entire 2D interface into the 3D scene; this will be especially important when exploring a dataset within an immersive virtual reality system, where conventional 2D interfaces are difficult to use.

2.3 Other Visualization Techniques

While our primary aim in this grant is to develop new user interfaces for scientific visualization applications, we have also spent some time developing new visualization techniques that seemed interesting and innovative. The "flux ball" and "smoke rings" techniques described below were developed within the same visualization system as our other widgets, and currently work with regular and curvilinear datasets. Implementing these techniques also exposed some new 3D interface design issues.

2.3.1 Flux Ball

The flux ball (Figure 7) is a method for visualizing the direction of a fluid flow as it passes through a region of space. In our case, we use a spherical region. As fluid flows into or out of the spherical region, we calculate the angle at which it crosses the boundary and compare this with the normal to the sphere's surface. If this angle is small, the fluid is flowing almost directly into or out of the region; when the angle is large, the fluid is flowing tangent to the surface. By sampling this angle at a number of points on the surface of the sphere, we can produce contour lines of similar angles. We draw these contours and color them according to the direction of flow and the magnitude of the angle. Blue indicates flow into the region, red indicates flow out of the region, and the intensity of the color indicates the angle (small angles are more intense than large). The final effect is a set of concentric contours around the sphere oriented in the direction of flow.

Figure 7: *Flux balls.* The three flux balls in this image are embedded in a dataset of convection currents in the Earth's mantle. Blue lines indicate flow out of a ball; red lines indicate flow into a ball. The black lines indicate where the flow is tangent to the surface of the ball. A few streamlines indicate the direction of flow as well.

In our initial implementation of this widget, the sample region was drawn as a fully opaque sphere and often obscured other visualizations and geometry in the scene. Also,

one could not see the contour lines on the side of the sphere facing away from the viewer. A solution to this may be to render the sphere transparently.

2.3.2 Smoke Rings

Smoke rings (Figure 8) are similar to streamlines but do not represent the entire path of a particle through the dataset. Instead, we arrange a set of particles in a ring and advect them all simultaneously through the dataset. At each integration step, we draw a line connecting all of the sample points together. Thus, at the first integration step, we see a ring-shaped object. As this ring of points is advected through the dataset, it deforms according to the vector field data. In order to maintain the ring's visual continuity, if any two adjacent points move too far apart from each other, new points are introduced to fill the gap. Just as with the rake widget, we can see how points that are initially in close formation diverge as they pass through the dataset, so that features such as vortices and divergences are revealed by the ring's deformation.

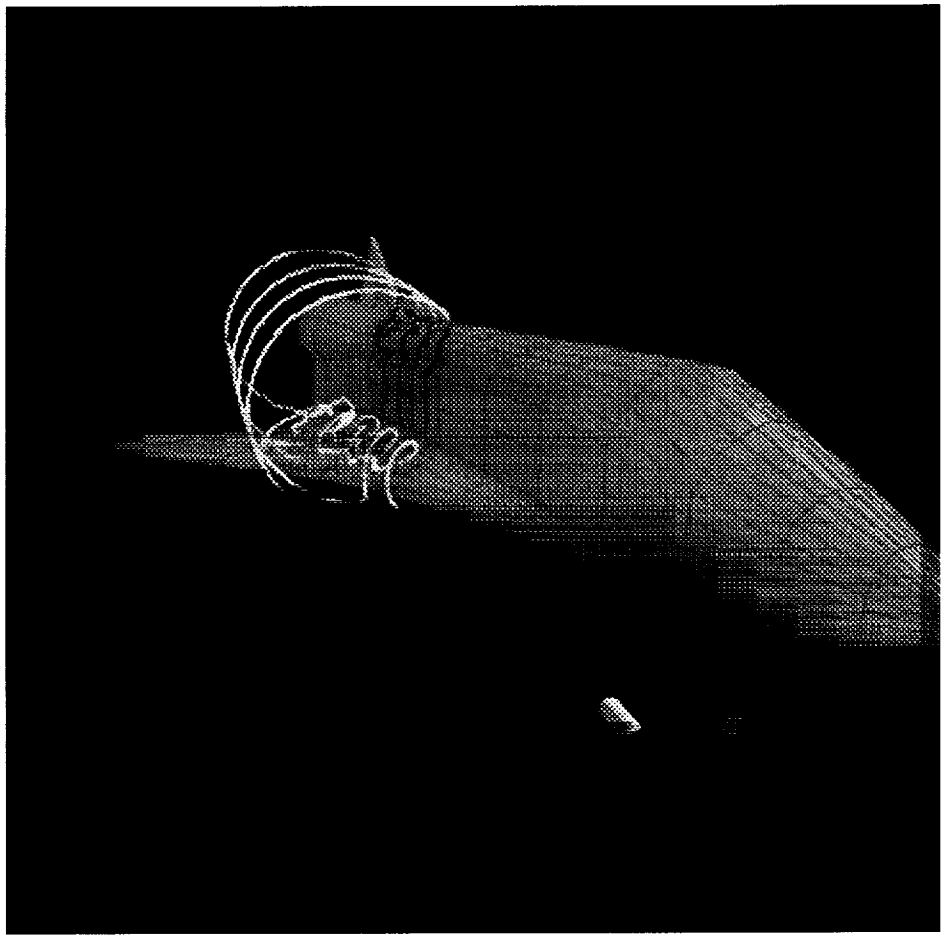


Figure 8: *Smoke Rings.* The smoke ring widget defines a set of sample points which are advected through the dataset. These points, which form a circle at first, flow through the dataset and the circle is deformed accordingly. Points are colored by velocity.

The smoke ring technique was developed by an undergraduate student during a summer internship in our Graphics Group. We are considering extending this technique to arbitrary surfaces advected through the field, using an oriented-particle system.

2.4 Annotation Systems

One of the common tasks of scientists is to maintain accurate records of interesting areas in a dataset, keeping track of a feature's importance and type. In the last year, a Master's student in the Graphics Group built an annotation system for time-varying fluid flow datasets to provide for the creation of these annotations and the subsequent querying and filtering of this dataset. This research was published at Visualization '94 [6].

This annotation system allows the scientist to create typed geometrical point and volume markers, and add associated data, including the date, the user making the annotation, the type of feature, and a description. These can then be filtered either through a 2D interface or through a 3D "Magic Lens" [2], which hides irrelevant annotations as the scientist moves the lens through the dataset.

2.5 User Studies

Last spring, another Master's student in the Graphics Group conducted a user study of some of the positioning widgets within our fluid flow visualization system. The goal of this study was to determine the effectiveness of various widgets and interaction techniques across a range of input devices. The subjects for the study were chosen with a high bias toward computer scientists, but with no concern for their knowledge of CFD concepts.

This work is modeled after a study by Jacob and Sibert [5], which involved the two-dimensional task of matching one square to another target square, and the one-dimensional task of matching the target square's size or color. These tasks were performed by each user, first with a conventional mouse, then with a three-dimensional mouse.

In that study, it was found that for matching position and color, the 2D mouse was more effective, but for controlling position and size, the 3D mouse was better, presumably because these two parameters are cognitively integrable. That is, the users were able to move more efficiently through the task space with the 3D mouse by simultaneously manipulating three separate dimensions. With the 2D mouse, they were required to separate the tasks into individual interactions.

In our study, users were asked to find a number of features (sources, sinks, and vortices) by moving streamlines through an analytically-generated 3D flow field. The positioning techniques available were screen-aligned translation for the 2D mouse, full 3D translation with the 3D mouse, and interactive shadows for both. It was found that the interactive shadows were only really useful when using a 2D mouse, and they confused people who tried to use them with the 3D mouse. Indeed, since the 3D mouse gave them three degrees of translation freedom by itself, the interactive properties of the shadow widgets did not help at all except as a depth cue.

2.6 Techniques for Real-Time Interaction

The computations required by scientific visualization techniques often cannot be done at interactive rates (generally at least ten frames per second). Furthermore, even if the computations can be done quickly enough, a visualization may be so graphically complex that it cannot be rendered in real time.

To address these concerns, we have experimented with time-critical interfaces for isosurfaces and streamlines, using techniques that compute only the local visualization while users interact with a probe, and then gradually complete the computation after the probe is released. This style of interaction is useful both for large static datasets and for time-varying datasets when it is difficult to animate the dataset more quickly than visualizations can be computed. For instance, if an isosurface of a time-varying dataset takes 30 seconds to compute, then each frame of an animation of this isosurface must be computed off-line and viewed later at full speed. By applying time-critical techniques, we can achieve the interactive visualization of complex datasets without memory- and time-intensive techniques.

A version of the direct-manipulation "grab-anywhere" streamlines described above was made time-critical by dynamically adjusting their lengths depending on the amount of computation possible in a given time period (usually about a tenth of a second). As the streamline is dragged, it integrates outward (both forward and backward) from the selected point as far as it can until the next frame must be started. This was part of a research project to minimize lag in virtual environments, to be published in Presence [10].

A complementary approach to achieving real-time interaction is to use multiprocessing techniques to parallelize computations. At the SIGGRAPH conference this past summer, we built a demonstration for the Sun booth based on our NASA-funded CFD research. In this application, we used the Shuttle dataset and parallelized the rake, cutting plane, hedgehog and particle advection visualization techniques on an eight-processor Sparc-10 workstation. We found this to be very effective in maintaining fast interaction speeds for scenes which were not extremely complex, achieving near-linear speedup.

2.7 Time-Critical Scheduling Algorithm

We have recently developed a framework to better integrate the time-critical visualization algorithms already developed (streamlines and isosurfaces) into our existing system. Our implementation [8] includes an algorithm for managing the scheduling of visualization techniques on single- and multi-processor workstations. This algorithm quickly optimizes a *benefit* function which is the product of several relevant factors:

- the inherent value of a given visualization technique
- a hysteresis function to encourage frame-to-frame visual continuity
- the benefit of spending additional time computing the visualization (e.g., computing more points along a streamline)
- the amount of user interaction with the visualization technique

This algorithm is used to schedule the allocation of compute-power for both computation and rendering of visualization techniques, with minimal pipelining of the computations. It has $O(n \log n)$ complexity when generating an optimistically-feasible schedule ($O(n^2)$ complexity for a guaranteed schedule), and iteratively refines its results, so can return a feasible schedule at any time. When allowed to run to completion, the scheduling algorithm achieves near-total, good usage of all the processors in the multiple-processor system.

We intend to carry out informal user studies to determine which aspects of the real-time scheduling are most important in practice. In addition, through working with perceptual psychologists to determine the visual importance of rendered objects and with

scientists to determine the semantic value of various types of visualizations, we would be able to develop much more relevant benefit functions.

Another aspect of time-critical rendering is the perceptually based selective degradation of streamlines and isosurfaces. For example, many of the polygons in a generated isosurface may be too far away from the viewer to be perceptually relevant, and could be culled in favor of those which are closer to the viewer. Fast algorithms for reducing the number of points in streamlines or the number of polygons in isosurfaces could prove very useful in maintaining rapid interaction with the datasets.

3 Brown Personnel

The Brown Graphics Group, directed by Professors Andries van Dam and John F. Hughes, is a team of Ph.D., Masters, and undergraduate students and full-time staff, all of whom work with the UGA system. Professor van Dam, the principal investigator of this research, will, in August, 1995 become the Director of the STC described in Section 1. He and John Hughes are co-authors of the standard computer graphics textbook, *Computer Graphics, Principles and Practice,* along with James Foley and Brown Ph.D. Steven Feiner. Van Dam is a co-founder of ACM SIGGRAPH and co-founder and first chairman of Brown University's Computer Science Department. The Graphics Group staff includes a Senior Research Scientist (Bob Zeleznik), a Research Scientist (Timothy Miller), two Interface Developers (Kenneth Herndon and Tom Meyer), an Interactive Illustrations Designer (Dan Robbins), a part-time Technical Administrator (Nate Huang), and an Educational Outreach Director (Anne Morgan Spalter). Our interface design team has experience in animation, modeling, graphic design, industrial design, and communication techniques. A number of undergraduate students complement the group by assisting on various research projects such as the 2D and 3D interface to the modeling and animation system. Five students work part-time to support computers, video-teleconferencing, and other AV equipment.

The principle researchers being funded by this grant are Kenneth Herndon and Tom Meyer. Additional research is being done by Dan Robbins, and students Currier McEwen and Matt Ayers.

4 Facilities and Equipment at Brown

The facilities at Brown include a variety of workstations from HP, IBM, DEC, Sun and SGI. Our Virtual Reality Lab contains a FakeSpace Labs BOOM, a Virtual Technologies CyberGlove, and an Ascension Long-range Bird tracker. We also use a StereoGraphics VR setup (LCD-shutter glasses, two Logitech 3D mice and a high-scan-rate monitor attached to a Sun SPARC 10 GT workstation). A full audio/video editing system is used to record footage directly from workstation screens and to edit videotapes.

We also maintain a World Wide Web site (`http:/www.cs.brown.edu/research/graphics`) which contains general information about our group and research projects.

5 Current Support

Title: Three-dimensional user interfaces for scientific visualization
Sponsor: NASA

Description: Develop novel 3D user interfaces for managing elements of scientific visualization applications.
Amount: $100,000/yr.
End date: April '96

Title: NSF/ARPA Science and Technology Center for Computer Graphics and Scientific Visualization
Sponsor: National Science Foundation/ARPA
Description: Advance the state of the art in computer graphics techniques through collaboration in a national distributed center.
Sites: Brown University, California Institute of Technology, Cornell University, University of North Carolina at Chapel Hill, and University of Utah
Amount: $500,000/yr.
End date: February '96

Title: Multiprocessor 3D interactive graphics
Sponsor: Sun Microsystems
Description: Investigate operating systems and graphics performance for multi-threaded, multi-processor graphics applications.
Amount: $100,000/yr.
End date: July '95

Title: Self as a first programming language
Sponsor: Sun Microsystems Group
Description: Investigate using the Self language as a first programming language for undergraduate students at Brown.
Amount: $70,000/yr.
End date: July '95

Title: Computer graphics research
Sponsor: Autodesk
Description: 3D user interface research
Amount: $100,000/yr.
End date: July '95

Title: Computer graphics research
Sponsor: Microsoft
Description: 3D and 4D graphics
Amount: $100,000/yr.
End date: July '95

Title: Computer graphics research
Sponsor: TACO
Description: General support of the graphics group's research program
Amount: $100,000/yr.
End date: December '94

6 References

[1] Bier, E.A. Snap-dragging in three dimensions. In *Proceedings of the 1990 Symposium on Interactive 3D Graphics*, published as *Computer Graphics*, Vol. 24, No. 2, pages 193-204, March 1990.

[2] Bier, E.A., Stone, M.C., Pier, K., Buxton W., DeRose, T. Toolglasses and Magic Lenses: The see-through interface. In *Computer Graphics (SIGGRAPH'93 Proceedings)*, Vol 27., pages 73–80, August 1993.

[3] Herndon, K.P., Meyer, T. 3D widgets for exploratory scientific visualization. In *Proceedings of the 1994 Symposium on User Interface Software and Technology*, pages 69–70, November 1994.

[4] Herndon, K.P., Zeleznik, R.C., Robbins, D.C., Conner, D.B., Snibbe, S.S., and van Dam, A. Interactive shadows. In *Proceedings of the 1992 Symposium on User Interface Software and Technology*, pages 1–6, November 1992.

[5] Jacob, R.J.K., Sibert, L.E. The perceptual structure of multidimensional input device selection. In *Proceedings of ACM CHI'92 Conference on Human Factors in Computing Systems*, pages 211–218, 1992.

[6] Loughlin, M.M., and Hughes, J.F. An annotation system for 3D fluid flow visualization. In *Proceedings of Visualization '94*, 1994.

[7] Meyer, T., and Globus, A. Direct manipulation of isosurfaces and cutting planes in virtual environments. Technical Report 93-54, Department of Computer Science, Brown University, 1993.

[8] Meyer, T., and Hughes, J. F. Scheduling time-critical graphics on multiple processors. Submitted to *SIGGRAPH'95*.

[9] Stevens, M.P., Zeleznik, R.C., and Hughes, J.F. An architecture for an extensible 3D interface toolkit. In *Proceedings of the 1994 Symposium on User Interface Software and Technology*, pages, November 1994.

[10] Wloka, M.M. Lag in multiprocessor virtual reality. PRESENCE: Teleoperators and Virtual Environments, Vol. 4, No. 1, 1995.

[11] Zeleznik, R.C., Conner D.B., Wloka, M.M., Aliaga, D.G., Huang, N.T., Hubbard, P.M., Knep, B., Kaufman, H., Hughes, J.F., and van Dam, A. An object-oriented framework for the integration of interactive animation techniques. In *Computer Graphics (SIGGRAPH'91 Proceedings)*, Vol. 25, No. 4, pages 105–112, July 1991.

3D Widgets for Exploratory Scientific Visualization

Kenneth P. Herndon and Tom Meyer

Brown University
Department of Computer Science
Providence, RI 02912
(401) 863-7693; {kph,twm}@cs.brown.edu

1 Introduction

Computational fluid dynamics (CFD) techniques are used to simulate flows of fluids like air or water around such objects as airplanes and automobiles. These techniques usually generate very large amounts of numerical data which are difficult to understand without using graphical scientific visualization techniques. There are a number of commercial scientific visualization applications available today which allow scientists to control visualization tools via textual and/or 2D user interfaces. However, these user interfaces are often difficult to use. We believe that 3D direct-manipulation techniques for interactively controlling visualization tools will provide opportunities for powerful and useful interfaces with which scientists can more effectively explore their datasets. A few systems have been developed which use these techniques, including [1].

In this paper, we will present a variety of 3D interaction techniques for manipulating parameters of visualization tools used to explore CFD datasets, and discuss in detail various techniques for positioning tools in a 3D scene. We generally call these techniques *3D widgets* [2].

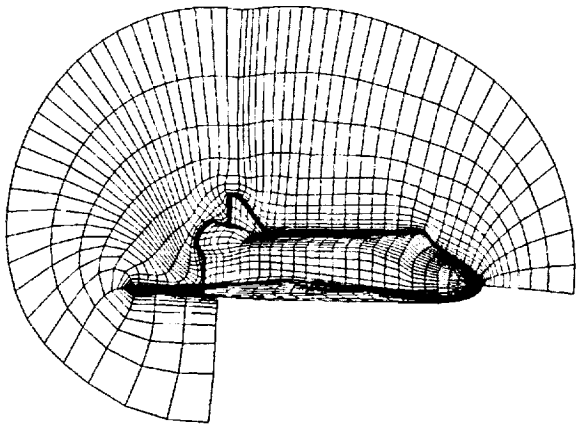


Figure 1: A 3D curvilinear grid for the Space Shuttle.

Our environment, built on top of the UGA system [4], supports both vector and scalar fields. The data may be arranged in a regular grid or it may be deformed using a curvilinear grid to provide more detail in areas with more complex flow. In a typical curvilinear dataset (Figure 1), the computation grid is wrapped around the body of an aircraft and scaled so that there are many more sample points in the boundary region (near the surface) than in other areas.

We are conducting this research project under contract to NASA in order to provide scientists there with more effective tools for exploring CFD datasets. No formal user studies have yet been conducted to verify the general usability of our interfaces.

2 3D widgets for scientific visualization

3D widgets are naturally suited for CFD visualization applications because the data are inherently 3D. Also, several of the visualization techniques commonly used in CFD visualization are based on real-world tools used in actual wind tunnels (*e.g.*, "rakes" of streamlines simulate smoke-emitting rakes). With these metaphors in mind, 3D widgets can be constructed to control parameters of commonly used visualization techniques.

In general, the design of a widget must consider two conflicting requirements: that the widget have adequate geometry to disclose its affordances; and that this geometry not be so complex that it obscures other objects in the scene. When exploring or analyzing a dataset, the visualizations (*i.e.*, streamlines, cutting planes, etc.) of the data are usually the most important elements in the scene. In these kinds of applications, it is crucial that 3D widgets provide only the necessary functionality with a minimum of geometry.

In general, a widget's degrees of freedom should correspond to the type of data it affects (*e.g.*, a widget which produces a scalar value should be constrained to a single degree of freedom, as in a slider or a knob). Also, widgets should provide useful visual feedback for the user's actions (*e.g.*, highlighting when selected).

We have implemented 3D widgets for the following visualization techniques: streamline and particle path; rake of streamlines or particle paths; array of tufts ("hedgehog"); scalar and vector probe; isosurface; and cutting plane.

Each 3D widget provides interactive access to a technique's parameters, such as position, orientation, resolution, etc. When possible, we align a widget's range of motion with the effect it has on a visualization technique. For example, the rake's resolution handle (Figure 2), which determines the spacing and number of streamlines displayed, slides along the bar of the rake; also, the arrow-shaped extent handles of the "hedgehog" are aligned with and move in three orthogonal directions. It is more difficult to create interfaces to some abstract parameters such as the integration step of a streamline.

3 Case study: Positioning widgets

A very common task for scientists is specifying the position of a 3D probe in a dataset (*e.g.*, placing the source of a streamline in a vector field). To demonstrate the range of choices in 3D widget design,

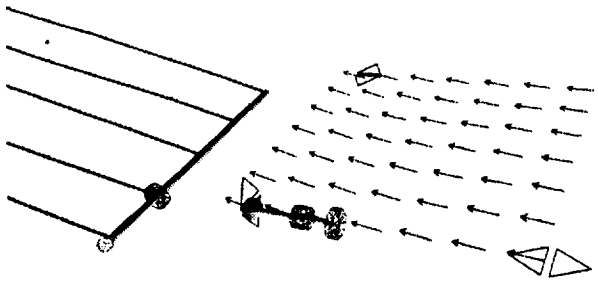


Figure 2: The rake (left) and hedgehog (right) 3D widgets. The cylindrical shapes are sliders which control resolution of streamlines or tufts. The arrow-shaped controls on the hedgehog modify the extent of the array of tufts in each of three dimensions.

we will discuss the designs of several techniques for positioning widgets which we have implemented in our system.

The default positioning technique is direct-manipulation screen-aligned translation. To move objects in three-space, we have added our "interactive shadow" [3] widgets to this environment. A combination of screen-space translation and interactive shadows allows the user to easily place an object in a 3D scene without having to move the camera. The "shadow" widgets also provide useful depth cues for 3D widgets and other objects in the scene.

However, it is easy for a geometry in the scene (*e.g.*, the Shuttle fuselage) to hide the "shadow" widgets and render them unusable. Another technique, called "object handles", attaches three objects (in our case, simple line segments) to the selected object and aligns them with the world coordinate system. These widgets provide much of the same functionality as the "interactive shadows", but do not provide any depth cues.

Each of these techniques use features of Cartesian coordinate space to position objects. While they are useful techniques in many situations, problems arise when using them to explore curvilinear datasets because the data was structured based on the geometry of the objects being modeled (as in the Space Shuttle), and it is often useful to move objects relative to this geometry. We have extended the handle metaphor to accomodate these situations.

"Grid-aligned handles" are especially useful for curvilinear datasets which are specially fitted to a physical model like an airfoil. As shown in Figure 3, the handles trace out nearby computation grid lines. When a handle is dragged, the selected object is constrained to move along the grid line. Using this technique, it is straightforward to translate objects along complex surfaces whose geometry is reflected in the computation grid, such as the leading edge of an airfoil. Furthermore, because these handles display the nearby structure of the grid, users can possibly gain a better understanding of the dataset as they explore it with these widgets.

It can also be useful to work with interaction techniques based on the data being visualized. For example, the vector probe widget in Figure 3 consists of a grey spherical sample point, an arrow which represents the direction of flow at that point, and a disk which represents the plane perpendicular to the vector. By dragging the arrow component, the sample point can be moved along the streamline formed by the flow through that point. The disk is used to move the sample point perpendicular to the flow, allowing the user to explore nearby streamlines in the flow field.

We can use this same general probe widget to visualize scalar data. In this case, the vector component displays the gradient of a scalar field. Pulling the vector changes the value at which the isosurface is computed; translating the disk moves the sample point along the isosurface.

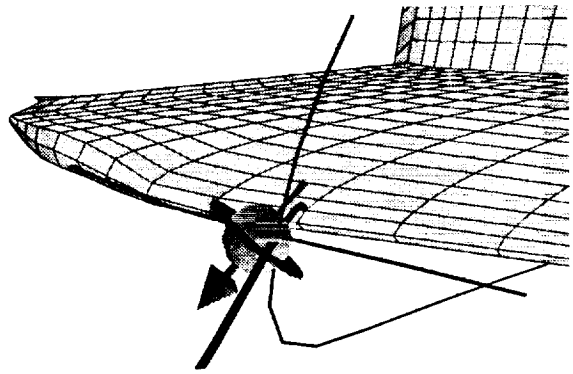


Figure 3: A vector probe widget and three grid-aligned object handles. The three lines extending through the probe widget are the handle widget and serve both as a Frenet frame for the point in the computation grid closest to the probe and as constrained translation widgets. The thicker grid handle extends outward from the surface of the wing.

4 Future Work

We are continuing to explore techniques to further simplify the graphical representations of our widgets without impeding their functionality. Additionally, the widgets described in this paper were rapidly protoptyped to explore the design space. After we have done user studies with different widget designs, we would like to redesign our tools so that they have a more consistent interface. The general probe widget, which can be used as an interface to a vector or scalar probe, a streamline, or an isosurface, is a step in this direction.

Acknowledgments

This work was supported primarily by NASA Ames. Support is also provided by the NSF/ARPA Science and Technology Center for Computer Graphics and Scientific Visualization, by ONR Contract N00014-91-J-4052, ARPA Order 8225, and by the sponsorship of IBM, NCR, Sun Microsystems, Hewlett Packard, and Digital Equipment Corporation. We thank Steve Bryson, Andries van Dam, and the members of the Brown University Graphics Group, especially Jeremy Katz and Lars Bishop, for their help and support.

References

[1] Steve Bryson and Creon Levitt. The virtual windtunnel: An environment for the exploration of three-dimensional unsteady flows. In *Visualization '91*, pages 17–24, 1991.

[2] D. Brookshire Conner, Scott S. Snibbe, Kenneth P. Herndon, Daniel C. Robbins, Robert C. Zeleznik, and Andries van Dam. Three-dimensional widgets. *Computer Graphics (1992 Symposium on Interactive 3D Graphics)*, 25(2):183–188, March 1992.

[3] Kenneth P. Herndon, Robert C. Zeleznik, Daniel C. Robbins, D. Brookshire Conner, Scott S. Snibbe, and Andries van Dam. Interactive shadows. *1992 UIST Proceedings*, pages 1–6, November 1992.

[4] Robert C. Zeleznik, D. Brookshire Conner, Matthias M. Wloka, Daniel G. Aliaga, Nathan T. Huang, Philip M. Hubbard, Brian Knep, Henry Kaufman, John F. Hughes, and Andries van Dam. An object-oriented framework for the integration of interactive animation techniques. *Computer Graphics (SIGGRAPH '91 Proceedings)*, 25(4):105–112, July 1991.

An Annotation System for 3D Fluid Flow Visualization

Maria M. Loughlin[1]
Cambridge Research Lab
Digital Equipment Corporation

John F. Hughes[2]
Department of Computer Science
Brown University

Abstract

Annotation is a key activity of data analysis. However, current systems for data analysis focus almost exclusively on visualization. We propose a system which integrates annotations into a visualization system. Annotations are embedded in 3D data space, using the Post-it [3] metaphor. This embedding allows contextual-based information storage and retrieval, and facilitates information sharing in collaborative environments. We provide a traditional database filter and a Magic Lens[4] filter to create specialized views of the data. The system has been customized for fluid flow applications, with features which allow users to store parameters of visualization tools and sketch 3D volumes.

1 Introduction

In a study to characterize the data analysis process, Springmeyer *et al.* [15] observed scientists analyzing different types of scientific data. The study found that recording results and histories of analysis sessions is a key activity of the data analysis process. In each session, the scientists recorded notes, and inspected previous notes. Two distinct types of annotating were observed:

- *recording*, or preserving contextual information throughout an investigation

  and

- *describing*, or capturing conclusions of the analysis sessions.

Despite the importance of annotation, current systems for data analysis emphasize visualization, focusing on the generation of visual displays. Little or no annotation support is available: for example, Springmeyer *et al.* noted that the recording media used by scientists in their study included notebooks, scratch paper, and Post-it notes.

In this paper, we describe a system that supports annotation as an integrated part of a fluid flow visualization system. Unlike typical annotations on static 2D images, our system embeds annotations in 3D data space. This immersion makes it easy to associate user comments with the features they describe. To avoid clutter and data hiding, annotations are represented by graphical annotation markers that have associated information. Therefore graphical attributes of the markers, such as size and color, can be used to differentiate annotations with different functions, authors, creation dates, etc.

Annotations can easily be added, edited and deleted. Also, multiple sets of annotations can simultaneously be loaded into a visualization. This allows scientists, collaborating on a data set, to use annotations as a form of communication, as well as a history of data analysis sessions. Annotation markers also aid scientists in navigating through the data space by providing landmarks at interesting positions. Figure 1(a)-(c) shows the visualization environment, annotation markers, and the annotation content panel. Figure 1(d) shows a Magic Lens filter which hides the annotation markers and widget handles. The implementation has been applied to three-dimensional Computational Fluid Dynamics (CFD) applications. However, the techniques can be used in visualization systems of many disciplines. The design can also be extended to 3D stereo and virtual-reality environments.

The rest of this paper is organized in five sections. In section 2 we review previous approaches to annotation. Section 3 describes design guidelines for annotation systems. Section 4 details our implementation of annotation within a 3D modeling and animation system. In the last two sections, we discuss possible future work and present our conclusions.

2 Background

Scientific visualization systems provide little, if any, support for annotation. For example, Application Visualization Sys-

[1]One Kendall Square, Cambridge, MA 02139. email: loughlin@crl.dec.com. phone: 617-621-6618

[2]Box 1910, Providence, RI 02912. email: jfh@cs.brown.edu. phone: 401-863-7638

[3]Post-it is a trademark of 3M.

[4]Magic Lens is a trademark of Xerox Corporation.

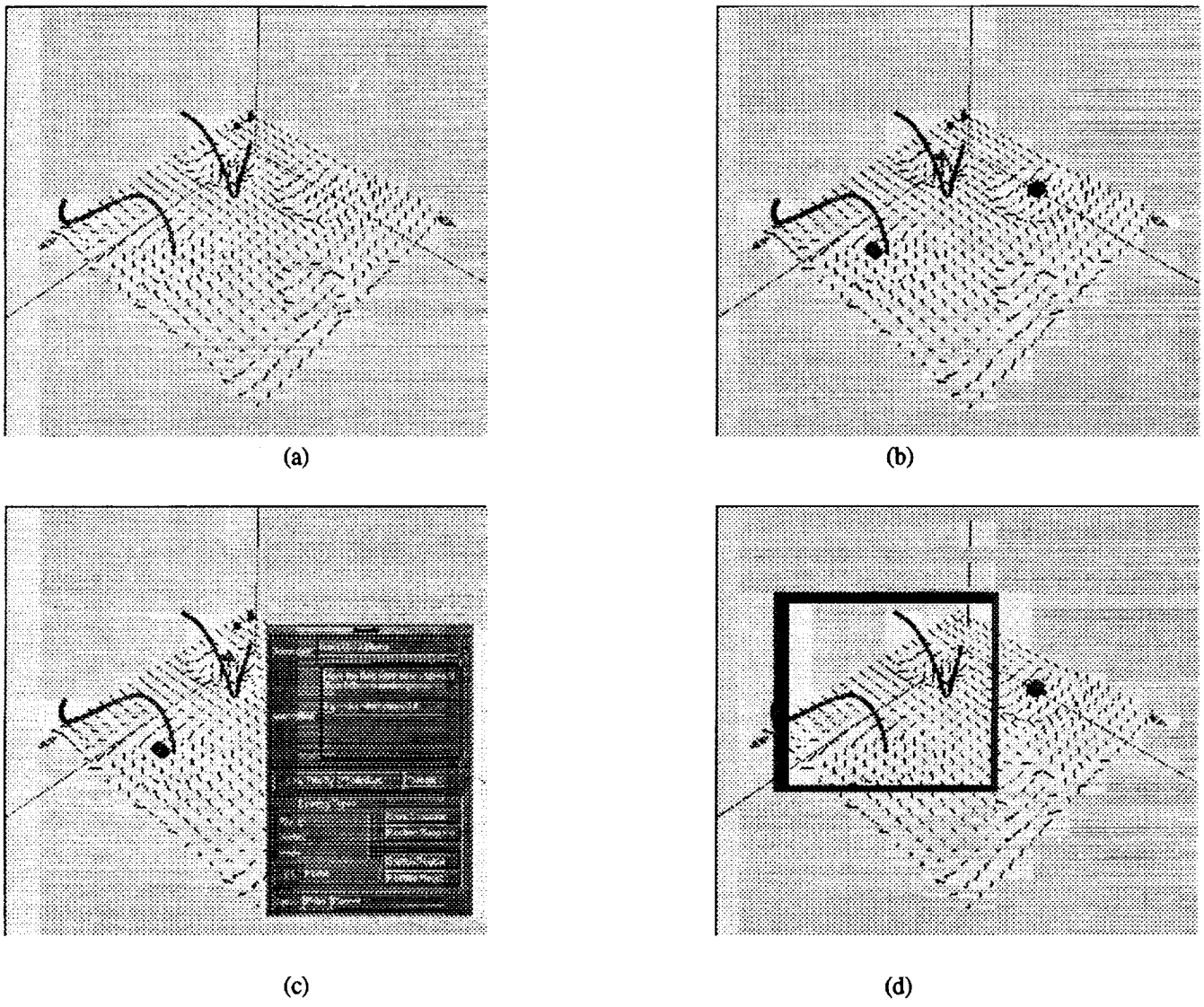


Figure 1: The visualization and annotation system
(a) hedgehog and streamlines showing 3D fluid flow, (b) annotation markers (small geometric objects) placed at points of high velocity, (c) annotation content panel, (d) Magic Lens filter hiding annotation markers and widget handles.

tem (AVS) [17] and Flow Analysis Software Toolkit (FAST) [1], two software environments for visualizing scientific data, facilitate attachment of labels to static 2D images. These systems also allow a user to record a sequence of interactions with the visualization. This support is useful for generating presentations from the data, but does not facilitate the recording and describing operations observed by Springmeyer *et al.*

Outside the scientific visualization domain, annotations of various sorts have been integrated in different applications.

MacDraw, a 2D paint program, introduced a notes feature, which allows static 2D annotations using the Post-it metaphor. Media View [12] [13], a multi-media publication system, extends the conventional paradigm of a document and allows annotations in all media components including text, line art, images, sound, video sequences, and computer animations. The format of annotations has been expanded, but their use is still limited to presentation of information in a static environment.

Document annotation is used as a means of communication in the Wang Laboratories multi-media communication system, Freestyle [8]. Freestyle's multi-media messages are based on images, including screen snapshots and hand-drawn sketches. Furthermore, this system allows synchronization of input modalities, such that messages can contain information about the process by which they were created. Freestyle advances the concept of annotations as communicators, but does not address the issues of clutter and management of annotations in the environment.

Verlinden *et al.* [18] developed an annotation system to explore communication in Virtual Reality (VR) environments. In general, annotation in immersive VR systems is restricted, as the user must interrupt the session to interact with objects in the real world, such as notebooks and computer monitors. Verlinden's system overcomes this problem by embedding verbal annotations in the VR space. The annotations are represented as visual 3D markers. When the user activates a marker, the verbal message stored with that marker is played. This system is unique in that it embeds annotations in 3D scenes, but it is limited to verbal annotations and provides no support for annotation filtering. It also limits annotations to a fixed position in a time-based environment.

3 Design Issues

We have extracted, both from the Springmeyer *et al.* study and from our own experience with scientific visualization, a set of three design guidelines that seem appropriate for an annotation system. These guidelines, discussed below, formed the basis for the design of our system.

Guideline 1: To support ongoing recording of contextual information, an annotation system must be an integral part of a visualization system. Effective placement and storage of annotations are required.

Traditionally, annotations to scientific visualizations are recorded on paper or in electronic files, and both the dataset and the files are labeled to mark their association. This separation of data and annotations means that some effort is required to find the data features described by annotations. The 3D data space of many scientific applications provides the context in which annotations should be placed. Recording annotations in this space capitalizes on human's spatial senses by facilitating the retrieval of information based on its spatial location in the visualization.

However insertion of annotations in the data space creates an immediate conflict between the annotation and visualization functions: both compete for screen territory. We do not wish to impose any restrictions on the amount of information that can be recorded. At the same time, since information is contained in the data itself, we do not wish data to be obscured by annotations. Our approach is to decompose an annotation into:

- an *annotation marker* or small geometric object that identifies the position of the annotation in the data space

  and

- an *annotation content* in which a user stores information.

The geometry and graphical attributes of markers are chosen so that they are easily distinguished from existing visualization tools. By clicking on a marker, a user can expand the associated annotation to read or edit its content. Separation of the annotation's content from the annotation marker in this way allows direct insertion of arbitrarily large annotations.

Guideline 2: Annotations must be powerful enough to capture information considered important by the user.

There are different types of information. Tanimoto [16] distinguishes between *data* (raw figures and measurements), *information* (refined data which may answer the users' questions) and *knowledge* (information in context). Bertin [3] classifies the levels of information in a similar way. He considers information as a relationship which can exist between elements, subsets or sets. The broader the relationship, the higher the level of information. We assume that an annotation system should be able to store information at each of these levels - scientists need to record both the data values at probe points in the data set, and a higher-level analysis of these figures.

Although some data, such as date of creation and author, are likely to be relevant to all applications, it is possible that knowledge can be captured only when an annotation system is customized for a specific application. The customization would ensure that annotations can represent information relevant in the context of the application. For example, if the data of a particular application is time-varying, the annotation system should provide time-varying annotations that can track the features being described.

In our annotation system, we provide support for different types of information in two ways. First, within each

annotation, scientists can record both numerical and textual details, and high-level information specific to fluid flow. This is discussed in section 4.4. Second, the system supports hierarchically-organized annotations. The hierarchical structure allows scientists to record facts in separate annotations, and group related annotations in sets that describe broader observations.

It is also important to consider the modalities that are available for capturing information in an annotation system. Two dimensional text, graphics and images are the standard annotation modalities; aural annotation is also a candidate. Chalfonte, in an experiment on the use of annotation for collaborative document authoring, found aural annotations a richer and more effective medium for high-level communication [5]. Freestyle shows that coordinating hand/cursor movements with textual and aural annotations is also effective.

In our current implementation, we use 2D text and 3D volumes to store information. In the future, we would like to use different interaction techniques for information capture.

Guideline 3: We need to consider the established rules of user interface (UI) design, because the UI of an annotation system will play a key role in determining its acceptance (or lack thereof) by scientists.

We considered many UI rules [7] and designed our annotation system accordingly. One rule states that a UI should allow users to work with minimal conscious attention to its tools. We achieve this goal by using a direct manipulation interface, that is, an interface in which the objects that can be manipulated are represented physically. For example, the volume of data affected by the Magic Lens filter can be controlled directly by moving and resizing the physical representation of the lens. Another design rule states that an interface should provide feedback, e.g., on the current settings of domain variables. In our system, annotation markers give visual feedback on the location of annotations and marker geometry gives feedback on annotation content.

Because the geometric data space of fluid flow applications has three dimensions, we considered design issues specific to 3D graphical user interfaces [6]. One issue is the complexity introduced by 3D viewing projections, visibility determination, etc. A second issue is that the degrees of freedom in the 3D world are not easily specified with common hardware input devices. A third issue is that a 3D interface can easily obscure itself. We use guidelines outlined by Snibbe *et al.* [14] to deal with these problems. For example, we provide shadows, constrained to move in a plane, to simplify positioning of annotation markers (see section 4.3.2). We provide feedback on the orientation of the data by optionally drawing the principal axes and planes. We also ensure that annotations do not obscure data, by making it easy for a user to change the viewpoint and resize or hide annotation markers.

4 Implementation

This section describes the annotation system we have implemented. We begin by setting a context for the system with a description of fluid flow visualization and the software development environment. Then we discuss the main components of the annotation system: the annotation markers, support for information capture, and interaction techniques.

4.1 Fluid Flow Visualizations

Computational fluid dynamics (CFD) uses high speed computers to simulate the characteristics of flow physics. Computed flow data is typically stored as a 3D grid of vector and scalar values (e.g., velocity, temperature, and vorticity values), which are static in a steady flow, and change over time in an unsteady flow. CFD visualization tools allow a scientist to examine the characteristics of the data with 3D computer displays.

Interaction with the visual representation is essential in the exploration and analysis of the data, and has three goals: *feature identification*, *scanning*, and *probing* [9]. Feature identification techniques help find flow features over the entire domain, and give the scientist a feel for the position of interesting parts of the flow volume. An example of this type of technique is a vector hedgehog, a three-dimensional array of velocity vectors. Scanning techniques are used to interactively search the domain, by varying one or more parameters, through space or through scalar and vector field values. Scanning techniques include cutting planes (planar surfaces which slice the 3D grid and show scalar field value at each grid point of the plane) and iso-surfaces (three-dimensional surfaces of a constant scalar value). Probing techniques are localized visualization tools, typically used to gather quantitative information in the final step of investigating a flow feature. Examples of probing tools include streamlines and particle paths, which show the path in which a particle would flow if positioned in a steady or unsteady fluid flow.

The Computer Graphics Group at Brown University has developed a flow visualization system, to study new modes of interaction with flow tools. The annotation system was developed as part of this flow visualization system. This provided a test-bed for techniques to integrate visualization and annotation functionality.

4.2 The Development Environment

The annotation system was developed using FLESH, an object oriented animation and modeling scripting language [11], and C++. In the FLESH language, scenes are described as collections of objects. The FLESH objects defined for the annotation system include geometric objects such as annotation markers, 3D volumes and lenses, and non-geometric objects, such as holders for collections of annotations and an annotation filter. Some of these FLESH classes have corresponding C++ classes, in which data is stored and compute-intensive operations performed. This allows us to benefit from the

power of an interpreted interactive prototyping modeling system and the efficiency of a compiled language.

4.3 Annotation Markers

Annotations are represented in the 3D data space by small geometric markers. Each marker has an associated content which the user can edit at any time.

4.3.1 Marker Geometry and Graphical Attributes

The geometry of a marker gives visual feedback on the content of the annotation. In the fluid flow visualizations system, the user can define annotation keywords (e.g., plume, vortex), and select a geometry to associate with each keyword. Then, when the user assigns a keyword to an annotation in the system, the annotation's marker takes the associated shape. It is likely that other mappings between graphical attributes of markers and annotation content would also be useful. For example, the color saturation of a marker could depend on the age or priority of the annotation.

The graphical attributes of annotations are also user-customizable. The size and color of all markers in one level of hierarchy can be changed. We predict that this feature would be useful if many scientists work collaboratively on a data set, and each scientist defines a unique color and size for her markers.

4.3.2 Marker Behavior

Since the function of a marker is simply to identify points of interest in the visualization, its behavior is quite simple. A marker is created when the user presses the annotation push-button. It appears at the point on which the user is focussed, making it easy for the user to position it near the feature of interest.

The scientist can translate and rotate markers with simple mouse movements. He can also project interactive shadows of the marker on the planes defined by the principal axes [10]. Each shadow is constrained to move in the plane in which it lies. If a user moves a shadow, the marker moves in a parallel plane. This constrained translation helps in precisely positioning a marker.

Markers can be highlighted in response to a filter request. In the current system, the color of a marker changes to a bright yellow when highlighted. This simple approach seems adequate. However, the user may change this highlight behavior, by, for example, having highlighted markers flash between alternating colors.

Since the features of unsteady fluid flows change over time, a user would like the annotation describing a particular feature to follow the feature's movement in the visualization. The current annotation system provides partial support for this by allowing the user to specify the position of an annotation at any number of points in time. The annotation markers then linearly interpolate between the specified positions in time.

4.4 Knowledge Stored

Our annotations can store generic information, as well as information specific to fluid flow applications. The generic information includes keyword, textual summary and description, author, and date. Some of this information (author and date) are captured implicitly when the annotation is created. The rest must be explicitly added after the scientist has opened the annotation by clicking on it. This data entry is performed via a 2D Motif panel of buttons and text widgets. We consulted with fluid flow experts to understand how the information content of annotations could be customized for fluid flow applications.

4.4.1 Parameters of Visualization Tools

One of the key additions to the annotation system suggested by the fluid flow experts results from the interactive nature of fluid flow analysis. As described earlier, a scientist must insert flow visualization tools (such as streamlines and iso-surfaces) in the data space to see the underlying data. Much time is spent determining which tools most effectively highlight a feature, and positioning and orienting both the tools and viewpoint to best show off the feature being described. Springmeyer *et al.* observed this activity of the data analysis process, and described it as *orientating* the data, or altering a representation to gain perspective.

To support this activity, our concept of an annotation was expanded to include parameters of flow visualization tools. When a user wishes to store the parameters of a set of tools, he or she presses a button to indicate that a set of tools is being saved, and then clicks on the tools of interest. The time-varying location, orientation, size, and other parameters of the tools are saved with the annotation. This can be repeated any number of times for different groupings of tools with different parameters. When an annotation is restored, the user is presented with a list of all saved sets of tools, and can recover each set of tools to see how they illustrate the annotated feature.

4.4.2 3D Volume Descriptions

It also became obvious that annotation markers, which are appropriate for locating point features in a visualization, are not sufficient for CFD applications. Fluid flows contain volume features, such as vortices (masses of flow with a whirling or circular motion) and plumes (mobile columns of flow). Users may want to associate an annotation with a region of the data space, rather than a single point in the space. We therefore need a way to sketch a volume in the data space. The volume-sketching method must be intuitive, so that flow scientists (who may not be interested in becoming artistic volume sculptors!), can easily describe the volume. Also, the resolution of the volumes sketched need only be as precise as the grid on which the flow field is defined.

We provide a simple method to sketch volumes. The user positions "pegs" that define the extreme vertices of the volume to be drawn. The pegs are created and moved within

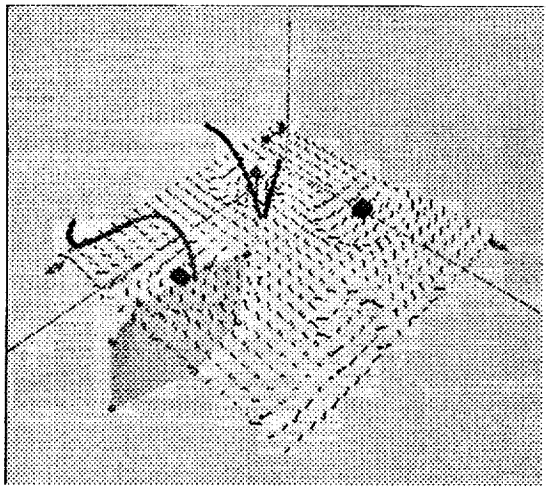


Figure 2: A volume defined as the convex hull of a set of pegs.

the visualization in a way similar to the creation and translation of annotation markers. When the user is done positioning pegs, the system computes their convex hull using the quickhull algorithm [2]. The boundary of the volume, defined by the convex hull, is rendered in either wireframe or transparent mode. Vertices can be added, deleted and moved, and the volume redrawn, until the volume is accurate. Figure 2 shows a volume which has been defined in this way.

This implementation provides a simple means to draw volumes. However, since it uses the convex hull of the pegs, certain shapes, such as a 3D "L" shape, cannot be sketched.

4.5 Retrieving the Annotations

Effective information retrieval and communication requires that a user can easily identify annotations relating to a specific topic, by a specific author, etc. The annotation system facilitates such data filtering in two ways.

First, a traditional database filter is provided. The user can specify data selection criteria (such as the annotation creation date, author, or keyword), via a 2D Motif panel. The markers of annotations that satisfy the search criteria are highlighted.

A second filter uses the Magic Lens metaphor introduced by Bier *et al.* [4]. A Magic Lens filter is a rectangular frame, placed in front of the visualization, that appears as if it moves on a transparent sheet of glass between the display and the cursor. The lens performs some function (which may use information from application-specific data-structures) on the application objects behind it.

Four functions are defined for the lens in the annotation system. The first sets the color of all objects, except annotation markers, to gray. This helps users find markers in a cluttered scene. The second lens function displays only the annotations that satisfy the criteria specified in the Motif database filter. The third lens function hides all annotation markers behind the lens. Finally, the default function hides all annotation markers and all interaction handles on the visualization tools behind the lens. Many other interesting lens functions could be defined. One such function could remove all fluid flow tools except those in the user sketched volume behind the lens.

We believe that the magic lens filter alleviates the problem of visualization and annotation functions sharing the same screen space. Using the lens, a scientist can tightly integrate the two functions when appropriate. When she wishes to focus exclusively on either visualization or annotation, the clutter introduced by the other component can be hidden.

5 Future Work

The work described in this paper could be expanded in a number of ways, in both the fluid flow application and in new environments.

There are a number of opportunities for the fluid flow application. The facility for recording parameters of visualization tools could be extended to record view parameters. Then, flow tools could automatically be viewed from the same viewpoint and with the same magnification as when their parameters were saved. Annotations could also become more active in the data investigation process. For example, annotation markers could be used as seed points for automatic flow feature-characterization code. The output of the feature-characterization code (i.e., specifications of the feature found) could then be added to the annotation content. Feature-characterization code could also be used to improve support for time-varying annotations. If the location of an annotation marker were constrained to the feature's position (as found by feature-characterization code), the marker would follow the movement of the feature over time.

We would also like to implement annotations in other applications and environments. For example, virtual reality environments pose many new research problems. User studies would have to be performed to determine which annotation modalities would be appropriate in this space. If textual annotations were appropriate, we would have to determine where to place the text: floating in space near the marker, or on 2D panels which exist in the virtual space, or perhaps in some other place. New interaction mechanisms for annotation markers and filters should also be developed.

Finally, we would like to expand the scope of annotations. Springmeyer *et al.* noted that scientists tend to record their interactions with visualization systems. Perhaps the annotation system could help in recording and examining these edit trails. Also, scientists spend time comparing different data sets. The current annotation system could be redesigned to fit in the context of more than one data set.

We hope that further experience with the current system and its extension to other applications and environments will allow us to evaluate our design guidelines, and develop principles for customization of a general purpose annotation system.

6 Conclusion

The importance of annotation in data analysis and the lack of annotation support in data analysis tools led us to develop a system that integrates annotation and visualization. In our system, annotations are embedded in the 3D space of CFD data. Co-location of annotations and data allows users to navigate through the information by spatial association. Each annotation is composed of a small geometric marker and a content that can include textual, graphical and other domain-specific information. This allows unobtrusive annotations with an unlimited amount of information. Filters are provided to help sort annotations and create customized views of the information.

Initial feedback from scientists leads us to believe that the close integration of annotation and visualization facilitates the ongoing recording activity observed by Springmeyer *et al.* At the same time, the ability to group annotations in disjoint sets and filter annotations supports the organization of analysis conclusions, i.e., the describing activity. Furthermore, annotations can be used as a means of communication between collaborating scientists, and as a way to present information in an educational tool.

7 Acknowledgments

The authors of this paper would like to thank the members of the Graphics Group at Brown for their helpful comments and support. Thanks also to the members of the Visualization group at Digital Equipment Corporation's Cambridge Research Lab for their review of this paper. The paper is based on the Master's thesis of the first author, whose attendance at Brown University was made possible by Digital Equipment Corporation's Graduate Engineering Education Program. This work was supported in part by grants from Digital Equipment Corporation, NSF, DARPA, IBM, NCR, Sun Microsystems, and HP.

References

[1] Bancroft, Gordon V., Merritt, Fergus J., Plessel, Todd C., Kelaita, Paul G., McCabe, R. Kevin, and Globus, Al. FAST: A Multi-Processed Environment for Visualization of Computational Fluid Dynamics. *Proceedings of the First IEEE Conference on Visualization*, pages 14–27, 1990.

[2] Barber, C. Bradford, Dobkin, David P., and Huhdanpaa, Hannu. The Quickhull Algorithm for Convex Hull. Technical Report GCG53, Geometry Center, U. Minnesota, July 1993.

[3] Bertin, Jacques. *Graphics and Graphic Information Processing*. Walter de Gruyter and Co., 1981.

[4] Bier, Eric A., Stone, Maureen C., Pier, Ken, Buxton, William, and DeRose, Tony D. Toolglass and Magic Lenses: The See-Through Interface. *Proceedings of SIGGRAPH '93*, pages 73–80, 1993.

[5] Chalfonte, Barbara L., Fish, Robert S., and Kraut, Robert E. Expressive Richness: A Comparison of Speech and Text as Media for Revision. *Proceedings of the ACM Computer Human Interaction Conference*, pages 21–26, 1991.

[6] Conner, D. Brookshire, Snibbe, Scott S., Herndon, Kenneth P., Robbins, Daniel C., Zeleznik, Robert C., and van Dam, Andries. Three-Dimensional Widgets. *Proceedings of the Symposium on Interactive 3D Graphics*, pages 183–188, 1992.

[7] Foley, James, van Dam, Andries, Feiner, Steven, and Hughes, John. *Computer Graphics Principles and Practice*. Addison W esley, 2nd edition, 1992.

[8] Francik, Ellen, Rudman, Susan E., Cooper, Donna, and Levine, Stephen. Putting Innovation to Work: Adoption Strategies for Multimedia Communication Systems. *Communications of the ACM*, 34(12):53–63, Dec. 1991.

[9] Haimes, Robert and Darmofal, Dave. Visualization in Computational Fluid Dynamics: A Case Study. *Proceedings of the Second IEEE Conference on Visualization*, pages 392–397, 1991.

[10] Herndon, Kenneth P. Interactive Shadows. *UIST Proceedings*, pages 1–6, November 1992.

[11] Meyer, Tom and Huang, Nate. Programming in FLESH. Technical report, Department of Computer Science, Brown University, 1993.

[12] Phillips, Richard L. An Interpersonal Multimedia Visualization System. *IEEE Computer Graphics and Applications*, pages 20–27, May 1991.

[13] Phillips, Richard L. MediaView, A General Multimedia Digital Publication System. *Communications of the ACM*, 34(7):74–83, July 1991.

[14] Snibbe, Scott S., Herndon, Kenneth P., Robbins, Daniel C., Conner, D. Brookshire, and van Dam, Andries. Using Deformations to Explore 3D Widget Design. *Proceedings of SIGGRAPH '92*, pages 351–352, 1992.

[15] Springmeyer, Rebecca R., Blattner, Meera. M., and Max, Nelson. L. A Characterization of the Scientific Data Analysis Process. *Proceedings of the Second IEEE Conference on V isualization*, pages 351–352, 1992.

[16] Tanimoto, Steven L. *The Elements of Artificial Intelligence*. Computer Science Press, 1990.

[17] Upson, C. and *et al.* The Application Visualization System: A Computational Environment for Scientific Visualization. *IEEE Computer Graphics and Applications*, 9(4):60–69, July 1989.

[18] Verlinden, Jouke C., Bolter, Jay David, and van der Mast, Charles. Voice Annotation: Adding Verbal Information to Virtual Environments. *Proceedings of the European Simulation Symposium*, pages 60–69, 1993.

Scheduling Time-Critical Graphics on Multiple Processors *

Tom Meyer and John F. Hughes
NSF/ARPA Science and Technology Center for
Computer Graphics and Scientific Visualization,
Brown Site
{twm,jfh}@cs.brown.edu

Abstract

This paper describes an algorithm for the scheduling of time-critical rendering and computation tasks on single- and multiple-processor architectures, with minimal pipelining. It was developed to manage scientific visualization scenes consisting of hundreds of objects, each of which can be computed and displayed at thousands of possible resolution levels. The algorithm generates the time-critical schedule using progressive-refinement techniques; it always returns a feasible schedule and, when allowed to run to completion, produces a near-optimal schedule which takes advantage of almost the entire multiple-processor system.

CR Categories:

Additional Keywords:

1 Introduction

Scientists who create complex datasets (e.g., large time-varying fluid-dynamics simulations, high-resolution MRI scans, and structural simulations) require correspondingly sophisticated ways of examining and visualizing this data. In such cases, a scientist may want to interactively manipulate and examine very complex visualizations, such as a rake with dozens or hundreds of streamlines. Maintaining the fast interaction rates required can be quite difficult, especially in immersive environments such as the Virtual Wind Tunnel at NASA Ames [BL91], where update rates of at least ten frames/second are required.

In order to support the task of exploratory visualization in these complex datasets, we have developed a time-aware scheduling algorithm to provide importance-based real-time computation and rendering of some common scientific-visualization techniques. This algorithm takes advantage of a dedicated graphics workstation with a single-threaded graphics pipeline and from one to several dozen processors, communicating using a shared-memory model. (Although some research systems, such as UNC's PixelFlow, will use multi-threaded graphics pipelines, no such system is commercially available yet.)

The techniques described in this paper extend to general graphics scheduling. Objects with nearly-continuous representations—streamlines can be computed at arbitrary timesteps, and for any number of steps, for example—are particularly well-suited to be scheduled using this algorithm.

This scheduling algorithm has the following advantages

- After an initial startup phase, it can terminate at any time during its incremental refinement phase, and will always return a feasible schedule.
- It results in near-maximal usage of single- and multiple-processor machines if allowed to run to the completion of the refinement phase.
- It pipelines all computations to be rendered as soon as possible, reducing lag times.
- It balances the benefit of spending time computing new data against the time required to redisplay existing data at its already-computed resolution.

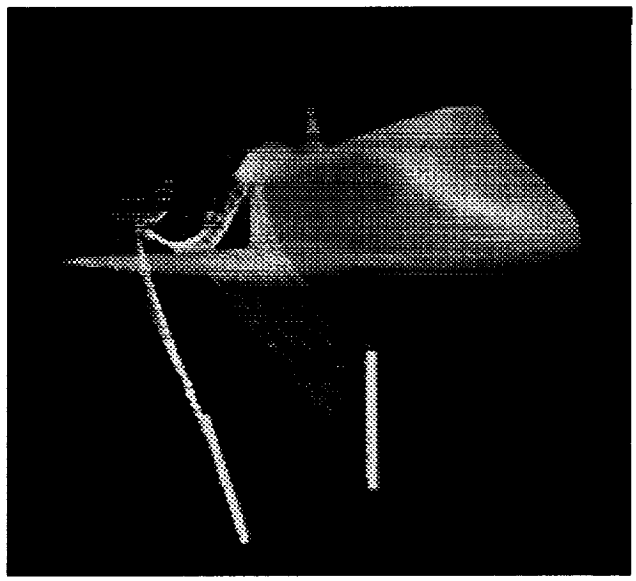


Figure 1: The target application, scientific visualization of complex computational fluid dynamics scenes in an immersive environment.

2 Previous Work

An large body of research on real-time scheduling exists, dating from the 1950's. A good introduction to the relevant issues is [SSNB94].

Classical real-time theories mainly deal with static scheduling problems in which the algorithm has complete knowledge of the demands placed on it, and where there are generally hard constraints

*This work was supported in part by grants from NASA grant NAG 2-830, NSF/ARPA, Sun Microsystems, Autodesk, Microsoft, and TACO.

which, if violated, could result in catastrophic failure (airplanes crashing, factories blowing up, etc). These types of problems are fairly well-understood, and many algorithms exist for solving them.

Although dynamic multiple-processor scheduling is becoming an active area, little work has been done on it. Almost all the interesting problems are NP-complete, and good approximation algorithms are only beginning to emerge.

Unfortunately, problems of particular interest to scientific visualization have not been much studied. The narrowness of the problem – scheduling multiple independent tasks-pairs (computation and rendering) with the two requirements that the "rendering" portions all take place on one machine, and that each rendering portion start only after the completion of its computation portion – makes it too specialized to warrant much attention, except from those who need solutions.

Because virtual environments require near-constant, high frame rates, several systems which address time-critical issues have been developed:

Richard Holloway's Viper system [Hol92] uses objects with predefined levels of resolution, and renders objects at a global level of resolution sufficient to display all of them in allotted time. It does not provide for individual levels of importance for the objects.

Funkhouser and Sequin describe a real-time scheduling algorithm for complex virtual walkthroughs in [FS93]. However, their algorithm only provides support for objects with a few pre-computed levels of representation and their faster algorithm only works well on objects with a convex-downward benefit function. Nonetheless, the ideas in that paper provide the starting point for our algorithm.

The multi-processing scheduling algorithm described by Rohlf and Helman in [RH94] schedules computation, culling and rendering of geometric data by using pipelining, which results in the addition of one frame's worth of lag to the system (lag is as bad as low frame rates in virtual environments). Additionally, they use a feedback-based model for managing scene complexity, so cannot bound frame times when the scene changes rapidly.

Little work has been done on combining computation and rendering, and on managing tradeoffs among computing expensive but useful information.

3 Benefit Function

For a set of rendering tasks, we need to determine the most useful amount of time to spend computing and rendering each one. We define a function $Benefit(t)$, which reflects the approximate value of spending an amount of time t computing and rendering a graphics task. We want to maximize

$$B = \sum_i Benefit_i(time_i)$$

subject to

$$\sum_i time_i \leq frametime$$

This function consists of a product of several other benefit values, computed on a per-item basis. For any item i, we decompose the benefit of alloting time t to rendering that item into a product of three parts:

$$Benefit_i(t) = Importance_i \cdot Processor_i(t) \cdot Hysteresis_i(t)$$

Importance is an importance value for the item, expressing the object's inherent value, closeness to the viewer, user interaction, and the visual focus of the viewer. Any number of perceptually-based metrics could be weighted into this; in the current implementation we assign high importance to streamlines with which the user is interacting. Defining useful metrics for determining an object's importance, both in perceptual and semantic terms, is beyond the scope of this paper.

Processor expresses the amount of value to be gained by spending that amount of startup, computation, and rendering time. We assume that this is a nondecreasing function, convex to the right of the "startup time," reflecting the idea that for most visualizations, "something is better than nothing, but fine detail is worth only a little more than coarse detail." Our implementation uses $\sqrt{t - t_s}$ for $t > t_s$, where t_s is the startup time, and 0 for $t \leq t_s$.

Hysteresis term is a sigmoid function designed to encourage inter-frame continuity. It varies smoothly from a value of 1 at some point $t < t_{\text{prev}}$ up to a value of $1 + \delta$ at t_{prev}, where t_{prev} is the time allocated to the task in the previous frame.

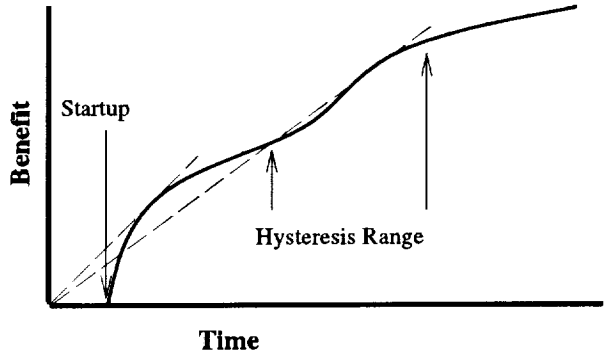


Figure 2: The benefit function has a fixed startup cost, rises quickly after that, and gradually falls off until the hysteresis point. The two dashed lines indicated the local maxima of the function $Benefit(t)/t$.

Since we want to maximize the sum of the benefits subject to the constraint that the sum of the times scheduled for all tasks is less than the frame time, we first examine the benefit per time unit for the tasks. If this benefit per time $\frac{B_i(t)}{t}$, which Funkhauser and Sequin called the *value*, is increasing at some value t, we would ideally like to allocate more time to task i: doing so would reduce the average cost of the benefits derived from executing task i. But if the benefit per time is decreasing, then allocating more time to task i will increase the average cost of the benefit, and should be done only if other tasks cannot benefit more from being given the additional time instead. We therefore consider the points where the value $\frac{B_i(t)}{t}$ is at a maximum as good starting points in the search for ideal time allocations. A complete rationale for this starting choice is given in [FS93]. As seen in Figure 2, the benefit/time function will have at most two local maxima – near the extreme points of $Processor_i(t)/t$ and $(Hysteresis_i(t) - 1)/t$. If these functions are expressed analytically, it is simple to compute these two points analytically. Note that as long as both functions are differentiable and have few local maxima, we can perform a similar analysis to obtain the starting points.

The functions that constitute $Benefit(t)$ are all defined only at a fixed, sparse set of values for t, since computation time cannot be allocated in quanta smaller than the clock cycle. Furthermore, the functions are likely to be step functions, constant on large intervals in the domain. To the extent that this is true, time spent in making small adjustments to the allocation of processor time is often wasted. On the other hand, by bounding the smallest step size we will take in adjusting processor allocations be of the same scale as the smallest interval on which the true benefit functions are constant, we can substantially avoid such waste, while still derviving the benefit of being able to use differentiable functions.

4 Scheduling Algorithm

The application into which the scheduler fits works as follows: In a typical frame-time, user input is gathered, the schedule computed during the last frame is executed, and the schedule of tasks to be performed during the next frame is computed, with the components of each benefit function modified according to the previous schedule (which influences the hysteresis factor) and the user interaction, which influences importance. This sections describes how the scheduler works.

4.1 Single-Processor Case

We use a two-phase incremental-refinement scheduling algorithm, based partly on Funkhouser and Sequin's algorithm [FS93].

Greedy Phase. The first phase is essential; it generates a feasible but not necessarily good schedule, and requires $O(n \log n)$ time. This makes it possible to bound the worst-case time of the scheduler and consider that amount of time as part of the frame time. (The scheduler can place itself into the generated schedule for the next frame). Of course, it is possible to have an extremely complex scene for which it would be impossible to execute even this phase during the frame time. In this case the schedule could be recomputed only once every few frames, at some loss of responsiveness.

The greedy initial phase generates, for each task i that has been selected for inclusion in the schedule, a pair $(t_{i,j}, Benefit(t_{i,j}))$ at each of the local maxima of the $Benefit(t)/t$ function. We sort these pairs by value $Benefit(t)/t$, and repeatedly take from the list the task whose value is greatest. If the task has not yet been scheduled and there is still available space, we add it to the work list; if it has been scheduled and the new value of t is greater than the previously scheduled one, we reschedule it at the new time (if there is space). This produces an initial packing which is at least half as good as the result from doing the NP-complete optimal packing[FS93].

Incremental Phase. During the second phase of the algorithm the scheduler iteratively refines its generated schedule, as time allows. However it can terminate at any time, since the feasibility of the schedule is never violated. This phase also has $O(n \log n)$ complexity, and may terminate before the time alloted.

The problem reduces to an N-dimensional gradient descent, where we try to find the maximum value of the derivative such that all of the benefit functions have the same slope (some may have a zero slope).

To solve this, we use a version of the Newton-Raphson technique to initialize each of the tasks with a stepvalue δ_i:

$$\delta_i = granularity \cdot \min(1, \kappa - \frac{Benefit_i'(t_i)}{Benefit_i''(t_i)}))$$

where $granularity$ is the starting value for the refinement, and κ is the current mean of the derivative values. If δ_i is smaller than the current smallest task size, we initialize it to the smallest task size. This technique moves slowly through areas where the benefit function has high curvature and quickly otherwise, by taking a step that's inversely proportional to the curvature, but proportional to the difference between the current derivative value and the desired derivative value.

While we can feasibly add a task i, we do so, generating a benefit of $Benefit_i'(t_i)$. We add δ_i to t_i, and then repeat. If we cannot feasibly add any task, we find the currently-scheduled task i with the smallest $Benefit_i'(t_i)$, we subtract δ_i from the time allotted to it. We then repeat the entire process. Every time a task has time added to it and then subtracted away, we halve the value of δ_i, until δ_i is smaller than the smallest task size, at which point the task i is removed from the list of candidates for improving the schedule. The algorithm terminates when the list is empty, or the available time is used up.

To determine if we should spend additional time refining the schedule before executing it, we compare the margin of change in the total benefit per iteration with the amount of time required to perform that iteration. If the scheduler should run for less time, we decrease its alloted time slightly, bounded by the worst-case time. Otherwise, we can increase its alloted time.

4.2 Multiple-Processor Case

Dedicated graphics multiple-processor workstations are becoming common, especially for high-end scientific-visualization applications. These machines allow light-weight processes which communicate using low-overhead shared memory and synchronization primitives; however, the rendering pipeline is fed from only one processor at a time.

Most multiple-processor scheduling algorithms are NP-complete (even the fairly simple case of 2 processors, no precedence constraints, and arbitrary computation times is NP-complete) [GJ75]. Since we want to schedule a set of tasks on several processors with precedence constraints, our problem is at least this hard.

We modify the single-processor greedy algorithm to quickly generate a feasible schedule for multiple processors in $O(n^2)$ time for a guaranteed schedule or $O(n \log n)$ time for an optimistic, probably-feasible one.

First, let us consider how we might build a good multiple-processor schedule. Generally we have two portions of the visualization task: a compute task taking time c and a render task taking time r (possibly with a cull task inserted between them). The compute task can run on any processor, but all render tasks must stay together. Also, any data must be computed before it can be rendered, so any good schedule would have to make sure that rendering tasks would not sit idle while waiting for data. Let us consider two tasks which are being computed on a single processor and rendered on another. If we order them so that the tasks with large values of $c - r$ (which we call the *excess compute time*) are last, we have the most room possible for additions, and minimize the startup differences and ending differences between the processors, as shown in Figure 3.

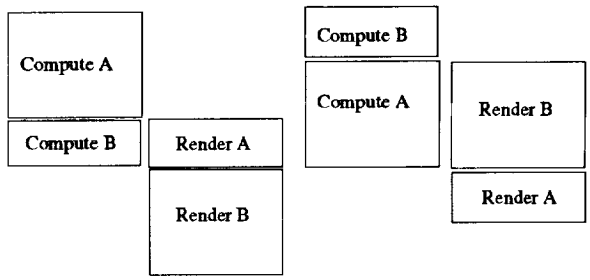


Figure 3: Ordering tasks by increasing excess compute time minimizes the makespan.

The multiple-processor algorithm works like the single-processor algorithm, except that we modify the insertion routine to verify that adding work to the schedule doesn't produce an infeasible schedule. We initially try to add each task to the rendering processor; if there is not enough room on a processor, we push tasks onto the next processor, starting with the task with the most excess compute on the current processor. In this way, we always minimize the total amount of computation time required before rendering can begin. Pushing a single task may cause a cascade of pushes, as shown in Figure 4, but we do not attempt to push a task again if a previous

push on that task has failed (there is probably still not enough room for it, since pushes only go in one direction), so we perform at most $O(p \cdot n)$ pushes (p is the number of processes), with each attempted push requiring an additional verification pass.

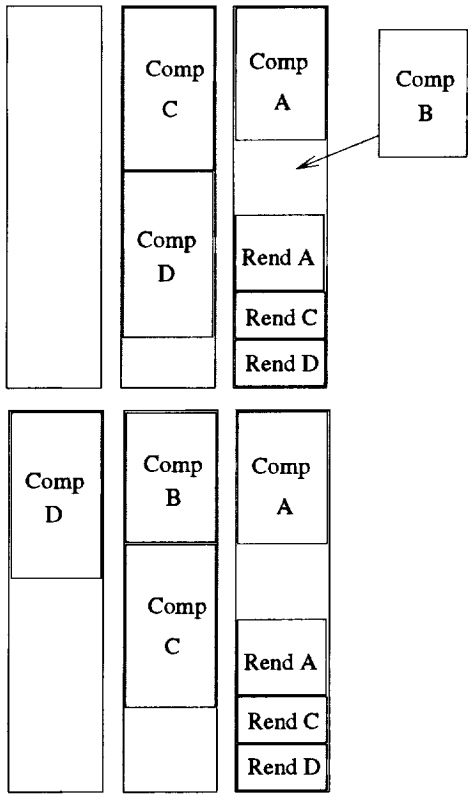


Figure 4: Pushing a task may cause a cascade of pushes. Here, inserting task B causes task D to move to another processor.

In order to guarantee feasibility, we need to look at the two possible ways in which an insertion could violate it:

- We must make sure that the sum of the work is less than the frametime, for each processor. Verifying schedule-size feasibility takes $O(1)$ time if done as each task is added to the list.
- Additionally, indivisible tasks cannot be rendered in time less than the sum of the startup, compute, and rendering times for that task. Consider a fine-grained compute task that generates small pieces of geometry (meshes, lines, or even individual polygons and line segments) at regular intervals c during computation, after the startup time s. Rendering of any of that task's data cannot begin until time $s + c$. If the time required to render a piece is r, the total time required to render x primitives is $s + c + (x - 1) \cdot \max(c, r) + r$. Any generated schedule which violates this requirement is infeasible.

  Verifying precedence relations takes $O(n)$ time, since it is necessary to check every rendering task which is scheduled before an inserted rendering task, as well as every computational task which is scheduled after an inserted computational task.

As described previously, when scheduling several tasks, we can lower the possibility of feasibility conflicts by ordering them from low to high excess computation. Where this value is equal, we define a consistent ordering of tasks so that the partial order of tasks is identical across both the compute and rendering phases. In actual practice this heuristic, when applied to scenes containing diverse types of objects, results in schedules which rarely violate the precedence relations and which achieve high processor usage. If one can tolerate the occasional slow frame, removing the precedence checking results in an $O(n \log n)$ algorithm.

Of course, in pathological cases, any render-dominated schedule may have only the same benefit as the single-processor schedule, but in most complex and diverse scenes, the scheduling algorithm is able to take advantage of the different computational and rendering demands of these tasks to generate a feasible, good schedule which uses almost the entire multi-processor system.

5 Implementation Results

Our actual implementation results are fairly preliminary at this stage, but we have already found some interesting results.

We used a two-processor Onyx to run our real-time usability tests. However, we were unable to isolate the processors from the vagaries of UNIX scheduling (we could only have isolated one processor from the operating system, since system tasks have to run on at least on processor), so obtained frame rates that varied between 8 and 12 frames per second when we attempted to schedule at 10 frames per second.

If we simulate a four-processor Onyx, allocating 100 milliseconds (one complete processor at 10 frames/second) for the scheduling phase, we fill 97% of the other three processors, when the task is compute-bound. With 50 milliseconds allocated for the scheduling task, the algorithm still manages to fill 92% of the available processor time. The benefit of the algorithm declines sharply after this point, however. Giving the scheduler 25 milliseconds resulted in only 65% processor usage, while 15 milliseconds dropped to 32% usage, resulting in less computational time than a single processor.

We intend to perform additional investigations to understand how the scheduling algorithm's behavior degrades under stress, and modify the search techniques which it uses to cope better with situations involving too much work or too little time to schedule.

6 Future Work

We have not yet completed the interleaved version of this algorithm, in which it schedules iself as a time-critical task onto the work queue, scheduling the next frame while the current frame is being completed. Several interesting problems occur here, involving feedback problems and how to deal with unexpected user input (the scheduler can't tell when a user will manipulate a large, complex visualization tool, so will necessarily lag slightly behind in that case).

It is also be extremely important to understand more about human perception in complex environments, to generate more realistic benefit functions. Additionally, we would like to work with scientists to determine the actual semantic benefit of each tool when performing varying types of tasks.

References

[BL91] Steve Bryson and Creon Levitt. The Virtual Windtunnel: An Environment for the Exploration of Three-Dimensional Unsteady Flows. In *Visualization '91*, pages 17–24, 1991.

[FS93] Thomas A. Funkhouser and Carlo H. Séquin. Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments.

In James T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 247–254, August 1993.

[GJ75] R. Garey and D. Johnson. Complexity Results for Multiprocessor Scheduling Under Resource Constraints. *SIAM Journal of Computing*, 1975.

[Hol92] Richard L. Holloway. Viper: A Quasi-Real-Time Virtual-Environment Application. Technical Report TR92-004, University of North Carolina, Chapel Hill, 1992.

[RH94] John Rohlf and James Helman. IRIS Performer: A High Performance Multiprocessing Toolkit for Real–Time 3D Graphics. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 381–395. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[SSNB94] John A. Stankovic, Marco Spuri, Marco Di Natale, and Giorgio Buttazzo. Implications of Classical Scheduling Results for Real-Time Systems. *to appear in IEEE Computing*, 1994. `ftp.cs.umass.edu, /pub/ccs/spring/impl_sch_rts.ps`.