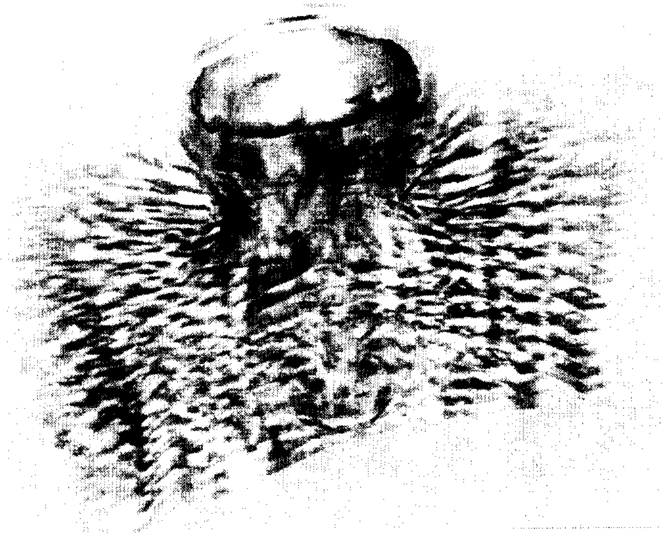
3

p- 125

NASA Conference Publication 3321

ICASE/LaRC Symposium on Visualizing Time-Varying Data

Edited by
D. C Banks, T. W. Crockett, and K. Stacy



Proceedings of a symposium sponsored by the National Aeronautics and Space Administration, Washington, D.C., and the Institute for Computer Applications in Science and Engineering (ICASE), Hampton, Virginia, and held in Williamsburg, Virginia September 18–19, 1995

January 1996

NASA Conference Publication 3321

ICASE/LaRC Symposium on Visualizing Time-Varying Data

Edited by
D. C. Banks
Mississippi State University • Mississippi State, Mississippi

T. W. Crockett
Institute for Computer Applications in Science and Engineering (ICASE) • Hampton, Virginia

K. Stacy
Langley Research Center • Hampton, Virginia

Proceedings of a symposium sponsored by the
National Aeronautics and Space Administration,
Washington, D.C., and the Institute for Computer
Applications in Science and Engineering (ICASE),
Hampton, Virginia, and held in Williamsburg, Virginia
September 18–19, 1995

National Aeronautics and Space Administration
Langley Research Center • Hampton, Virginia 23681-0001

January 1996

Cover Photo

Visualization of the flow field near a high density contour in an interstellar cloud collision simulation.

(From *Flow Visualization Using Moving Textures*, N. Max and B. Becker, LLNL)

This publication is available from the following sources:

NASA Center for AeroSpace Information
800 Elkridge Landing Road
Linthicum Heights, MD 21090-2934
(301) 621-0390

National Technical Information Service (NTIS)
5285 Port Royal Road
Springfield, VA 22161-2171
(703) 487-4650

Preface

Time-varying datasets present difficult problems for both analysis and visualization. For example, the data may be terabytes in size, distributed across mass storage systems at several sites, with time scales ranging from femtoseconds to eons. In response to these challenges, ICASE and NASA Langley Research Center, in cooperation with ACM SIGGRAPH, organized the first Symposium on Visualizing Time-Varying Data. The purpose was to bring the producers of time-varying data together with visualization specialists to assess open issues in the field, present new solutions, and encourage collaborative problem-solving.

These proceedings contain the peer-reviewed papers which were presented at the Symposium. They cover a broad range of topics, from methods for modeling and compressing data to systems for visualizing CFD simulations and World Wide Web traffic. Because the subject matter is inherently dynamic, a paper proceedings cannot adequately convey all aspects of the work. The accompanying video proceedings provide additional context for several of the papers.

In addition to the contributed papers, the Symposium featured a dozen informal "home videos" of work in progress, as well as several live demonstrations of visualization systems. We also solicited contributions of time-varying datasets which pose challenging visualization problems. Abstracts of the informal presentations are included here. The complete program, the contributed datasets, and other materials are available on the World Wide Web at "*http://www.icase.edu/workshops/vtvd/*".

We wish to thank Jay Lambiotte, Bill von Ofenheim, and the Scientific Applications Branch at NASA LaRC for their support of this Symposium. Brian Hahn of the Newport News office of Silicon Graphics, Inc. furnished computer equipment to support the live demonstrations. Finally, Emily Todd of ICASE provided invaluable assistance in managing the logistics of the Symposium and assembling the proceedings.

David Banks, *Mississippi State University*
Tom Crockett, *ICASE*
Kathy Stacy, *NASA Langley Research Center*

Symposium Organizers

Symposium Co-Chairs:

David C. Banks, *Mississippi State University*
Kathryn Stacy, *NASA Langley Research Center*

Organizing Committee:

Mary Adams (Demonstrations), *NASA Langley Research Center*
Leon Clancy (Computer Systems), *ICASE*
Thomas Crockett (Datasets), *ICASE*
Kwan-Liu Ma (Publicity), *ICASE*
Piyush Mehrotra, *ICASE*
Kurt Severance (Audio/Visual), *NASA Langley Research Center*

Program Committee:

Lambertus Hesselink (Program Chair), *Stanford University*
Roger Crawfis, *Lawrence Livermore National Laboratory*
Robert Haimes, *Massachusetts Institute of Technology*
Chuck Hansen, *Los Alamos National Laboratory*
David Lane, *Computer Sciences Corporation, NASA Ames Research Center*
Nelson Max, *Lawrence Livermore National Laboratory*
Duane Melson, *NASA Langley Research Center*
Lloyd Treinish, *IBM T. J. Watson Research Center*
Velvin Watson, *NASA Ames Research Center*

Reviewers:

David Banks, *Mississippi State University*
Roger Crawfis, *Lawrence Livermore National Laboratory*
Robert Haimes, *Massachusetts Institute of Technology*
Chuck Hansen, *Los Alamos National Laboratories*
David Lane, *NASA Ames Research Center*
Nelson Max, *Lawrence Livermore National Laboratory*
Duane Melson, *NASA Langley Research Center*
Samuel Uselton, *NASA Ames Research Center*
Velvin Watson, *NASA Ames Research Center*

Contents

Papers

Visualizing and Modeling Categorical Time Series Data

Randy Ribler   Anup Mathur
Marc Abrams
Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061-0106
{ribler,mathur,abrams}@cs.vt.edu

August 25, 1995

Abstract

Categorical time series data can not be effectively visualized and modeled using methods developed for ordinal data. The arbitrary mapping of categorical data to ordinal values can have a number of undesirable consequences. New techniques for visualizing and modeling categorical time series data are described, and examples are presented using computer and communications network traces.

1 Introduction

Visualization tools allow scientists to comprehend very large data sets, and to discover relationships which otherwise are difficult to detect. Unfortunately, not all types of data can easily be visualized using existing tools. In particular, most of the large arsenal of tools that are available for the visualization and modeling of numeric time series data are inappropriate for categorical time series data. This paper identifies the particular problems that are encountered when visualizing and modeling categorical time series, and proposes new methods which are useful in many domains. Example problems from computer and communications network performance analysis are examined.

Categorical data is identified by category rather than by ordinal value. The data set can be partitioned into a number of categories. Each element in the data set belongs to exactly one category. For example, gender data is categorical data. Each item in a data set of people can be placed in one of the categories from the set {Male, Female}.

Categorical data arises naturally in many types of analysis. The analysis of computer trace files was the original motivation for this work. These files contain time series information describing the *state* of an executing computer program. Some examples of categorical data contained in these traces are:

- the name of the currently executing procedure in a program
- the name of the disk drive in use
- the names of programs loaded in memory
- the name of a remote computer sending information

Trace files provide low-level descriptions of the time dependent state transitions which occur during program execution. But trace files are often very large, and important information may be lost when they are summarized or modeled with statistics such as averages and variances.

A trace file is an n length sequence of ordered pairs $\{T_1, S_1\}, \{T_2, S_2\}, \ldots, \{T_n, S_n\}$, where $T_i (1 \leq i \leq n)$, is a time and S_i is a system state vector which records pertinent system information. To simplify the

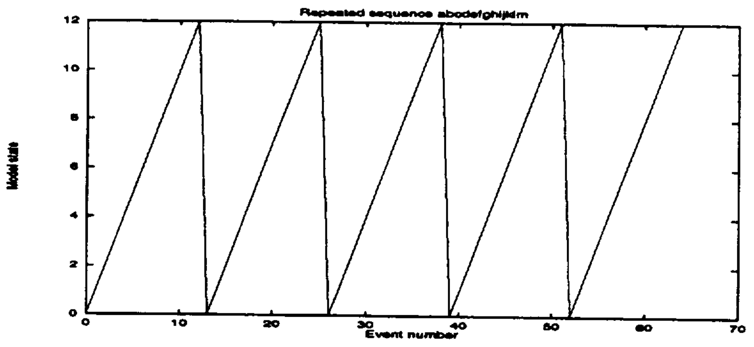


Figure 1: Sequence "abcdefghijklm" with one mapping of letters to numbers

analysis, and to create visualizations that are more intuitive, the system state vectors is usually broken down into its component members, yielding a separate scalar time series for each member of the state vector. In situations where it is important to consider the state vector as a unit, each unique vector is identified by a unique *model state*, which is a single categorical value [ADM92]. In either case, the reduction allows us to analyze a time series of scalar values. We use the term *event* to refer to the individual entries in the time series. We use the term *state* to refer to the category to which an individual event belongs.

2 Problem Statement

2.1 Problem 1: Visualization

Visualization is generally accomplished through mapping data values to a coordinate system, and displaying a 2-dimensional projection of a 2, 3, or more dimensional object. Often the same coordinate system that is used to model a physical phenomenon is used to produce its visualization. Each data point has a spatial relationship with every other data point in the set. This relationship can be expressed as a vector.

Generally, spatial relationships between *categories* of data do not exist or are incomplete. We can not compute a distance and a direction between *names* of things. This limitation renders the majority of conventional visualization techniques inappropriate for use with categorical data.

Imposing spatial relationships through the arbitrary assignment of ordinal values to each category can produce results that are either misleading or not particularly revealing. Figure 1 shows one mapping of a repeated sequence of the states "abcdefghijklm." This mapping is obtained by assigning sequential numbers to each of the letters in the sequence. The repeated ramps seem to match our intuition about how a repeated sequence should look when the letters are mapped to numbers. Figure 2 shows a mapping of different numbers to the same sequence of letters. Although it is clear from both graphs that a pattern exists, the graph in Figure 2 contains additional oscillations which are misleading. When there are a large number of unique states in the series, it becomes impossible to distinguish this phenomenon from real oscillations in the data.

The fact that two different mapping can produce such different visualizations is not surprising when we consider that n categories can be mapped to n ordinal values in $n!$ different ways. It seems unlikely that all $n!$ different views will convey the same information equally well.

It is tempting to try to use the view itself as a basis for coordinate assignment. A mapping that minimizes the distance between the state transitions, where distance is defined as the difference between the ordinal values assigned to the successive states, might at first seem desirable. But because the same

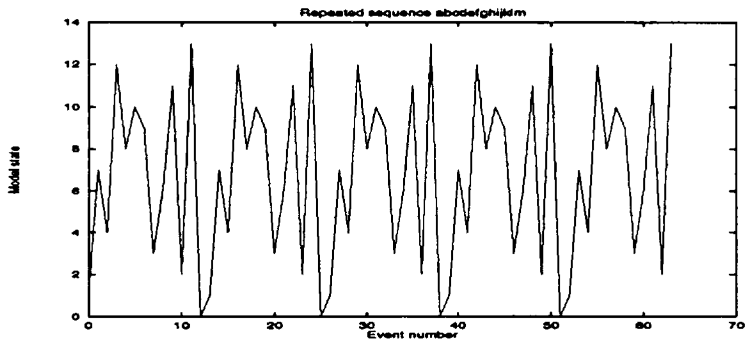


Figure 2: The same sequence as Figure 1, with another mapping

mapping must be used throughout the trace, a mapping that minimizes distances globally may still display very poor performance locally. In addition, it can be shown that determining this mapping, known as Optimal Linear Sequencing, is NP-hard [GJ79].

2.2 Problem 2: Modeling

We can not directly model categorical time series for the same reasons that we can not directly visualize them. That is, conventional models assume that ordinal relationships can be exploited. Another significant problem is that many models of time series data rely on time invariant probability distributions, and yet, it is often important to model and understand the transient behavior present in traces. When we model a trace, we view it as a realization of a stochastic process. In our models, the probability mass functions used to determine the stochastic process are time-dependent.

3 Traces Used for Illustration

Table 1 summarizes the traces used to demonstrate our visualizations and models.

Table 1: Example Traces

Traces		
Trace Name	Number of States	Unique States
Gaussian Elimination	409	31
World Wide Web (WWW) Traffic	229,256	2973
Dining Philosophers	5793	588
Cache Hit Trace of WWW Traffic	37	2

3.1 Gaussian Elimination

The Gaussian Elimination trace was obtained from an instrumented implementation which solves a 64×64 system of equations using 32 processors on an NCUBE multicomputer [RW93]. The system state recorded in the trace is the number of the processor that last sent a message. Process numbers are categorical because their exists no total ordering on their values – one processor is not "less than" another.

3.2 World Wide Web Traffic

The World Wide Web (WWW) traffic trace records the names of files which were accessed on a server used to store educational materials for computer science classes at Virginia Tech. This large trace includes the names of all the files accessed over a period of 75 days [AWA+95].

3.3 Five Dining Philosophers

The five dining philosophers trace was created by a parallel program running on a Sequent Symmetry at Argonne National Labs. The program simulates a resource contention problem from operating system design: five philosophers sit in a circle, eat spaghetti, and contend for a limited number of forks. At any point, each philosopher is in one of eight possible states (e.g., eating, or waiting for a fork). Upon a state change, the trace records a categorical data vector of length five, where each element in the vector records the state of one of the five philosophers. Each unique vector is mapped to a unique *model state*, which is a single categorical value [ADM92].

3.4 Cache Hits

Cache (or proxy) servers are used within organizations connected to the World Wide Web (WWW) to keep local copies of frequently used files from remote WWW sites. The cache hit trace for WWW traffic records a "hit" when a requested file is available locally, and records a "miss" when a file must be accessed remotely from the WWW. We use a small section of this trace, displayed in Table 2, to exemplify visualizations and modeling concepts.

Table 2: WWW Cache Hit trace

Event Number	Time	State
0	451420	miss
1	452660	miss
2	453270	hit
3	453300	miss
4	453460	hit
5	453690	miss
6	454090	miss
7	454140	miss
8	454170	miss
9	454290	miss
10	454410	miss
11	454510	miss
12	454640	miss
13	454730	miss
14	454750	miss
15	454890	miss
16	455460	miss
17	455480	miss
18	455510	miss
19	455860	miss
20	456250	miss
21	456340	miss
22	456380	hit
23	456850	miss
24	456869	miss
25	456870	miss
26	456871	miss
27	456872	miss
28	456880	miss
29	456900	miss
30	457190	miss
31	458050	miss
32	458490	miss
33	459210	miss
34	459300	miss
35	459320	miss
36	459440	miss
37	459450	miss
38	459490	hit

4 Five New Visualizations of Categorical Time Series

Because it is not productive to map categorical data directly to ordinal values, we propose a strategy which identifies numerical relationships between occurrences of categorical values in a time-series, and uses these relationships as a means to visualization. This section describes a number of visualizations which exploit the repetitions of states in the series to produce appropriate visualizations. We first illustrate each visualization with the cache hit subtrace from Table 2 for pedagogical purposes, then with another trace from Table 1 to illustrate a practical application.

4.1 Cumulative Categorical Periodogram

The periodogram is a popular view for ordinal data. The discrete Fourier transform (DFT) transforms *numeric* time-series data into its component sine waves, which are used to produce the periodogram. But the concept of component sine waves is inapplicable to categorical time series data. We need a different definition of periodicity for categorical time series.

Intuitively, a categorical time series achieves a cycle when it returns to a previously encountered state. The *period* of this cycle can be defined as either the number of intervening events plus one, or as the time between the two occurrences of the state. We refer to the first definition as the *untimed* period, and to the second as the *timed* period.

4.1.1 Untimed Cumulative Categorical Periodogram

Recall that a trace is a sequence, $\{T_1, S_1\}, \{T_2, S_2\}, \ldots, \{T_n, S_n\}$. We compute an *Untimed Cumulative Categorical Periodogram (CCP)* by determining the number of times cycles with each of the possible untimed periods (1 to $n-1$) occur. The number of occurrences of cycles with period i in a sequence S of length n is given by

$$\mathrm{Occurrences_i} = \sum_{j=i}^{n} \begin{cases} 1 & \text{if } S_j = S_{j-i}, \\ 0 & \text{otherwise.} \end{cases}$$

Occurrences$_i$ can be computed for $1 \le i \le n$ in O(n) time and O(number of categories) space.

Figure 3 shows the CCP generated from the cache hit trace in Table 2. The x-axis represents periods and has domain $[0, n)$. The y-axis represents occurrences$_i$, for $0 \le \; i \le \; n$. The events numbered 3, 4, 5, and 23 contribute to the bar at $x = 2$ in the periodigram, giving it the value 4. Events 22 and 38 are responsible for the bars at 18 and 16 respectively. All other events, with the exceptions of events 0 and 2, contribute to the first bin. Events 0 and 2 are the first occurrences of states and therefore do not represent the completion of a cycle.

Figure 4 shows the CCP generated from the five dining philosophers trace. The x-axis is logarithmic to help distinguish the lower frequencies. The y-axis is logarithmic so that we can clearly see a wide range of amplitudes. This particular graph identifies strong periodic behavior at periods near 6, 30, and 60.

4.1.2 Geometric Distribution of CCP

In uniformly distributed random data, the probability that a given event completes a cycle of length l is pq^{l-1}, where p is the reciprocal of the number of uniformly distributed states, and q is $1 - p$. This means that the periods of the cycles present in random data will have a geometric distribution, where the expected value for the number of occurrences of cycles of length l in a sequence of length n is

$$\mathrm{E(Occurrences_l) = (pq^{l-1})(n-1)}.$$

where Occurrences$_l$ is the number of occurrences of states which recur with period l.

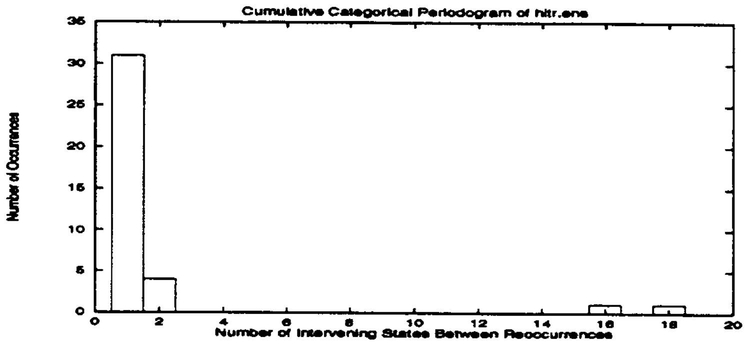


Figure 3: Untimed Cumulative Categorical Periodogram of Cache Hits

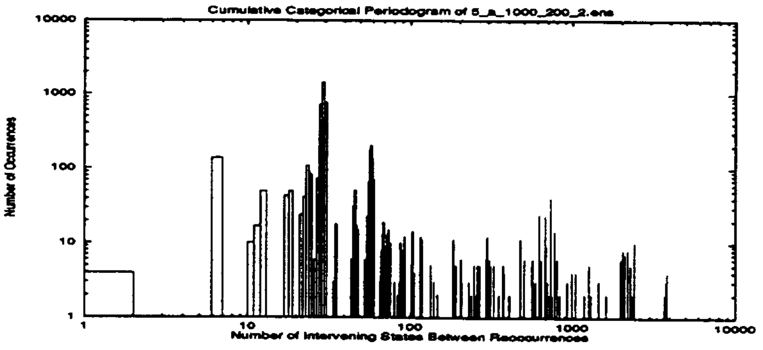


Figure 4: Untimed Cumulative Categorical Periodogram of Dining Philosophers

However, we expect to find strong periodic components in the computer and communications network traffic we analyze, and these periods will tend to be present for contiguous stretches. When periodic components of a trace have period l every lth event will also have period l. The probability of seeing k repetitions of a given period l in random data is:

$$(pq^{l-1})^k$$

This quickly becomes very small as k gets large, so when we see localized periodic behavior in a trace, we can be very confident that it is not from random chance. Still it may be desirable to normalize to correct for this bias toward geometric distributions. This would allow random data to produce a relatively flat curve. Normalization may be accomplished by subtracting the expected value from each bin and dividing by the standard error.

$$\text{NormOccurrences}_l = \frac{(\text{Occurrences}_l - \text{E}(\text{Occurrences}_l))}{\sqrt{\text{Var}(\text{Occurrences}_l)}}$$

4.1.3 Timed CCP

The Timed CCP differs from the untimed CCP in that it defines the period of a cycle as the *time* between the reoccurrences of a state. Figure 5 shows the timed CCP generated from the same trace of the five dining philosophers simulation. This view shows that most of the periodic behavior occurs in between 3500 and 5500 time units.

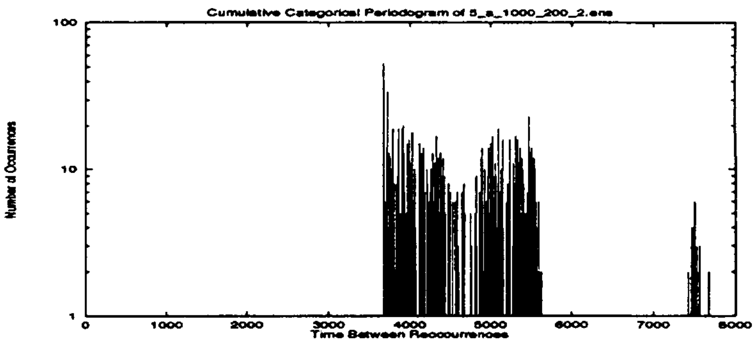


Figure 5: Timed Cumulative Categorical Periodogram of Dining Philosophers

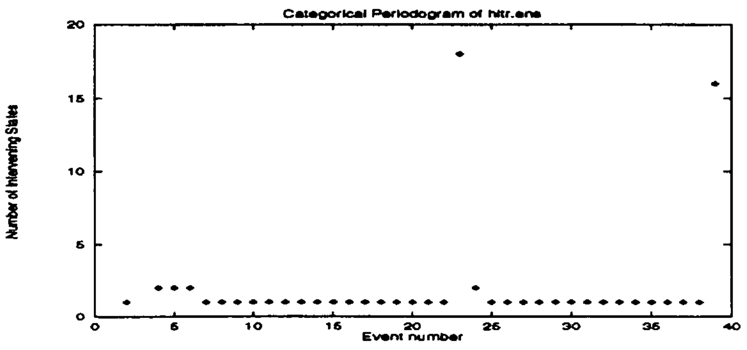


Figure 6: Categorical Periodogram of Cache Hits

4.2 Categorical Periodogram (CP)

The CCP provides a visualization of the total number of occurrences of each period. However, it is often useful to see exactly where the cycles occur in the time series. Figure 6 shows the *categorical periodogram* of our cache hit example. In this view we see not the events themselves, but the period since the previous occurrence of each state. A dot is plotted on the graph for every event that completes a cycle. The x-coordinate of the entry is the event sequence number, and the y-coordinate is the period of the cycle which has been completed. For example, the point $(1, 1)$ is plotted because in Table 2, $S_1 = S_0 =$"miss," thus the period of "miss" at event one is one. The point $(4, 2)$ is plotted because $S_4 = S_2 =$"hit," yielding a period of two. The categorical periodogram is designed to generate horizontal lines across sections of the graph that contain strong periodic components.

An example of a categorical periodogram from trace data is shown in Figure 7, which shows the CP for the dining philosophers trace. Several strong periodic components can be seen. Two central components with periods of approximately 30 and 60 are particularly prominent, while some shorter components at the left and right, which result from processes starting and stopping, also stand out clearly.

4.2.1 Timed Categorical Periodogram

The timed categorical periodogram, like the timed CCP, uses time between repetitions of states rather than number of intervening events to determine period. Figure 8 shows the timed categorical periodogram for the five dining philosophers problem. Several sequences with strong periodic behavior can also be

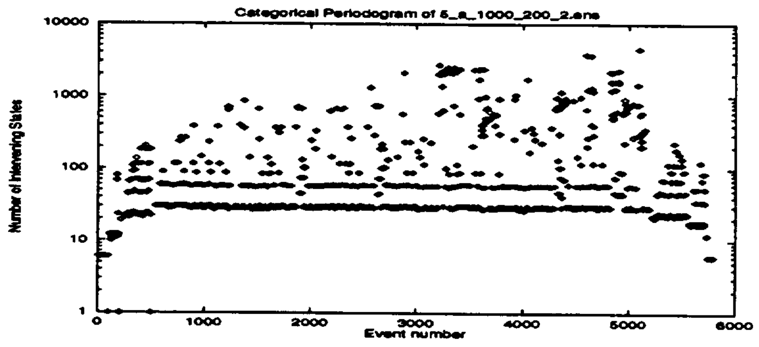


Figure 7: Untimed Categorical Periodogram

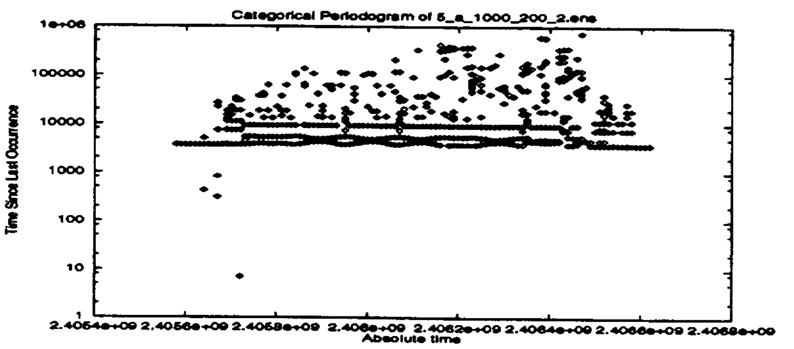


Figure 8: Timed Categorical Periodogram

seen here. In addition, some oscillating features can be seen. When a frequency is present in a trace for a contiguous time interval, *lines* will form from the concentration of plotted points. When these lines appear to move upward it is an indication of periods becoming longer, and when lines move downward, it is an indication of periods becoming shorter. There appears to be a strong relationship between two components in this trace. As the period of one component increases, the other decreases.

Figure 9 shows the oscillating section from the dining philosophers trace in greater detail and with a linear rather than a log y-axis.

4.3 Mass Evolution Graph

It is difficult to identify which states comprise periods of interest using the categorical periodogram. A single horizontal line may be made up of a series of different states. For example, the subsequence "abcabcabc" will produce a solid horizontal line at $y = 3$. If, later in the sequence, these same states are included in a subsequence of period six, there will be no way to identify the fact that these individual states have a new period of reoccurrence. The Mass Evolution Graph (MEG) shows how the periodicity of each state changes in time. The x-axis is the event number, and the y-axis indicates the number of times the state has previously appeared in the trace.

Figure 10 shows the untimed MEG for the sub-trace of Table 2. The two different states (cache hit and cache miss) are each represented by a different line. The ith occurrence of a state is plotted at y-coordinate i. The x-coordinate is the event sequence number. MEG lines that are perfectly straight identify periodic behavior. This can be seen in the segment of the trace that contains only cache misses.

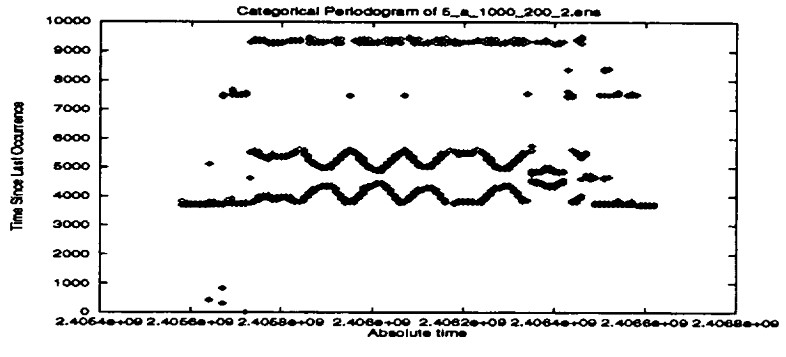


Figure 9: Oscillating Section of Timed Categorical Periodogram

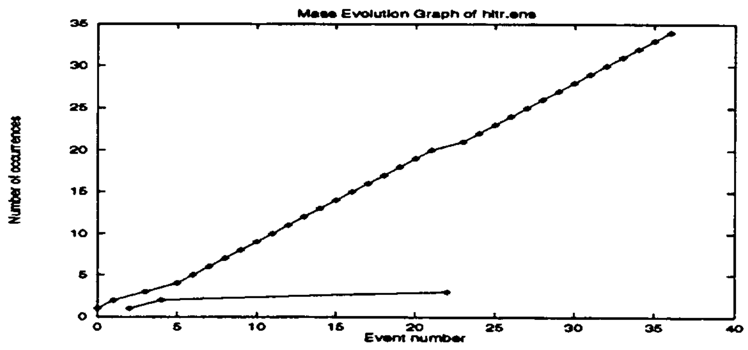


Figure 10: Mass Evolution Graph of Cache Hits

In a system where a state s occurs with a probability $p(s)$, we expect that a linear regression of the s path in the MEG will have a slope equal to $p(s)$. We can construct a MEG using either untimed or timed definitions of period.

The MEG frequently extracts characteristics of a trace that are not easily discernible in any other view. For example, Figure 11 shows that the Gaussian Elimination trace contains a series of distinct *phases*, that is, it has distinct sections of the trace which have different probability density functions. Places in the graph where lines touch the x-axis indicate times when model states appear for the first time.

The slope of the MEG line between points (x_i, y_i) and (x_{i+1}, y_{i+1}) is:

$$\frac{(y_{i+1} - yi)}{(x_{i+1} - x_i)} = \frac{1}{(x_{i+1} - x_i)},$$

which is equal to the frequency, or the reciprocal of the period in the categorical periodogram. So the derivative of the mass evolution graph is equal to the reciprocal of the categorical periodogram. When the i^{th} model state in the sequence appears at period p in the categorical periodogram, the i^{th} model state in the MEG forms a line of slope $1/p$ with the preceding occurrence of that model state.

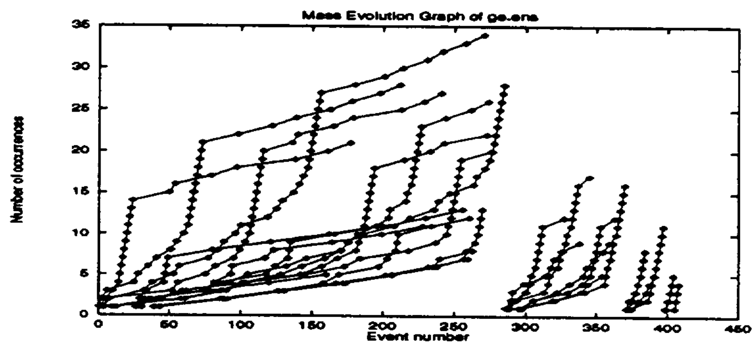


Figure 11: Mass Evolution Graph of Gaussian Elimination Trace

4.4 Correlation of Categorical Time Series

4.4.1 Components of Categorical Correlation

Correlation for numeric data is defined based on the *strength of the relationship* [Ott88] between an independent and dependent variable. So for example, if the dependent variable gets larger when the independent variable gets larger, there is said to be a strong correlation between the two values. Categorical data does not share this concept of larger and smaller values (which is larger "a" or "b?") so another definition for categorical data is required. The definition should reflect our strong intuitive feelings about correlated categorical data. For example, it seems obvious that two sequences, "abcdefg" and "abcdefg," should posses the highest degree (i.e. perfect) correlation. Similarly "abcabc" *seems* to be correlated to "xyzxyz," but *seems* to be more highly correlated to "cbacba" and also correlated to "abcabd." These concepts of correlation depend not on the one-to-one relationship between independent and dependent variables, but on the similarities in *sequences* of independent/dependent variables.

Our intuitions, though strong in some instances, seem weak when we ask questions like, "Is 'abcabc' more highly correlated to 'abcefg' or 'cbacba?'" These questions are difficult because they compare different aspects of our intuitive idea of correlation. For example, "abcefg" matches "abcabc" for the first three elements in the sequence, but "cbacba" has a one-to-one matching for each element in the sequence and it also uses the same alphabet.

These different aspects suggest that the correlation should be a *vector* rather than a single value, where each element of the vector corresponds to particular aspects of categorical correlation. The approach here is to implement and test a categorical correlation made up of several components, and attempt to evaluate the efficacy of each. All of the individual components range from 0 to 1, where 1 indicates perfect correlation.

4.4.2 Correlating Subsequences

Rather than computing a correlation for the entire trace, we utilize a pair of moving windows, one window for each trace, to determine the correlation of two subsequences. For each event, a correlation value is computed which reflects the correlation of the two subsequences that are centered at that event. The window size determines the length of the two subsequences. This method provides a view of how the correlations of local subsequences change over time.

4.4.3 Parameters for Categorical Correlation

Several parameters will be used to describe categorical correlation. To determine the correlation for two categorical sequences S and S' of length n, the following parameters will be used:

s_k is the k^{th} element in sequence S, $1 \leq k \leq n$
s'_k is the k^{th} element in sequence S', $1 \leq k \leq n$
w is the size of the windows that will be compared
W is a subsequence in the window on S
W' is a subsequence in the window on S'
W_i is the i^{th} state in W, $1 \leq i \leq w$
$||A||$ is the size of the alphabet used (i.e. number of unique states)

4.4.4 Position Correlation

The simplest measure of correlation is the number of items in the two subsequences that match both in category and position, divided by the number of items in the subsequence. We refer to this aspect as *position correlation.*

$$\text{PositionCorrelation} = \sum_{i=1}^{w} \begin{cases} \frac{1}{w} & \text{if } W_i = W'_i, \\ 0 & \text{otherwise} \end{cases}$$

4.4.5 State Set Correlation

The *state set correlation* is the number of states the two windows have in common, divided by the number of items in the subsequence. Let A represent the union of the alphabets of states present in the two windows. Let i be an index for an arbitrary numbering of the set A. Let C_i be the number of occurrences of A_i in sequence W. Let C'_i be the number of occurrences of A_i in the sequence W'.

$$\text{StateSetCorrelation} = \sum_{1}^{||A||} \frac{min(C_i, C'_i)}{w}$$

4.4.6 Transitions in Common

Transitions in common attempts to measure the degree to which the two windows make the same transitions between contiguous states. So if both windows contain many instances of the same subsequence, the transitions in common will tend to be high. Let $T_{i,j}$ equal the number of transitions from state A_i to state A_j in the sequence W. Let $T'_{i,j}$ equal the number of transitions from state A_i to state A_j in the sequence W'.

$$\text{TransitionsInCommon} = \sum_{i=1}^{||A||} \sum_{j=1}^{||A||} \frac{min(T_{i,j}, T'_{i,j})}{(w-1)}$$

We can compute a *categorical correlation vector* which contains a component for each of the measures describe above. The following algorithm computes this categorical correlation vector.

A window W of size w is centered around one event in the S sequence and a similar window W' is centered around the corresponding event in the S' sequence. A categorical correlation vector is computed for the two subsequences in W and W'. Each window is advanced by one state and the process is repeated. The result is a sequence of correlation vectors. These may be visualized as a graph where each component of the correlation vector appears as a separately plotted line in a 2-dimensional graph. Because the windows can not be centered on the initial events (there are not enough events to fill the windows), the windows are initially zero filled. This produces a bias in the initial correlations computed. In most applications these initial correlations should be discarded.

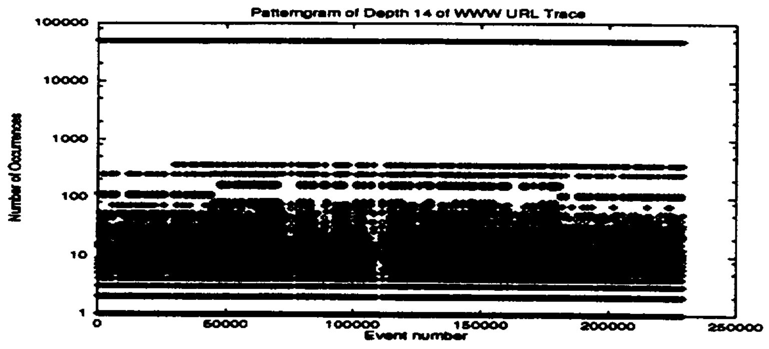


Figure 12: Trie Visualization of WWW Trace

4.5 Pattern Visualization for Categorical Time Series

4.5.1 Trie Visualization

A *pattern* is a sequence of events that consists of a specified sequence of states. The trie visualization is designed to find patterns in categorical time series. Because we do not know what patterns to look for, we need a method that will find *interesting* patterns automatically. Depending on the application, interesting patterns are patterns that occur frequently, infrequently, or within a certain range of repetitions.

If we define k as the maximum length of patterns that we want to detect, we can solve the problem by partitioning the trace into k-space, where a section of the trace: T_i to T_{i+k} is mapped to k-space coordinate $(T_i, T_{i+1}, .., T_{i+k})$. If we use a k-dimensional array s that is initialized to zero, and increment the value $s[T_i, T_{i+1}, ..T_{i+k}]$ each time that the corresponding subsequence is encountered, we will generate a sparse k-dimensional matrix which will contain values other than 0 only in locations corresponding to sequences that occur in the trace. The entries in the k-dimensional matrix are equal to the number of times the corresponding patterns occur. We can compute s in O(n) time and space.

Given s we can produce a visualization which shows, for each event i in the sequence, the number of times the subsequence beginning at event i and continuing to event $i + k$ occurs in the entire sequence.

4.5.2 Examples Using Tries

The trie visualization method can be used in a variety of applications. We have used it to analyze World Wide Web traffic, and to detect sections of code that are common to multiple computer programs

We can use the Trie technique to view the repetitions present in the WWW Traffic trace. Figure 12 shows, for each event, the number of times the 14 state subsequence beginning at that event occurs within the entire trace. There are a large number of sequences that are repeated 10-30 times. Surprisingly, there is at least one sequence that occurs over 50,000 times in the trace. This sequence consists of repeated accesses to the same file.

5 Modeling Approach

Having proposed a number of visualizations methods, we next consider the use of one of them (MEG) to generate a model. This section describes how the MEG is used as a basis for modeling. The goal of MEG based modeling is to produce a time-dependent probability mass function (pmf) $f_i(t)$ for each state i recorded in the trace. The first step towards recovering $f_i(t)$ is to plot a curve with the *Number of Occurrences* of state i as the abscissa and *Time* or *Event Number* as the ordinate. The slope of this curve at any instant t is directly proportional to $f_i(t)$. We denote the set of such curves for all unique states in the trace as the Mass Evolution Graph (MEG). The next step in our approach is to fit a curve to each path in the MEG using linear regression techniques and compute $s_i(t)$, the time-dependent slope function for each fitted curve. Since $s_i(t)$ represents the proportion of time state i occurs, instead of $s_i(t)$ we use the normalized $s_i(t)$, $s_i{}'(t)$, defined as follows:

$$s_i{}'(t) = \frac{s_i(t)}{\sum_i s_i(t)}$$

Thus $\sum_i s_i{}'(t) = 1$. $s_i{}'(t)$ is the estimated probability mass function for state i.

Table 3 below shows the normalized estimated pmf for the WWW cache hit trace. The columns *From Event* and *To Event* in this table define an event interval for the pmf. The interval is closed at *From Event* and open at *To Event*. The column *P(0)* is the probability of a cache miss during the corresponding event interval in the table. Similarly the column *P(1)* is the probability of a cache hit during the corresponding event interval in the table. The first line of Table 3 says that between event numbers 0 and 2, $p(0) = 1$ and $p(1) = 0$.

Table 3: Model for the Cache Hit trace

Estimated pmf			
From Event	To Event	$P(0)$	$P(1)$
0	2	1.00	0.00
2	5	0.50	0.50
5	6	0.90	0.10
6	22	0.95	0.05
22	26	1.00	0.00

The steps in the process of building a MEG model from a trace are discussed in detail below.

5.1 Partitioning Paths in the MEG

A path i in the MEG is a realization of the time-dependent pmf, $f_i(t)$, for the corresponding state. Our goal is to obtain an estimate of this pmf. As discussed in section 5 above, the pmf for a state i at event t is directly proportional to the slope of the corresponding MEG path $s_i(t)$ at that event. This holds true for all events except where the time-dependent pmf $f_i(t)$ is not differentiable. Denote such events as *partitioning events*. Partitioning events define the boundaries of *event-intervals* such that:

- during each event-interval the function $f_i(t)$ has a constant value, and
- for each consecutive event-interval the function $f_i(t)$ has a distinct value.

To estimate the function $f_i(t)$ we must determine all the partitioning events for it. In a MEG path, events where the slope of the path changes significantly are strong indicators of a partitioning event for the corresponding state. In this step we determine the partitioning events for the pmf of a state by partitioning the corresponding MEG path into segments of nearly-constant slope. We use two algorithms that form a UNIX-style pipeline to partition a path in the MEG into segments having a constant slope. The first algorithm computes the slope at every event for each path in the MEG. The input to this algorithm is the MEG. The second algorithm partitions each path in the MEG into segments of constant slope. The input to this algorithm are the MEG and the output from the first algorithm.

The partitioning algorithm inspects the slope at each consecutive pair of events in a MEG path. If the slope at event E_i differs from the slope at the immediately previous event E_{i-1} by more than the user defined parameter *threshold* then the event E_i is a partitioning event. Another user defined parameter *runlength* enforces a lower bound on the number of events in a segment (e.g., if *runlength* $= 2$ then a segment must have at least two points).

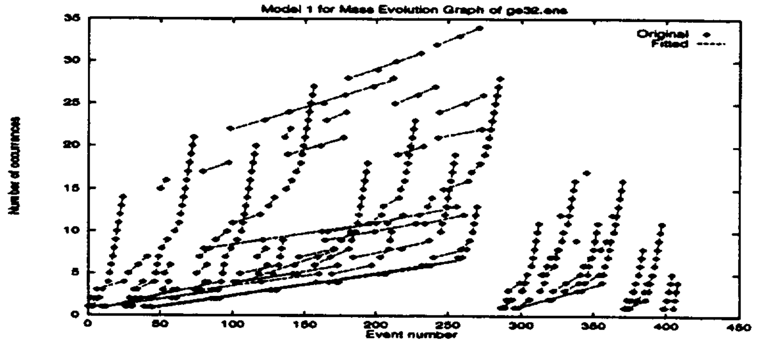


Figure 13: MEG of the Gaussian Elimination Trace After Fitting Straight Lines to Partitioned Segments

5.2 Fitting straight line segments to MEG paths

This step can be described by the following simple algorithm:

```
for every path i in the MEG {
   for every partitioned segment j in path i {
      fit a straight line to segment j using linear regression;
   } /* endfor */
} /* end for */
```

Figure 13 shows the MEG for the Gaussian Elimination trace after each path in it has been partitioned (using the approach described in section 5.1) and each partitioned segment has been fitted with a straight line.

Note that a single straight line can be fitted to the entire MEG path. This saves us from the partitioning step described in the previous section (section 5.1). However, this usually leads to a less accurate model.

5.3 Generating the Model

Generating the model after fitting straight lines to segments involves inferring an empirical probability mass function using the fitted lines. To illustrate the mechanism for generating the model we start with a simpler scenario, namely a single straight line is fitted to every path in the MEG output. Recall that every path in the MEG output corresponds to a state in the trace. Also the slope of a MEG path is proportional to the pmf for corresponding state. Using the above we can derive the probabilities for the simple scenario by using the slopes of the straight lines fitting each path in the MEG output. Recall that we already have these *FittedSlopes* from the previous step described in section 5.2.

The same idea is extended to MEG paths that are partitioned into segments. Consider the scenario where a MEG path i (corresponding to state i) is partitioned into k segments and a straight line is fitted to each of these k segments. From the *FittedSlopes* of each of the k segments we can build a time-dependent estimated pmf for state i. Applied to all paths (and hence all states) in the MEG output this generates an estimated time-dependent probability mass function for all states.

6 Model Validation

This section describes how we validate the MEG model. We also demonstrate how well our model compares to a *time-homogeneous* semi-Markov model which does not take time-varying behavior into account. The model generated from the Gaussian Elimination trace is used as an example.

6.1 Tests for Model Validation

We use two tests for model validation – categorical correlation and homogeneity.

6.1.1 Categorical Correlation

Categorical correlation (section 4.4) measures how closely categorical time series resemble one another. We use categorical correlation plots to gauge the accuracy of our model by measuring the correlation between a trace synthesized from our model and the original trace.

6.1.2 Test of Homogeneity

Another validation test we use is the test of homogeneity. It is a statistical test which tests the following null hypothesis:

- H_0 : A trace synthesized from our model is generated from the same distribution that generated the original trace.

We use the Kruskal-Wallis rank sum test with correction for ties [Ott88]. The Kruskal-Wallis test is non-parametric test, so we do not need to make any distributional assumptions about the trace data.

To validate our model using the Kruskal-Wallis test we first synthesize traces from our model. A synthesized trace is then compared with the original trace using the Kruskal-Wallis test. To make the test more rigorous we partition the synthesized and the original traces into ten partitions and then apply the Kruskal-Wallis test to each such partition in the synthesized trace and the corresponding partition in the original trace. Denote by n the number of partition pairs for which H_0 is not rejected. If $n = 10$ then all ten partitions of the synthesized trace match their corresponding partitions in the original trace.

6.2 Validation Results

The results from the validation tests for the model generated for the Gaussian Elimination trace are described below.

6.2.1 Categorical Correlation Plots

The categorical correlation plots are constructed using a moving window 32 events wide. The ordinate in these plots shows the categorical correlation component as a fraction that represents the degree of correlation between traces. The plots in Figures 14 and 15, display the categorical correlation between the original Gaussian Elimination trace (*ge32.original*) and a trace synthesized from the MEG model (*ge32.MEG*). Also shown on each plot is the categorical correlation between the original Gaussian Elimination trace (*ge32.original*) and a trace synthesized from a semi-markov Model (*ge32.semi-markov*). Figures 14 and 15, illustrate that for each of the categorical correlation components displayed, the trace synthesized from the MEG model follows the original trace more accurately than the trace synthesized from the semi-markov model.

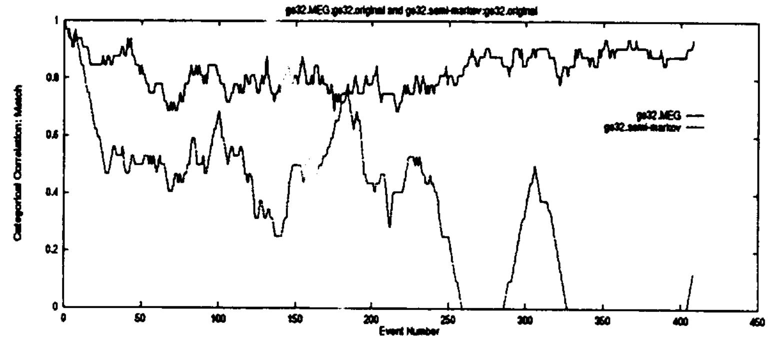


Figure 14: Categorical Correlation Plot: Match Component

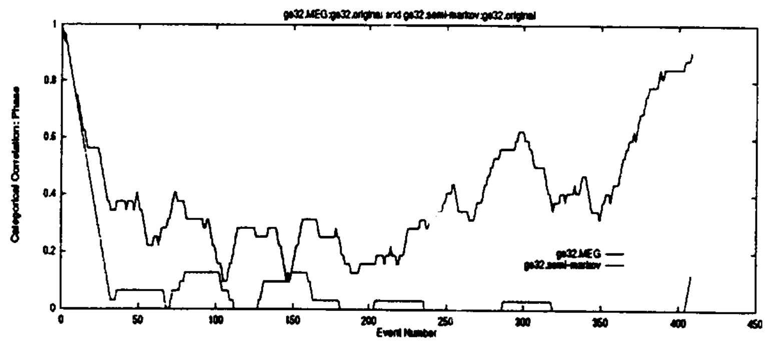


Figure 15: Categorical Correlation Plot: Phase Component

6.2.2 Results of the Kruskal-Wallis test

Table 4 shows the results of the Kruskal-Wallis test on original Gaussian Elimination trace (*ge32.original*), a trace synthesized from the MEG model (*ge32.MEG*), and a trace synthesized from the semi-Markov model (*ge32.semi-markov*). As shown in Table 4 only 5 (out of 10) of the trace ge32.semi-markov pass the Kruskal-Wallis test when compared to ge32.original. In a similar test all 10 segments of the trace ge32.MEG passed.

Table 4: Results of the Kruskal-Wallis test for the Gaussian Elimination Trace

Kruskal-Wallis test results ($\alpha = 0.05, \chi^2 = 3.84146$)			
Model	No. segments passing the test	Mean of H'	Standard Deviation of H'
Semi-Markov	5 of 10	25.12	27.67
MEG	10 of 10	0.34	0.38

7 Conclusions and Future Work

In this paper we have demonstrated a number of visualization techniques for categorical time series data, and a modeling approach which can build accurate models of time-dependent data. These techniques can be used to help draw insights from very large data sets.

Because it is risky to draw conclusions about a population of traces from a single trace, all of these techniques are currently being expanded to support *ensembles* of data. Many of the visualizations are also being expanded to provide 3-dimensional views of the categorical attributes that we have described.

All of the views and models described in this paper have been incorporated into *Chitra95*, a third generation performance analysis and visualization tool developed at Virginia Tech [ALG94, ABRV94], available via WWW from http://www.cs.vt.edu/~chitra.

References

[ABRV94] M. Abrams, A. Batongbacal, R. Ribler, and D. Vazirani. CHITRA94: A tool to dynamically characterize ensembles of traces for input data modeling and output analysis. Technical Report TR 94-21, Computer Sci. Dept., Virginia Tech, Blacksburg, VA 24061-0106, June 1994. Available from World Wide Web location http://www.cs.vt.edu/~chitra.

[ADM92] M. Abrams, N. Doraswamy, and A. Mathur. Chitra: Visual analysis of parallel and distributed programs in the time, event, and frequency domain. *IEEE Trans. on Parallel and Distributed Systems*, 3(6):672–685, November 1992.

[ALG94] M. Abrams, T. Lee, and K. Ganugapati. Constructing software performance models from trace data with Chitra92. *submitted to Software - Practice and Experience*, February 1994. Computer Sci. Dept., Virginia Tech, TR 94-07.

[AWA+95] M. Abrams, S. Williams, G. Abdulla, S. Patel, R. Ribler, and E. Fox. Multimedia traffic analysis using chitra95. Technical Report TR 95-05, Computer Sci. Dept., Virginia Tech, Blacksburg, VA 24061-0106, May 1995. Available from World Wide Web location http:// succeed/95multimediaAWAFPR/95multimediaAWAFPR.html.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability.* W.H. Freeman and Company, New York, 1979.

[Ott88] L. Ott. *An Introduction to Statistical Methods and Data Analysis.* PWS-Kent, Boston, 3rd edition, 1988.

[RW93] Diane T. Rover and Abdul Waheed. Multiple-domain analysis methods. In *ACM/ONR Workshop on Parallel Debugging and Performance*, proceedings appeared in *ACM SIGPLAN Notices,* 28(12), Dec. 1993, pages 53–63, San Diego, May 1993. ACM.

A TOOL FOR HIERARCHICAL REPRESENTATION AND VISUALIZATION OF TIME-VARYING DATA[1]

Pak Chung Wong
Department of Computer Science
University of New Hampshire
Durham, NH

R. Daniel Bergeron
Department of Computer Science
University of New Hampshire
Durham, NH

SUMMARY

A wavelet-based data visualization tool is presented to support multiresolution analysis of very large multidimensional multivariate scientific data. It has the ability to display scientific data in a fine to coarse hierarchical fashion. In addition, the tool has error tracking facilities that enable scientists to visualize error information of an individual coarse approximation and the whole hierarchy.

INTRODUCTION

Developing very large scientific data visualization tools is a sophisticated process. Although a variety of visualization tools exist for visualizing multidimensional multivariate scientific data [WB95a], none of them really address the data size problem effectively.

It is often the case that scientists are primarily interested in analyzing only small subsets of the data at the highest resolution obtained. The sheer quantity of data, however, makes it infeasible for them to explore the data at this fine resolution. Very large scientific data needs to be reduced to a reasonable size to make it useful. A key requirement, therefore, is to efficiently survey the full data set at a lower resolution and then be able to identify and analyze interesting subsets of the data in greater detail.

We are developing a visualization prototype that supports multiresolution analysis of very large scientific data. It is a data management tool as well as a data visualization tool. Currently our emphasis is placed on one-dimensional time-varying data. We have developed an error tracking sub-system [WB95b] and a partitioned wavelet transform mechanism [WB95d] to aid scientific data analysis. All indications are that wavelets are very powerful tools [WB95a, WB95c], not just for very large data management, but also for error tracking as well as missing values predictions [Hou94].

The prototype is implemented in C++ with a Motif front-end. It supports very large data visualization in a variety of ways:

[1]This work has been supported in part by the National Science Foundation under grant IRI-9117153.

- it accepts data in *CDF* [Nat94], as well as system-defined binary and *ASCII* formats;
- it provides orthogonal wavelet support with up to ten vanishing moments;
- it supports progressive refinement data analysis with resolution as fine as one data item per pixel;
- it allows users to retrieve data and select display resolution interactively;
- it keeps track of the accumulated data loss as well as data loss from individual resolutions during wavelet transforms;
- it displays data loss errors;
- it provides a variety of colormaps.

In the remainder of this paper, we discuss the prototype in greater detail and suggest how it may contribute to the understanding of very large scientific data.

INTERACTIVE DATA EXPLORATION MODEL

The concept of hierarchical data representation is critical for data exploration because it may be too expensive for scientists to *re-visit* the original data frequently. The size of the data may also prevent them from *storing* it locally. Besides, reduced data stored in random access memory such as a CD-ROM can be accessed freely without much memory burden. This version of data might be the best, i.e., highest resolution, the scientist can access locally.

A more important aspect of the hierarchical representation is the notion of *interactive data exploration*. Scientists can first pin-point the interesting subsets of data from the resolutions that suit their applications and memory capacity. They can then acquire finer but localized data from other data banks through channels such as *ftp*, *World Wide Web*, or any distributed CD-ROM library.

TIME-VARYING SCIENTIFIC DATA

Our target data is obtained from the NASA International Solar-Terrestrial Physics (ISTP) [God92] program. The data is stored on CD-ROM in *CDF*. The medium-sized sample data used for illustrations in this paper is extracted from the CD-ROM **USA_NASA_DDF_ISTP_KP_0003** recorded from the spacecraft GEOTAIL. This data set contains electron average energy data recorded every 64 seconds around the earth for the first three months of 1994. It has a total of $2^{17} = 131,072$ integers.

WAVELETS

Although wavelets play an important role in our design, we do not describe wavelet theory in detail in this paper. The reader is referred to [Str89, Dau92, Chu92] for more details. For novices, we recommend [WB94].

Without loss of generality, a wavelet is described as a filter matrix that accepts a data stream with N items, and generates $\frac{N}{2}$ *approximation* values and $\frac{N}{2}$ *detail* values. The approximation is a coarse summary of the original data. A hierarchy of coarse approximations is generated when this process is applied iteratively on the approximations to get increasingly coarse data. Figure 1 depicts a simple example of

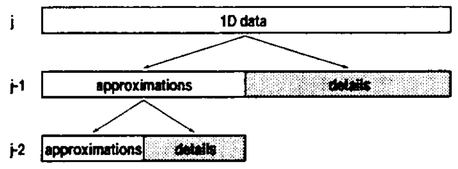
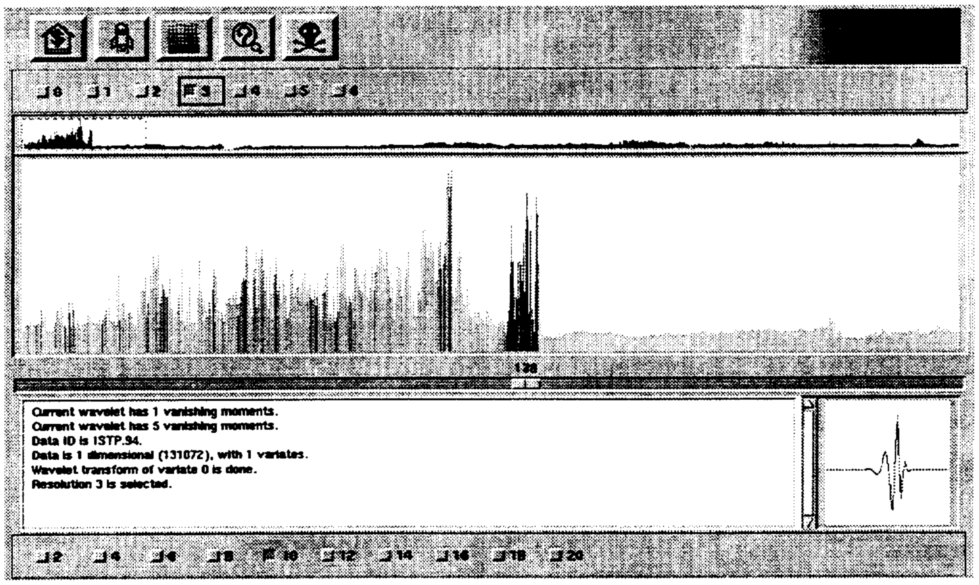


Figure 1: Wavelet decomposition on a 1D dataset.

a multiresolution hierarchy generated by wavelet decompositions. Wavelet transforms are invertible. If both the approximations and the details of any one resolution are available, it is possible to have a *lossless* reconstruction of the approximations of the next finer resolution.

USER INTERFACE OVERVIEW

The system front-end is divided into eight window panes. A copy of a typical system display is shown in Figure 2 and Color Plate 1.



```
Current wavelet has 1 vanishing moments.
Current wavelet has 5 vanishing moments.
Data ID is ISTP.94.
Data is 1 dimensional (131072), with 1 variates.
Wavelet transform of variate 0 is done.
Resolution 3 is selected.
```

Figure 2: The system displays a coarse approximation of a one-dimensional dataset with 131,072 items.

The first pane contains five command buttons, as shown in Figure 3. The first button is the *data bank* command, which initiates all the system I/O operations. The next one is the *wavelet* command button. It looks up the wavelet selection table and generates the multiresolution wavelet approximations accordingly. The middle button allows scientists to select colormaps. It is followed by a context-sensitive *help* command button, and the *exit* button.

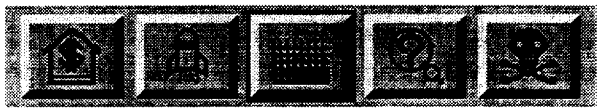


Figure 3: System command buttons.

The colormap window pane in the upper right of Figure 2 displays all the colors currently available for data mapping. Figure 4 and Color Plate 2 depict five pre-defined colormaps described by Levkowitz and

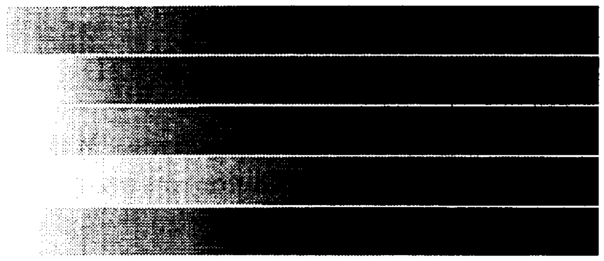


Figure 4: From top to bottom: the rainbow scale, the heated-object scale, the magenta scale, the blue scale, and the linearized optimal color scale.

Herman [LH92] for visualization. They are the rainbow scale, the heated-object scale, the magenta scale, the blue scale, and the linearized optimal color scale.

The resolution selection buttons pane is shown in Figure 5. In this example, the one-dimensional input

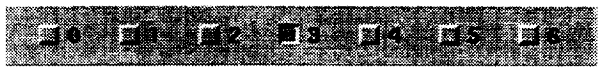


Figure 5: Multiresolution buttons.

data has a total of $131,072$ items. The coarsest resolution supported by the system is $1,024$. That means there are a total of 7 resolutions available, with resolution 0 being the coarsest.

Figure 6 depicts a portion of the data range selection pane. The background of the slider is the coarsest

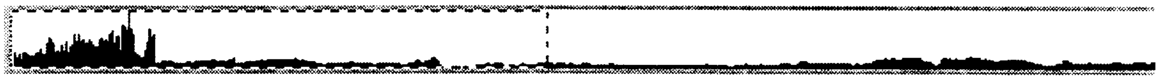
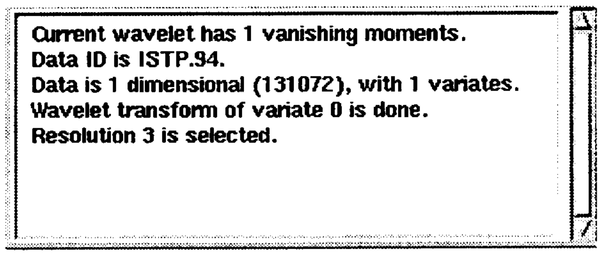


Figure 6: Data range selection slider.

approximation of the original data. It serves as a rough guideline for navigating through very large scientific data. The rectangular shaped rubberband marks the range of data being selected. The position of the rubberband is controlled by the mouse. The box size depends on the display resolution and the size of the original data.

We skip the main data display window pane for now, and move on to the next control slider. The index reflected by the slider is used to map data to color. The current prototype only allows very simple color mapping.



```
Current wavelet has 1 vanishing moments.
Data ID is ISTP.94.
Data is 1 dimensional (131072), with 1 variates.
Wavelet transform of variate 0 is done.
Resolution 3 is selected.
```

Figure 7: System message display.

The system provides a scrollable message box, which is depicted in Figure 7. It is used to display system status including data ID, metadata, colormap, wavelet, as well as error messages.

The wavelet window pane displays the wavelet currently being used. Figure 8 depicts a Daubechies

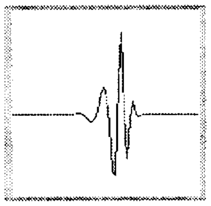


Figure 8: Wavelet display.

wavelet [Dau92] with five vanishing moments.

The wavelet selection pane contains ten selection buttons, as shown in Figure 9. The system currently

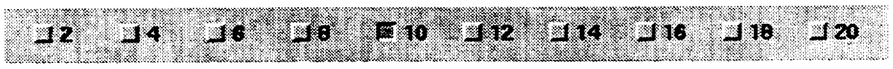


Figure 9: Wavelet selection buttons.

provides compactly supported orthogonal wavelets with vanishing moments from 1 to 10. This includes the Haar wavelet, and the nine wavelets of the Daubechies family.

MULTIRESOLUTION VISUALIZATION

The main window pane in Figure 2 is the data visualization area of the system. Since one-dimensional time-varying data is the primary target of this prototype, we limit the visualization options to simple polyline plots. This basic technique, although very simple, is a powerful tool to convey information and characteristics of time-varying data [Cle93].

Figure 10 and Color Plate 3 show the one-dimensional ISTP data set described earlier. A highly localized Daubechies wavelet is used to generate the multiresolution hierarchy, from the 0^{th} resolution with $2^{10} = 1,024$ to the 6^{th} resolution with $2^{16} = 65,536$ items. A rainbow colormap, as depicted in Figure 4, is used for the display.

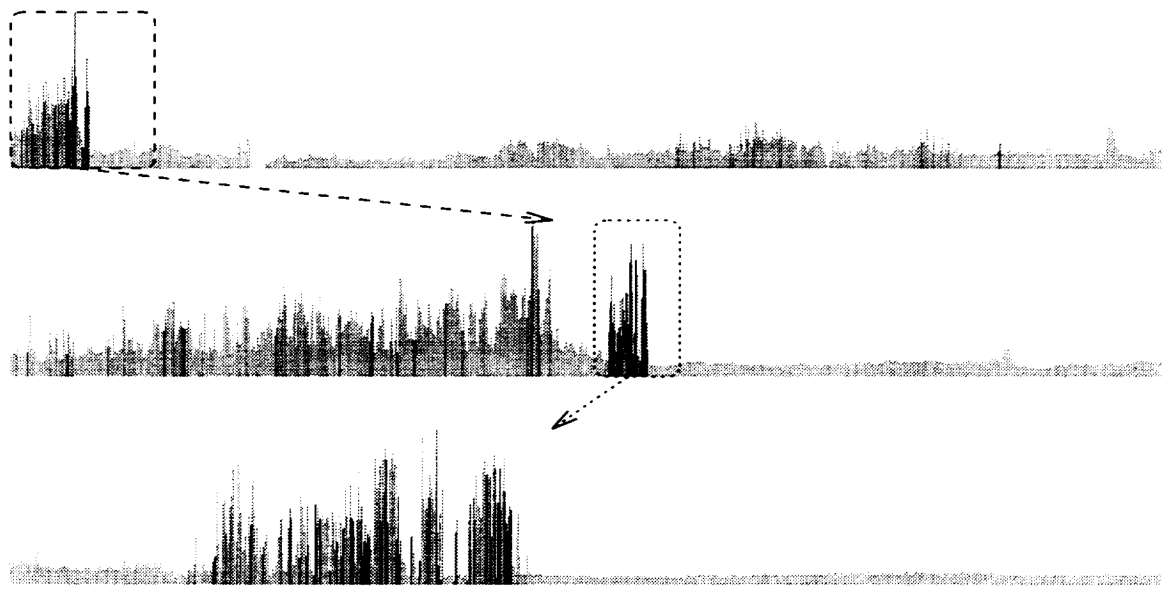


Figure 10: Top: The coarsest approximation of the data at the 0^{th} resolution. Dotted rectangles are zooming windows. The height indicates the data value and the color shows its corresponding approximation error. Middle: The zoomed data at the 3^{rd} resolution. Bottom: The zoomed data at the 6^{th} resolution.

We start from the coarsest (0^{th}) resolution with $1,024$ items. After the interesting spot (the highly fluctuating data with some of the highest data values) is identified, we study the marked data at the 3^{rd} resolution, three levels up in the hierarchy. The zooming to finer resolutions (i.e., more coefficients) continues until it reaches the 6^{th} resolution with $65,536$ items. At this point, we are positioned to study the original data.

VISUALIZATION OF ERROR

Our system provides a mechanism for tracking and visualizing data loss of an approximation hierarchy. By using a wavelet representation, the data loss can be obtained from the details of each resolution. This is described in detail in [WB95b].

The interesting spot identified in Figure 10 contains highly fluctuating data with very high data values. Both of these characteristics contribute to the very high information loss of the approximation. This is reflected by the darker colors (green/blue) of the error display in the figure. By using the color to represent the accumulated error, we are able to understand more about the quality of any portion of the approximation. In [WB95b], we present an example in which a combination of data and error display actually reveals hidden information which does not show up in the data plot itself.

CONCLUSIONS AND FUTURE WORK

We discuss the notion of multiresolution representation of very large time-varying data. The idea is illustrated with a wavelet-based analysis tool. The prototype demonstrates the feasibility of using a hierarchical data representation based on wavelets in an interactive exploratory visualization environment.

Our intermediate goal is to extend this work from handling one-dimensional time-varying data to multidimensional multivariate data. The increase of data parameters, especially the dimensional parameters, complicates the wavelet algorithms and our error tracking mechanism, and makes the visualization more complicated. The additional quantity of data also affects the computation demands, but most of this time is required for pre-processing.

Acknowledgements

The authors' work has been supported in part by the National Science Foundation under grant IRI-9117153. The NASA ISTP data is courtesy of Dr. Terry Onsager of the Institute for the Study of Earth, Oceans, and Space at the University of New Hampshire. The colormaps are courtesy of Professor Haim Levkowitz of the Department of Computer Science at the University of Massachusetts at Lowell.

References

[Chu92] Charles K. Chui. *An Introduction to Wavelets.* Wavelet Analysis and its Applications – Volume 1. Academic Press, 1992.

[Cle93] William S. Cleveland. *Visualizing Data.* Hobart Press, Summit, New Jersey, 1993.

[Dau92] Ingrid Daubechies. *Ten Lectures on Wavelets.* SIAM, Philadelphia, Pennsylvania, 1992.

[God92] Goddard Space Flight Center, National Aeronautics and Space Administration, Greenbelt, Maryland. *International Solar–Terrestrail Physics (ISTP) Key Parameter Generation Software (KPGS) Standards and Conventions – Version 1.1*, December 1992.

[Hou94] Christian Houdré. Wavelets, probability, and statistics: Some bridges. In John J. Benedetto and Michael W. Frazier, editors, *Wavelets, Mathematics and Applications*, chapter 9, pages 365–398. CRC Press, 1994.

[LH92] Haim Levkowitz and Gabor T. Herman. Color scales for image data. *IEEE Computer Graphics and Applications*, 12(1):72–80, January 1992.

[Nat94] National Space Science Data Center, Greenbelt, Maryland. *CDF User's Guide, Version 2.4*, February 1994.

[Str89] Gilbert Strang. Wavelets and dilation equations: A brief introduction. *SIAM Review*, 31(4):614–627, December 1989.

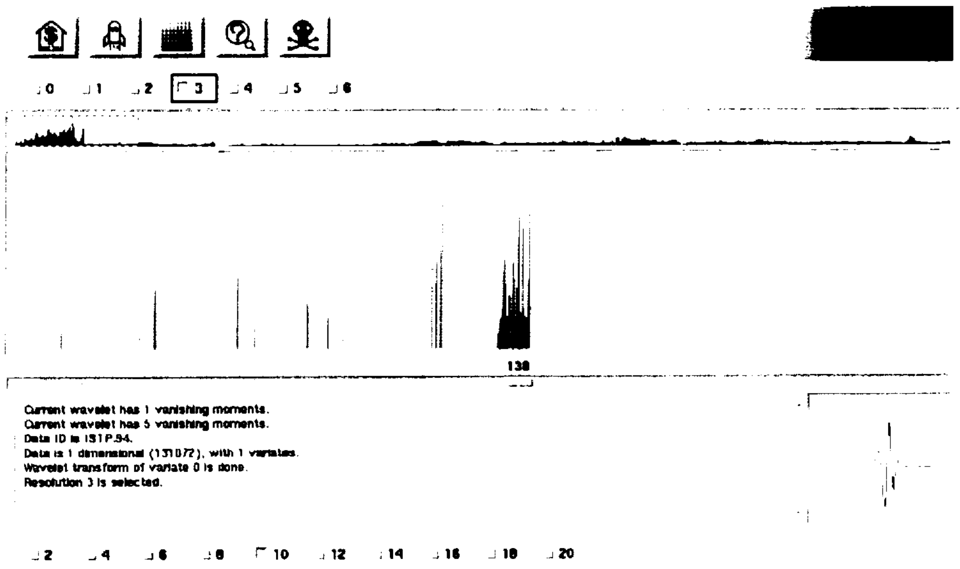
[WB94] Pak Chung Wong and R. Daniel Bergeron. A child's garden of wavelet transforms. Technical report, Computer Science Department, University of New Hampshire, Durham, New Hampshire, 1994.

[WB95a] Pak Chung Wong and R. Daniel Bergeron. 30 years of multidimensional multivariate visualization. In *Proceedings of Dagstuhl Workshop on Scientific Visualization*. IEEE Computer Science Press, 1995.

[WB95b] Pak Chung Wong and R. Daniel Bergeron. Authenticity analysis of wavelet approximations in visualization. In *Proceedings of Visualization '95*, 1995.

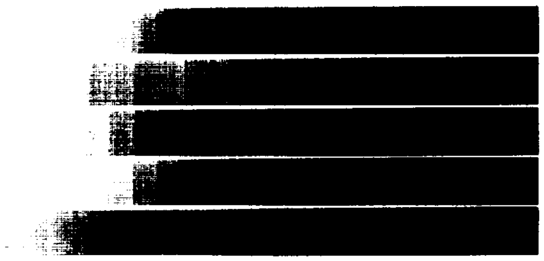
[WB95c] Pak Chung Wong and R. Daniel Bergeron. Hierarchical representation of very large data sets for visualization using wavelets. In *Proceedings of Dagstuhl Workshop on Scientific Visualization*. IEEE Computer Society Press, 1995.

[WB95d] Pak Chung Wong and R. Daniel Bergeron. A model for adaptive multiresolution representation of very large multidimensional multivariate scientific datasets using wavelets. Technical report, Computer Science Department, University of New Hampshire, Durham, New Hampshire, 1995. In preperation.

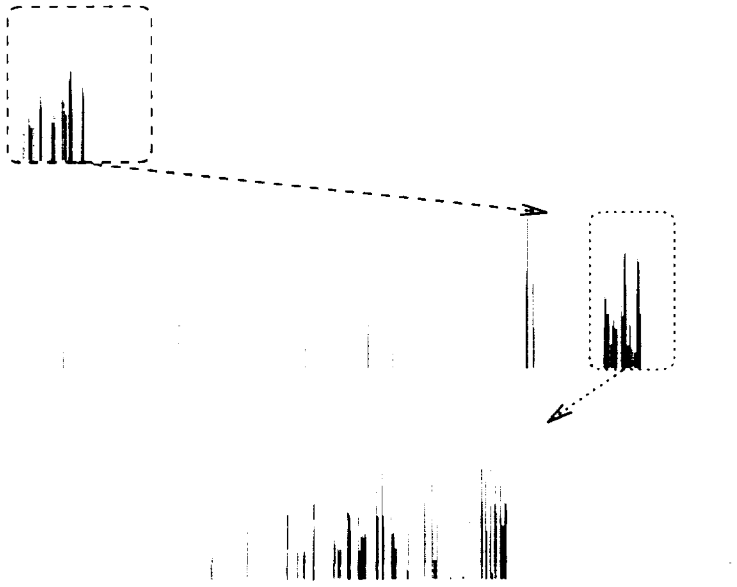


Current wavelet has 1 vanishing moments.
Current wavelet has 5 vanishing moments.
Data ID is IS1P.94.
Data is 1 dimensional (131072), with 1 variates.
Wavelet transform of variate 0 is done.
Resolution 3 is selected.

Color Plate 1.



Color Plate 2.



Color Plate 3.

REAL-TIME VISUALIZATION OF WORLD WIDE WEB TRAFFIC[*]

Will H. Scullin Thomas T. Kwan Daniel A. Reed
Department of Computer Science
University of Illinois
Urbana, Illinois 61801

SUMMARY

The World Wide Web (WWW) server at the National Center for Supercomputing Applications (NCSA) is one of the most heavily accessed WWW servers in the world. To increase our understanding of how users access this server and to provide a basis for assessing server and system software optimizations, we analyzed NCSA's server performance using the *Avatar* virtual reality performance visualization system. The large volume of performance data generated each day, coupled with rapidly changing server access patterns, makes real-time analysis both attractive and highly desirable. This paper presents the results of our analysis and discusses the implications for understanding time varying data in a virtual environment.

1 INTRODUCTION

The ease of use of the World Wide Web (WWW) [2] as a global information system has made it an indispensable tool for the worldwide Internet community. The widespread distribution of WWW servers and client browsers (e.g., NCSA Mosaic) not only fueled the explosion of interest in the WWW but also led to phenomenal growth in network traffic. This paper describes our experiences with real-time data presentation techniques for understanding this growth and its implications for next generation WWW servers.

1.1 World Wide Web Growth

Network statistics from Merit, the NSFNet backbone management group, show that WWW traffic is the largest and by far the fastest growing segment of the Internet, and growing numbers of government and commercial groups are making hundreds of gigabytes of data available via WWW servers. In particular, because NCSA, along with CERN, the European Laboratory for Particle Physics, helped pioneer the development of early server and client browser software, NCSA has been one of the most heavily accessed WWW sites in the world. The WWW servers at NCSA have experienced tremendous traffic growth, with the request rate growing at a compounded rate of about eleven percent per month for many months [10].

To support continued growth, next generation WWW servers must be capable of concurrently serving multiple request streams while managing a potentially multi-gigabyte data base of multimedia information. This places demands on the servers' underlying operating system and file system that lie far outside today's normal operating regime. In essence, WWW servers must become more adaptive and intelligent. The first step on this path is understanding extant access patterns and responses. Based on this understanding, one can then develop more efficient and intelligent server and system file caching and prefetching strategies.

1.2 Performance Analysis

Understanding extant access patterns and server responses requires the analysis of both the application stimulus (e.g., number and types of requests) and the system response (e.g., processor utilization and network statistics). At NCSA, this aggregate data rate, including both server logs and sampled processor and network performance statistics, exceeds 200 megabytes per day. From

[*]Supported in part by the National Science Foundation under grants NSF IRI 92-12976 and NSF CDA 94-01124, and by the Advanced Research Projects Agency under ARPA contracts DAVT63-91-C-0029 and DABT63-93-C-0040.

this data, one can easily calculate 40–50 important, time varying performance metrics. Understanding the correlations among these metrics, and their evolution with changing access patterns, is a daunting task.

To alleviate this problem, we have exploited and extended the Pablo performance analysis toolkit [14] to develop *Avatar* [15] — a flexible, extensible framework for using virtual reality to analyze complex performance data. *Avatar* uses data immersion to provide a fast, intuitive mechanism for evaluating data from a variety of perspectives and for manipulating characteristics of the data display. Because it can operate in real-time with a stream of arriving server requests, *Avatar* has enabled us to quickly discover transient data correlations among subsets of the WWW server performance data.

1.3 Paper Organization

The remainder of this paper is organized as follows. In §2, we describe the WWW server data collection process. Following this, in §3, we describe the *Avatar* software used to visualize and interact with the time varying WWW data. In §4, we analyze this data to identify user request patterns and correlations and discuss the implications of our findings. This is followed in §5 by a brief description of our current research directions and in §6 by a summary of related work. Finally, in §7, we conclude with a summary of our observations.

2 NCSA WWW PERFORMANCE DATA

Every day, NCSA's WWW server continuously collects performance data to enable system administrators and researchers to understand the characteristics and growth rates of the WWW access traffic. This data provides valuable insights into current server performance bottlenecks and hints about the evolution of access patterns. In aggregate, the data volume exceeds 200 megabytes/day, with 30–40 requests per second arriving at peak times. Below, we first briefly describe NCSA's WWW server architecture, followed by a description of the performance data and how it is captured for real-time analysis.

2.1 Server Architecture

To date, the backbone of the NCSA WWW service has been a group of dedicated Hewlett-Packard HP 735 workstations. As of May 1995, NCSA has eleven HP 735's dedicated as WWW servers. In the NCSA configuration, the local disk on each HP 735 stores performance data and log files. The WWW servers are connected to Andrew (AFS®)[†] [16] file servers via a 100 megabit/second Fiber Distribution Data Interface (FDDI) ring; see Figure 1. The FDDI ring connects to the rest of NCSA and, via a T3 link, to the Internet.

At NCSA, the WWW document tree is shared among the servers and stored by the Andrew File System (AFS). This center-wide AFS file system is also shared by many hundreds of client workstations and supercomputers. The current NCSA AFS file system has three SUN Sparc 10 file servers, each with 120 gigabytes of disk space. AFS provides a single, consistent view of the file system to each WWW server, allowing each WWW server to access all of the WWW document tree.

Because AFS manages the shared document tree, the individual WWW servers need not and do not know either the number or identity of the other servers. A round-robin Domain Name System (DNS) enables the servers to independently receive WWW requests; for additional details on the server architecture, see Katz *et al* [9].

[†]The Andrew File System is a product of Transarc, Corporation.

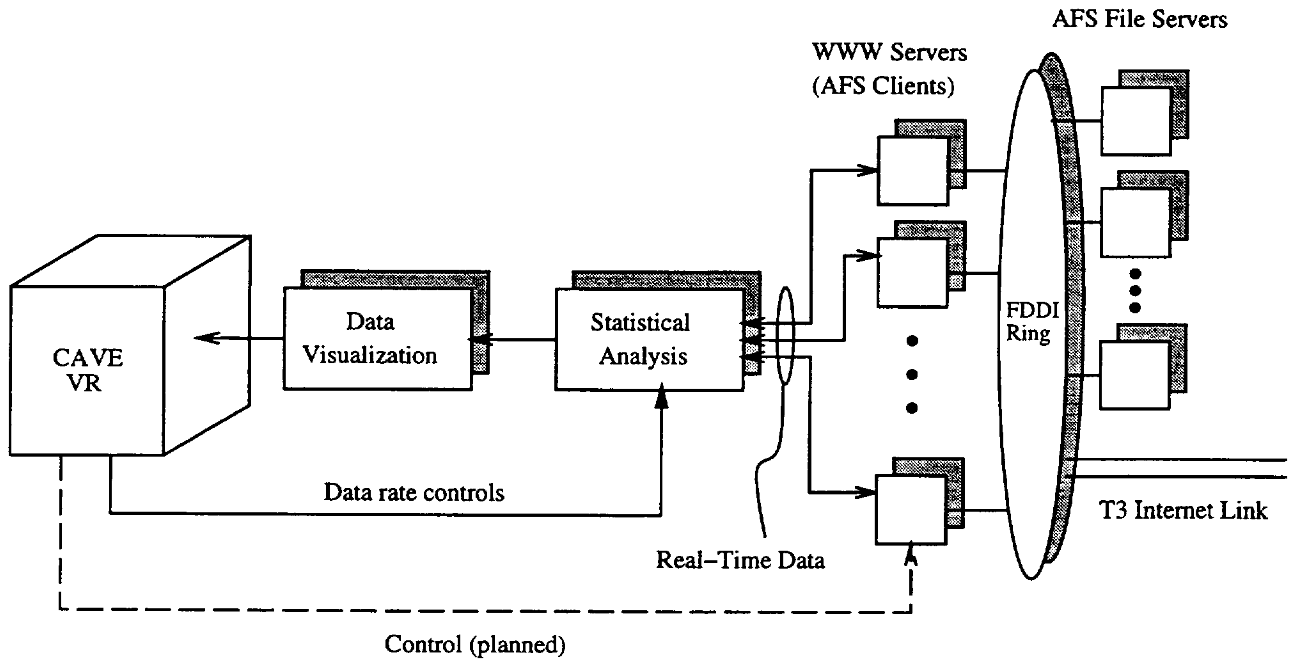


Figure 1: 2 Real-time data collection and analysis architecture.

2.2 World Wide Web Data

Each WWW server executes a Hypertext Transfer Protocol (HTTP) daemon (httpd) that responds to requests from remote WWW clients. To support on-line and off-line performance analysis, each WWW server collects the following time varying data: (1) the standard access logs from the HTTP daemon, (2) samples of virtual memory statistics, obtained by recording Unix *vmstat* data once each minute, and (3) samples of network packet counts, obtained by recording Unix *netstat* data once each minute. Below, we describe these three data sources in more detail.

2.2.1 Access Logs

NCSA's HTTP daemon produces a log entry for each request. Each entry includes the IP address or domain name of the client, a timestamp (with one second resolution), the identity of the request (i.e., the document requested), the size of the document requested, and other information. From this information, one can compute several time varying access pattern metrics, including document types, request rates, data volumes, and locations.

For example, because the WWW server logs contain the name of each requested document, and a file extension is used to identify the document type, it is possible to classify the data as requests for text, images, audio, video, and scientific data. For example, the text category includes hypertext markup language (HTML) documents, plain text, and postscript files; the image category includes `GIF`, X bitmap (`xbm`), `JPEG`, and `RGB` files; the audio category includes `au`, `aiff`, and `aifc` files; the video category includes `MPEG` and `QuickTime` files; and the data category includes Hierarchical Data Format (HDF) files.

Because the WWW server logs also contain the IP address of the requesting system, one can classify the requests based on originating domain. For example, common U.S. Internet domains include education (`edu`), commercial (`com`), and government (`gov`); requests from each domain can be viewed as a separate metric. To reduce the number of possible metrics, we introduced a

"Europe" pseudo-domain that includes requests from European countries that generated a non-trivial number of WWW requests.[‡] The remaining requests are grouped in a separate "other" pseudo-domain. As WWW traffic continues to increase, we expect it will be necessary to further refine this characterization by partitioning the requests based on geographic location.[§]

At present, we can extract the following metrics from the WWW server logs: the server identifier, the time, the number of requests and bytes transferred in each of seven file categories (text, image, audio, video, HDF, other, and total), and, similarly, the number of requests and bytes transferred for six domain categories (edu, com, gov, Europe, other, and total). These thirteen metrics define a subset of the data dimensions for each of the eleven component WWW servers; additional metrics are defined by virtual memory and network traffic, described below.

2.2.2 Virtual Memory Statistics

In addition to the access logs, each of the component NCSA WWW servers also records data from the *vmstat* utility. Although the details of this data vary across Unix systems, the data usually contains a measure of system processor usage (e.g., user time, system time, context switches per second) and paging activity (e.g., page ins, page outs, and address translation faults). Data from *vmstat* is collected at one minute intervals, and contributes another eighteen metrics to the time varying server performance data.

2.2.3 Network Statistics

NCSA's servers also record samples from the *netstat* utility. This utility reports the count of network packets and errors. As with the *vmstat* data, information is collected at one minute intervals, and provides some insights into the behavior of the network interfaces on the servers. This data shows, for instance, that the number of packets is highly correlated with the number of requests; we will return to this issue in §4. The *netstat* utility contributes an additional two metrics (number of input output network packets) to the performance data.

In aggregate, the HTTP server logs, the virtual memory statistics, and the network statistics define a high-dimensional performance data space. One minute samples of this data define the location and motion of the eleven component WWW servers within the metric space. Our goal is understanding the time varying characteristics of this metric space.

2.3 Real-time Data Analysis

Figure 1 shows the real-time data capture and analysis system we have developed. The access logs, virtual memory statistics, and network statistics are periodically transferred via TCP connections from the WWW servers to an analysis program. The analysis program computes statistics for each performance metric and merges the data into records. Because the virtual memory statistics and the network statistics both have a one minute resolution, statistical data summarization is performed on the access logs (which have one second resolution) to yield performance metric samples once each minute.

After computing the statistics, the analysis program produces data records in the Pablo Self Defining Data Format (SDDF); SDDF is described in greater detail in §3.4.1. The SDDF records are then transferred to the *Avatar* visualization module for rendering in the NCSA CAVE [5], an unencumbered environment for immersive data analysis. In the following section, we will describe the visualization module in detail.

[‡] At present, these countries include Belgium, Finland, France, Germany, Italy, the Netherlands, Norway, Spain, Sweden, Switzerland, and the United Kingdom.

[§] Indeed, since the this paper was first written, we have further subdivided the categories and mapped them to specific geographic regions.

3 *Avatar* DATA IMMERSION ENVIRONMENT

Avatar is a flexible, extensible framework that uses virtual reality to support analysis of complex, high-dimensional, time varying data. To date, the focus of our work has been performance data analysis. However, the *Avatar* infrastructure is sufficiently general to support analysis of high-dimensional time varying data from any source.

Below, we first describe one of *Avatar*'s metaphors for data interaction and it implementation; we then describe how users can interact with the *Avatar* virtual environment. This is followed by a brief description of the input data format.

3.1 Scattercube Matrix Metaphor

In our problem domain, there are roughly forty metrics for each of the (currently) eleven WWW servers, the metrics change each minute, and there are tens of megabytes of statistical data each day. Moreover, some of the performance metrics are discrete, others are continuous, and their dynamic ranges can differ by multiple orders of magnitude.

Because human visualization and spatialization skills evolved to recognize two and three-dimensional projections, understanding the relations among such abstract, multivariate data is a difficult task. In short, the conundrum of high-dimensional data visualization is finding presentation metaphors that successfully project the data in meaningful ways.

Avatar draws on a long history of data display research in the statistical graphics community [4] by extending the notion of a two-dimensional scatterplot matrix to three dimensions. In a two-dimensional scatterplot matrix, all possible pairs of dimensions for a set of n-dimensional data are plotted against each other in separate scatterplots arranged in an n by n matrix. This shows all possible two-dimensional data projections and can be used to determine data dispersion and bivariate correlations.

As an example, Figure 2 shows a scatterplot matrix of eight variables. This figure illustrates three important attributes of scatterplot matrices. First, the matrix is symmetric; the upper and lower triangles are simple transpositions. This allows one to view whichever projection is more convenient. Second, the diagonal is degenerate — both metrics are the same for these scatterplots. In Figure 2, these degenerate scatterplots have been replaced with plots [4] showing the mean and variance of the associated metric. However, if the data were plotted in the diagonal scatterplots, its dispersion would be visible on the diagonal. Third, data clustered in one scatterplot need not be clustered in other scatterplots. Indeed, this will occur only if the data is clustered in all dimensions.

A scattercube matrix extends the notion of showing all bivariate projections in the two-dimensional scatterplot matrix to showing all combinations of trivariate projections in individual scattercubes. Like the scatterplot matrix, this also allows one to study data dispersion and correlations, but with an added dimension. Moreover, it maps naturally to a virtual environment where one can walk around and inside the data. In such an environment, each scattercube forms a virtual room of data. Figure 5 shows the exterior of a six-dimensional scattercube matrix; the cube color scheme is explained below.

Like a scatterplot matrix, a scattercube matrix has symmetries, though they are more complex, with bivariate and trivariate degeneracies and multiple transpositions. In a two-dimensional scatterplot matrix, degeneracies occur only along the diagonal. In a three-dimensional scattercube matrix, degeneracy occurs along three diagonal planes; in all scattercubes in each of these planes, two of the three metric axes are identical. As an example, in Figure 5 the three degenerate planes are shown in blue, green, and red. The intersection of the three planes, the set of white cubes that cut through the diagonal of the cube of cubes, is additionally degenerate — all three metric axes are

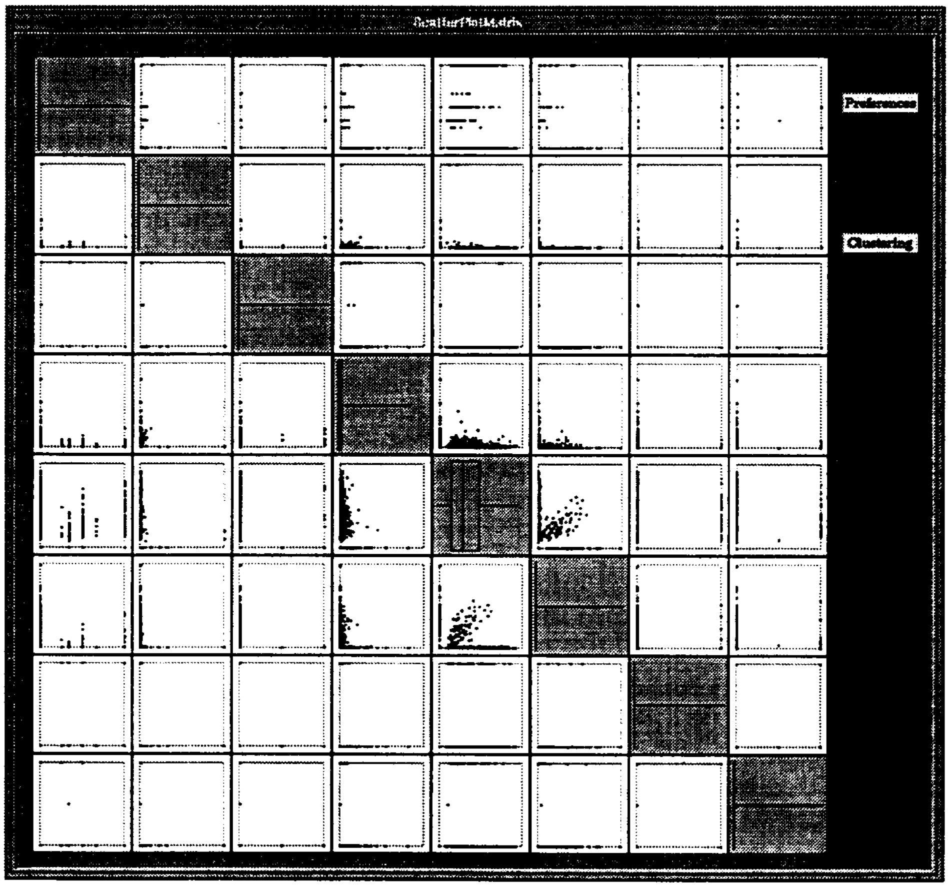


Figure 2: Scatterplot matrix.

identical. Finally, in Figure 5, the non-degenerate cubes are violet, and there are multiple transpositions of the non-degenerate groups.[¶]

Our implementation of the scattercube matrix is dynamic, with the location and attributes of each of the displayed data values continually updated. Geometrically, the behaviors of the p entities define a set of p curves in an n-dimensional performance metric space, and the measured data define a series of irregularly spaced points on each entity's behavioral trajectory. Hence, understanding the movement of the data points is as important as understanding their spatial relation at any temporal point.

To help analyze data point trajectories, it is possible to display these trajectories using history ribbons. These history ribbons serve two roles. First, they are markers of data paths, much as massless particles and stream lines show air movement in visualizations of computational fluid dynamic calculations. Second, they fulfill the same role as *brushing* in statistical graphics [4] — interactively enabling history ribbons for a subset of the data points allows one to see if the selected points cluster in one or more scattercubes.

In our implementation of history ribbons, more recent positions are rendered using bright colors; darker colors show older positions. As an example, Figure 6 shows an overview of the scattercube

[¶]Typically, *Avatar* does not display degenerate cubes, both to reduce rendering time and to increase the visibility of the remaining, non-degenerate scattercubes.

with degenerate cubes removed and history lines (in blue) activated. Figure 8 shows a closeup of a single scattercube with history ribbons enabled; in this figure, a subset of the points are clustered, with one outlier clearly visible.

3.2 *Avatar* Implementation

To realize the *Avatar* scattercube metaphor just described, we have developed a virtual world implementation that operates with either (a) a head mounted display (HMD) and six degree of freedom tracker, (b) the CAVE[||] [5] virtual reality theater, or (c) a workstation display with stereo glasses. Sonification software supports real-time mapping of data to sound using either the *Porsonify* [11] sonification library or the CAVE's native sound library. The HMD version supports three dimensional sound spatialization via a Crystal Rivers Technologies Convolvotron [18, 17].

All versions track head and hand position, orientation, and movement, and render new scenes in real-time based on user movement. A tracked mouse allows one to interact with three-dimensional objects. Users can move inside a single scattercube by walking and between scattercubes by "flying." The latter is accomplished by pointing the tracked mouse in the desired direction and clicking a button. Unfortunately, space precludes a complete description of the *Avatar* implementation and its capabilities. Below, we briefly summarize those aspects most relevant to user interaction and data analysis; for additional implementation details, see [15].

3.3 Immersive Interaction

One of the more difficult implementation problems in virtual reality is user interaction. Capitalizing on new hardware technology and the kinematic and haptic senses requires a judicious balance of new and familiar interaction techniques. Below, we describe *Avatar*'s menus and window system, data point and data dimension selection, navigation, sonification, and voice recognition features.

3.3.1 Menus and Windows

In *Avatar* the primary interaction mode remains a set of menus and windows. Figure 7 shows an example of the *Avatar* window and menu within a single scattercube. For the sake of simplicity and familiarity, these windows and menus resemble those of a standard workstation or PC windowing system. The primary difference lies in the pointing device used in virtual environments.

Unlike a desktop mouse, which is untouched except when needed, tracked, three-dimensional mice are continually held by a user. Consequently, they are much more susceptible to accidental movement. Current, temporary solutions to this problem have been making menu items very large, providing as much feedback on current cursor location as possible (e.g., by sound cues when menu items are selected), and being patient.

A disadvantage of large windows and menus is that they can obstruct the user's vision of surrounding imagery. Consequently, *Avatar* allows the user to temporarily disable the window and menu interface to provide an unobstructed view of the data display. With the menus disabled, the user can still interact with the system by walking or flying, and by using voice commands; see §3.3.6.

3.3.2 Data Point Selection

In *Avatar*, the individual scattercubes typically are between 8 to 10 feet (2.5 to 3 meters) on a side. This means that one can comfortably walk about a single scattercube and view the scattercube's data from multiple perspectives. Using the tracked mouse, one can select an

[||]The CAVE is a room-sized cube of rear-projection displays that allows users to walk about unencumbered by a head-mounted display.

individual data point simply by pricking it. By selecting a point, the user can enable or disable history ribbons or query the point for additional information.

As with scatterplot brushing, data point selection or de-selection in one scattercube is reflected in all other scattercubes. This allows the user to select all the data points in a cluster in the current scattercube, fly to another scattercube, and see if the cluster exists in other subsets of the metric space.

3.3.3 Dimension Selection

Ideally, the size of the displayed scattercube matrix would be $n \times n \times n$, where n is the number of data dimensions. However, limitations of the graphics rendering hardware and memory currently restrict the number of scattercubes to at most a few dozen, depending on the number of data points. Because human factors studies show that a minimum of roughly ten frames per second is required to maintain any illusion of direct interaction, real-time display of all scattercubes for a forty-dimensional data set is not practical.

To support display and study of high-dimensional data sets while still not compromising the frame rate, we instead allow users to interactively change the metrics mapped to the axes of an individual scattercube. A simple dialog provides a list of all available metrics; picking an item from the list causes the desired axis to change. For consistency with standard scatterplot layout, all scattercubes in the axis plane change. As described below, this still permits flying along a coordinate plane to see data changes with respect to metric changes.

3.3.4 Navigation

Avatar allows one to scale the rendered scenes over a large range. For example, it is possible to scale the scattercube matrix such that all scattercubes fit within a volume slightly larger than one's head. At this scale it is possible to move between scattercubes simply by by head movement. However, we have found that the most effective scale maps each scattercube to a space roughly the size of a small room. At this scale, movement within a scattercube is best accomplished by walking. However, the layout of the scattercubes mandates some other mechanism for intercube movement.

After informal experiments, we selected a mechanism for intercube movement that combines flying (unrestricted movement in three dimensions) with warping (jumping from point a to point b). The user points the tracked mouse in the desired direction, pushes a button, and is transported from their current position in one scattercube to the same position the adjoining scattercube. This restricted form of flying keeps the scattercube floor and walls aligned with physical barriers external to the virtual environment, while still conveying a feeling of movement.

As we noted earlier, Figure 6, with the degenerate scattercubes removed, shows the complex symmetry of the scattercube matrix. A scatterplot matrix is symmetric across the degenerate diagonal, but a scattercube matrix is symmetric across all its degenerate planes, for a total of six symmetric regions. While seemingly redundant, these regions are important because they allow traversal of entire columns of metrics in any direction. Being able to traverse an entire column allows a user compare two metrics against all the other remaining metrics by simply moving in a straight line. If the symmetries were removed, this comparison would still be possible, but the path for doing this could be quite complex.

Both navigation and metric selection menus allow the user to change the metrics mapped to the axes of a cube, either by moving to a different scattercube or by changing the metrics in the current scattercube. Although seemingly redundant, they really serve different purposes. Navigation is within a context — one can see into neighboring scattercubes, the cubes are placed in a logical order, and one can quickly move through that ordering in any direction. Menus are more useful when the one wants to view a specific set of metrics.

3.3.5 Sonification

Avatar's visual displays can be augmented with sonification of data attributes. For instance, within each scattercube, the mean value of the data points in a particular dimension can be mapped to the pitch of a continuous or periodic sound. Alternatively, one can map the variance of the data or the data centroid to pitch or other sound attributes. Several such sound sources can operate simultaneously in each scattercube.

With sound spatialization hardware [18], sounds can appear to emanate from either fixed or moving points (e.g., by mapping the mean of the data to pitch and placing the pitch at the temporally varying location of the mean). Moreover, it is possible to move between scattercubes and obtain some intuition about data dispersion merely by listening to the sound changes within each cube; see [17] for more details. While sound does not provide the level of detail possible with graphics, sonic cues do not require the direct attention of the user to be useful and effective.

3.3.6 Voice Recognition

Finally, *Avatar* supports a speaker-dependent voice recognition and synthesis system. The voice recognition hardware matches utterances with a previously trained vocabulary and injects associated commands (e.g., scaling, rotation, or display opacity). Although not capable of complex voice recognition, this mechanism can be used in a manner similar to keyboard shortcuts in a desktop application and can be used even when the menu interfaces of §3.3.1 are hidden.

3.4 *Avatar* Data Inputs

To maximize portability and extensibility, *Avatar* has been designed to process high-dimensional data described by the Pablo Self Defining Data Format (SDDF) [1]. This data can originate from a file or can be real-time data transmitted via a network socket. Below, we describe SDDF, illustrate its representation of WWW data, and describe mechanisms for real-time WWW data reduction and control.

3.4.1 Self Defining Data Format (SDDF)

Avatar uses the Pablo** Self Defining Data Format (SDDF) [1] for its data representation. SDDF is a meta-data format, in the style of HDF [13] or netCDF. As such, SDDF consists of two parts: data record descriptors and data record instances defined by those descriptors. The record descriptors define the number and sizes of the data elements in each record. The actual data consists of a sequence of descriptor tags and record instances. The tags identify the record descriptor that should be used to parse the byte stream that follows.

Figure 3 shows a typical SDDF record descriptor; the header consists of pairs of attribute names and their types. For the WWW data of §2, each pair represents a different data dimension. A corresponding SDDF record is shown in Figure 4. There is a one-one mapping between fields in the header and fields in the record.

The Pablo [14] software contains a variety of tools for generating and processing ASCII and binary SDDF files, and it supports transmission of records over standard Unix sockets. There also are routines to convert several other popular data formats to SDDF.

3.4.2 Input Data Rates

When the input data is from a file, the user has a great deal of control over what rate the data is presented. The user can control how much data is read from the file at one time, and how frequently. It is possible to stop the data, step through the data file in user-defined increments, or start again from the beginning.

**Pablo is a registered trademark of the Board of Trustees of the University of Illinois.

```
SDDFA
#1:
"Mosaic_Metric" {
      int    "server_number";
      int    "minute_of_day";
      int    "in_packets";
      int    "out_packets";
      int    "page_ins";
      int    "page_outs";
      int    "device_interrupts_per_sec";
      int    "sys_calls_per_sec";
      int    "context_switch_per_sec";
      int    "user_time";
      int    "system_time";
      int    "idle time";
};;
```

Figure 3: SDDF record descriptor (*netstat* and *vmstat*).

```
"Mosaic_Metric" { 1, 1300, 1632, 1853, 46, 0, 251, 1404, 46, 1, 3, 95 };;
```

Figure 4: Single SDDF record.

In contrast, with real-time data one has very limited control over the data generation rate. If the data generation rate exceeds the display rate, the data must be buffered; this is only feasible if the aggregate data volume is modest. Ideally, the data analysis software should allow the user to interact with the data source to control many aspects of the data generation. For instance, using a dialog display in the virtual environment, a user might vary the size of an averaging window. For large window sizes, the data rate becomes small, albeit with consequent loss of fine display resolution and extra overhead for data reduction.

The real-time data analysis software of Figure 1 allows the user of *Avatar* to adjust the frequency of real-time data updates by changing the size of averaging windows. By default these windows are one minute long. However, their size can be increased to decrease the real-time data rate.

4 DATA DISCOVERY AND EXPERIENCES

Using the real-time data reduction software shown in Figure 1 and *Avatar*'s scattercube metaphor, we have explored the characteristics of WWW traffic. Below, we describe the qualitative results obtained by analyzing WWW performance data in the *Avatar* virtual environment. We also summarize our experiences with *Avatar* and real-time data. For a more extensive analysis of WWW traffic based on *Avatar* and other statistical techniques, see [10].

4.1 WWW Data Analysis

Although we had earlier conducted a statistical analysis of the WWW server logs, while being immersed in *Avatar*'s scattercube representation of the WWW data, we quickly discovered several data correlations that deepened our understanding of the WWW client stimuli and WWW server responses. The majority of these insights were based on seeing the temporal evolution of request characteristics and server responses. In our experience, understanding temporal behavior is much more difficult without dynamic graphics. We illustrate this experience with three examples.

Using *Avatar*, we could track request evolution across a complete twenty-four hour period, either by replaying previously captured data or by real-time display. By choosing a scattercube whose axes correspond to time of day, number of requests, and pseudo-domain, we could study the origin and number of requests during a day. This representation showed that the dominant, temporal feature of the request patterns is the standard business day, both in the United States and in Europe – the majority of the activity in the scattercubes occurs when the time-of-day metric falls within 8–5 local time.

As we noted in §1, for many months the request rate to the NCSA WWW server grew at a compounded rate of about eleven percent per month. However, in addition to the rate, the characteristics of the growth have important implications for WWW server implementation. Although audio and video account for only a small percent of the total number of requests, they account for a disproportionally large portion of total data volume.

In a scattercube whose axes corresponded to the volume of data served in response to audio, video, and text requests, the effects of video requests were striking. Each time a video request is serviced, the head of the history ribbon for the associated server leaps to the center of the scattercube. Moving to related scattercubes with server memory metrics showed the deleterious effects of such requests. We concluded that as more images and video clips are available online, managing system buffer caches to accommodate even a relatively small number of requests for such large objects will be a major challenge.

The scattercube metaphor has also enabled us to quickly discover qualitative correlations among server performance metrics. Near-perfect correlations are striking; the WWW server history ribbons often lie along the diagonal in the scattercubes whose axes are server metrics. For example, Figure 8 shows the correlation between the number of input and output network packets; the scattercube with these two metrics (along the major axes) has diagonal ribbons. Intuitively, this means that most requests are satisfied with a small amount of data.

Figure 8 does show one anomaly — the load on one of the servers (the blue ribbon) is much higher than that on the other servers. In the scattercube, this is represented as a outlying data point and ribbon that does not follow the general movement of the data points from the other servers.[††] The quick discovery of such anomalous behavior is often essential to the fine tuning of systems with a heavy load.

4.2 Real-time Data

Real-time data analysis has proven both an advantage and a liability. The biggest advantage is the ability to see current behavior. Because one of *Avatar*'s operating modes is on a workstation with stereo glasses, it can serve as a WWW server diagnostic station. In the future, we believe this will permit real-time interactive adjustment of server parameters to maximize performance; this is the dashed line in Figure 1.

[††]Similar behavior occurs on most days, although the heavy load moves randomly from server to server. We conjecture that some remote host is caching the IP address of a particular WWW server rather than periodically requesting a new IP address from the NCSA DNS server.

The liability is that in a real-time environment, it is not possible to freeze the generation of the WWW log data — new user requests continually arrive at the WWW servers. However, the *Avatar* user interface allows us to either record the data for off-line analysis or to specify the size of the time window in which statistics are computed. Because of the relatively large number of WWW servers at NCSA, we must ensure that data (from all the servers) we process has approximately the same timestamp. This way, activities among servers can be easily correlated in *Avatar*.

In our experiments, we selected a time interval increment of five minutes. That is, log data from each WWW server is collected and processed such that the log records processed reflect activities within the same five minute time window on each WWW server. In a round-robin order, the logs from each WWW server were collected, transferred, and processed to enable us to update and view all server activities that are correlated in time.

5 RESEARCH DIRECTIONS

Although our immersive virtual reality system already allows software developers and performance analysts to interact with executing software and modify software parameters, much work remains. We continue to enhance *Avatar* and experiment with new display metaphors to make understanding time varying data a less daunting task. Below, we discuss our on-going research efforts and possible enhancements to *Avatar*. These efforts include new instrumentation, new display metaphors, and three-dimensional projection pursuit.

5.1 WWW Instrumentation

Although the server logs and standard Unix utilities can provide considerable insight into server behavior, understanding the fine-scale causes for server performance requires more detailed data. We are instrumenting the NCSA HTTP server using the Pablo performance instrumentation toolkit [14] to capture timestamped event traces of file and network input/output operations, sizes, durations, and times and to generate statistical summaries of data in user-specified windows of time.

The primary motivation of this work is to understand the operating regime for current WWW servers and how operating system implementation idiosyncrasies limit server performance. However, the enormous volume of performance data from the additional instrumentation makes correlating diverse metrics, assessing policy alternatives, and gaining behavioral insights even more difficult.

5.2 Display Metaphors

To analyze the detailed server performance data just described, we plan to expand *Avatar*'s repertoire of data presentation metaphors to include an abstract rendering of the NCSA WWW server architecture. By showing the components of the server architecture, augmented with data on their performance, this display will enable us to understand how the servers respond to specific types of requests and how bottlenecks shift through the server components. Such understanding is a prelude to designing better server caching algorithms.

In addition, to better understand the geographic distribution of requests we have developed a "WWW globe" that maps the IP addresses of incoming requests to their geographic location and computes time varying statistics for geographic regions. Our preliminary experience suggests that this metaphor is effective in the study of relations among location, number and volume of requests and how those request characteristics relate to network bandwidths.

5.3 Projection Pursuit

Although there are already a large number of dimensions in the WWW data set, more detailed server instrumentation will produce data with more metrics and higher dimensions (e.g., one hundred or more). To cope with this ever growing dimensionality, one needs more powerful statistical data analysis techniques that can identify data correlations prior to data display.

Projection pursuit [8] is one such powerful technique. It can be viewed as a generalization of principal components analysis that finds one or more optimum projections of the data variables such that the data exhibits clusters with respect to those dimensions. Although the majority of projection pursuit techniques are intended for two-dimensional projection, there are techniques for three-dimensional projection pursuit [12]. In principle, such techniques are ideally matched to the scattercube metaphor.

By collapsing 40-100 dimensions to only three, all the data can be shown in a single scattercube. However, each of the resulting three dimensions is now a linear combination of all the original dimensions. We plan to augment the existing scattercube metaphor with three-dimensional projection pursuit and develop display techniques that will allow the user to identify the relative contribution of each original dimension to the projection, and then move to the scattercubes that display this data in the original metric space.

6 RELATED WORK

Our research draws on a long history of statistical graphics and virtual reality research. Cleveland [4] is a cogent summary of the statistics community's techniques for visualizing irregular data, Huber [7] describes early experiences with three-dimensional scatterplots. Our work differs in its generalization of scatterplot matrices to encompass three-dimensional scatterplots and the integration of history lines to show phase behavior.

The closest analog to our work within the virtual reality community is Feiner and Besher's work on multi-dimensional data spaces for visualization of financial data. AutoVision [3] and it's predecessor, n-Vision [6], use "worlds within worlds" to display N-dimensional data. Both create a hierarchy of three-dimensional displays, where within a display one can recursively nest other displays by selecting a point. Our work differs by imposing no hierarchy on the data dimensions. All are treated as equals, and one need not assign an *a priori* order or importance.

7 CONCLUSIONS

In this paper, we described how real-time performance data is captured from NCSA's World Wide Web (WWW) servers and how this time varying data is displayed and analyzed with the *Avatar* virtual environment. *Avatar* has enabled us to explore novel paradigms for displaying time varying data, for interacting with this data via immersive techniques, and for understanding the evolution of WWW access patterns and server responses. In particular, *Avatar* allowed us to quickly discover data correlations and effortlessly navigate through high-dimensional data.

Based on this and our experiences with other high-dimensional data sets, we believe *Avatar* is a promising tool for visualizing and understanding time varying data. However, much work remains, and we continue to enhance *Avatar* and experiment with new display metaphors to make understanding time varying data a less daunting task.

ACKNOWLEDGMENTS

We thank Bob McGrath for installing our daemon on NCSA's WWW servers and Chris Elford for porting the SDDF converter to the SGI platform.

REFERENCES

1. Aydt, R. A. SDDF: The Pablo Self-Describing Data Format. Tech. rep., University of Illinois at Urbana-Champaign, Department of Computer Science, Sept. 1993.

2. Bt rners-Lee, T., Cailliau, R., Luotonen, A., Nielsen, H., and Secret, A. The World-Wide Web. *Communications of the ACM 37*, 8 (Aug. 1994), 76–82.

3. Bes :ers, C., and Feiner, S. AutoVisual: Rule-Based Design of Interactive Multivariate Visualizations. *IEEE Computer Graphics and Applications 13*, 4 (July 1993), 41–49.

4. Clevt!and, W. S., and MiGill, M. E., Eds. *Dynamic Graphics for Statistics.* Wadsworth & Brook:s/Cole, 1988.

5. Cruz-Neira, C., D.J.Sandin, and DeFanti, T. Surround-Screen Projection-Based Virtual Reality The Design and Implementation of the CAVE. In *SIGGRAPH '93 Proceedings* (Aug. 1993), Association for Computing Machinery.

6. Feiner, S , and Beshers, C. Visualizing n-Dimensional Virtual Worlds with n-Vision. In *ACM SIGGRAPH Computer Graphics* (Mar. 1990), vol. 24, pp. 37–39.

7. Huber, P. J. Experiences with Three-Dimensional Scatterplots. In *Dynamic Graphics for Statistics*, W. S. Cleveland and M. E. MiGill, Eds. Wadsworth & Brooks/Cole, 1988, pp. 448–453.

8. Hurley, C., and Buja, A. Analyzing High-Dimensional Data with Motion Graphics. *SIAM Journal of Scientific and Statistical Computing 11*, 6 (Nov. 1990), 1193–1211.

9. Katz, E. D., Butler, M., and McGrath, R. A Scalable HTTP Server: The NCSA Prototype. In *Proceedings of First International WWW Conference* (May 1994).

10. Kwan, T. T., McGrath, R. E., and Reed, D. A. User Access Patterns to NCSA's World Wide Web Server. Tech. rep., University of Illinois at Urbana–Champaign, Department of Computer Science, February 1995 (available at `http://www-pablo.cs.uiuc.edu/Papers/WWW.ps.Z`).

11. Madhyastha, T. M. A Portable System for Data Sonification. Master's thesis, University of Illinois at Urbana -Champaign, Department of Computer Science, May 1992.

12. Nason, G. Three-dimensional Projection Pursuit. Tech. rep., Department of Mathematics, University of Bristol, 1994.

13. NCSA. *NCSA HDF Version 3.3.* University of Illinois at Urbana-Champaign, National Center for Supercomputing Applications, Feb. 1994.

14. Reed, D. A. Experimental Performance Analysis of Parallel Systems: Techniques and Open Problems. In *Proceedings of the 7th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation* (May 1994), pp. 25–51.

15. Reed, D. A., Shields, K. A., Tavera, L. F., Scullin, W. H., and Elford, C. L. Virtual Reality and Parallel Systems Performance Analysis. *IEEE Computer* (*to appear*, currently available as `http://www-pablo.cs.uiuc.edu/Projects/VR/`, 1995).

16. Satyanarayanan, M. Scalable, Secure, and Highly Available Distributed File Access. *IEEE Transactions on Computers 23*, 5 (May 1990), 9–21.

17. Tavera, L. F. Three Dimensional Sound for Data Presentation in a Virtual Reality Environment. Master's thesis, University of Illinois at Urbana–Champaign, Department of Computer Science, Dec. 1994.

18. Wenzel, E. M., Wightman, F. L., and Foster, S. H. A Virtual Display System for Conveying Three-dimensional Acoustic Information. In *Proceedings of the Human Factors Society* (1988), pp. 86 – 90.

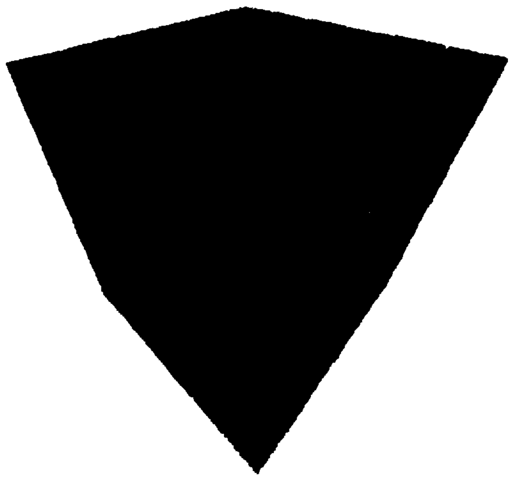


Figure 5: Scattercube matrix.

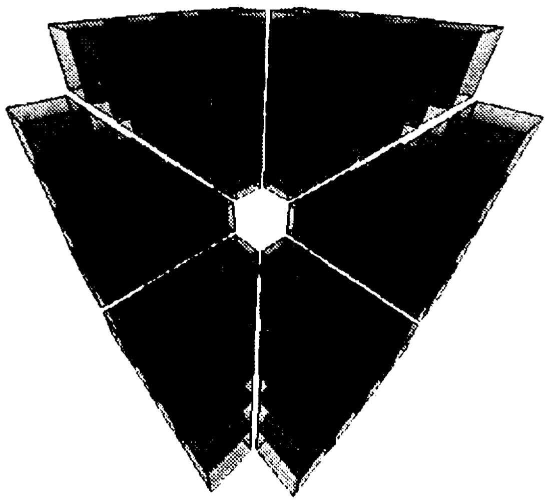


Figure 6: Scattercube (degenerate cubes removed).

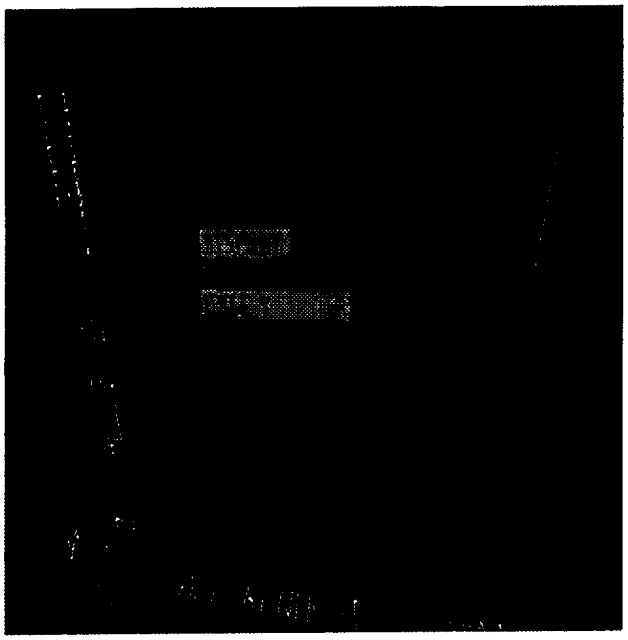


Figure 7: *Avatar* windows and menu.

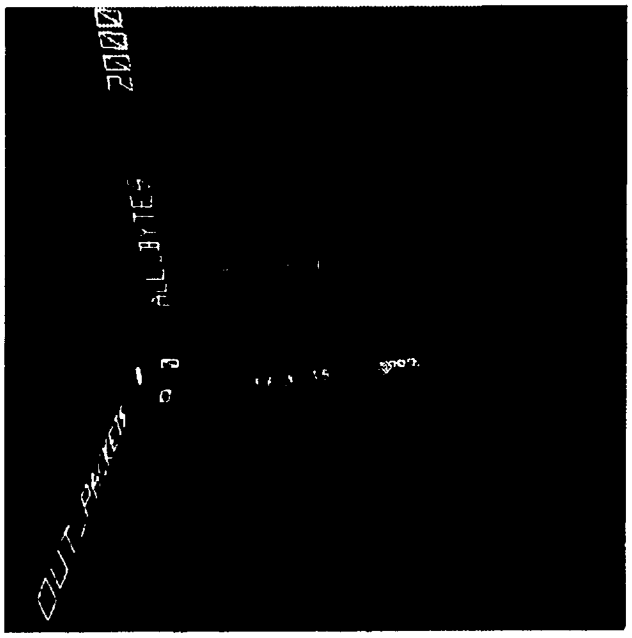


Figure 8: *Avatar* server imbalance.

VISUALIZING CAUSALITY

Eric J. Davies
University of Waterloo
Waterloo, Ontario

William B. Cowan
University of Waterloo
Waterloo, Ontario

SUMMARY

The Temporal Network Visualizer is an experimental system to perform real-time visualization of time varying network oriented data. It builds upon the concept of stripcharts to augment a user's memory of past events, enhancing the user's ability to recognize patterns that occur across time as well as space, patterns that we associate with causality. Examples are given for visualizing transactions in an economic simulation and traffic in an actual local area network. The paper concludes with a description of our preliminary user testing and its results.

PROBLEM AREA

One of the most important relationships we can uncover in time-varying data is that of causality. When a computing system fails, a building collapses or any of a variety of unexpected events occur, we go to great pains to discover what preceding events were the likely causes.

Statistics has long been a powerful tool in this. Through its devices, we seek correlations between events. After having found a significant correlation between a set of events, researchers turn to more expensive and time consuming laboratory procedures to uncover the underlying mechanism behind the correlation. This pattern is clearly visible in the history of AIDS research; high risk activities were first correlated with incidences of AIDS and later a probable mechanism was found in the HIV virus. In terms of discovering causal relationships, visualization of time-varying data may prove to be a similarly powerful tool.

Our interests are in online visualizations; visualization done with data that is continuously collected during the course of the visualization as opposed to that done with a set of data that is available at the beginning. This mandates that our visualizations be animated[1]. The common way to animate time varying data is to map the narrative time (time in the semantic or application domain) to the presentation time. For instance, an animation of the respiratory system might use color that changes during the presentation time to represent CO_2 concentrations that changed during the narrative

[1]With an online visualization, at t_i the system has available only data representing times t_0 to t_i and so can only represent that portion of time. At some later point, say t_j, a new graph must be produced if the display is to represent the newly collected data. Therefore, the display must change if it is to reflect the currently known data, and hence be animated.

time. As expanded upon in the following subsections, this approach has its shortcomings in several areas; scaling, limitations of human short term storage (memory), and limitations in how 'global' our instaneous perception is.

Scaling

In many application domains, events are on time scales that are not well matched to the time scale of human perception. As a result, it is necessary to provide a scaling of the temporal data. Very fast events, like the progress of a sub-atomic particle through a detector, need to be expanded or 'slowed down'. Very slow events, such as climate change, can take decades and hence must be compressed before viewing. However, when the observer expects to take quick action based on information presented in the visualization, an air traffic controller for example, elasticity of time cannot be provided without deleterious consequences.

Applications, such as air traffic control, are on-line visualizations since they have a temporal constraint requiring the display to reflect the current data (less some reasonable lag time from processing and display). We will call this constraint the real-time requirement. The real-time requirement means that scaling must take place in other domains in order to provide functionality similar to that provided by temporal scaling.

Short Term Storage

In each image presented, the observer may be able to notice spatial relationships between items. However, perceiving relationships that contain a temporal component (in addition to a spatial component) is much more difficult, particularly if the amount of extraneous information (noise) is significant.

As a simple but concrete example, imagine sitting in a darkened room with a set of lights $l_1, l_2, \ldots, l_n$ such that each l_{i+1} is wired to flash a set interval after l_i flashes. As the interval increases, it becomes more difficult to see a relationship between l_i and l_{i+1}. If additional randomly flashing lights are added to the room, preceiving the relationships between the lights becomes harder still.

Globality

Static visualizations, such as those commonly printed in journals, can have an arbitrarily large amount of detail because readers can take as long as needed to examine the images. This is not true of an animated visualization, particularly one monitoring a real-time source of data. The crucial information may only be displayed while the user's attention is focused on a different portion of the display.

The locality of human perception has been exploited in aircraft simulators, where eye tracking is used to present a detailed image only where the 'pilot' is looking. In the case of a visualizaton, the situation is the reverse, we need to direct the observer's eye to portions of the display we think the observer will be is interested in so that the events are seen.

What Is Needed

We have shown that there are three fundamental properties that are valuable when we visualize temporal relationships: the visual representation should allow the time component to be scaled, the recognition of relationships should not rely on the user's short term storage, and the representation should assist the user with perceiving relevent patterns (such as potential causal relationships). The Temporal Network Visualizer (TNV) is a system which has these properties.

The remainder of this paper discusses how we have given these properties to TNV, and how we have applied the visualizer to two different application domains that are time oriented: local area network traffic and economic modeling. We conclude with preliminary results of our user testing.

BASIC APPROACH

The difficulty in solving the light partioning problem described earlier is that the short term storage associated with visual processing is limited and degrades with increased detail. Our solution is to transform the problem from being one of memory to one of recognizing visual patterns. To accomplish this transformation, we borrow from the concept of stripcharts.

A strip chart is a simple XY graph used to plot scalar data in on-line visualizations. The scalar values are mapped to the Y axis and narrative time to the X axis. The data points are scrolled along the X axis such that the most recent data is closest to the Y axis. Temporal scaling is achieved by varying the rate at which the data is scrolled along the X axis. Common examples of strip charts are the output medium of seismographs and traditional lie detectors.

The nature of strip charts is that they make it easy to perceive patterns involving a small number of time dependent variables (subsequently referred to as *channels*, a term borrowed from keyframed animation). Accordingly, variations on stripcharts have been used in program execution visualization systems such as the Hermes debugger [1] and the Animation Choreographer [2].

However, as the number of channels increases the effectiveness of strip chart oriented displays diminishes. It is harder to compare two items when there is intervening space and distractors between them, and these two quantities are a function of the number of channels.

We assume that the data to be visualized represents relationships between a relatively small number of objects. The basis of our approach is to use a 'nodes and edges' style graph to represent the objects and relationships respectively. The edges of the graph become pairs of quasi-strip charts which we subsequently refer to as edge graphs. As an example, Figure 1 depicts three objects and their relationships. The edge graph for the relationship on directed edge AB is drawn along one side of the edge AB, while the relationship on directed edge BA is drawn along the opposite side.

ENHANCEMENTS TO THE BASIC APPROACH

Several improvements are possible to enhance the visualization in the following areas:

- the space problem is aided by the use of anti-matter mode and grouping; and

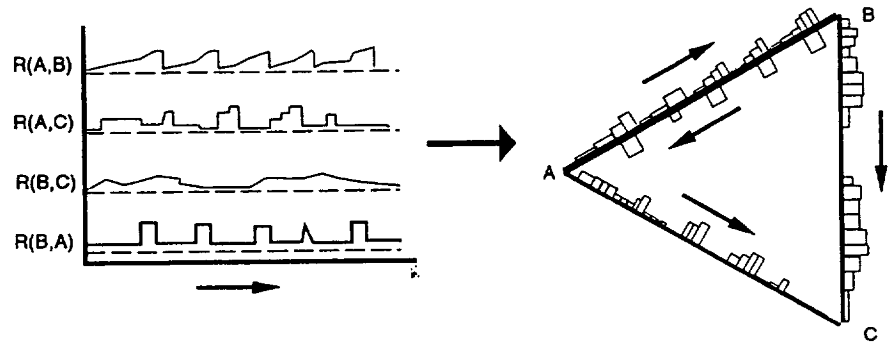


Figure 1: Transformation of stripcharts to edge graphs.

- the online perceptual problem, such as the presence of distractors, is solved by grouping, color-tagging, and a replay facility.

Anti-matter Mode

A drawback to the basic representation of edge graphs is a difficulty in tracking possible causal relationships. The historical information displayed by the edge graph is not necessarily positioned to correlated easily with information from other edge graphs.

As an example, consider a system of three objects (A, B and C), edges AB and BC, and a relationship between edges such that an event on AB is followed by an event on BC[2]. When the block representing an event on AB is close to A, the block representing an event on BC is close to B, as shown in Figure 2. If an observer notices the event on BC, the observer may not associate it with the event on AB because the first event is outside of the spatial locality of the second event.

To solve the locality problem (for some cases) the visualizer has an 'anti-matter' option. With the option enabled, each edge is split into two edge graphs; the data proceed from the first object in the relation to the midpoint of the edge, and mirror images of the data proceed simultaneously from the second object to the midpoint (in the opposite direction). To distinguish between the two edge graphs, the first is drawn with green blocks while the mirror is drawn with purple blocks. The intent is that when the observer sees new activity on BC, the green block will be near B, and a purple (anti-matter) block on AB will also be near B and hence the observer will more easily notice related activity has occurred. Figure 3 shows the previous example with anti-matter option enabled.

Grouping

Eventually, as the number of objects increases the amount of screen real-estate is not enough to display everything, no matter how efficiently the screen real-estate is used. Using a larger screen helps to an extent, but only so much detail can be handled at once by an observer, particularly when the data is animated.

[2]Assume no random events on BC.

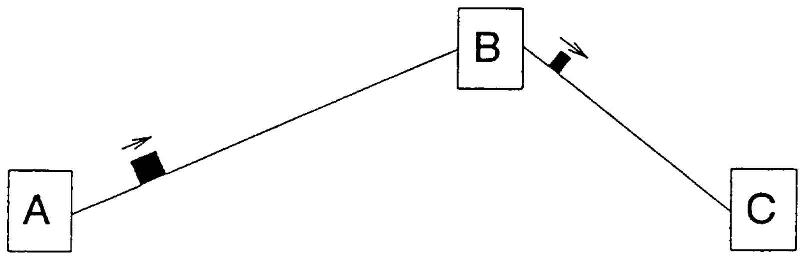


Figure 2: A client/server relay sequence **without** the anti-matter option enabled.

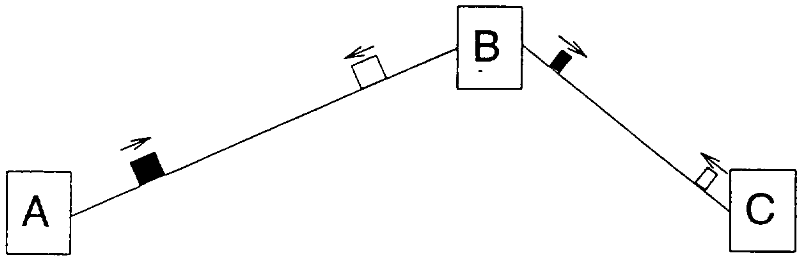


Figure 3: A client/server relay sequence **with** the anti-matter option enabled.

Rather than employing a 'region of interest' approach, such as fisheye views [3] or the perspective wall [4][3], TNV uses two different schemes to deal with excessive detail. A visibility toggle allows a user to hide an object and all the connecting edges. An ellipsing mechanism supplies a drag-and-drop interface to encapsulate objects into groups. Both schemes can operate while the visualization is in progress with the use of a direct manipulation view called the 'grouping' window which contains icons of all the objects. This is illustrated in Figure 4.

Color-tagging

Color-tagging is a causality-detection strategy which takes advantage of the human capacity for pre-attentive processing. Pre-attentive processing is that portion of visual processing that we are able to do in parallel, processing that takes a constant time to perform, independent of scene complexity[4].

Our approach is to highlight significant events on edges in order to attract a viewer's attention. Our chosen highlighting method is to color the events a radically different hue, since color is one of the visual attributes that we are able to process pre-attentively.

Currently we have three detection mechanisms in place. Each mechanism unifies the color of two events if one event falls within a temporal window relative to the other.

Reply-detection: tags a pair of events if they travel along the same edge but in opposite directions (shown in Figure 5). Reply-detection is useful for spotting symmetrically positioned events (i.e., message/acknowledgement sequences in ethernet traffic).

[3]Fisheye views and perspective wall approachs achieve space contraction by distorting the basic geometry of the display. Distorting the edge graphs compromises the accuracy of the information displayed.

[4]Additional background on pre-attentive processing can be found in [5].

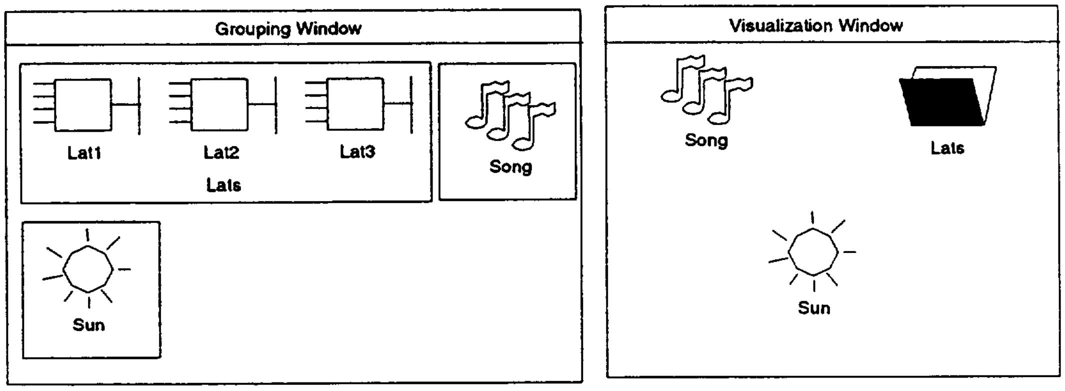


Figure 4: Icons as they appear in the grouping window (left), and in the window containing the visualization (right).

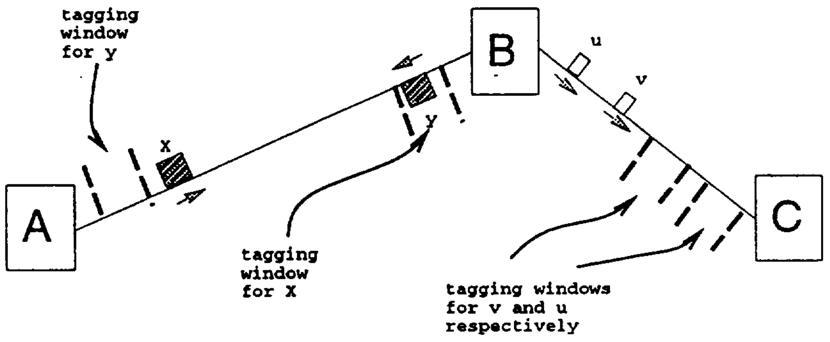


Figure 5: Color-tagging of symmetrically positioned events. Blocks x and y are highlighted.

Relay-detection: tags a pair of events if one is travelling to a the same node the other is leaving from (shown in Figure 6). Relay-detection eases discovery of transitive chains of events (such as trickle down effects in economic data).

Hidden-cause/effect-detection: tags pairs of events if they travel on edges that do not share any common objects. It is useful in picking out relationships which are not explicitly revealed by the topology of the data.

Time Control

A television set is an example of a strictly real time device. Lacking any image buffering capability, a television set can only monitor a video signal with no (humanly) perceivable delay. However, the popularity of the video cassette recorder in the consumer market is an indication of the desirability of occassionally loosening real-time constraints.

Most of the visualizer's time control interface (shown in Figure 7) is an imitation of the panel of a VCR. Additional replay features are possible since we are playing from a random access storage device (memory) instead of a serial access device (tape). The additional features are:

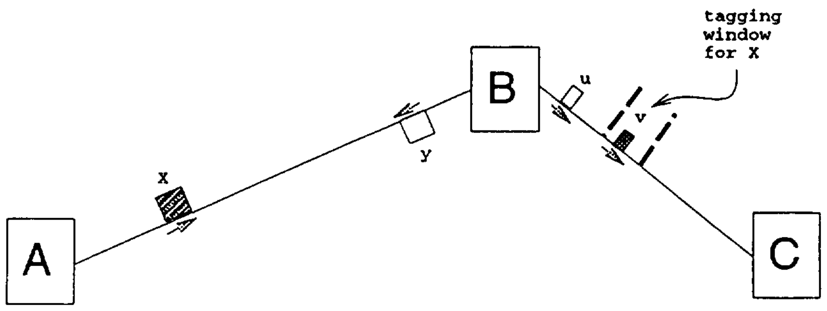
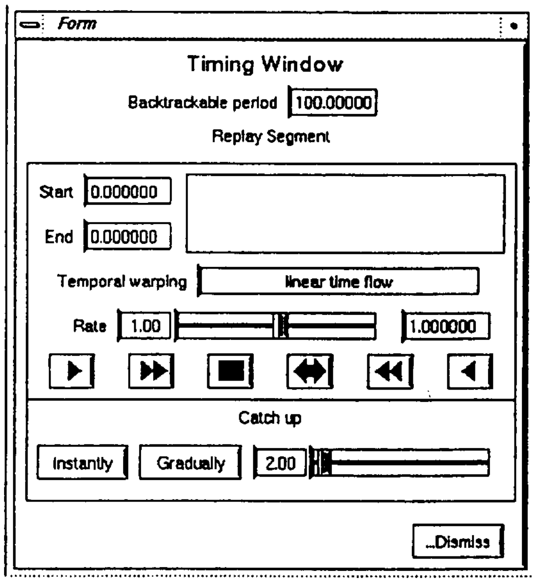


Figure 6: Color tagging of transitive chains of events. Blocks x and v are highlighted.



Backtrackable period 100.00000

Start 0.000000

End 0.000000

Temporal warping: linear time flow

Rate 1.00 1.000000

Catch up 2.00

Figure 7: Interface for time control in the visualizer.

- repeatedly replay a segment of time until stopped;
- the same as above, but in reverse time;
- oscillate between two points in time; and
- a continuous range of playback rates.

For a visualization system, the ability to replay in reverse time is a strong feature. Playing in reverse time flips the temporal order of cause and effect chains. As an analogy, imagine a room of people throwing balls against walls, one of which eventually knocks over a vase. Watched in reverse time, one can see the vase 'get up' to meet the colliding ball, which then bounces off several walls to return to the hands of the person who originally threw the ball. Playing in reverse time eliminates the dependency on the observer's memory to assemble transitive cause and effect chains. extraneous detail.

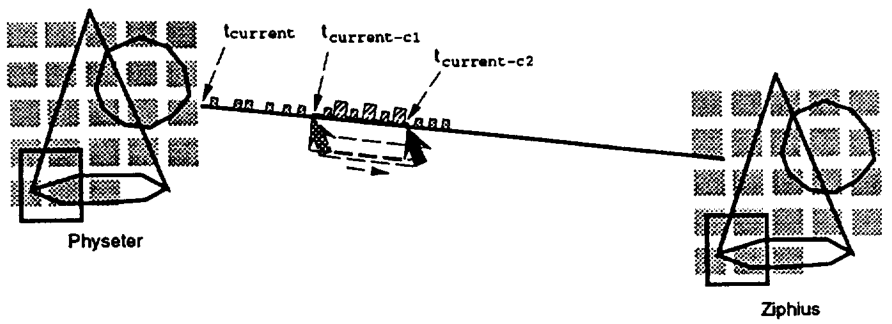


Figure 8: Selecting a region of time by dragging the cursor along an edge.

Unlike the VCR, TNV has a convenient mechanism to select points in time for replay. The mechanism exploits the mapping between a point on an edge graph and a point in narrative time.

The point closest to the source object represents the current time while points further away represent time increasingly further into the past. For example, in Figure 8, when observer notices an interesting pattern on an edge, the observer can drag the cursor along the portion of the edge containing the pattern to specify start and end times, and then press a replay button.

After a section of time has replayed, there remains the issue of returning to the current time. In other words, how is the real time constraint re-tighten? A VCR equipped TV uses a discontinous time technique. When the stop button on the VCR is pressed, the VCR passes through (to the TV) the present-time external video signal. Under this model, the observer misses activity that occurred during the replay.

Compact disc (CD) players however, offer an alternative. When manually searching for a particular song, some CD players allow a listener to play material at faster rates until the desired section is located[5]. TNV offers the options to catch up immediately (VCR style) and gradually (CD style). The gradual catch up preserves the temporal context that is lost under the dis-continuous technique.

APPLICATIONS IN THE REAL WORLD

The expression, "No man is an island", becomes truer each day as the twin forces of technology and capitalism accelerate us towards increasing levels of connectivity. The internet, long used by the academic community to exchange information, is beginning to do the same for individuals and businessess. Similarly, free trade agreements are increasing commerce across borders. Both the internet (under the name of 'the Information Superhighway') and economic trade (national and international) are items of great interest today to individuals, businesses, and governments. As they continue to grow, it will be increasingly important to monitor the ongoing activity in each to catch early developments of problems like the internet worm or the stock market crash of 1987.

Effectively monitoring either of these will require controlling the display of vast amounts of information, maintaining a real-time display, and enhancing the observer's ability to pick out causal relationships. While our system is too limited in its processing speed and capacity to handle either

[5]In order to keep the pitch from being affected as it is when you play a record at the wrong speed, compact disc players actually present short samples taken from non-contiguous regions of the track.

the entire internet or a global economy (the data collection alone would be a major technical feat in either case), it does support the concepts to enable this on a smaller scale. As examples, the following subsections discuss how we have used TNV to monitor traffic on a local area subnet and to visualize an economic simulation.

Monitoring A Network

One difficulty of network visualizations that employ motion, as TNV does, is the preconceived metaphors of the observers. The intuitive interpretation of edge graphs is that each block traveling along an edge represents an ethernet packet traveling from one device to another. Such a display is not possible with a real-time constraint because the time scale is far faster than current monitors can handle, let alone the human visual system[6].

TNV visualizes not the actual packets but the **volume** of packets from one machine to another. To do this we uses two programs in addition to TNV:

Snoop, a vendor supplied packet capture program on SUN's[7] to supply the raw data;

ProcSnoopCnt or **ProcSnoopSize**, filter programs written in C which summarize the raw data provided by Snoop. The programs supply a count of the number of packets and the amount of actual data in bytes (respectively) passed between each pair of devices on the network, sampled over one second intervals.

The data is shipped over the network to an SGI workstation running TNV.

This strategy has trade-offs; sending the data from one machine to another induces additional traffic on the network (which is visible in the visualization), but it also allows the visualizing system to run remotely from the network being monitored[8].

The Computer Graphics Lab of the University of Waterloo has a hetereogenous set of computing devices attached to its local subnet. Devices range from ancient multiprocessor systems to Sun Sparcstations, DEC Alphas, and an SGI Onyx. Fortunately, not all of these machines are particularly active.

As Figure 9 shows, even without the edge graphs and the spiral graphs, the layout is crowded. To conserve enough window real-estate for visualizing the traffic between the devices, we use the grouping mechanism:

- twelve older workstations (represented by rocking chair icons) are grouped into a single entity labeled 'old stuff';

- three harmony workstations (represented by the musical notes icons) into 'harmony';

[6]Modern machines are more than fast enough for most communication exchanges to be sent and replied to in the period of a single vertical retrace of a conventional CRT display.

[7]SUN and Sparcstation are trademarks of Sun Microsystems. DEC and Alpha are trademarks of Digital Equipment Corporation. SGI and Onyx are trademarks of Silicon Graphics.

[8]This was done while demonstrating TNV at CASCON '94. The visualization system was run on a local machine in Toronto, Ont, while Snoop was capturing packets on a subnet in Waterloo, Ont. The data was shipped across the internet to Toronto to provide a visualization of live data.

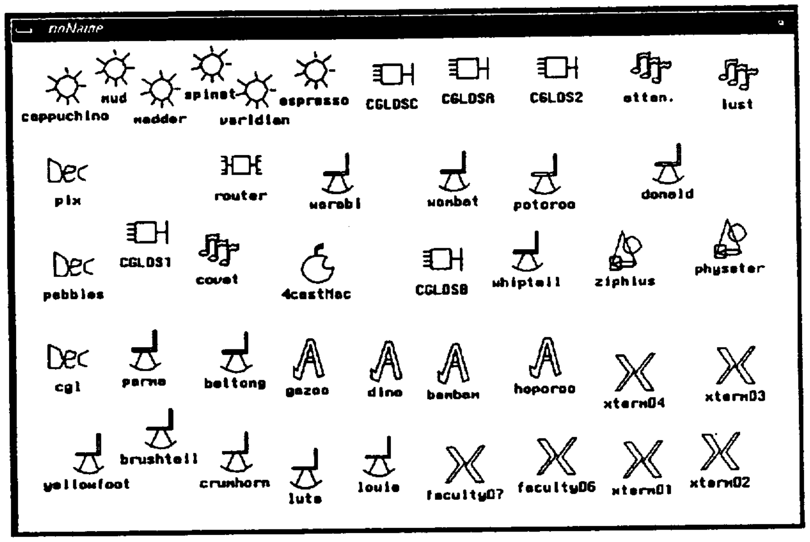


Figure 9: The CGL local subnet, before edges are added.

- six sun workstations (represented by shining sun icons) into 'suns';
- five DEC LAT server ports (CGLDSX) into 'lats'; and
- six X-windows terminals (represented by the X logo) into 'xterms',

resulting in the layout shown in Figure 10.

Enabling the anti-matter option has an unexpected benefit in network visualization; the ability to discriminate between point-to-point and broadcast communications. Point-to-point communications are usually intended to be reliable and hence for every original packet put onto the network by device A for device B, there must be a packet placed on the network containing an acknowledgement from device B for device A. Because the acknowledgement must follow the original message with very little delay[9] the original message and its reply occur simultaneously (in a human time scale).

In terms of the TNV display, a group of packets from device A to device B causes a green block to be drawn on one side of the edge AB and close to A while the replies to the packets cause a green block to be drawn on the opposite side and close to B. The anti-matter option adds two additional purple blocks to mirror the message block and reply block. The mirror of the message block is aligned with the reply block while the mirror of the reply block is aligned with the message block, as shown in Figure 11.

While most communication on an ethernet is point-to-point, some devices broadcast. Visualization of the CGL network resulted in the observation that several terminal servers periodically broadcast packets[10].

The real-time nature of network visualization is important because often the observer is seeking to correlate the activity shown in the visualization with actions and behaviors occurring in the real

[9]If the acknowledgement is not received in a sufficiently short interval, a repeat of the message is sent under the assumption that the original message is lost.

[10]Examination of the contents of actual messages with the Snoop tool revealed that the LAT devices broadcast a list of the services they provide.

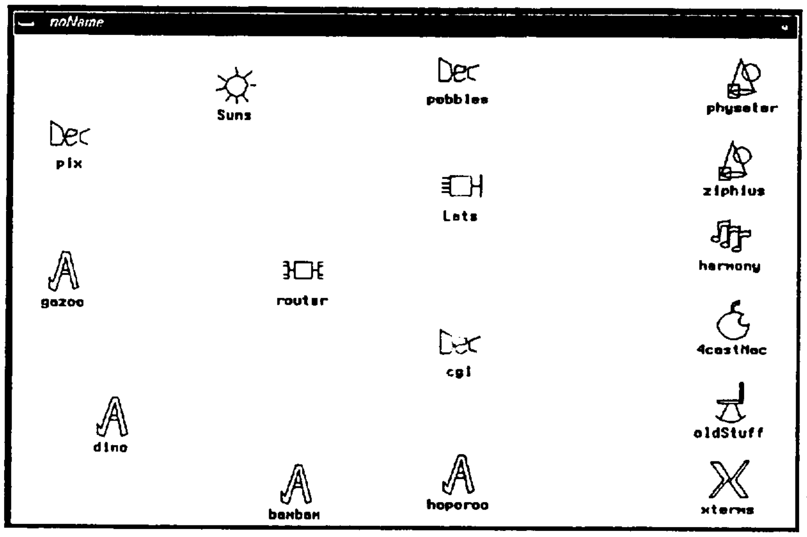


Figure 10: The CGL local subnet, after machines are grouped.

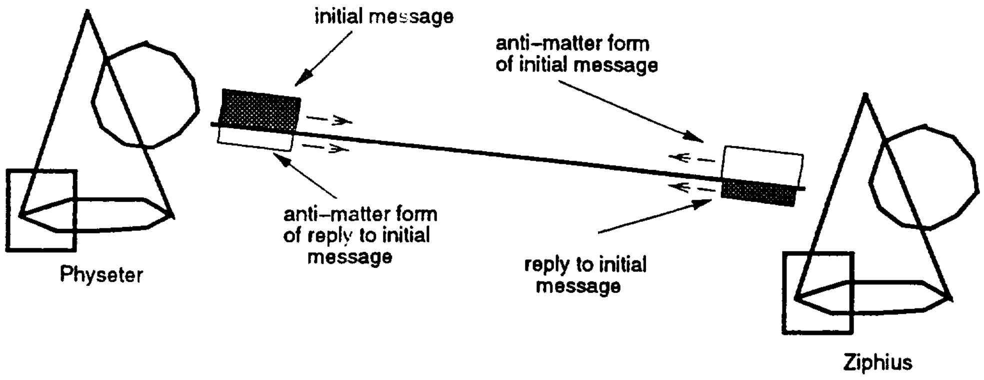

initial message
anti–matter form of initial message
anti–matter form of reply to initial message
reply to initial message
Physeter
Ziphius

Figure 11: The appearance of acknowledged packets in the network visualization.

world (not reflected in the visualization). A short time lag between an event and its visualization is unavoidable since even the fastest machine requires time to collect and process data. However, a system in which the lag stretchs or shrinks is a problem because it dynamically changes the apparent relationships we are trying to discover. In the network visualization, the TNV display lags clock time by exactly four seconds, ensuring that data is collected, processed, and in memory before the TNV actually needs it for the display.

To handle temporary data drop-outs (i.e. when a lag time of four seconds is insufficient due to network congestion), portions of the edges are drawn with dashed lines to indicate that data has not been collected for portions of the represented time period. Without this feature, it is possible to have the rather ironic (and misleading) effect of traffic levels appearing to be zero when the levels are actually high enough to prevent the transfer of data from Snoop.

Visualizing An Economic Simulation

The original purpose of the system was to visualize financial data. However, for the purposes of testing the system, we produced a simple economic simulation to generate fake data. Simulated data has several benefits over real data:

- the data represents a closed system, which is hard to find in the real economic world;
- the level of noise in the data is controllable;
- since the simulation is fairly simple and has no unknown quantities, (unlike the real world) the relationships are predictable.

Our simulation represents 3 countries, each of which contains: four producers which employ labor, pay taxes, sell products and in some cases buy products from other producers; one source of labor/consumers which buys products and pays taxes; and a government which buys products and employs labor.

The simulation is a simplistic model of reality since the number of producers, the spending habits of the consumers, and the price setting scheme of the producers is unrealistic. However, it does model the 'trickle-down' effect; when one labor/consumer group has an influx of capital, it spends more, resulting in an outflux of capital to governments and producers, who in turn spend more.

TNV is supplied with two sets of values: the amount of capital held by each party, and the amount of capital exchanged between each pair of parties. The amount of capital is represented by spiral graphs (a variation of a bar graph). Positive values are represented by green boxes, while negative values (debt in the case of the economic visualization) are drawn in red. The exchange of capital between two parties is shown by the edge graphs connecting them.

Other Applications

A very suitable application for TNV would be monitoring telephone exchanges systems. Objects could either represent individual lines or entire exchanges. The ability to reverse time could be used to infer where calls were originating from, without the use of caller-id.

Monitoring electric power grids would also be a possibility. Power grids operate on two different time scales; one corresponding to normal use, and one corresponding to breakdowns. In normal use, large changes in power consumption would be on the scale of minutes to hours, tied to uses of business and consumer products such as lighting, heating, air conditioners and industrial processes. On the other hand, breakdowns operate in a scale of seconds.

EMPIRICAL TESTING

Evaluating how well a set of subjects can discern useful information from real-world network data initially seems like an obvious and simple way to test the quality of our visualization. Unfortunately, extracting such network information from a visualization is a complex task which incorporates several more primitive subtasks. The choice of the subtasks is dependent on the personal strategies developed by the subjects.

In appreciation of this, our testing philosphy is to monitor how well users perform each primitive subtask involved. The philosphy is based on the assumptions that facilitating the individual subtasks similarly facilitates the whole task, and that the subtasks are less prone to the effects of personal strategies. Currently, we have preliminary results for the subtask of recognizing the edge in the graph that has intentionally symmetric behavior [11]. We will subsequently refer to an edge with symmetric behavior as the *signal* edge, while to others will be referred to as *noise* edges.

To keep the length of the tests manageable, each type of display (anti-matter, color-tagged, color-tagged anti-matter, and a stripchart) were tested with a complete graph on five objects arranged on the perimeter of a circle. The signal edge generated a pair of events (one in each direction) every half second with a probability 0.15. The noise edges generated the same level of traffic, but the likelihood of false positives (randomly paired events) on a particular edge was 0.15^2 or 0.0225 in any half second interval. Four subjects were asked to identify the objects at either end of the signal edge via a numeric keypad after a fixed interval of watching the display.

Figure 13 shows that the color-tagged display and the color-tagged anti-matter display allowed the most accurate selections, and were the least degraded by shorter inspection intervals. This is to be expected, since using color as a tag allows greater parallel processing/filtering by observers.

The stripchart display, used as a control group, consisted of five subgraphs, each with four stripcharts. The top surface of each stripchart represented traffic in one direction, while the bottom surface represented traffic in the other direction. The display in Figure 12 features a graph in which the non-random edge is between objects 1 and 4. The anti-matter display and the stripchart display were not significantly different which is not surprising since the anti-matter display is essentially a distorted version of the strip chart.

Sadly, few real world applications consist of only five objects and 10 edges. The low level of complexity would not be useful in most cases, even with the grouping mechanism. Accordingly, additional tests, increasing the number of objects, were done to see how the color-tagging with anti-matter scheme scaled. The results are shown in Figure 13. In the first test, the complete graph on five objects was replaced by a complete graph on eight objects (28 edges). Subject performance

[11]This is the type of behavior observed on an ethernet between sites that are engaged in point-to-point communication.

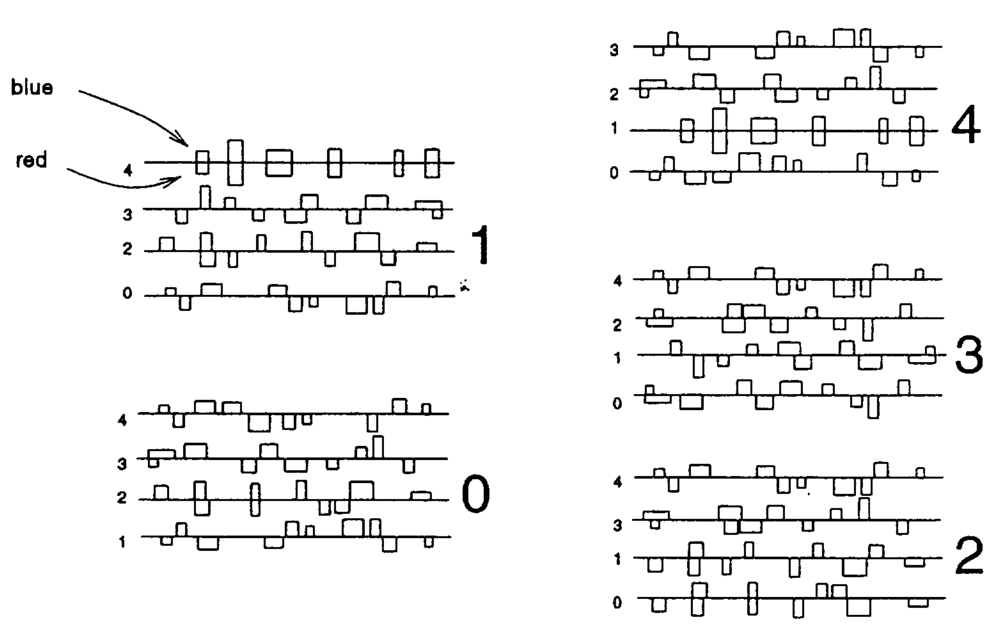


Figure 12: Example of the stripchart display. The signal edge is between objects 1 and 4.

dropped to the level of the stripchart, possibly resulting from the 3 fold increase in the number of false positives[12], requiring additional serial processing.

If the drop in performance was purely a matter of the number of edges, increasing the number of objects without increasing the number of edges would have no visible effect. This conjecture was tested by giving subjects a non-complete graph with 15 objects and 28 edges. Performance dropped, but not nearly to the extent that it did with the increase from 5 to 8 objects.

The preceding is an example of the experiments we are currently conducting. It is our intention that each feature we implement is empirically tested for the effect on subjects' abilities to extract information from the display.

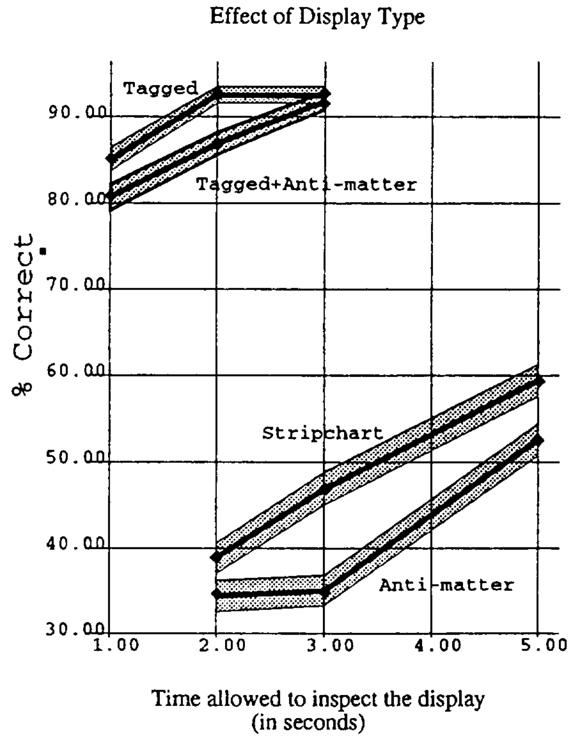
CONCLUSIONS AND FURTHER WORK

This paper has several goals: to convince the reader that enhancing a user's ability to perceive causality is a worthwhile pursuit; to explain how conventional visualization may fall short, and to explore better ways of visualizing causality.

Our user testing is still in its beginning stages. However, with the data collected from it, we will be able to refine our current set of tests and quantify the effectiveness of our tool in helping observers perceive other causal relationships than the simple symmetric case.

Acknowledgements

[12]The number of random edges increased from 9 to 27, while the probability of a false positive on a random edge remained the same. In terms of average signal-to-noise ratios, increasing the number of edges dropped the S/N ratio from 0.83 to 0.26.



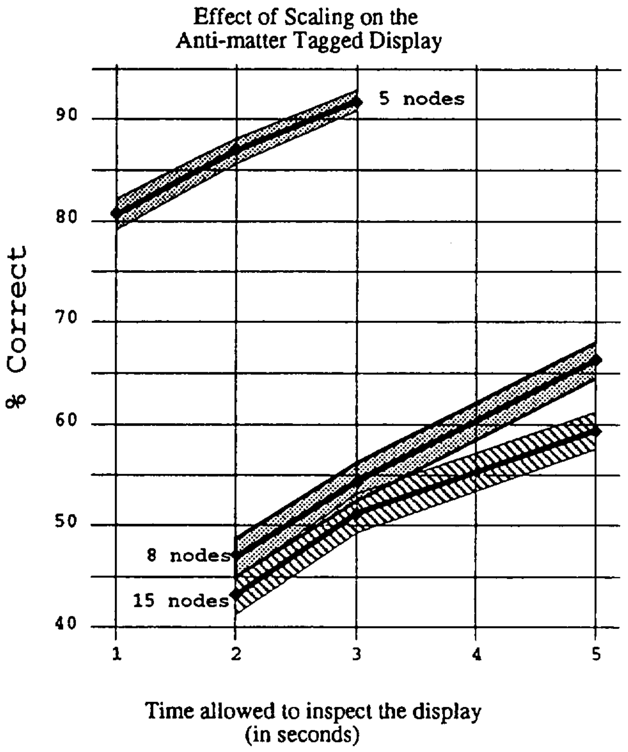


Figure 13: User testing results for different display types and scaling. The shaded areas represent a 99% confidence interval for the value of the proportion estimator.

Many thanks are owed to members of the Computer Graphics Lab at the University of Waterloo, and other students there who participated in our experiments. Additional thanks are due to Karen Sleumer for her tireless proof-reading of this paper. This research was supported by grants from NSERC and ITRC.

References

[1] David J. Taylor. A prototype debugger for hermes. In *Proceedings of the 1992 CAS Conference*, volume 1, 1992.

[2] Eileen Kraemer and John T. Stasko. Toward flexible control of the temporal mapping from concurrent program events to animations. Technical report, Graphics, Visualization, and Usability Center, Georgia Institute of Technology, 1994.

[3] George W. Furnas. Generalized fisheye views. In *CHI'86 Proceedings*, pages 16–23, April 1986.

[4] Jock D. Mackinlay, G.G. Robertson, and S.K. Card. The perspective wall: Detail and context smoothly integrated. In *CHI'91 Conference on Human Factors in Computing Systems*, pages 173-179, April 1991.

[5] Cathryn J. Downing. Expectancy and visual-spatial attention: Effects on perceptual quality. *Journal of Experimental Psychology: Human Perception and Performance*, 14(2):188–202, 1988.

Unsteady Visualization of Grand Challenge Size CFD Problems: Traditional Post-Processing vs. Co-Processing

Robert Haimes
Massachusetts Institute of Technology
Cambridge, MA
haimes@orville.mit.edu

ABSTRACT

The traditional steps taken in the analysis of steady-state Computational Fluid Dynamics results break down when dealing with time-varying problems. A single transient application can require enormous resources that can stress any computer center. The common approach that most individuals take is a form of sub-sampling (usually in time) with regard to disk requirements and not to the physical problem. This can lead to erroneous results when viewed and give the investigator (or worse – the designer) an incorrect impression of the unsteady flow regime.

This paper discusses two available visualization packages that can perform co-processing. Through their architecture, and some coding, close coupling of a CFD solver with the visualization allows for concurrent execution of both. The pros and cons of each system are exposed. And finally, a hybrid post-processing / co-processing scheme is discussed that can, in a production environment, resolve the resource / accuracy issues.

TRADITIONAL COMPUTATIONAL ANALYSIS

The computational steps traditional used for Computational Fluid Dynamics (CFD) analysis (or when CFD is used in design) are:

- Grid Generation

  The surfaces of the object(s) to be analyzed (usually derived from a CAD system) specify part of the domain of interest. Usually for the analysis of external aerodynamics, the airplane is surrounded by other surfaces that extend many body lengths away from the craft. This enclosed volume is then discretized (subdivided) in one of three ways. Unstructured meshes are built by having the subdivisions be comprised of tetrahedral elements. Another technique breaks up the domain into sub-domains that are hexahedral. These sub-domains are further subdivided in a regular manner so that individual cells in the block can be indexed via an integer triad. The structured block schemes may have the blocks abut or overlap. Finally, some systems use meshes that are not body fit. In this class of grid generation, the surfaces of the object intersect a hierarchical (embedded) Cartesian mesh. For all of these methods, the number of nodes produced for complex geometries can be on the order of millions. See Aftosmis[1] for a fine treatise on the state of the art in grid generation.

- Flow Solver

  The flow solver takes as input the grid generated by the first step. Because of the different styles of gridding, the solver is usually written with ability to use only one of the volume discretization methods. In fact there are no standard file types, so most solvers are written in close cooperation with the grid generator. The flow solver usually integrates either the Euler or Navier-Stokes equations (these are hyperbolic partial differential equations) in an iterative manner, storing the results either at the nodes in the mesh or in the element centers. The output of the solver is usually a file that contains the solution. Again there are no file standards. PLOT3D[2] files can be used by grid generators and solvers that use structured blocks and store the solution at the nodes. In practice, this file type is not used by the solver writer because it contains no information on what to do at the block interfaces (the boundary conditions to be applied). Therefore, even this subclass of solvers writes the solution to disk in a manner that only the solver can use (to restart) and this file is converted to a PLOT3D file for post-processing.

- Post-processing Visualization

  After the solution procedure is complete, the output from the grid generator and flow solver is displayed and examined in a graphical manner by the third step in this process. Usually workstation-class equipment with 3D graphics adapters are used to quickly render the output from data extraction techniques. The tools (such as iso-surfacing, geometric cuts and streamlines) allow the examination of the volumetric data produced by the solver. Even for steady-state solutions, much time is usually required to scan, poke and probe the data in order to understand the structure of the flow field. In general, most visualization packages read and display PLOT3D data files as well as a limited suite of output from specific packages. There are some systems that can deal naturally with unstructured meshes and few that can handle hybrid structured/unstructured solutions. No general package can, at this point, deal with the output from Cartesian systems.

The 3-step process (described above) has been the view of CFD analysis for years as individuals and teams have worked on each component. This work has had the assumption that a steady-state problem is being tackled. Therefore, this process works well for steady-state solutions. But, when it is applied to a transient system there are two crucial steps that get easily overwhelmed. First, if the grid changes in time, the grid generation process can take enormous amount of human interaction. This area will not be covered by this paper.

The second limiter to effectively using the traditional CFD analysis steps in transient problems is understanding the results. The production of a movie is the method currently employed. This is accomplished by treating each saved snap-shot of the solution (on disk) as a steady-state visualization task. Image (window) dumps are produced from the same view (or from a fly-by) with the same tools active. These images can be played back a some later time to give the viewer information on what is changing in time. Movies can be a very efficient method for communicating what is going on in the changing flow field, but is ineffective as a vehicle for understanding the flow physics. Rarely are we smart enough to 'story-board' our movie in a way that displays salient flow features without finding them first.

We have learned that a single view or picture is not enough to understand complex flow features; why should a single snap-shot in a movie convey any more?

PROBLEMS WITH POST-PROCESSING

If we follow the traditional computational analysis steps for CFD (and assume the simple case that the grid is not changing in time) and we wish to construct an interactive visualizer we need to be aware of the following:

- Disk space requirements

  A single snap-shot must contain at least the values (primitive variables) stored at the appropriate locations within the mesh. For most simple 3D Euler solvers that means 5 floating point words. Navier-Stokes solutions with turbulence models may contain 7 state-variables. The number can increase with the modeling of multi-phase flows, chemistry and/or electomagnetic systems. If we examine a 5 equation system with 1 million nodes (with the field variables stored at the nodes) a single snap-shot will require 20 megabytes. If 1000 time-steps are needed for the simulation (and the grid is not moving), 20 gigabytes are required to record the entire simulation. This means that the workstation performing the visualization of this simulation requires vast amounts of disk space. The disk space should be local because access to this much data over a distributed file system can also be a limitation.

- Disk speed vs. computational speeds

  The time required to read the complete solution of a saved time frame from disk is now longer than the compute time for a set number of iterations from an explicit solver. Depending on the hardware and solver an iteration of an implicit code may also take less time than reading the solution from disk. If one examines the performance improvements in the last decade or two, it is easy to see that depending on disk performance (vs. CPU improvement) may be a bad 'bet' for enhancing interactivity. Workstation performance continues to double every 18 months. The price per megabyte of disk drives has dropped at amazing rates, but the performance of commodity drives has only gone from about 1 megabyte/sec in 1985 to about 5 megabytes/sec in 1995. To augment disk performance, technologies may be applied like disk striping, RAID, or systems like Thinking Machine's DataVault, which all depend on using multiple disks and controllers functioning in parallel. But most workstations currently have SCSI interfaces that limit data transfer rates to about 5 megabytes/sec (SCSI II) per chain. High performance workstations that have other methods may only be able to move 20 megabytes/sec through an I/O channel. Therefore, to post-process on a normal workstation, it may take 4 seconds per iteration, just to read the solution for the above example.

- Cluster and Parallel Machine I/O problems

  Disk access time is much worse within current parallel machines and clusters of workstations that are acting in concert to solve a single problem. In this case we are not trying to read the volume of data, but are running the solver to write it out. I/O is the bottleneck for a parallel machine with a front-end. The machine probably has the ability to compute in the gigaFLOP range (being able to generate much data very quickly) but all this data has to be funneled to a single machine and put on disk by that machine. Clusters of workstations usually depend upon distributed file systems. In this case the disk access time is usually not the bottleneck, but the network becomes the pacing hardware. An IBM SP2 is a prime example of the difficulties of writing the solution out every iteration. The machine has a high-speed interconnect, but the distributed file system does not use it. There are other access points into each node. Most SP2s have an Ethernet port for every node and some also have FDDI connections. These traditional network interfaces must be used for the file system. The SP2 can also be used in the traditional front-end paradigm if one of the nodes has a disk farm. In this model, the high-speed interconnect can be used with explicit message passing to the I/O node that does the writing. This obviously requires special code and knowledge of the underlying hardware. In our above example, it would take about 20 seconds to write one time frame (Ethernet hardware – distributed file system). If the machine is dedicated (no other tasks running), then that wastes 20 seconds of cycles.

- Numerics of particle traces

  Most visualization tools can work on a single snap shot of the data but some visualization tools for transient problems require dealing with time. One such tool is the integration of particle paths through a changing vector field. After a careful analysis (numerical stability and accuracy) of integration schemes[3] it has been shown that there exist certain time-step limitations to insure that the path calculated is correct. Even for higher order integration methods, the limitation is on the order of the time step used for the CFD calculation. This is because of a physical limit, the time-scale of the flow. What this means for the visualization system is that in order to get accurate particle traces, the velocity field must be examined close to every time step the solver takes.

Because of the disk space requirements and the time to write the solution to disk, the authors of unsteady flow solvers perform some sort of sub-sampling. This sub-sampling can either be spatial or temporal. Because the traditional approach is to deal with the data as if it were many steady-state solutions, this sub-sampling I/O is almost always temporal. The individual running the simulation determines the frequency to write the complete solution based on the available disk space. In many cases, important transitions are missed. Also since the solution is coarsely sampled in time, streaklines (unsteady particle paths as discussed above) almost always produces erroneous results. The problem with sub-sampling is that the time-step selected for the visualization becomes dependent on the available disk space and not the physical problem.

With the huge storage equipment (and therefore financial) burden on the compute facility it is no wonder that only large national labs routinely visualize results from unsteady problems. We must adopt another visualization paradigm in order to overcome the limitations produced by post-processing.

CO-PROCESSING VISUALIZATION

A solution to the above problems is co-processing. The concurrent solving with interactive visualization can relieve the compute arena of the disk space, speed and sampling limitations. Co-processing does require a complete re-thinking of the architecture of the visualization system and poses the following issues:

- Coupling to the solver

  The solver must communicate with the visualization system. This can be accomplished by one of three methods:

  Disk files: In this approach the solver task communicates with the visualization task by data written to disk. This method is rejected for the disk timing and I/O arguments discussed above.

  Shared memory: In this approach the solver and visualization system communicate via shared memory. This has the advantage that the solver and visualization task are separate and the communication is fast. The disadvantage is that the solver must be written with the shared memory interface. The data that is exposed must be shared in a way that the visualization task knows where and how to get individual data elements. Also, some method is required to mark the data invalid as it is being updated.

  Application Programming Interface (API): This method couples the visualization task (or some portion of the visualization system) with the solver. This coupling is done at the programming level. The advantages to this approach is that all the data is shared (there is only one task), no special operating system level requirements are needed, and it can be used with solvers written in different languages. The challenge is to develop a software architecture that is general, non-invasive, and that allows the visualization system and solver to function independently.

- Additional resources

  Whichever approach is selected, an additional burden is placed on the computer resources. Now both the solver and at least a portion of the visualization system is active on the computer. The visualization system should not place a large encumbrance on either the memory subsystem or require large amounts of CPU cycles.

- Robustness

  The visualization system is running concurrently with the solver. Either it is part of the same task, or has access to crucial data. The visualization must be robust and not interfere with the solver's functioning. A problem with the visualization portion of the system must not crash the solver.

- Interactivity

  It is important for interactivity that the resources required by a co-processing visualization system be a minimum. The cost of CPU cycles and additional memory are the barometer for the frame rate. To accomplish fast frame rates, it becomes a necessity to classify the type of transient problem under consideration. If the visualization system is properly designed, this classification can be used to determine when the results of a tool should be saved and when recalculation is necessary. The following classification is used throughout this paper:

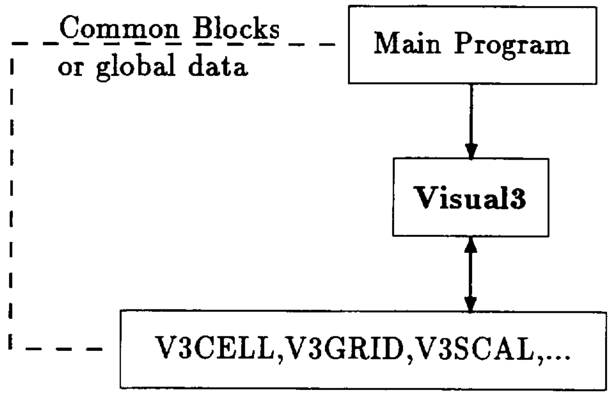


Figure 1: Control diagram of **Visual3**

Data Unsteady: In this type of application the grid structure and position are fixed in time. The data defined at the nodes (both scalar and vector) changes with each time step. An example is when a boundary condition in the domain is changing.

Grid Unsteady: These cases are 'Data Unsteady'; in addition the grid coordinates associated with each node are also allowed to move with each snapshot. An example of this is stator/rotor interaction in turbomachinery. The stator and rotor grids are separate, with the rotor grid sliding past the stator grid. In this case the stator portion is actually 'Data Unsteady' and the rotor grid moves in a radial manner.

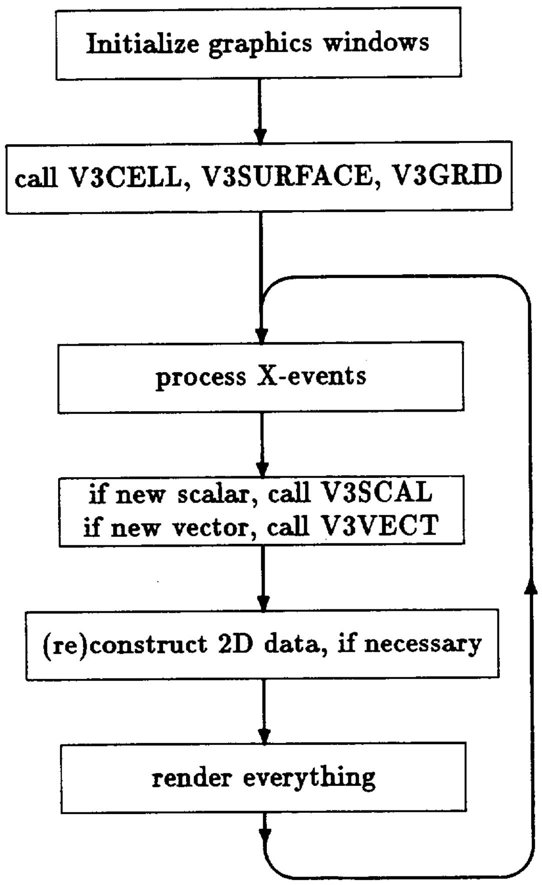
Structure Unsteady: If the number of nodes, number of cells or cell connectivity changes from iteration to iteration, then the case is 'Structure Unsteady'. An example of this mode is store-separation or the solid rockets moving away from the shuttle during lift-off.

CO-PROCESSING USING **Visual3**

The manner in which a solver interacts with **Visual3**[4, 5] is through the API coupling method. It is best understood by first looking at a steady-state post-processing application, then examining how this architecture is changed for an unsteady post-processing application and finally looking at co-processing.

As shown in Fig. 1 the steady-state driver program first initializes and activates **Visual3**. **Visual3**, now in control, gets the data it requires from the driver by calling a small number of programmer-supplied subroutines. These call-back routines are connected to the main program only through COMMON blocks in FORTRAN or through global data in C. The reason for this unusual approach is that it keeps the interface between the driver and **Visual3** as simple as possible, and allows the programmer to plot new scalar functions by simply changing the code which supplies the scalar data.

In the simplest steady-state visualization application, a main routine reads the data from disk and calls V3_INIT. The application also contains the subroutines, V3SURFACE (define the domain surfaces), V3GRID (specify the 3D coordinates for the nodes), V3SCAL (specify the scalar values for the nodes), and if appropriate V3CELL (defines the volume discretization) and V3VECT (specifies the vector values at the nodes), that are called by **Visual3**. Figure 2 shows the flow diagram for the control of **Visual3**. In the initialization phase V3CELL, V3SURFACE and V3GRID are called once to obtain the data structure and grid coordinates, which do not change. After initialization, **Visual3** enters a large animation loop. At the beginning of the loop the first step is to process the list of X-events (keystrokes, dial movements, and cursor movements) that have occurred since the beginning of the previous loop. Processing this list can result in changes in the transformation matrix being used to view the scene, or a change of desired scalar or vector variable, or the invocation



Initialize graphics windows
call V3CELL, V3SURFACE, V3GRID
process X-events
if new scalar, call V3SCAL
if new vector, call V3VECT
(re)construct 2D data, if necessary
render everything

Figure 2: Flow diagram of **Visual3** for steady application

of some new plotting function. The second step is to call V3SCAL if the scalar variable has changed, and V3VECT if the vector variable has changed. The third step is to construct any 2D structures required for cutting surfaces and iso-surfaces. The final step is to perform the rendering, the drawing of the images to all of the graphics windows. The third and fourth steps are usually the most computationally and graphically demanding, and are optimized by keeping careful note of what has changed since the previous loop.

In post-processing unsteady visualization, the program control is similar to that in steady-state visualization. It is assumed that the unsteady data has been pre-computed or needs only a minor level of computational effort. Again the driver program initializes **Visual3** during the transfer of control. Program control is returned to the calling routine only when **Visual3** is terminated by the user.

There are currently two supported types of unsteadiness. At present the 'Structure Unsteady' mode is not allowed. The modifications to the control loop in **Visual3** to handle both 'Data Unsteady' and 'Grid Unsteady' modes are very minor, as shown in Figure 3. Before the data collection phase, the call-back V3UPDATE is executed. This allows the programmer to update the data as appropriate for the problem.

Another change is that V3GRID is called if the case is 'Grid Unsteady' so that new grid coordinates can be obtained. In any case, V3SCAL and V3VECT are always called, even if their definition has not changed, because the values in the field have been updated.

The operating system on SGI workstations, IRIX, supports a parallel processing feature called

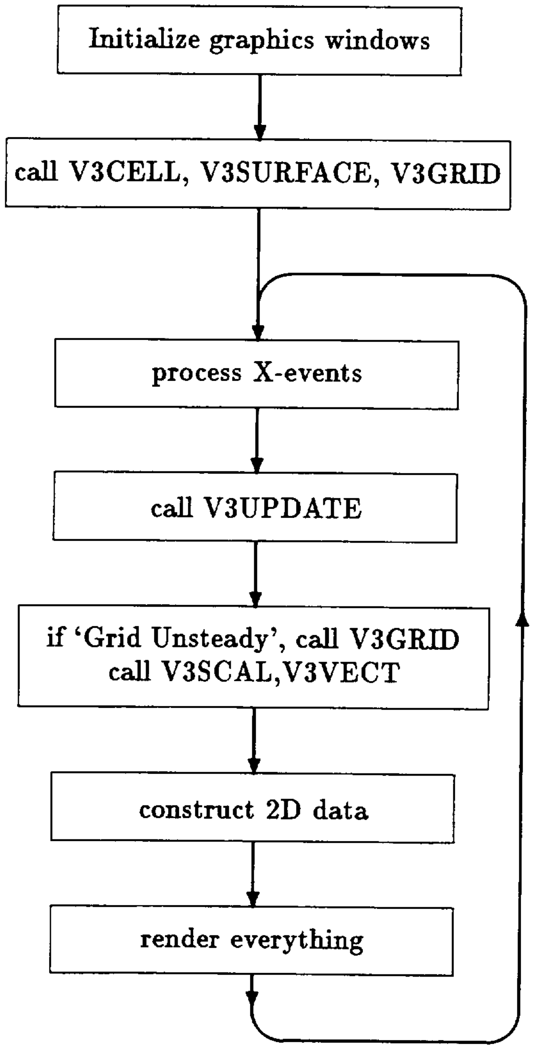

Initialize graphics windows
call V3CELL, V3SURFACE, V3GRID
process X-events
call V3UPDATE
if 'Grid Unsteady', call V3GRID
call V3SCAL,V3VECT
construct 2D data
render everything

Figure 3: Flow diagram of **Visual3** for post-processing unsteady application

'threads'. On machines with more than one processor, threads can run on different CPUs. These Symmetrical Multi-Processor (SMP) computers include PowerSeries, Onyx and PowerOnyx machines. Multi-threading provides a mechanism by which a process can split into two or more parts that run concurrently and share the same address space. **Visual3** can use this mechanism to allow co-processing with a solver. In this mode, when the programmer calls for visualization initialization, **Visual3** generates a new thread which runs the graphics, before returning to the calling program. At this point there are now two (or more) threads running, one (set) executing the solver code, and the other thread executing **Visual3**.

To avoid data conflicts between the two, special locking mechanisms have been set up to insure that the solver does not update its data at the same time that **Visual3** is fetching it by the call-back mechanism. The programmer controls when new data is available to **Visual3** by calls to a hand-shake routine. When the solver needs to update the flow variables, at the end of one timestep for example, a call is made to this hand-shake routine so that **Visual3** can be notified that the data is being updated and is not currently available. If **Visual3** is in the process of accessing the data, then the hand-shake routine waits until the access is completed before seizing control. Once

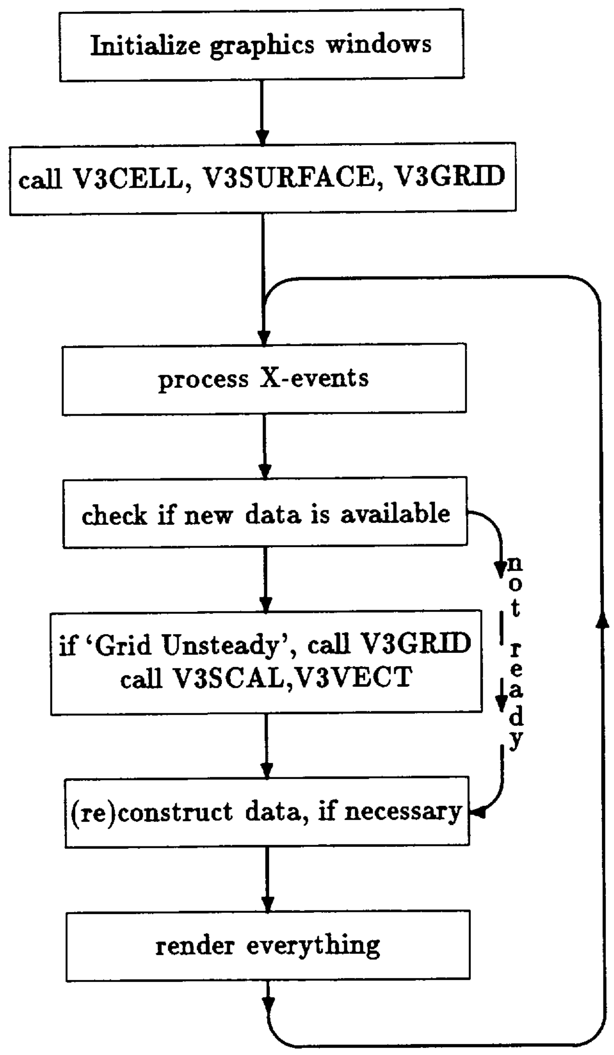

Initialize graphics windows
call V3CELL, V3SURFACE, V3GRID
process X-events
check if new data is available
not ready
if 'Grid Unsteady', call V3GRID
call V3SCAL,V3VECT
(re)construct data, if necessary
render everything

Figure 4: Flow diagram of **Visual3** for co-processing unsteady application

the solver has finished the data update, it again executes the hand-shake routine to notify **Visual3** that the data is available while setting the simulation time. The simulation time is used for particle integrations.

From the point of view of **Visual3**, the modifications to the control loop are shown in Figure 4. After processing the X-event list, and before updating the data, it checks to see if the data is available. If it is not, then **Visual3** jumps to the rendering phase so that interactivity is maintained.

CO-PROCESSING AND **pV3**

A distributed visualization tool-kit that performs co-processing, **pV3**[6], has been developed. **pV3** builds heavily from the technology developed for **Visual3**. This re-tooling was necessary to support the visualization of transient problems without having to dump the entire volume of data to disk every iteration. This system also relieves the equipment-specific requirements of using **Visual3** in the co-processing mode. **pV3** is designed to allow the solver to run on other equipment than the graphics workstation. In fact, the solver can execute in a cluster environment.

pV3 segregates the visualization system into two parts. The first part is the task that actually displays the visualization results onto the graphics workstation, the server. The second portion is a library of routines that allows the solver (solver portions or solvers) to communicate with the visualization server by providing windows to the data at critical times during the simulation. This client library separates all visualization based calculations from the flow solver, so that the solver programmer only needs to feed **pV3** the current simulation data.

pV3 has been designed to minimize network traffic. The client-side library extracts lower dimensional data required by the requested visualization tool from the volume of data in-place. This distilled data is transferred to the graphics workstation. To further reduce the communication burden posed by the visualization session, the transient problem classification described above is used. Only the extracted data that has changed from the last iteration is sent over the network.

An added benefit of this architecture is that most of the visualization compute tasks run in parallel for co-processing in a cluster environment or on a Massively Parallel Processor (MPP). This is because most visualization tools can be cast into operations performed on all the cells within a partition. This means that these tools are embarrassingly parallel. If the partitions are balanced on the basis of cells, then the result of much of the visualization computation is also executed in parallel with some balance. Therefore much better performance for the visualization is achieved.

Each client may be assigned a different class of unsteadiness. This tells the system if data needs to be recalculated and re-transmitted. For example, if a sub-domain is 'Data Unsteady' and a geometric cut is desired (such as a planar cut), only the field data at the nodes (scalar and/or vector) need to be re-transmitted to the server every iteration. The geometry (node locations and mesh data to support the surface) is communicated only when the cut is initiated. If, in the above example, the case is 'Grid Unsteady' all data needs to sent every iteration to the server.

To maximize **pV3**'s effectiveness, client-side libraries exist for most major workstation vendors, MPP's and super-computer manufacturers. Unlike **Visual3**'s co-processing, the computer resources that would usually be used in the execution of the solver can be used. In fact, **pV3** has been designed so that the process of coupling to the solver is simple and noninvasive. The solver's control structure is maintained with the visualization subservient. In the distributed setting, client(s) perform their appropriate calculations for the problem, then as the solution is updated, supplied call-back routines are executed by the visualization library.

The programming model for a typical iterative CFD application can be seen in Figure 5. In general, at the top of each time step boundary conditions are calculated and set, and the solver computes the change to the solution for the entire volume. Then the change to the field is applied.

The coupling is performed by adding two calls to the **pV3** client library in the solver code. The first call is for initialization which informs the visualization system about the following:

Volume discretization – the number of disjoint elements and their type and the number and size of structured blocks.

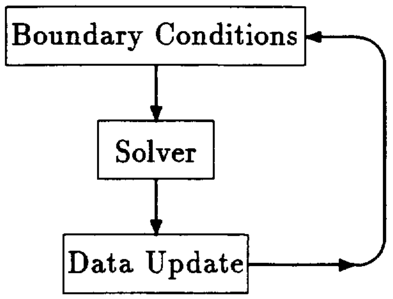


Figure 5: Archetypical Iterative CFD Solver

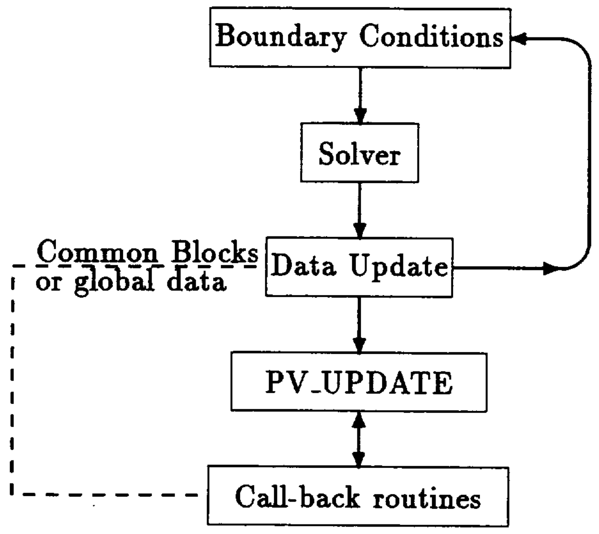


Figure 6: Flow diagram of a **pV3** client

Unsteady mode – 'Steady-state', 'Grid', 'Data' or 'Structure Unsteady'.

Visualization field variables – the number, type, name and limits for each scalar, vector and threshold field.

Programmed cuts – any non-planar geometric cuts defined for this case.

After a successful initialization, the process is entered into the **pV3** client group.

This second call (to PV_UPDATE) informs the visualization system of two things: (1) the simulation time for time-accurate solvers and (2) the solution has been updated and synchronized. This is the mechanism that allows **pV3** to operate, see Figure 6. Without the server running, every time the solution is updated and this call is executed, a check is made for any members in the server group. If none are found, this call returns. When the visualization task starts, it enrolls in the server group, and waits for initialization messages from the clients. The next time the client goes through an iteration, and calls the **pV3** update routine, an initialization message is sent to the server. The server processes all initialization messages, figures out the number of clients and the visualization session begins. Each subsequent time the client goes through the loop and makes this **pV3** call, visualization state messages and tool requests are gathered. Then the appropriate data is calculated, collected and sent to the graphics workstation.

Because PV_UPDATE is where all requests from the graphics workstation are performed, the overall response to state changes and the interactive latency associated with these changes (as well as the next time frame) depend on how frequently this routine is called. About one call per second is optimal. If the solution is updated significantly faster then much of the compute cycles will be used for the visualization, moving the solution slowly forward in time. In this case it is advisable to call PV_UPDATE only every N times the solver updates the solution. When skipping iterations, some analysis of how many iterations can be missed is required, if particle integrations are required.

The more difficult case is when the solution update rate is much slower than optimal. In this situation, there are two choices: (1) live with large lags between user requests and the result of the request seen on the screen or (2) set up another task between the solver and the **pV3** server. This software's responsibility is to communicate with the solver. It should be the task to make all the **pV3** client-side calls. This secondary task must communicate with the solver (and not through disk files) and therefore must be on the same machine to avoid large network transfers. If the machine supports multi-threading, the task can be a second thread and perform double-buffering of the solution space, so no data need be transferred. These methods are a trade-off of computer resources for interactivity.

When the user is finished with the visualization, the server sends a termination message, leaves the server group, and exits. The clients receive the message and clean up any memory allocated for the visualization. The **pV3** client update call reverts to looking for members in the server group.

The visualization system gets the solver's data via call-back routines supplied with the solver. These routines furnish details about the volume discretization, define the surfaces (partition as well as computational boundaries), the node locations and the requested scalar and vector fields. These are virtually the same call-backs used for a **Visual3** driver.

During the design of **pV3**, the state of the solver (client side) was always viewed as the most important component. No error condition, exception or unusual state within the visualization portion of the client effects the solver. To maintain this robustness, no visualization state is stored within the client, all tools are handled via a specific request for data and the response is calculated and sent to the server. If any condition occurs that would cause some problem within the visualization system, the **pV3** server shuts itself down. This allows the simulation to continue, with at most a delay equal to **pV3**'s time-out constant. The **pV3** server can always be restarted to re-connect to the solver at any later time.

This software architecture has the following advantages:

Control – the solver is always in control, and the control mechanisms used during code development can be maintained.

Minimum intrusion – Only two lines need to be added (not including the call-back routines) to the solver code to allow co-processing visualization. Therefore, it can be removed easily for bug isolation.

Steering – The ability to steer the solution is provided via text strings passed from the server to all clients. The action is taken by a programmer supplied call-back that gets the string, interprets the command(s) and performs the requested action.

USING **Visual3**'s ARCHITECTURE

Most of the problems with using **Visual3** for co-processing are that the design point for the code was steady-state or post-processing unsteady visualization. The issues of robustness (for the running solver) were not considered. If the **Visual3** thread aborts, the solver may continue, but the memory allocated for the visualization session does not get freed-up and the visualization session can not be restarted. In this case the thread (and possibly a processor on the SMP machine) will not be used for the simulation. This is generally true, if the visualization has aborted or is functioning. One processor is used, out of the pool, for the visualization (or shared with what is currently running on the machine).

In most cases, where design decisions pitted memory usage vs. a speed trade-off, speed was selected, again because the execution model was not co-processing. That means that if the size of the problem would just fit in the machine without visualization, it would page when the visualization was activated. **pV3** requires much less memory for the same size problem because the trade-offs were more carefully analyzed.

Finally, **Visual3** can be run in the co-processing mode on only a small subset of machines with available ports. Currently this only includes SGI SMP machines with 3D graphics hardware.

PROBLEMS WITH **pV3** CO-PROCESSING

pV3's architecture does pose some difficulties to the code integrator. These involve informing the system on how to patch sub-domains together (for instantaneous streamlines and particle integrations) and in the definition of surfaces of interest (such as wing, body, tail, etc.). Having the programmer define the surfaces allows **pV3** to automatically collect them together for the visualization. The programmer can lessen the sub-domain connectivity burden by trading-off computational efficiency for having **pV3** attempt to continue integrated paths anywhere throughout the domain.

The compute portion of most of the visualization tools is embarrassingly parallel. This gives an advantage in response when the application has been partitioned. The disadvantage is that during the execution of the tools that are serial (such as instantaneous streamlines or particle path integrations) the entire cluster may be stalled. This is a situation that must be minimized. Much effort has been placed in the design of **pV3** to overlap the wait time required for these serial operations to complete with other visualization computation. This involves pulling requests associated with serial tools off the message queue, if they exist, without regard to order. This technique works as long as there are many visualization requests that are not serial. A complete description of the methods used for integrating in partitioned volumes can be seen in Sujudi and Haimes[7].

When the domain has been partitioned and load balanced for the solver, the visualization will upset that balance. There is no attempt within **pV3**'s client-side library to address this problem. This would be contrary to the design goals of minimizing the data movement and leaving the solver (and and the solver's data) alone. Therefore, while the visualization is active, the solver remains load balanced, but the visualization compute and balance is defined by what tools have been activated, the load that generates in each partition, and the action of the serial tools (as discussed above).

The most significant problem with **pV3** in its current form is that it is purely interactive. If someone is not watching the solution as it progresses in time, the information is lost. This is because data is never stored for review. Movies can be made at the server during the visualization session, but this action can slow down the entire solver-visualization concurrent processing.

FEATURE EXTRACTION

In the past, feature extraction and identification were interesting concepts, but not required to understand the underlying physics of a steady 3D problem. This is because the results of the more traditional tools like iso-surfaces, cuts and streamlines were more interactive and easily abstracted so they could be represented to the investigator. These tools worked and properly conveyed the collected information at the expense of much user interaction. For unsteady problems, the investigator does not have the luxury of spending much time scanning only one 'snap-shot' in the simulation. Automated assistance is required in pointing out areas of potential interest in the flow. This must not require a heavy compute burden (for co-processing the visualization should not significantly slow down the solution procedure). Methods must be developed to abstract the feature and display it in a manner that makes sense physically.

Some success has been made in this field. For example, a method that finds the vortical structures has been developed[8]. This is important for flow regimes that are vortex dominated (most of these are unsteady) such as flow over delta wings and flow in turbomachinery. Tracking the core can give insight into controlling unsteady lift and fluctuating loadings due to core/surface interactions. This particular algorithm has been designed so that no serial operations are required, it is embarrassingly parallel, deterministic (with no 'knobs') and the output is minimal. The method operates on a cell at a time in the domain and disjoint lines are created where the core of swirling flow is found. Only these line segments need to be sent to the server to be displayed, distilling the entire vector field to a tiny amount of data.

More feature extraction techniques are required that have the same characteristics: embarrassingly parallel, deterministic and minimal output.

CONCLUSIONS

pV3 is an important tool for solver development and debugging[9, 10]. But as discussed above, the problem with **pV3** in a production environment or for batch execution is that the user may not be around to fire-up the server and view the results. An important addition to the **pV3** suite of software (currently in the design and implementation phase) is a 'batch' server. The client side will remain unchanged. The solver need not know if the results are currently being viewed or to be viewed at some later time.

Therefore, when a 'batch' job starts, the 'batch' **pV3** server is also started. Data is read on where and what tools and probes are to be active and their locations. The results (tool extracts) are collected and written to disk for play-back later. This is different from the normal post-processing in that the entire volume of data is not written to disk every iteration. Features (with the properties described above) can also be extracted and dumped along with the traditional tools.

The end result is something that is not interactive in the placement of tools, but can be thought of as analogous to a wind-tunnel experiment. The investigator must be smart in the placement of probes to extract data of interest. If an important area is missed (or only found after viewing the results) the user will have to re-run the 'tunnel' adding (or changing the location) of the probes. A post-processing viewer is also required to read and display the extracts. This viewer will be highly interactive in dealing with time. This is due to the fact that the amount of data has been reduced many orders-of-magnitude.

This suggests a method for resolving the problems in dealing with transient CFD calculations. The traditional method used of storing the entire solution space every N iterations should be maintained. N may be able to be increased, because co-processing will be used for time critical components. This includes particle integrations, feature extracts, and the result of any other tools desired at a high fidelity in time. An interactive post-processing viewer is required, that can handle a mixture of field quantities and extracts, and is flexible in dealing with simulation time. Time is traversed with the frequent re-rendering of the co-processed extracts while the normal post-processing of the field quantities is refreshed at the time when the data is available. The time required to read the saved solution from disk (or over the network) can be overlapped with the rendering of the high fidelity extracts to give good overall interactivity.

If important transitions are missed, then the solver and **pV3** can be re-executed from the closest saved time-frame to interactively examine the changing flow field.

References

[1] M. J. Aftosmis. Emerging CFD Technologies and Aerospace Vehicle Design. NASA Workshop on Surface Modeling, Grid Generation, and Releated Issues in CFD Solutions, May 1995.

[2] P. Buning and J. L. Steger. Graphics and Flow Visualization in Computational Fluid Dynamics. AIAA Paper 85-1507, 1985.

[3] D. Darmofal and R. Haimes. An Analysis of 3-D Particle Path Integration Algorithms. AIAA Paper 95-1713, June 1995.

[4] M. B. Giles and R. Haimes. Advanced Interactive Visualization for CFD. *Computing Systems in Engineering*, 1(1):51–62, 1990.

[5] R. Haimes and M. Giles. VISUAL3: Interactive Unsteady Unstructured 3D Visualization. AIAA Paper 91-0794, January 1991.

[6] R. Haimes. pV3: A Distributed System for Large-Scale Unsteady CFD Visualization. AIAA Paper 94-0321, January 1994.

[7] D. Sujudi and R. Haimes. Integration of Particle Paths and Streamlines in a Spatially-Decomposed Computation. Proceedings of Parallel CFD '95, June 1995.

[8] D. Sujudi and R. Haimes. Identification of Swirling Flow in 3-D Vector Fields. AIAA Paper 95-1715, June 1995.

[9] P. Crumpton and R. Haimes. Parallel Visualisation of Unstructured Grids. Proceedings of Parallel CFD '95, June 1995.

[10] R. Haimes and T. Barth. Application of the pV3 Co-processing Visualization Environment to 3-D Unstructured Mesh Calculations on the IBM SP2 Parallel Computer. Proceedings of The Computational Aerosciences Workshop 95, NASA Ames Research Center, March 1995.

FLOW VISUALIZATION USING MOVING TEXTURES*

Nelson Max
Lawrence Livermore National Laboratory
Livermore, California

Barry Becker
Lawrence Livermore National Laboratory
Livermore, California

SUMMARY

We present a method for visualizing 2D and 3D flows by animating textures on triangles, taking advantage of texture mapping hardware. We discuss the problems when the flow is time-varying, and present solutions.

INTRODUCTION

An intuitive way to visualize a flow is to watch particles or textures move in the flow. The early color table animation of [1] was an example of this technique. More recently, van Wijk [2] has proposed advecting and motion blurring particles by the flow field. The LIC method [3, 4, 5] uses integrals of white noise textures along streamlines, moving the weighting function in the integrals from frame to frame to animate the texture motion. The motion blur of the particles and the directional texture blurring from the LIC integration create anisotropic textures which indicate the flow even in a still frame. However they are computationally intensive, and cannot generate animation in real time. The textured splats of Crawfis [6] use a loop of cycling texture maps with precomputed advecting motion blurred spots, and take advantage of texture mapping hardware. These are composited in back to front order in a local region near each data point, and oriented in the direction of the velocity vector, so that the precomputed advection cycle indicates the flow.

In this paper, we show how texture mapping hardware can produce near-real-time texture motion,

***This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract number W-7405-ENG-48, with specific support from an internal LDRD grant. We thank Roger Crawfis for helpful suggestions.**

using a polygon grid, and one fixed texture. However, we make no attempt to indicate the flow direction in a still frame. As discussed below, any anisotropic stretching comes from the velocity gradient, not the velocity itself.

The basic idea is to advect the texture by the flow field. In [7] we gave an indication of the wind velocity by advecting the 3D texture coordinates on the polygon vertices of a cloudiness contour surface in a climate simulation. This was slow, because the 3D texture was rendered in software, and because advecting the texture was difficult for time-varying flows. In this paper, we replace the 3D textures by 2D texture maps compatible with hardware rendering, and give techniques for handling time-varying flows more efficiently.

The next section gives our technique for the case of 2D steady flows, and the following one discusses the problems of texture distortion. Then we discuss the problems with extending our method to time-varying flows, and our two solutions. Next we develop compositing methods for visualizing 3D flows. The final section gives our results and conclusions.

TEXTURE ADVECTION FOR STEADY 2D FLOWS

We start with a mathematical definition of texture advection, and then show how it can be approximated by hardware texture-mapped polygon rendering.

Let $F^t(x, y)$ represent the steady flow solution of the differential equation

$$\frac{dF^t(x, y)}{dt} = V\left(F^t(x, y)\right) \tag{1}$$

where $V(x, y)$ is the velocity field being visualized. Thus point P is carried by the flow to the point $F^t(P)$ after a time delay t. The flow F^t satisfies the composition rule

$$F^{s+t}(P) = F^s\left(F^t(P)\right) \tag{2}$$

for both positive and negative s and t. Thus $(F^t)^{-1}(P) = F^{-t}(P)$.

In this paper, we will assume that the initial texture coordinates at $t = 0$ are the same as the (x, y) coordinates of the region R being rendered. In practice, the texture is usually defined in a different (u, v) coordinate system related to (x, y) by translation and scaling, but for simplicity we will ignore the difference.

If $T(x, y)$ is a 2D texture being advected by the flow, then a new texture $T_t(x, y)$ is defined by

$$T_t(x, y) = T\left(\left(F^t\right)^{-1}(x, y)\right) = T\left(F^{-t}(x, y)\right).$$

Thus, to compute T_t at a point P, we go backwards along the streamline through P, to find the point Q such that $F^t(Q) = P$, and then evaluate the texture function T at Q. When animated, this will give the appearance that the initial texture T is being carried along by the flow. By equation (2) above, $F^{-(t+\Delta t)}(P) = F^{-\Delta t}\left(F^{-t}(P)\right)$. Thus the streamlines $F^{-t}(P)$ needed for the texture coordinates can be computed incrementally.

There are two problems with this formulation when the domain of definition for $V(x, y)$ or $T(x, y)$ is limited to a finite region R in which the velocity data or texture is available. First of all, if the streamline $F^{-t}(P)$ leaves the region R, the necessary velocities are not available to continue the integration. One must either extrapolate the known velocities outside R, or continue the streamline as a straight line using the last valid velocity. Fortunately, either of these extrapolation methods will give a correctly moving texture in animation. This is because the visible texture motion at a point P inside R is determined only by the velocity at P, and the extrapolation of the streamline beyond R serves only to determine what texture will be brought in from "off screen".

Second, even if $F^{-t}(P)$ is extended outside R, the texture may not be known there. The standard solution to this is to take $T(x, y)$ to be a periodic function in both x and y, so that it is defined for all (x, y). Most texture mapping hardware is capable of generating this sort of wraparound texture, by using modular arithmetic (or truncation of high order bits) to compute the appropriate texture map address from the x and y values. There are also tools to generate textures which wrap around without apparent seams [8].

To adapt this technique to hardware polygon rendering, the 2D region R is divided up into a regular grid of triangles, and the texture coordinates $F^{-t}(P_i)$ are only computed for the vertices P_i of the grid. During the hardware scan conversion, texturing, and shading process, the texture coordinates at each pixel are interpolated from those at the vertices, and the appropriate texture pixels are accessed. For triangles, the standard bilinear interpolation, which is not rotation invariant, reduces to linear interpolation, which is. For anti-aliasing, the hardware can use the higher order bits of the texture coordinates to weight an average of four adjacent texture map values (or four values in each of the two most-nearly-appropriate-resolution versions of the texture, if MIP mapping [9] is employed.)

TEXTURE DISTORTION

The flow $F^{-t}(P)$ can change the shape of a triangle, so that it becomes long and thin in texture space, as shown in figure 1. In the direction where the triangle is stretched by F^{-t}, the texture will be compressed by F^t. This distortion will not be present if the velocity is constant, so that F^{-t} and F^t are both translations. The distortion instead indicates anisotropies in the derivatives of V. For incompressible 2D flows, stretching in one direction will be compensated by compression in a perpendicular direction. For compressible flows, there may be stretching in all directions at some positions, and shrinking in all directions at others.

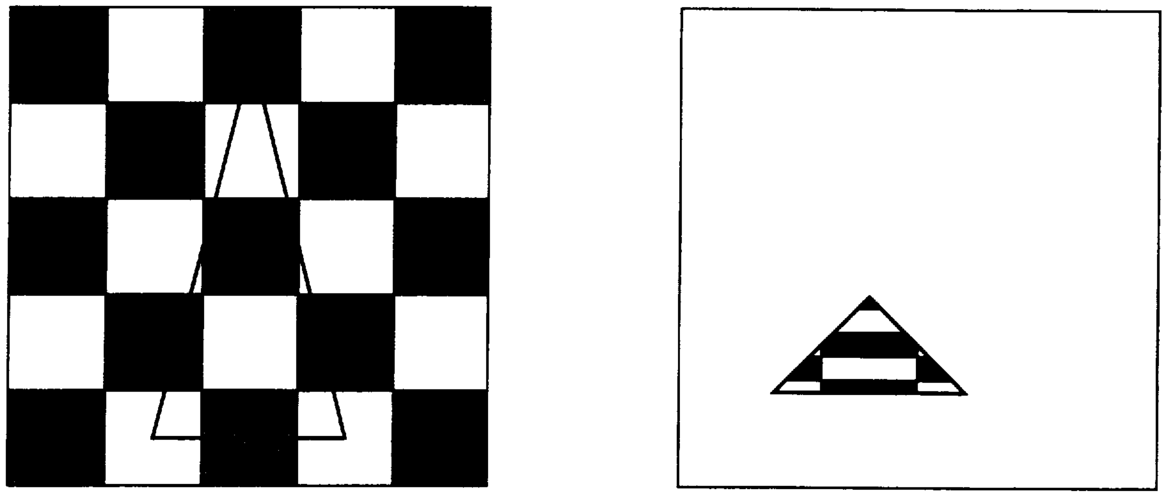


Figure 1. The triangle on the right is mapped to the texture on the left, which ends up being compressed vertically when the triangle is rendered.

During the animation of the texture advection, this distortion continues to build up, so that eventually the visualization will become useless. Therefore we periodically restart the texture coordinates back at their original positions in the regular grid. To avoid the sudden jump this would cause in the animation, we gradually fade up the new texture and fade down the old one, according to the weighting curves in figure 2. Each texture starts with weight zero, fades up over the old texture until it alone is present, and then fades

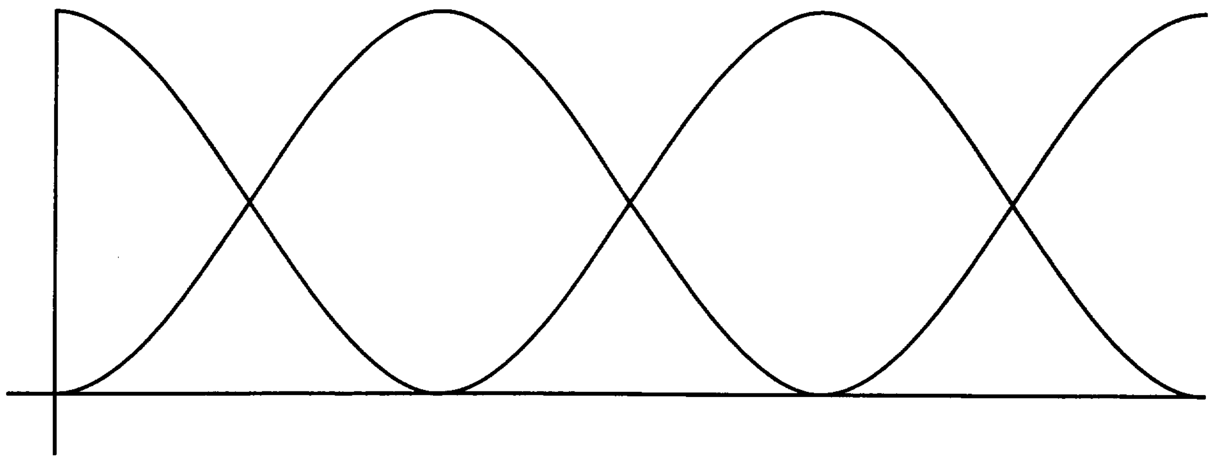


Figure 2. Three cycles of the weighting curves for fading the textures up and down.

down as an even newer texture takes its place. This "cross dissolve" can be done in hardware, using α compositing [10]. If the textures are random, and contain an adequate range of spatial frequencies, this cross dissolve will not disturb the perception of continuously flowing motion.

Since each texture is used for only a short time, the distortion does not become extreme. For a steady flow, one cross dissolve cycle ends with the same image at which it began, so an animation loop may be created which can be cycled rapidly and repeatedly on a workstation screen. Similar precomputed loops are possible with the surface particle [2], LIC [3], and textured splat [6] techniques.

TEXTURE ADVECTION FOR UNSTEADY 2D FLOWS

If the velocity V depends on t, the differential equation

$$\frac{dF^t(x,y)}{dt} = V\left(F^t(x,y),t\right) \tag{3}$$

defines a flow which no longer satisfies equation (2). For a fixed initial position Q, the curve $F^t(Q)$ is a particle trace $C(t)$ as in [11], rather than a streamline. To find the texture coordinates for P at time t_0 we need to find the point Q such that $F^{t_0}(Q) = P$. We must go backwards along the particle trace, and thus solve the differential equation

$$\frac{dC(t)}{dt} = V(C(t),t) \tag{4}$$

for the t range $0 \le t \le t_0$, with "final" condition $C(t_0) = P$, and then set $Q = C(0)$. With the change of variables $u = t_0 - t$, this is equivalent to the differential equation

$$\frac{dC(u)}{du} = -V(C(u),t_0-u) \tag{5}$$

for the u range $0 \le u \le t_0$, with initial condition $C(0) = P$. Then $Q = C(t_0)$.

In the case of unsteady flow, the differential equations (5) for different t_0 are not related and define completely different particle traces, so incremental methods can no longer be used. In [7] we integrated equation (5) anew for each frame time t_0. To find the texture coordinates for frame t_0, we had to access the time varying velocity data for the whole t range $0 \le t \le t_0$, which is very inefficient for large data sets. Here we propose two more practical methods.

The first method is to derive a differential equation for the flow $G^t(x,y) = \left(F^t\right)^{-1}(x,y)$. This flow maps a point P to the texture coordinate point Q needed at frame time t, that is, the point with $F^t(Q) = P$. Thus we have

$$F^t\left(G^t(P)\right) = P. \tag{6}$$

Let G^t_x and G^t_y be the x and y components of the vector-valued function $G^t(x, y)$, and similarly let F^t_x and F^t_y be the components of F^t. Then by differentiating the components of equation (6) with respect to t by the chain rule, we get the pair of equations

$$\frac{\partial F^t_x}{\partial t} + \frac{\partial F^t_x}{\partial x}\frac{\partial G^t_x}{\partial t} + \frac{\partial F^t_x}{\partial y}\frac{\partial G^t_y}{\partial t} = 0,$$
$$\frac{\partial F^t_y}{\partial t} + \frac{\partial F^t_y}{\partial x}\frac{\partial G^t_x}{\partial t} + \frac{\partial F^t_y}{\partial y}\frac{\partial G^t_y}{\partial t} = 0.$$

Now by equation (3), $\frac{\partial F^t_x}{\partial t} = V_x$ and $\frac{\partial F^t_y}{\partial t} = V_y$, where V_x and V_y are the components of the velocity field at position $F^t\left(G^t(P)\right) = P$ and time t. Therefore we have

$$M\begin{pmatrix} \frac{\partial G^t_x}{\partial t} \\ \frac{\partial G^t_y}{\partial t} \end{pmatrix} = \begin{pmatrix} -V_x \\ -V_y \end{pmatrix}$$

where M is the Jacobian matrix for the flow $F^t(x, y)$:

$$M = \begin{bmatrix} \frac{\partial F^t_x}{\partial x} & \frac{\partial F^t_x}{\partial y} \\ \frac{\partial F^t_y}{\partial x} & \frac{\partial F^t_y}{\partial y} \end{bmatrix}.$$

Thus

$$\begin{pmatrix} \frac{\partial G^t_x}{\partial t} \\ \frac{\partial G^t_y}{\partial t} \end{pmatrix} = M^{-1}\begin{pmatrix} -V_x \\ -V_y \end{pmatrix}.$$

But since $G^t(x, y) = \left(F^t\right)^{-1}(x, y)$, the matrix M^{-1} is the Jacobian matrix J for $G^t(x, y)$:

$$J = \begin{bmatrix} \frac{\partial G^t_x}{\partial x} & \frac{\partial G^t_x}{\partial y} \\ \frac{\partial G^t_y}{\partial x} & \frac{\partial G^t_y}{\partial y} \end{bmatrix}.$$

Thus $G^t(x, y)$ satisfies the partial differential equations:

$$\begin{aligned}\frac{\partial G_x^t(x,y)}{\partial t} &= -\frac{\partial G_x^t(x,y)}{\partial x}V_x - \frac{\partial G_x^t(x,y)}{\partial y}V_y \\ \frac{\partial G_y^t(x,y)}{\partial t} &= -\frac{\partial G_y^t(x,y)}{\partial x}V_x - \frac{\partial G_y^t(x,y)}{\partial y}V_y.\end{aligned} \tag{7}$$

These differential equations essentially say that the flow $G^t(x, y)$ is determined from the negative of the velocity V, as transformed into the texture coordinate system appropriate for $t = 0$, so they determine the texture flow necessary to give the desired apparent velocity at time t. The initial condition for G^t at $t = 0$ is that $G^0(P) = P$, that is, G^0 is the identity map. Equations (7) can be integrated incrementally in time by Euler's method. If $G^t(P_i)$ is known at time t for all vertices on a regular grid, the partials in the Jacobian matrix $J(P_i)$ can be estimated from central differences between the G^t values at adjacent grid vertices. (For vertices at the boundary of R, one-sided differences must be used.) Then, using the current velocity $V(G^t(P_i), t)$, increments $\Delta G_x = \frac{\partial G_x^t}{\partial t}\Delta t$ and $\Delta G_y = \frac{\partial G_y^t}{\partial t}\Delta t$ are found for the components of G^t. If necessary, Δt can be a fraction of the time step between frames, and/or the vertex grid used for solving equations (7) can be finer than the triangle grid used in rendering the texture, in order to make the solution more accurate.

The vertex grid spacing will affect the accuracy of the finite difference approximations to the partial derivatives like $\frac{\partial G_y^t}{\partial x}$. This accuracy is critical, because small errors in these partials will cause errors in position in the next frame, which may compound the errors in the partials, and cause them to grow exponentially from frame to frame. Here again, it is useful to fade out the current advected texture and fade in a new texture whose coordinates are reinitialized to the identity map, so that the integration errors cannot accumulate for too long.

The second method for handling unsteady flows is to move the triangle vertices by the flow $F^t(x, y)$, keeping their texture coordinates constant. This advects the texture directly, by moving the triangles, and carrying the texture along with them. To do this, we incrementally integrate equation (3), and no partial derivative estimates are needed for a Jacobian. However we again have a problem at the edges of the region R. The boundary vertices may move inside R, leaving gaps at the edges, or may move outside, causing too much texture to be rendered. The excess rendering is easily prevented by clipping all triangles to the boundary of R. The gaps can be eliminated by creating extra guard polygons around the edges of R, widening it to a larger region S. Whenever any vertex on the boundary of S crosses into R, a new row of guard polygons is added to the affected side of S. Again it is useful to integrate only over a limited time interval before reinitializing the texture coordinates, to avoid creating too many extra polygons.

FLOWS IN 3D

In three dimensions, one could advect 3D texture coordinates, but 3D texturing is not widely available.

We have instead used 2D textures on parallel section planes. We made the textured planes semi-transparent, and composited them from back to front using the α compositing hardware in our workstation. (This is how 3D texture mapping is usually implemented in hardware.) For the methods which change only the texture coordinates, we used the 2D projection of the velocity onto the section plane. For the method which moves the triangle vertices, we used the true 3D velocity, allowing the section surfaces to warp out of planarity.

Combining the compositing for the cross-dissolve of figure 2 with the compositing of the separate texture planes can lead to problems in the accumulated opacity. Given two objects with opacities α_1 and α_2, the resulting opacity from compositing both objects is $\alpha_1 + \alpha_2 - \alpha_1\alpha_2$. (See [10] or multiply the transparencies.) Suppose $f_1(t)$ and $f_2(t)$ are the two weighting curves shown in figure 2, with $f_1 + f_2 = 1$, and α is the desired section plane opacity. If we just take the two component opacities to be $\alpha_1 = \alpha f_1$ and $\alpha_2 = \alpha f_2$, the result is a composite opacity

$$\alpha_C = \alpha f_1 + \alpha f_2 - \alpha^2 f_1 f_2 = \alpha - \alpha f_1 f_2$$

The unwanted last term causes a periodic pulsation in α_C.

A solution is to use exponentials, which have better multiplicative properties. Define an "optical depth" $l = -\ln(1 - \alpha)$, so that $\alpha = 1 - e^{-l}$, and let $\alpha_1 = 1 - e^{-f_1 l}$ and $\alpha_2 = 1 - e^{-f_2 l}$. The resulting composite opacity is then

$$\begin{aligned}\alpha_C &= \alpha_1 + \alpha_2 - \alpha_1\alpha_2 \\ &= 1 - e^{-f_1 l} + 1 - e^{-f_2 l} - \left(1 - e^{-f_1 l}\right)\left(1 - e^{-f_2 l}\right) \\ &= 1 - e^{-(f_1 + f_2)\, l} = 1 - e^{-l} = \alpha\end{aligned}$$

as desired.

Another problem with compositing texture planes of constant transparency is that the frontmost planes will eventually obscure the ones to the rear if the data volume gets large. One solution is to use variable-transparency textures, so that some regions of the texture are completely transparent. Another is to specify the transparency on triangle vertices using a separate scalar data variable which can select out regions of interest where the texture motion should be visible. In [7] we used percent cloudiness contour surfaces to specify the location of the advecting software-rendered texture. With our new hardware based technique, this cloudiness variable is used to specify the vertex transparency, and produces similar realism in much less time.

IMPLEMENTATION AND RESULTS

The different types of moving textures discussed were implemented as a class hierarchy in C++. Inventor [12] quadmeshes were used to represent texture layers. An Iris Explorer module was then constructed in order to make use of color maps and data readers.

Figure 3 shows what happens when the vertices themselves are advected. The whole surface distorts, even in the direction perpendicular to the plane. In Figure 4 the texture coordinates are advected backwards while the vertices are held fixed. This gives the impression of motion in the direction of flow. Unfortunately the texture distorts too much over a long period of time. Also the texture vertices may move outside the defined domain. A solution to the first problem is to fade in a second texture with the texture coordinates reset to their original positions. The resulting cross dissolve is shown in Figure 5. The opacity for each texture is computed using exponentials, as discussed above, so there is no distracting variation in the overall intensity during animation. To avoid the problem of having to sample outside the domain, we used the inverse flow G^t for the texture coordinates, as explained above, while keeping the vertices fixed (Figure 6). This method also gives bad results over time if we do not periodically fade in a new unadvected texture as shown figure 7. Figure 8 illustrates how flow moves through particles of aerogel, a material with very low density which is a good thermal insulator. Figure 9 shows a frame from an animation of global wind data on a spherical mesh. The opaque regions represent high percent cloudiness. Although the vector field is static, the texture (but not the colors) appear to move in the direction of flow. Figures 10 and 11 depict steady flow near a high density contour in an interstellar cloud collision simulation (data courtesey of Richard Klein). Figure 10 has moving vertices, while figure 11 has moving texture coordinates. The color indicates density. A frame from an animation of unsteady wind data over Indonesia on a curvilinear mesh is shown in Figure 12. Percent cloudiness is mapped to color and opacity.

We ran our software on an SGI Onyx supporting hardware texture mapping. For a 32 by 32 slice of a volume (as in the aerogel example) we were able to achieve about four frames per second. To rotate a complete 50x40x10 volume, like the one shown in Figure 9, about 15 seconds was required.

REFERENCES

1. Shoup, Richard: *Color Table Animation.* Computer Graphics Vol. 13, No. 2 (August 1979) pp. 8 - 13

2. van Wijk, Jarke: *Flow Visualization With Surface Particles.* IEEE Computer Graphics and Applications, Vol. 13, No. 4 (July 1993) pp. 18 - 24.

3. Cabral, Brian; and Leedom, Lieth: *Imaging Vector Fields Using Line Integral Convolution.* Computer Graphics Proceedings, Annual Conference Series (1993) pp. 263 - 270.

4. Forssell, Lisa: *Visualizing Flow over Curvilinear Grid Surfaces using Line Integral Convolution.* Proceedings of IEEE Visualization '94, pp. 240 - 247.

5. Stalling, Detlev; and Hege, Hans-Christian: *Fast and Resolution Independent Line Integral Convolution*. ACM Computer Graphics Proceedings, Annual Conference Series, 1995, pp. 249 - 256.

6. Crawfis, Roger; and Max, Nelson: *Texture Splats for 3D Scalar and Vector Field Visualization*. Proceedings of IEEE Visualization '93, pp. 261 - 265.

7. Max, Nelson; Crawfis, Roger; and Williams, Dean: *Visualizing Wind Velocities by Advecting Cloud Textures*. Proceedings of IEEE Visualization '92, pp. 179 - 184.

8. Heeger, David; and Bergen, James: *Pyramid-Based Texture Analysis and Synthesis*. ACM Computer Graphics Proceedings, Annual Conference Series, 1995, pp. 229 - 238.

9. Williams, Lance: *Pyramidal Parametrics*. Computer Graphics Vol. 17 No. 3 (July 1983) pp. 1 - 11.

10. Porter Tom; and Duff, Tom: *Compositing Digital Images*. Computer Graphics Vol. 18, No. 4 (July 1984) pp. 253 - 259.

11. Lane, David: *UFAT - A Particle Tracer for Time-Dependent Flow Fields*. Proceedings of IEEE Visualization '94, pp. 257 - 264.

12. Wernecke, Josie: *The Inventor Mentor*. Addison -Wesley Publ. Co., Inc., 1994.

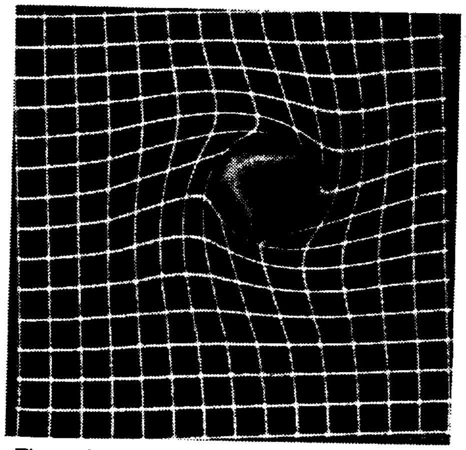


Figure 3. Actual vertices are advected in 3D.

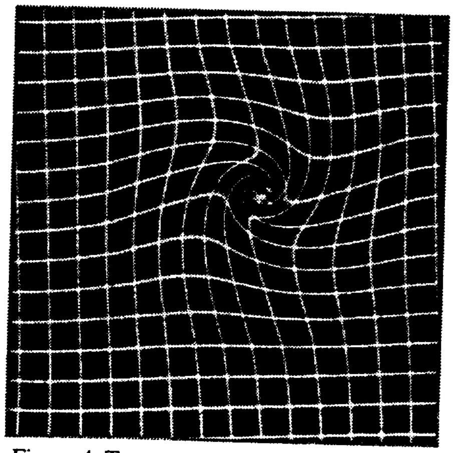


Figure 4. Texture coordinates are advected backwards.

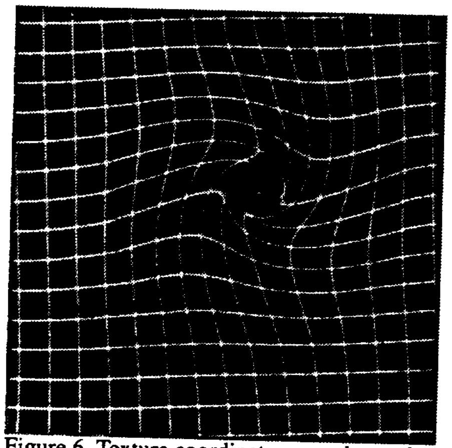


Figure 6. Texture coordinates are advected using vectors transformed by the local jacobian matrix, while vertices are held fixed.

Figure 5. Same method as figure 4, but with a new texture fading in as soon as the other becomes too distorted.

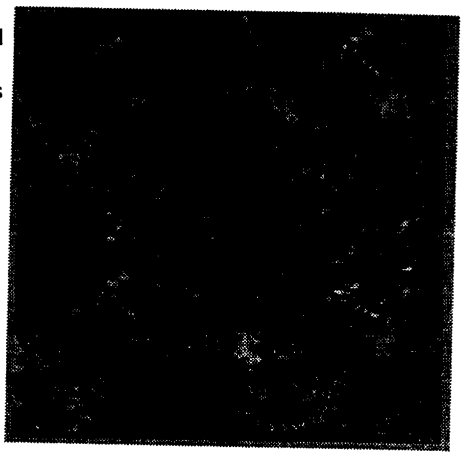


Figure 7. Same method as figure 6, but with a new texture fading in as soon as the other becomes too distorted.

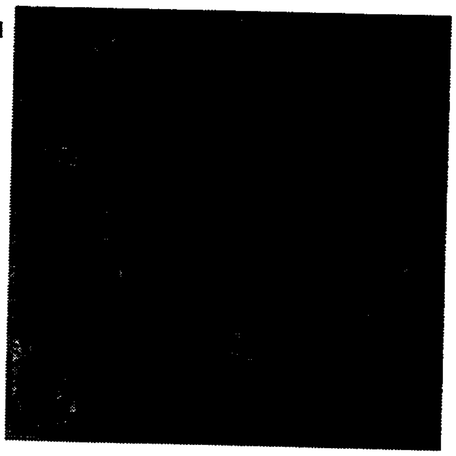


Figure 8. Method of figure 6 applied to a **steady** flow moving through particles of aerogel and using a colored texture.

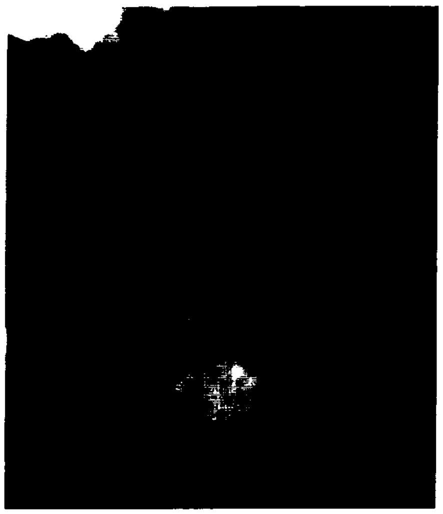


Figure 9. Method of figure 5 applied to a **steady** flow depicting wind data on a spherical mesh. Color and opacity from percent cloudiness.

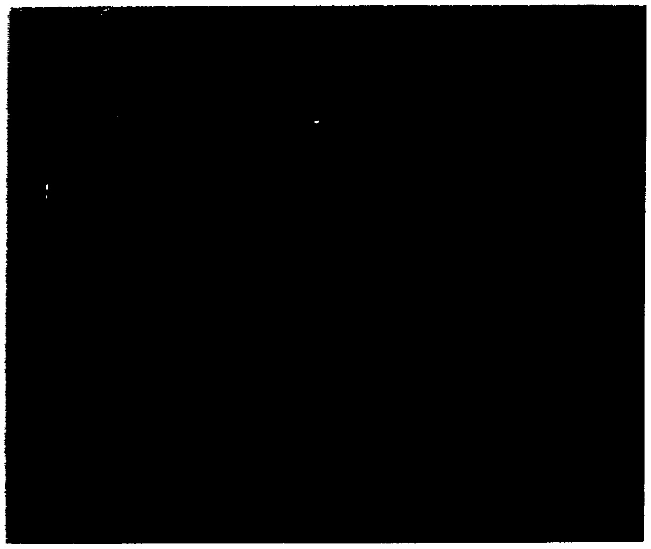


Figure 10. Several layers of textures advected using the method of figure 3. The layers are colored by density and move near a high density solid contour surface.

Figure 11. Method of figure 6 applied to a **steady** flow representing a field from an interstellar cloud collision simulation.

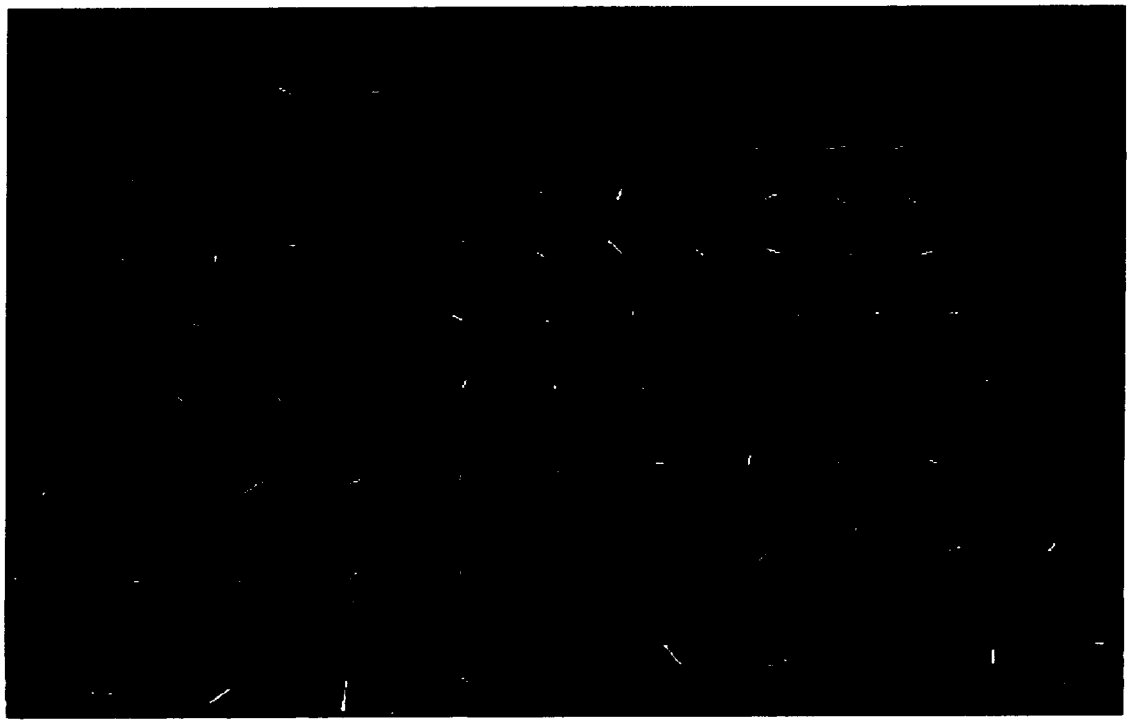


Figure 12. Method of figure 7 applied to an **unsteady** flow representing global climate data. Color and opacity indicate percent cloudiness. Both the winds and percent cloudiness vary in time.

VOLUMETRIC RAY CASTING OF TIME VARYING DATA SETS

Vineet Goel[1]
HP-Laboratories, 3U-4
1501 Page Mill Rd
Palo Alto, CA 94304

Amar Mukherjee
Department of Computer Science
University of Central Florida
Orlando, FL 32816

SUMMARY

In this paper, we present a fast technique for rendering scalar characteristics of time varying data sets. In this context, we present a block decomposition algorithm which decomposes changed data points in two consecutive time stamps, in rectangular blocks. These rectangular blocks are sorted in increasing visibility order for a given viewing direction. The sorted rectangular blocks are then projected on to the image plane to compute the color and opacity values of changed ray segments. These values are combined with the color and opacity values of rays in previous time stamp to generate an image for the current time stamp. The proposed algorithm allows any screen resolution. We also present a compact storage scheme for time varying data sets using the block decomposition algorithm. This scheme saves about 98.2% of the memory space for electrical wave propagation data sets.

INTRODUCTION

Volume Visualization is the subfield of scientific visualization which computes 2D information from 3D data sets. This 3D data set can be static or time varying. Time varying data sets arise in 3D flow simulation [7, 5, 11, 2, 12], recording of neuron activities in the brain at different time stamps [1, 15], simulation of electrical wave propagation in the heart [4], etc. The resulting 3D data is directly rendered (called *Direct Volume Rendering (DVR)*) in order to generate high quality images. The different methods used for direct volume rendering are ray-casting [9, 10], integration methods, splatting [19], and V-buffer [18] rendering. Direct volume rendering is a computationally expensive task as full volume data has to be traversed for a given viewing direction.

In FTB computation, the order of sample points computation is 1 through n for a ray r. At the ith sample point, the following computations are done.

$$\begin{aligned} C_r(i+1) &= c_i\alpha_i(1-A_{\alpha_i}) + C_r(i) \\ 1 - A_{\alpha_{i+1}} &= (1-\alpha_i)(1-A_{\alpha_i}) \end{aligned} \tag{1}$$

Where c_i and α_i are the color and opacity values of sample i. The quantity $C_r(i)$ is the color value

[1]Previously, Ph.D. student at University of Central Florida

contributed by sample 1 through $i-1$ for a ray r. The quantities A_{α_0} and $C_r(0)$ are zero. The final color value of the ray is given by $C_r(n+1)$.

The Eqn. 1 can be written in matrix form as

$$\begin{pmatrix} C_r(i+1) \\ 1-A_{\alpha_{i+1}} \end{pmatrix} = \begin{pmatrix} 1 & c_i\alpha_i \\ 0 & 1-\alpha_i \end{pmatrix} \begin{pmatrix} C_r(i) \\ 1-A_{\alpha_i} \end{pmatrix} \tag{2}$$

which can be written as:

$$M_{i+1,0} = E_i M_{i,0} \tag{3}$$

where $M_{0,0} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$,

$$E_i = \begin{pmatrix} 1 & c_i\alpha_i \\ 0 & 1-\alpha_i \end{pmatrix} \tag{4}$$

and

$$M_{i,0} = \begin{pmatrix} C_r(i) \\ 1-A_{\alpha_i} \end{pmatrix} \tag{5}$$

Given a ray, suppose we want to compute the accumulated color and opacity values of a ray segment from sample points (i) through $(i+j)$, From equation 2,

$$\begin{pmatrix} C_r(i+j+1) \\ 1-A_{\alpha_{i+j+1}} \end{pmatrix} = \begin{pmatrix} 1 & c_{i+j}\alpha_{i+j} \\ 0 & 1-\alpha_{i+j} \end{pmatrix} \begin{pmatrix} 1 & c_{i+j-1}\alpha_{i+j-1} \\ 0 & 1-\alpha_{i+j-1} \end{pmatrix} \cdots \begin{pmatrix} 1 & c_i\alpha_i \\ 0 & 1-\alpha_i \end{pmatrix} \begin{pmatrix} C_r(i) \\ 1-A_{\alpha_i} \end{pmatrix} \tag{6}$$

We can represent it as

$$\begin{pmatrix} C_r(i+j+1) \\ 1-A_{\alpha_{i+j+1}} \end{pmatrix} = \begin{pmatrix} 1 & C(i,j) \\ 0 & A(i,j) \end{pmatrix} \begin{pmatrix} C_r(i) \\ 1-A_{\alpha_i} \end{pmatrix} \tag{7}$$

where $C(i,j)$, $A(i,j)$ is accumulated color and opacity values for a ray segment from sample (i) through $(i+j)$. Therefore,

$$\frac{C_r(i+j+1)-C_r(i)}{1-A_{\alpha_i}} = C(i,j)$$
$$\frac{1-A_{\alpha_{i,j+1}}}{1-A_{\alpha_i}} = A(i,j) \tag{8}$$

Consider a ray r which is segmented in p parts consisting of sample points useful in computing color and opacity values of a ray, (i.e., skipping empty voxels). Let, these p segments be (i_0,i_1), (i_2,i_3) , (i_4,i_5), ... $(i_{2p-2},0)$, where $n > i_1 > i_2 > i_3 > ... > i_{2p-2} > 0$, as the ray passes through p sorted blocks and $(i,j), i > j$ denotes sample points $j, j+1, ..., i$ in the segment. It's color and opacity values are computed using the product of p matrices as

$$M_{n+1,0} = C_{i_0,i_1} C_{i_2,i_3} ... C_{i_{2p-2},0} M_{0,0} \tag{9}$$

where $C_{i,j} = E_{i-1}E_{i-2}...E_j, i > j$, where i and j denote the beginning and end cells of a segment. Computation of a ray segment is equivalent to computation of matrices $C_{i,j}$. The final ray value is computed as the product of matrices as in Eqn. 9.

Assume that two consecutive ray segments r_1 and r_2 of a ray r which are ordered as r_2, r_1, have values C_{i_1,i_2} and C_{i_3,i_4}, respectively, where $i_1 > i_2 > i_3 > i_4$. r_1 **over** r_2 (i.e. r_1 composed over r_2) is defined as $C_{i_1,i_2}C_{i_3,i_4}$. If r_1 and r_2 are ordered as r_1, r_2 then r_2 **over** r_1 is defined as $C_{i_3,i_4}C_{i_1,i_2}$ [14]. The quantities C_{i_1,i_2} and C_{i_3,i_4} are called **partial ray values** for ray r.

These equations will be used in the context of volumetric ray casting of time varying data sets. In this paper, we deal with visualizing only the scalar characteristics of a time varying data set using the volumetric ray casting technique. This technique has been used for visualizing electrical wave propagation in heart [4, 16] and it can be used in visualizing the activities of neurons in the brain using fMRI (functional MRI) data sets [1, 15]. Shen and Johnson [16] proposed a method where only changed data points are projected onto the image plane to find those rays which need to be recomputed. Here, a changed data point is projected onto the image plane to find the affected pixels which are four neighboring pixels surrounding the projected point. Therefore, screen resolution is assumed to be the same as the resolution of 3D data set. Moreover in this method, changed rays are fully traversed in the volume data set to compute their final color values. Therefore, a changed ray may traverse empty and/or unchanged data points. Shen and Johnson [16] also proposed storing only the changed data point values and their positions in a *differential* file for compact storage of time varying data sets.

In this paper, we propose a new sequential algorithm for rendering time varying data sets. We use a block decomposition algorithm for compact storage of the time varying data sets. In particular, we present the following results for compact storage scheme and rendering of time varying data sets.

- A novel idea of rendering time varying data sets using a block decomposition algorithm.
- A compact storage scheme for time varying data sets. The typical data set size is 128^3 at each time stamp. This requires 210 MB of memory space to store time varying data set over 100 time stamps. Our proposed storage scheme requires only 3.7MB of memory space for electrical wave propagation in the heart data set.
- The proposed algorithm allows any screen resolution unlike the existing algorithm [16] where screen resolution is the same as the resolution of a volume data set.
- The implementation results suggest that the performance of our algorithm is comparable to the algorithm by Shen and Johnson [16].

BLOCK DECOMPOSITION

In this section, we describe the block decomposition algorithm in the context of 2D data sets which is then extended to block decomposition for 3D binary data sets. We have proved that the number of blocks decomposed by the proposed algorithm for 2D data sets has an upper bound of three times the minimum number of blocks [6]. We conjecture that the upper bound of the number of blocks decomposed for 3D data sets is four times the minimum number of blocks. This algorithm has linear time complexity with respect to number of blocks for both 2D and 3D data sets.

Nomenclature and Definitions

A 2D data set of size $n \times n$ is represented by I with a positive integer value $I(y,x) \geq 0$ representing a pixel at the yth row and xth column of the 2D data set where $1 \leq x \leq n, 1 \leq y \leq n$.

A 3D binary data set of size $n \times n \times n$ is represented by V with a positive integer value $V(z,y,x)$ at location (x,y,z) where $1 \leq x \leq n, 1 \leq y \leq n, 1 \leq z \leq n$. The 3D data set can be thought of as a stack of n 2D data sets of size $n \times n$.

The ith row of a 2D data set I is a set of pixels called ith **scan-line** Q with value $Q(x) = I(i,x)$ at location x where $1 \leq x \leq n$. Similarly, a kth **slice** of 3D data set V is a 2D data set I with $I(y,x) = V(k,y,x)$.

A **line segment** L in a scan-line Q is defined as a maximum set of contiguous pixels having the same integer value $d_L > 0$. The minimum and maximum x coordinates in a line segment L are represented by $X_s(L)$ and $X_e(L)$, respectively.

A **2D-block** R in a 2D data set I is the set of pixels with the same integer value $d_R > 0$, contained in a rectangle whose minimum x and y coordinates are represented by $X_s(R)$ and $Y_s(R)$, respectively and maximum x and y coordinates are $X_e(R)$ and $Y_e(R)$, respectively. Similarly, a **3D-block** B in a 3D data set V is a rectangular block of voxels with value $d_B \geq 1$ whose minimum x, y and z coordinates are $X_s(B), Y_s(B)$ and $Z_s(B)$, respectively and maximum x, y and z coordinates are $X_e(B), Y_e(B)$ and $Z_e(B)$, respectively. Note that, unlike 2D-block, the voxel values in 3D-block required to be non-zero but not necessary equal.

Functions $X_s(P), Y_s(P)$, and $Z_s(P)$ give minimum x, y and z coordinates respectively, wherever applicable, of the entity P where P can be a line segment L, a 2D-block R, or a 3D-block B. Similarly, functions $X_e(P), Y_e(P)$, and $Z_e(p)$ give maximum x, y and z coordinates respectively, of the entity P. Function $B\_val(P)$ gives the value for entity P which is equal to d_L, d_R, and d_B for a line segment L, a 2D-block R, and a 3D-block B, respectively.

Two line segments L_1 and L_2 in different rows are said to be aligned if $X_s(L_1) = X_s(L_2)$, $X_e(L_1) = X_e(L_2)$, and $B\_val(L_1) = B\_val(L_2)$.

A scan-line Q can be decomposed into a unique minimum set $\mathcal{L}$ of non-overlapping line segment. A 2D data set I can be decomposed into a set $\mathcal{R}$ of non-overlapping 2D-blocks. This decomposition may not be unique. Similarly, a 3D data set V can be decomposed in several ways into a set $\mathcal{V}$ of non-overlapping 3D-blocks.

In the following section, we present algorithms to obtain block decomposition of a 2D data sets in 2D-blocks.

2D-Block Decomposition

A method of finding a block decomposition of 2D data set is to extend a method by Ferrari et.al. [3, 8, 13] of finding minimal number of rectangles in 2D binary images. This method first finds all the

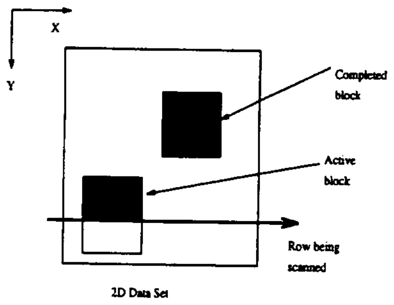


Figure 1: Active and deactive 2D-blocks in 2D data set

concave vertices in the data set and then uses matching technique with complexity $O(e^{1.5}\log e \log\log e)$ to find 2D-blocks [8]. Here, e is the number of concave vertices. This process is time consuming and concave vertices have to be determined first. We propose an $O(e)$ algorithm which finds a decomposition of the rectilinear region having at most three times the minimum number of 2D-blocks, while raster scanning the 2D data set.

While raster scanning a 2D (or 3D) data set, we will recognize two types of blocks, **active** and **complete**. The complete blocks are those which have already been identified by the scanning process in our decomposition and the active blocks are those which are still being processed, as illustrated in Fig. 1.

Each scan line of 2D data set I is decomposed into line segments while raster scanning the data set. The line segments in consecutive scan-lines of data set having same pixel values are combined to form blocks. While combining, line segments can be broken into more than one line segments. For example in Fig. 2, ith scan line has line segments a and c and $(i+1)$th scan line has line segments b and d. While combining line segments in scan lines i and $(i+1)$, line segment a is broken into $a1$ and $a2$, line segment b is broken into $b1$ and $b2$ and line segment c is broken into $c1$ and $c2$. Line segments $a2$ and $b1$ form part of a block. Line segment $a1$ is followed by zero entries in scan line $(i+1)$, therefore the block corresponding to $a1$ is complete. A new block starts with line segment $b2$. Similarly, a block corresponding to $c1$ is complete, and line segments $c2$ and d are combined to form a part of a block. The active blocks correspond to $a2$, $b2$, and $c2$ during scanning $(i+1)$th row.

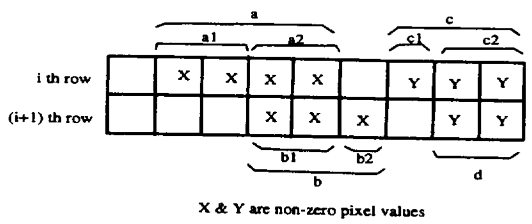


Figure 2: Combining line segments in two consecutive scan lines.

The idea behind the decomposition algorithm for two-dimensional data set can be explained with respect to the 8×8 data set as shown in Fig. 3(a). The data set is raster scanned top to bottom. Two buffers BV and BC each of size $n \times 1$ are used. BV holds the pixel value of the current scan line, that is, $BV(j) = I(i,j)$ if the ith row of the data set is being scanned currently. The buffer BC holds a *count* value associated with the pixel value $I(i,j)$ as follows.

$$
\begin{array}{rcll}
BC(j) & = & k & \text{if the } k \text{ pixel values including the current pixel value at the } j\text{th} \\
 & & & \text{column of the image are identified as equal and non-zero, i.e.,} \\
 & & & I(i,j) = I(i-1,j) = \ldots = I(i-k+1,j) \neq 0 \\
 & & & \text{and } I(i,j) \neq I(i-k,j) \\
 & = & 1 & \text{if } I(i,j) \neq 0 \text{ and } I(i,j) \neq I(i-1,j) \\
 & = & 0 & \text{otherwise}
\end{array} \tag{10}
$$

Initially, all buffers are set to zero. The values of BV and BC for the first three scan lines are shown in Fig. 3(b). When we enter the fourth scan line, we recognize that $I(4,2) = 0$ indicating that a completed block needs to be identified. We proceed to finish scanning of the fourth scan line, pixel by pixel, after copying BV and BC to two temporary buffers TV and TC respectively, pixel by pixel only if $I(i,j) \neq BV(j)$. The situation is illustrated in Fig. 3(b). TV and TC now contain all the information regarding the completed blocks. The line segments of TC correspond to blocks whose pixel value equals the pixel value of the corresponding segment in TV and whose height equals the corresponding entry of the line segment in TC. Thus at this point, we have identified two blocks each of size 2×2 containing a pixel value of 1. When we finish scanning the last row of the image, the content of TV and TC will be as shown in Fig. 3(c), identifying the lower left block of size 2×2 containing pixel value 2, the middle block of size 6×2 containing pixel value 2 and the lower right block of size 2×2 containing pixel value 3. The final block decomposition is shown in Fig. 3(d).

In general, after scanning ith row of the 2D data set, the line segments in the buffer TC represent completed blocks only if the content of TC buffer has changed during ith raster scan. For each such line segment L, there is a block R such that

$$
\begin{aligned}
&X_s(R) = X_s(L), \\
&X_e(R) = X_e(L) \\
&Y_s(R) = i - 1 - B\_val(L), \\
&Y_e(R) = i - 1
\end{aligned}
$$

The value d_R of block R is equal to the pixel value in the TV for the corresponding line segment L.

To formalize the description of the algorithm, whose computation is based only on the current pixel value $I(i,j)$, the condition expressed by Eqn. 10, can be restated as a *sequence* of operations:

$$
\begin{array}{llcll}
1. & BC(j) & \leftarrow & BC(j)+1 & \text{if } BV(j) = I(i,j) \neq 0 \\
 & & \leftarrow & 1 & \text{if } (I(i,j) \neq 0) \text{ and } (I(i,j) \neq BV(j)) \\
 & & \leftarrow & 0 & \text{otherwise} \\
2. & BV(j) & \leftarrow & I(i,j) &
\end{array}
$$

We also need a control bit F which is used to initiate the operation to output the completed blocks. This bit is reset to zero at the beginning and is set to one for the ith row if and only if for any j, the new value of $BC(j)$ is less than or equal to current value of $BC(j)$. The 2D-block decomposition algorithm returns a set $\mathcal{R}$ of blocks stored in the form of a linked list. Each record of the linked list

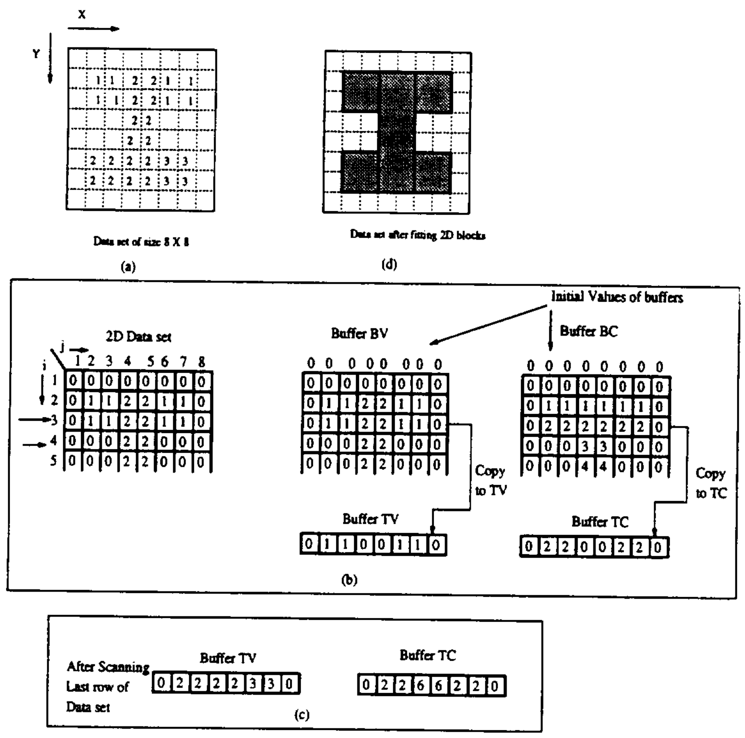


Figure 3: (a) Sample 8 × 8 data set. (b) Values of BF, BC, TF and TC buffers after processing 3rd and 4th row of data set. (c) Content of buffers TV and TC after scanning last row of the 2D data set. (d) Blocks decomposed in original data set.

stores parameters $X_s(R), Y_s(R), X_e(R), Y_e(R)$, and d_R of a block R and pointer to next record in the linked list.

If we exclude raster scanning time, the time complexity of the 2D-block decomposition algorithm is $O(|\mathcal{R}|)$ [6], while finding the minimum number of blocks in a rectilinear region takes $O(|\mathcal{R}|^{1.5} \log |\mathcal{R}| \log\log |\mathcal{R}|)$ [8]. We have proved that that this algorithm finds number of blocks less than three times the minimum number of blocks for any arbitrary rectilinear region [6].

3D-Block Decomposition

In this section, we present a generalization of the block decomposition algorithm to three dimensions. We simplify the problem by assuming that all non-zero voxel values are mapped to one, essentially reducing the data set to a 3-dimensional binary data set. The main idea of the algorithm is as follows.

The 3D data set is processed slice by slice. Each slice is raster scanned in the same manner as a 2D data set. If a set of non-zero pixels in a slice are followed by zero pixels at the corresponding pixel locations in the next slice, a set of 3D-blocks are completed. As an example, consider the binary 3D data set representation of object in Fig. 4(a), by five slices in the z-direction in Fig. 4(b). The shaded region represents non-zero entries. For the sake of clarity, the depth information (the count of number of contiguous non-zero voxels in z-direction) are also written on the corresponding voxel location at each slice. We notice that the pixels at $(2,3,1), (2,2,2), (2,3,2)$ in the slice $z = 2$ are followed by

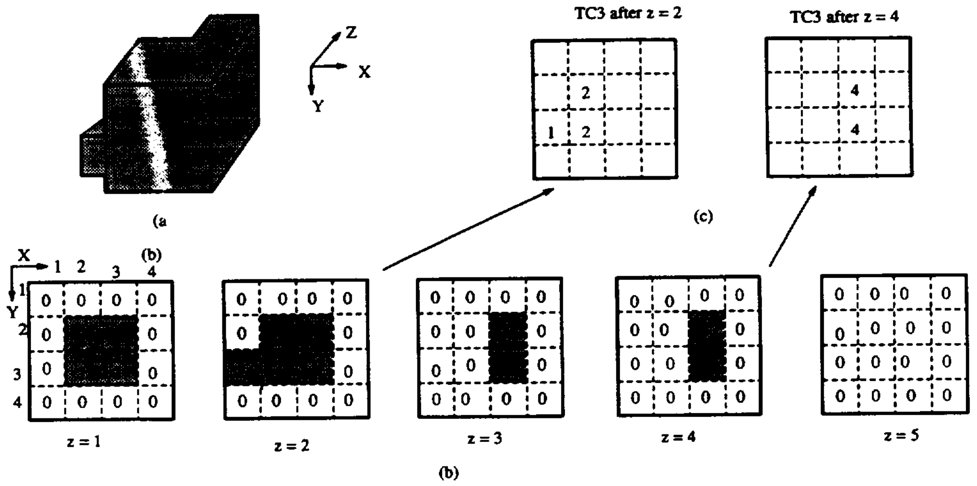


Figure 4: (a) A 3D object, (b) Slices in 3D data set representing object. The buffer $BC3$ is shown superimposed on the slices. (c) Buffer containing depth information for completed blocks, at the end of scanning 3rd slice.

zero pixels in slice $z = 3$ in Fig. 4(b). For these pixel locations, the count values do not increase monotonically from the slice at $z = 2$ to the slice at $z = 3$. This indicates a set of completed blocks and we can now extract these count values to a 2D buffer $TC3$ as shown in Fig. 4(c). We apply the **2D-Block-Decomposition** algorithm of the previous section to buffer $TC3$. Each 2D-block R in $\mathcal{R}$ has value d_R equal to depth value in the buffer. For each 2D-block R, a 3D-block B can be constructed using $X_s(R), X_e(R), Y_s(R), Y_e(R)$ values of R as $X_s(B), X_e(B), Y_s(B), Y_e(B)$ values, respectively of B. The $Z_s(B)$ and $Z_e(B)$ value of the 3D-block B can be found using the current slice number k and d_R value of 2D-block R, assigning $Z_s(B)$ equal to $k - 1 - d_R$ and $Z_e(B)$ equal to $k - 1$. Thus for example, we identify two blocks of size $1 \times 1 \times 1$ and $1 \times 2 \times 2$ at $z = 3$ and one block of size $1 \times 2 \times 4$ ar $z = 5$.

In the algorithm **3D-Block-Decomposition**, Two buffers $BC3$ and $TC3$ each of size $m \times m$ are used. Buffer $BC3$ holds a count value associated with location $V(k, i, j)$ as follows.

$$\begin{aligned} BC3(i,j) &= p \quad \text{if } V(k,i,j) = V(k-1,i,j) = \ldots = I(k-p+1,i,j) = 1 \\ &= 1 \quad \text{if } V(k,i,j) = 1 \text{ and } V(k-1,i,j) = 0 \\ &= 0 \quad \text{otherwise} \end{aligned}$$

Initially, both the buffers $BC3$ and $TC3$ are set to zero. As a zero pixel in a slice follows non-zero pixel at corresponding $x = j, y = i$ location in the previous slice, the count value of $BC3(i, j)$ is copied to $TC3(i, j)$ and $BC3(i, j)$ is set to zero.

A control bit G is used to initiate the operation of output of completed blocks. It is set to zero initially, and set to one if at some slice values in $BC3$ are copied to buffer $TC3$. At the end of scan of a slice k, if bit G is one, it implies that $TC3$ has depth information for some completed blocks. The **2D-Block-Decomposition** algorithm is used to find 2D-blocks in $TC3$. For each 2D-block R, a 3D-block B is constructed using following equations.

$X_s(B) = X_s(R)$,
$X_e(B) = X_e(R)$,
$Y_s(B) = Y_s(R)$,
$Y_e(B) = Y_e(R)$
$Z_s(B) = k - B\_val(R) - 1$,
$Z_e(B) = k - 1$

The 3D-block decomposition algorithm returns a set $\mathcal{B}$ of 3D-blocks stored in the form of a linked list. Each record of the linked list stores parameters $X_s(B), Y_s(B), X_e(B), Y_e(B), Z_s(B), Z_e(B)$ of a 3D-block B and pointer to next record in the linked list.

The time complexity of the algorithm is $O(n^3 + \mid \mathcal{B} \mid)$, where $O(n^3)$ is the time taken to raster scan the 3D data set and $O(\mid \mathcal{B} \mid)$ is the time for blocks output. Finding the minimum number of blocks in a rectilinear volume is NP-Hard. It is our conjecture [6] that this algorithm finds number of blocks less than four times the minimum number of blocks for any arbitrary rectilinear volume.

COMPACT REPRESENTATION OF TIME VARYING DATA SETS

The main idea behind the proposed method for compact storage of time varying data sets is to compare volume data of two consecutive time stamps and to decompose only the changed data points in 3D-blocks. This is done by using a slight modification of **3D-Block-Decompostion** algorithm. Let a sequence of 3D time varying data sets be denoted by $\mathcal{V}$, where the size of each volume data set is n^3. A volume data set at time stamp t is denoted by V_t, where $1 \le t \le \mid \mathcal{V} \mid$, where $\mid \mathcal{V} \mid$ denotes the number of volume data sets in $\mathcal{V}$.

The volume data at time stamp zero is denoted by V_0. Let B^c be a block containing only those data points which have changed. The block B^c is stored in a file called *Block* by storing its starting coordinates $(X_s(B^c), Y_s(B^c), Z_s(B^c))$ and ending coordinates $(X_e(B^c), Y_e(B^c), Z_e(B^c))$. These are followed by data values of changed data points within the block B^c. When all the changed data blocks in volume data sets in two consecutive time stamps are recorded in the *Block* file, a marker with value -1 is written in the *Block* file to indicate the end of the changed data points between two consecutive time stamps.

Let the total number of changed data points and changed data blocks over all the volume data sets be n^c and n^b. The time complexity of the algorithm is $O(\mid \mathcal{V} \mid n^3 + n^b + n^c)$, where $O((\mid \mathcal{V} \mid n^3)$ is the time taken to raster scan the 3D data set and $O(n^b + n^c)$ is the time for blocks output. The total memory required for storing sequence of time varying data sets is $O(2n^b + n^c + \mid \mathcal{V} \mid)$, where storing starting and ending positions of a block requires $O(2n^b)$ memory space. While the existing algorithm [16] requires $O(2n^c)$ memory space in order to store positions and data values of changed data points. Therefore, for a time varying data sequence where changed data points are mostly clustered, such as in fMRI data sequence recording neuron activities in the brain [1, 15], the proposed algorithm requires less memory compare to method proposed by Shen and Johnson [16].

Table 1: Block decomposition time and number of changed data points at different time stamps.

Time Stamp	Number of changed data points	Number of blocks	Block Decomposition time(sec)
10	2394	1046	2.69
20	8057	3705	2.89
30	9299	4609	2.97
40	9898	4834	3.05
50	7300	3690	3.00
60	2536	1338	2.83
70	1160	572	2.71
80	684	369	2.62
90	179	108	2.54

Results

We have implemented the proposed time varying data sets storage algorithm on Sun-Sparc (40 MHz). Table 1 gives the number of changed blocks, block decomposition time and number of changed data points at different time stamp for a sequence of electrical wave propagation data sets. Each data set is of size 128^3 and there are total 100 data sets including the data set at time stamp zero. The time to raster scan a volume data in this sequence is 2.10 seconds which is included in the block decomposition time.

The total sequence of 99 volume data sets is stored in 1.6 MB using this algorithm. The size of the volume data set V_0 at time stamp zero is 2.10 MB. Therefore, the total memory required to store the sequence time varying data set is 3.7 MB, resulting in a 98.2% saving of the storage space. The existing algorithm by Shen and Johnson [16] requires 4.18 MB of storage space. In these data sets, the changed data points are scattered, thereby giving a large number of changed blocks. For a time varying data set sequence, where changed data points are comparatively more clustered, the proposed technique will save the storage space by an even larger factor.

VOLUMETRIC RAY CASTING OF TIME VARYING DATA SETS

The main idea behind the proposed volumetric ray casting algorithm is as follows. Suppose at time stamp t, a ray r is segmented into p parts consisting of sample points useful in computing color and opacity value of ray r (i.e., skipping empty voxels). Let these p segments be (i_1, i_2), (i_3, i_4) , (i_5, i_6), ... (i_{2p-1}, i_{2p}), where $(i,j), i > j$ denotes sample points $j, j+1, ..., i$ in the segment at the time stamp t and $i_k \geq i_{k+1}$ for all k. A segment containing only the changed sample points called *changed* ray segment. A ray containing at least one changed ray segment at a time t is called a *modified* ray. A list of ray segments for a ray may contain some changed ray segments at time t. The new color value of the ray is computed using Eqn. 9 where only $C_{i,j}$ needs to be computed if (i,j) is a changed ray segment.

At the time stamp 0, the image can be generated for a specified viewing direction using the fast rendering algorithm [6]. To obtain an image at time stamp t from the image at time $t-1$, the following steps are performed.

1. *Modification:* The changed blocks at time stamp t are read from the *block* file and current volume data points are updated using the new values of changed data points in blocks. This results in volume data set at time stamp t. The block parameters $X_s(B^c)$, $Y_s(B^c)$, $Z_s(B^c)$, $X_e(B^c)$, $Y_e(B^c)$, and $Z_s(B^c)$ are stored in a linked list representing the set of blocks $\mathcal{B}^c$ as described earlier.

2. *Sorting:* The set of the changed blocks $\mathcal{B}^c$ is sorted for specified viewing direction using the block sorting technique.

3. *Projection:* Each of the changed blocks B^c is projected onto the image plane and the changed ray segments are computed. The starting color and opacity value for a changed segment (i,j) of ray r is set to 1.0 and 0.0 respectively.

4. *Merging:* For each modified ray, values of changed segments are combined with values of unchanged segments, to find the new color and opacity values of a modified ray using Eqn. 9.

In the next section, we describe the technique for sorting blocks according to their visibility order.

SORTING OF BLOCKS

We present a method for sorting blocks according to their visibility order for orthographic and perspective projections. We show that for any viewing direction, the blocks can be totally ordered and the viewing directions can be categorized into a small number of equivalent categories. Furthermore, for orthographic projections all rays belong to only one category of viewing directions. We then show that visibility order of blocks for nay viewing category is same.

Orthographic Projection

We will first describe the sorting of blocks in 2D data set and then extend the method for blocks in 3D data set. The direction of a ray can be specified by a unit viewing vector (x_v, y_v), where x_v is x-component and y_v is y-component of the unit vector. All the viewing vectors can be divided into four *categories* as $(+,+), (+,-), (-,+)$, and $(-,-)$, where $+$ stands for viewing component being greater than or equal to zero and $-$ stands for viewing component being less than zero. For orthographic projection for which all the rays are parallel, the rays have same unit viewing vector (x_v, y_v). We now prove the following theorem.

Theorem 1: Any two rays r_1 and r_2 in a viewing category, traversing the set of blocks $\mathcal{R}_1$ and $\mathcal{R}_2$, respectively, traverse blocks in $\mathcal{R}$ in same order, where $\mathcal{R} = \mathcal{R}_1 \bigcap \mathcal{R}_2$ and $\mid \mathcal{R} \mid \geq 2$.

Proof: Let $R_i, R_j \in \mathcal{R}$ and rays r_1, r_2 belong to the viewing category $(+,-)$.

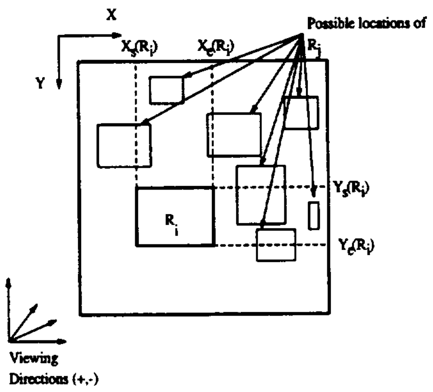


Figure 5: Possible locations blocks R_j for a ray to traverse R_i first and then R_j.

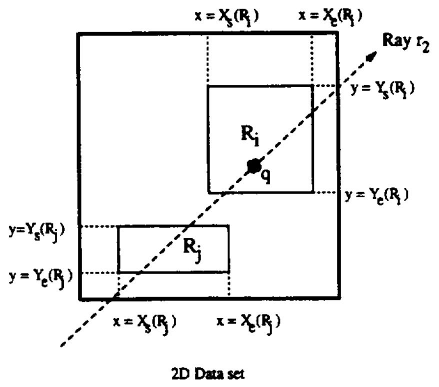


Figure 6: Ray r_2 traversing block R_j first and then block R_i.

Let r_1 traverse R_i first and then R_j, the block R_j could be located as shown in Fig. 5. The top and right boundaries of R_i define three regions (dotted lines) and R_j could be totally or partially contained in these regions. Therefore,

$$X_s(R_j) > X_e(R_i) \quad \text{or} \quad Y_e(R_j) < Y_s(R_i) \tag{11}$$

Let r_2 pass through first R_j and then R_i. Consider, a point q at location (x_q, y_q) on ray r_2 after ray entered block R_j but before exiting from R_i, as shown in Fig. 6, then

$$x_q > X_s(R_j) \quad \text{and} \quad y_q < Y_e(R_j) \tag{12}$$

and

$$x_q < X_e(R_i) \quad \text{and} \quad y_q > Y_s(R_i) \tag{13}$$

from Eqns. 12 & 13,

$$X_e(R_i) > X_s(R_j) \quad \text{and} \quad Y_s(R_i) < Y_e(R_j) \tag{14}$$

Eqn. 14 contradicts Eqn. 11. Therefore, rays r_1 and r_2 pass blocks R_i and R_j in the same order. We can prove the theorem for rays in other viewing categories $(+,+), (-,-)$, and $(-,+)$. in the same manner. □

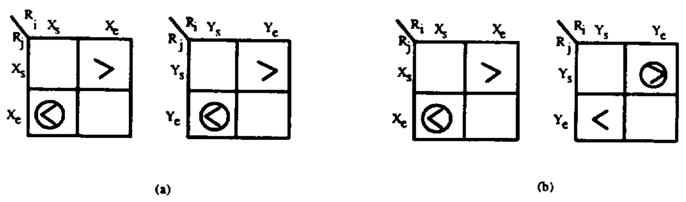


Figure 7: Tables for determining the visibility order of blocks R_i and R_j for viewing categories (a) $(+,+)$, (b) $(+,-)$.

Therefore, from Eqn. 11, the visibility order of two blocks R_i and R_j can be determined by comparing either the X or Y coordinates of the boundaries as given by tables in Fig. 7. The condition $X_s(R_j) > X_e(R_i)$ is shown in Fig. 7(a) by marking with the symbol $>$ in the entry corresponding to X_s row for R_j and X_e column for R_i for the viewing category $(+,+)$. If any of the uncircled condition is true then the order is (R_i, R_j). If any of the circled condition is true the order is (R_j, R_i). When a circled and an uncircled condition are true, then any of the ordering (R_i, R_j) and (R_j, R_i) can be taken because the two blocks have no rays in common. Given (R_i, R_j) and (R_j, R_k) as sorted order for three blocks R_i, R_j and R_k, their sorted order is (R_i, R_j, R_k); thus the ordering relation between two blocks is a total ordering relation and the blocks can be sorted using this relation. The visibility order condition for the viewing category $(+,-)$ are shown in Fig. 7(b). The visibility order for blocks for categories $(-,-)$ and $(-,+)$ is reverse of visibility order for categories $(+,+)$ and $(+,-)$, respectively.

For 3D data sets, (x_v, y_v, z_v) are viewing direction components in x, y, and z directions. All the viewing directions fall under one of the eight categories $(+,+,+)$, $(+,+,-)$, $(+,-,+)$, $(-,+,+)$, $(+,-,-)$, $(-,+,-)$, $(-,-,+)$, $(-,-,-)$. Note that viewing categories $(-,-,-)$, $(+,-,-)$, $(-,+,-)$, and $(-,-,+)$ are opposite of $(+,+,+)$, $(-,+,+)$, $(+,-,+)$ and $(+,+,-)$, respectively. Therefore, we need to order all the blocks only for four distinct visibility orders $(+,+,+)$, $(-,+,+)$, $(+,-,+)$ and $(+,+,-)$. We state the following theorem which is an extension of theorem 2 for 3-dimensions.

Theorem 2: Any two rays r_1 and r_2 in a 3D-viewing category, passing through the set of blocks $\mathcal{B}_1$ and $\mathcal{B}_2$, respectively, pass through blocks in $\mathcal{B}$ in same order, where $\mathcal{B} = \mathcal{B}_1 \bigcap \mathcal{B}_2$ and $|\mathcal{B}| \geq 2$.

The proof for above theorem is generalization of theorem 1.

The condition for viewing category $(-,+,+)$ can be stated as follows. For other viewing categories similar conditions can be derived.

```
if $X_s(B_j) > X_e(B_i)$ then $(B_j, B_i)$
else if $X_e(B_j) < X_s(B_i)$ then $(B_i, B_j)$
else if $Y_e(B_j) < Y_s(B_i)$ then $(B_j, B_i)$
else if $Y_s(B_j) > Y_e(B_i)$ then $(B_i, B_j)$
else if $Z_e(B_j) > Z_s(B_i)$ then $(B_j, B_i)$
else $(B_i, B_j)$
```

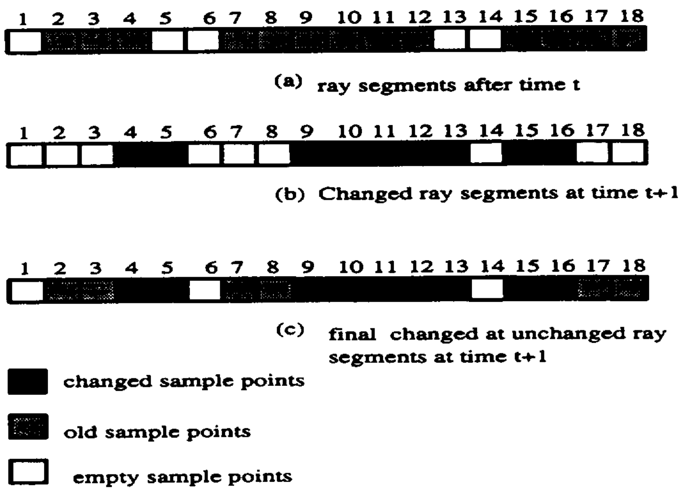

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
(a) ray segments after time t
(b) Changed ray segments at time t+1
(c) final changed at unchanged ray segments at time t+1
changed sample points
old sample points
empty sample points

Figure 8: (a) Ray segments at time $t = 1$. (b) Changed ray segments at time $t = 1$. (c) Final ray segments for ray r.

Similar sorting techniques can be applied for perspective projection [6].

Merging

In this section, we describe the method of combining segments of a modified ray r at time t, with its changed ray segments at time $t + 1$ resulting in a sequence of ray segments useful in computing color value of the ray. We illustrate this process by an example.

Assume that for ray r, the sample points at $t = 0$ are shown in Fig. 8(a). The ray segments $(18, 15), (12, 7)$, and $(4, 2)$ are computed using Eqn. 1 and the final ray value of ray r at time stamp $t = 0$ is computed using Eqn. 9. Assume that at time $t = 1$, the changed ray segments for ray r are $(16, 15), (13, 9), (5, 4)$, as shown in Fig. 8(b). These changed segments of ray r have to be combined with the ray segments of ray r at time $t = 0$, in order to compute the final color value of the ray. For example, ray segment $(18, 15)$ at time $t = 0$ is overlapped by the changed ray $(16, 15)$ at time $t = 1$. These two ray segments are combined by dividing ray segment $(18, 15)$ into ray segments $(18, 17)$ and $(16, 15)$. The segment $(16, 15)$ at $t = 0$ is completely contained in segment $(16, 15)$ at $t = 1$, therefore, the new value of the segment $(16, 15)$ is the same as the segment value at $t = 1$. Therefore, merging of segments $(18, 15)$ at time $t = 0$ and $(16, 15)$ at $t = 1$ results in segments $(18, 17)$ and $(16, 15)$. Ray segment $(12, 7)$ at $t = 0$ is combined with ray segment $(13, 9)$ at $t = 1$ in the similar fashion, resulting in a new segment list $(13, 9)$ and $(8, 7)$, where segment $(13, 9)$ contains new values of sample points at $t = 1$. Similarly, ray segment $(4, 2)$ at time $t = 0$ and segment $(5, 4)$ at time $t = 0$ are combined, resulting in segments $(5, 4), (3, 2)$ to be in the segment list of ray r. Therefore, the final ray segments list for ray r for time $t = 1$ is $(18, 17), (16, 15), (13, 9), (8, 7), (5, 4), (3, 2)$. This list is used in computing the final color value of ray r for time $t = 1$ using Eqn. 9. At time $t = 2$, this list is used as above for merging with changed segment list for ray r at time $t = 2$.

The worst case complexity of this step is $O(n)$, when number of samples are $O(n)$ which form $O(n)$

changed ray segments. The color and opacity values of each segment are computed using Eqn. 8.

IMPLEMENTATION

We have implemented the proposed algorithm on Sun-sparc (40 MHz) and used electrical wave propagation data set for testing the performance of our algorithm. In our implementation, in the beginning each ray is assigned a buffer for storing accumulated color and opacity values at each sample point along a ray. The size of the buffer is the same as the maximum number of sample points in a ray which is computed by intersecting the ray with the planes bounding the volume data set. The ith location in the buffer contain the accumulated color and opacity values for non-empty sample points j through i ($j \leq i$), where the $(j-1)$th sample point is empty. For example, in Fig. 8, the 10th location of the buffer for ray r contains accumulated color and opacity values of sample points 7 through 10 at time $t = 0$. But in order to compute the color and opacity values of segment $(12, 7)$, Eqn. 8 is used which requires the starting color and opacity values for this segment. At time $t = 0$, the starting color and opacity values of all the segments are 0.0. and 1.0, respectively. The starting color and opacity values of each segment are stored in a buffer called *head*, attached with each ray segment. At the next time stamp, new accumulated values may be written to a ray buffer and merging step may change starting sample point of a previous ray segment. For example in Fig. 8(b), merging segment $(18, 15)$ and the new segment $(16, 15)$ at time $t = 1$, results in segments $(18, 17)$ and $(16, 15)$. To compute the color and opacity values of segment $(18, 17)$, the accumulated color and opacity values of the 16th sample point at time $t = 0$ are required. Therefore, the accumulated color and opacity values of the 16th sample point are the starting values for segment $(18, 17)$. The new starting value of a segment is copied from the buffer *tail* attached to each ray segment. In this *tail* buffer, the color and opacity value at the last sample point of the segment at the previous time stamp is stored. This is done in merging step. As an example in Fig. 8(c) , the *tail* value of segment $(16, 15)$ is the color and opacity value of the 16th sample at time $t = 0$. These color and opacity values are the starting values for segment $(18, 17)$.

Therefore, each ray segment has following structure.

```
struct col_op {
float color,opacity; /* color and opacity values */
};
struct ray_seg {
    int start, end;  /* starting and ending sample point
                        indices for ray segment        */
    struct col_op head, tail;
};
```

The color and opacity values of each new segment are computed using Eqn. 8 which uses the last sample point values and *head* values of the segment.

Table 2 gives the rendering time at different time stamps. Fig. 9 depicts some of the images generated at different time stamps. It is clear from this table that as the number of changed block increases,

Table 2: Block sorting, projection, merging and total rendering time for electrical wave propagation data sets. All times are in seconds.

Time Stamp	Projection Time	Sorting Time	Merging and Rendering Time
10	0.04	0.46	0.11
20	0.14	1.28	0.26
30	0.20	1.57	0.40
40	0.19	1.56	0.46
50	0.16	1.33	0.45
60	0.05	0.50	0.27
70	0.02	0.22	0.15
80	0.01	0.14	0.14
90	0.00	0.03	0.07

projection time increases. An image is generated in about 2.0 seconds for a 5% change in data points. Multiple images can be generated in a second using high speed workstations. In the case of sequence of time varying data sets where changed data points are more clustered, further speed up is expected as the number of changed blocks will be relatively small. The advantage of this technique of rendering time varying data sets is that the user can generate images of any resolution unlike in the method proposed by Shen and Johnson [16], where image resolution is the same as that of volume data sets.

CONCLUSION

In this paper, We have presented a storage scheme for time varying data sets which achieves a 98.2% savings of memory space for electrical wave propagation data sets. We have then presented an algorithm for direct rendering of compact data structures. The implementation results suggest that the performance of this algorithm is comparable to the existing algorithm by Shen and Johnson [16], but our algorithm allows any arbitrary image resolution. The performance of the proposed algorithm for time varying data sets is expected to be better for data sets containing more clustered changed data points such as in data sets recording neuron activities in the brain [1, 15].

References

[1] J. D. Cohen, D. C. Noll, and W. Schneider. Functional magnetic resonance imaging: Overview and methods for psychological research. *Behavior Research Methods, Instruments and Computers*, 25(2):101–113, 1993.

[2] R. Crawfis and N. Max. Texture splats for 3d scalar and vector field visualization. In *Proceedings of Visualization*, pages 261-265, 1993.

[3] L. Ferrari, P. V. Sankar, and J. Sklansky. Minimal rectangular partitions of digitalized blobs. *Computer Vision, Graphics and Image Processing*, 28:58–71, 1984.

[4] P. Ghapure and C. R¿ Johnson. A 3d cellular automata model of the heart. In *Proceedings of 15th Annual IEEE EMBS Int. Conf.*, 1993.

[5] A. Globus, C. Levit, and T. Lasinski. A tool for visualizing the topology of three dimensional vector fields. In *Proceedings of Visualization*, pages 33–40, 1991.

[6] Vineet Goel. *Volume Rendering Algorithms and Architectures.* Ph.D. Dissertation, Univ. of Central Florida, Orlando, FL, 1995.

[7] J. L. Halman and L. Hesselink. Visualizing vector field topology in fluid flows. *IEEE Computer Graphics and Applications*, 11(3):36–46, 1991.

[8] H. Imai and T. Asano. Efficient algorithms for geometric graph search problems. *SIAM J. of Computing*, 15(2):478–494, May 1986.

[9] M. Levoy. Display of surface from volume data. *IEEE Computer Graphics and Applications*, 8(3):29-37, May 1988.

[10] M. Levoy. Efficient ray tracing of volume data. *ACM Trans. on Graphics*, 9(3), July 1990.

[11] K. L. Ma and P. J. Smith. Virtual smoke: An interactive 3d flow visualization technique. In *Proceedings of Visualization*, pages 46–53, 1992.

[12] N. Max, B. Becker, and R. Crawfis. Flow volume for interactive vector field visualization. In *Proceedings of Visualization*, pages 19–23, 1993.

[13] T. Ohtsuki. Minimum dissection of rectilinear regions. In *IEEE International Symposium on Circuits and Systems*, pages 1210–1213, Rome, 1982.

[14] T. Porter and T. Duff. Compositing digital images. *Computer Graphics*, pages 253–259, July 1984.

[15] W. Schnieder, D. C. Noll, and J. D. Cohen. Functional topographic mapping of the cortical ribbon in human vision and conventional mri scanner. *Nature*, 365:150–153, 1993.

[16] Han Wei Shen and Christopher R. Johnson. Differential volume rendering: A fast volume visualization technique for flow animation. In *Proceedings of Visualization*, pages 180–187, 1994.

[17] P. G. Swann and S. K. Semwal. Volume rendering of flow visualization point data. In *Proceedings of Visualization*, pages 25–32, 1991.

[18] C. Upson and M. Keeler. The v-buffer: Visible volume rendering. *Computer Graphics*, 22(4):59–64, July 1990.

[19] L. Westover. Footprint evaluation for volume rendering. *Computer Graphics*, 24(4):367–376, August 1990.

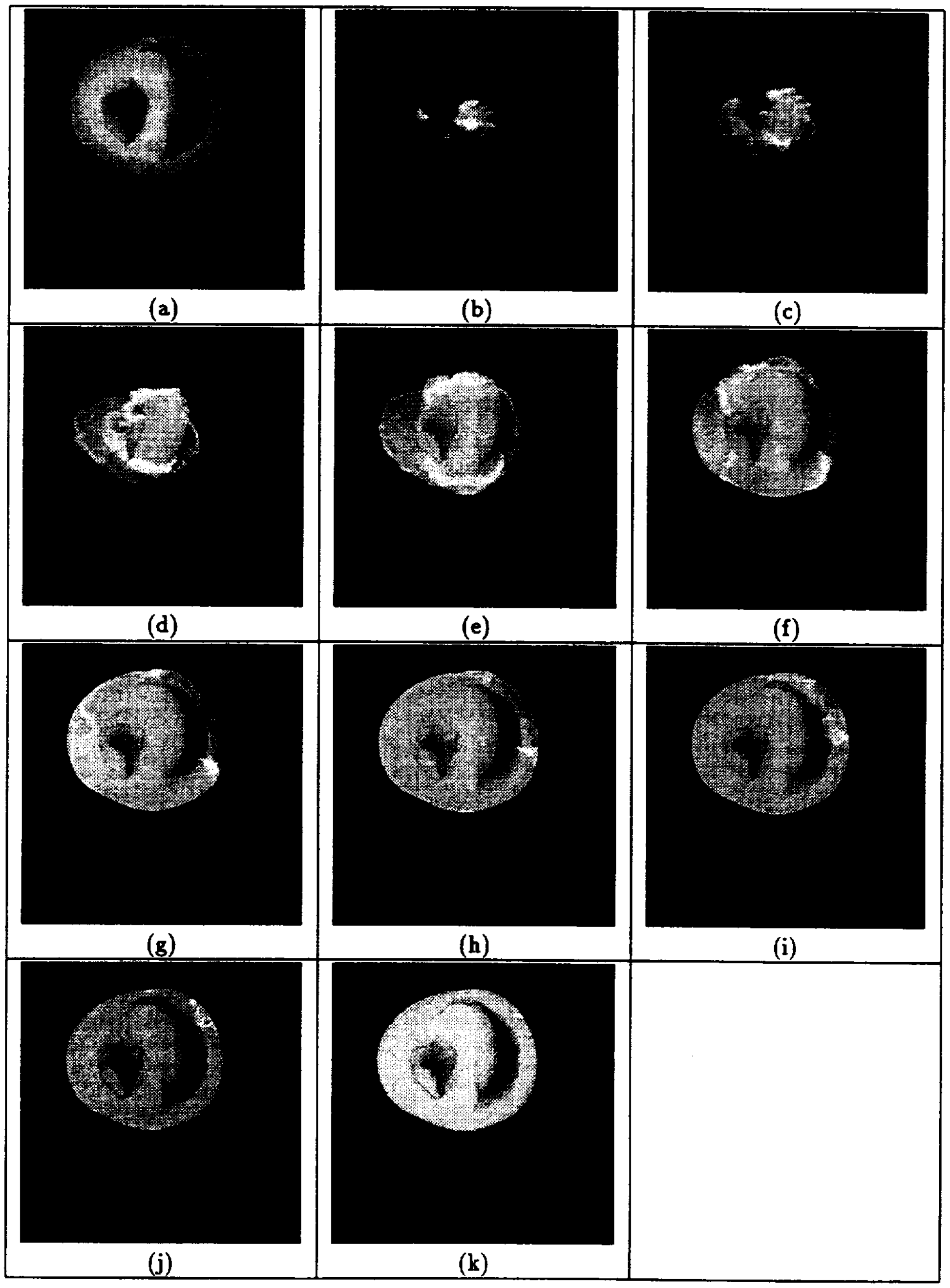


Figure 9: (a) Image at time stamp 0, (b) Image at time stamp 10, (c) Image at time stamp 20, (d) Image at time stamp 30. (e) Image at time stamp 40, (f) Image at time stamp 50, (g) Image at time stamp 60, (h) Image at time stamp 70. (i) Image at time stamp 80, (j) Image at time stamp 90, (k) Image at time stamp 99.

Video Presentations

Eddy Visualization for Various Ocean Models

Robert J. Moorhead, Zhifan Zhu, Kelly Gaither, and John VanderZwagg
Engineering Research Center
Mississippi State University

A feature detection algorithm is developed to extract eddies from a multi-year simulation of the ocean dynamics in the Pacific Ocean. The extraction algorithm finds the center of potential eddies using a topology-based approach. The extent of the eddy is determined by growing a geometric model until the local flow field and the tangent of the surface exceed a threshold. Potential eddies are then rejected on the basis of size and other physical parameters. The individual instances are then tracked over time. More eddies are then rejected based on a persistence threshold. The end result is approximately a thousand-fold decrease in data that must be visualized.

A movie of the resulting phenomena will be shown and the technique presented.

Visualizing Time-Varying Volume Data Using Parallel Computers

Kwan-Liu Ma
ICASE

Tzi-cker Chiueh
State University of New York at Stony Brook

Most previous research on parallel volume rendering on distributed-memory parallel architectures has ignored I/O and focused only on the rendering algorithms. This presentation describes the design of a strategy to reduce the I/O overhead in the overall rendering process for time-varying data by exploiting both inter- and intra-volume parallelism. Given a generic parallel volume renderer and an N-processor machine, there are three possible approaches to turn it into a parallel volume animator for time-varying data. The first approach simply runs the parallel volume renderer on the sequence of data sets one after another. At any point in time, the entire N-processor machine is dedicated to rendering a particular volume. Therefore, only the parallelism associated with rendering a single data volume, i.e. intra-volume parallelism, has been exploited. The second approach takes the exact opposite approach by rendering N data volumes simultaneously, each on one processor. This approach thus only exploits inter-volume parallelism. As the optimal rendering performance can only be achieved by carefully balancing two performance factors: resource utilization efficiency and parallelism, both intra-volume and inter-volume parallelism should be exploited. The third approach thus takes the hybrid approach, in which N processor nodes are partitioned into M groups, each of which renders one data volume at a time. Tests have been performed on the 72-node Intel Paragon located in the NASA Langley Research Center by using a 64-time-step 128x128x128 voxels data set. Preliminary test results will be reported.

ARL Scientific Visualization

William Mattson and Rick Angelini
U. S. Army Research Laboratory

The Sense and Destroy Armor (SADARM) projectile is a smart munition containing two smaller submunitions that deploy, sense enemy targets, and fire a shaped charge toward the target. Wavefronts Dynamation and Advanced Visualizer were used to model and animate the SADARM submunitions and their deployment process. The animation was used in conjunction with the CFD calculations to depict the submunition collision solution. Computational fluid dynamics has been applied to determine the unsteady aerodynamics of the SADARM submunition separation. Flow Field computations for this multibody problem have been performed at a transonic Mach number, M=0.80 using the Cray C-90 at Waterways Experiment Station. The visualization was done using Bop-View.

A molecular dynamics model of shock and detonation of a two dimensional energetic crystal. The animation was used to visualize the method and mechanisms of the chemical reactions of detonation. The model consists of a relaxed heteronuclear crystal at rest in a herringbone lattice, struck by a flyer plate producing a shock front that is driven by chemical reactions immediately behind the front. Both the molecular dynamics and visualization were done with in house code.

An interior ballistics model of the stresses and strains of a projectile moving down a gun barrel. Projectile design is an expensive and hazardous engineering task. This study was done to study the causes and effects of unstable projectiles. The visualization was used to show the stresses and ringing of the gun tube caused by minor collisions between projectile and gun barrel. The model was prepared with Patran, and the finite element analysis was done by Dyna3D. Visualization was performed with Ensight.

Visualization of ERBE Data

Larry Matthias and Paula Detweiler
Lockheed Engineering and Sciences Company

This video presents an overview of the Earth Radiation Budget Experiment (ERBE) project for NASA Langley Research Center. It explains, using 3D animation, what types of data were collected from the satellite instruments and the major scientific contributions of the project to atmospheric science.

This video visualizes the following data:

- Longwave Radiation
- Shortwave Radiation
- Computed Net Radiation
- Cloud Forcing

Volumetric Visualization Techniques Applied to the Time Evolution of a Turbulent Supersonic Flow over a Flat Plate at Mach 2.25

James Patten and Gordon Erlebacher
ICASE

In this movie, we demonstrate volumetric visualization techniques applied to the time evolution of a turbulent supersonic flow over a flat plate at Mach 2.25. We plot the vorticity magnitude to help visualize the bursting phenomena, which occurs very close to the wall.

SVP Time-Dependent Visualization Selections

Mary Vickerman
NASA Lewis Research Center

This video shows clips of various time-dependent visualizations that we have created using a software library developed at NASA Lewis called SVP (Scientific Visualization Package). The visualizations cover a range of research areas including: CFD, materials properties, solid lubricants, and molecular dynamics.

Virtual Facilities: Visualizing Experimental Data

Richard J. Schwartz
ViGYAN Inc.

Non-intrusive Global Measurement Systems, such as Particle Image Velocimetry, Pressure and Temperature Sensitive Paints, and Doppler Global Velocimetry are examples of technologies available today to provide massive quantities of time varying information about flow over a large surface area or volume. The resultant data sets can be overwhelming, making traditional presentation methods, such as graphs and tables, inadequate for the task. However, data sets can be displayed by computer animations. By utilizing image processing and three-dimensional computer modeling software, a virtual environment can be created to show the data in situ. Therefore, a data set can be visualized along with the test environment, namely a wind tunnel model and the wind tunnel itself. The technique described has been successfully implemented using PC-based computers with commercial three-dimensional modeling and rendering software.

Applications of Parallel Rendering for Visualizing Time-Varying Data

Thomas W. Crockett
ICASE

Richard G. Wilmoth
NASA Langley Research Center

David C. Banks
Mississippi State University

Bart A. Singer
High Technology Corporation

William J. Bene
Old Dominion University /
Computer Sciences Corporation

Patricia J. Crossno
University of New Mexico /
Sandia National Laboratories

Over the last few years, a number of researchers in the field of computer graphics have been exploring the use of parallel supercomputers for image generation. This work has progressed to the point that it is now being applied to a variety of visualization problems involving computationally-intensive simulations and large-scale datasets. In this video, we briefly introduce the concept of parallel rendering, and describe several scenarios in which it may be useful for visualizing time-varying data. We illustrate our discussion with several animations which were produced using PGL, a parallel graphics library under development at ICASE. The examples include visualizations of rarefied and turbulent flowfields and an explosive welding process.

DeVise -- A Tool for Visually Querying Sequence Data

M. Cheng, M. Livny, and R. Ramakrishnan
Computer Sciences Department
University of Wisconsin—Madison

Visually exploring sequence data, in particular, time-series data, is important in many domains. DeVise is a tool developed at UW-Madison that supports the concept of a visual query over such data. The novel features of DeVise include its flexibility in designing visual presentations of the data, and its ability to deal with very large datasets by transparently bringing in pages from disk to main memory as needed. In this talk, I will motivate and describe our work in this area, with several examples that use DeVise.

A High Resolution North Atlantic Ocean Circulation Calculation

Matthew O'Keefe
University of Minnesota

In our new Laboratory for Computational Science and Engineering we are visualizing very high resolution ocean circulation calculations (0.08 degrees). I would like to show a variety of animations of surface temperature, vorticity, salinity that are primarily 2D and also some 3D fly-throughs of large time-dependent ocean flows. We are working on a quaternion-based automatic fly-through movie generation tool that focused on 3D flows.

Our group has designed a PowerWall display system that allows us to paste together multiple high-resolution screens yielding a single very high resolution display. We used this technology to display a variety of high-res time dependent flows at Supercomputing 95.

Visualization of Numerical Unsteady Flows

David Lane
Computer Sciences Corporation

Numerical simulations of complex 3D time-dependent (unsteady) flows are becoming increasingly feasible because of the progress in computing systems. Unfortunately, many existing flow visualization systems were developed for time-independent (steady) solutions and do not adequately depict solutions from unsteady flow simulations. Furthermore, most systems only handle one time step of the solutions individually and do not consider the time-dependent nature of the solutions. For example, instantaneous streamlines are computed by tracking the particles using one time step of the solution. However, for streaklines and timelines, particles need to be tracked through all time steps.

For a complex 3D flow simulation, it is common to generate a grid system with several millions of grid points and to have tens of thousands of time steps. The disk requirement for storing the flow data can easily be tens of gigabytes. Visualizing solutions of this magnitude is a challenging problem with today's computer hardware technology. Even interactive visualization of one time step of the flow data can be a problem for some existing flow visualization systems because of the size of the grid.

The Unsteady Flow Analysis Toolkit (UFAT) developed at NASA Ames Research Center to compute time-dependent particle traces from unsteady CFD solutions is described. The system computes particle traces (streaklines) by integrating through the time steps. This system has been used by several NASA scientists to visualize their CFD time-dependent solutions. The flow visualization results are shown.

Displaying the Rotation of a Vortex Tube

David Banks
Mississippi State University

Tom Crockett
ICASE

Will Bene
Computer Sciences Corporation

Bart Singer
High Technology Corporation

This video visualizes the development of vortical structures in an unsteady flow. A direct numerical simulation provides the raw data, and the vortex core is identified by a predictor-gradient scheme using vorticity and pressure-gradients. A vortex tube can be reconstructed for graphical display on a workstation, but interpolating the tubes geometry between given time steps requires substantial computation. We are using a parallel machine both to interpolate the gross geometry of the tube between time steps and also to deform the surface shape of the tube by a small amount. The deformation (in the form of small grooves) provides a set of visual landmarks to indicate the direction and magnitude of rotation on the surface of the tube. This visualization technique substantially assists in revealing the dynamics of the tube and on the tube.

Time-Varying Datasets

3-D Unsteady Navier-Stokes Computations of Flow Through a Ducted-Propeller Blade Passage

Robert T. Biedron
Analytical Services & Materials, Inc.

http://www.icase.edu/workshops/vtvd/datasets/rotorblade.html

An upwind, approximate factorization scheme was used to solve the thin-layer Navier-Stokes equations for the flow through a single passage in a ducted propeller. The passage consists of a rotating fan blade and a stationary guide vane, bounded on the bottom by the hub and on the top by the cowl. Periodicity of 22.5 degrees is enforced in the circumferential direction, yielding a computation representative of a ducted propeller with 16 fan blades and 16 guide vanes. A patched/overset grid system comprised of 4 structured blocks was used for the computation.

The dataset displays the relative motion of the fan blades and guide vanes modeled through the use of a "sliding block" patched-grid interface. The location of the fan blade relative to the guide vane is recomputed at each iteration. The dataset depicts the solution at every fifth time step of the computation.

The Aerospace Flight Vehicle Pitch-Over Dataset

Bill Kleb
NASA Langley Research Center

http://www.icase.edu/workshops/vtvd/datasets/avpitchup.html

The dataset is a computational fluid dynamic (CFD) simulation of a proposed vertical take-off/ vertical landing, single-stage to orbit flight vehicle. This is one of the many vehicle concepts being considered as a replacement for the Space Shuttle. Just after re-entry into the earth's atmosphere, but prior to landing vertically, the vehicle must reverse it's direction of flight from being nose-first to base-first so that it can fire its engines for landing. This dataset represents a simulation of this pitch-over maneuver. The CFD code used to generate the dataset is a modified version of the 3D3U code written by J. T. Batina of NASA Langley. The code was originally created to enable simulation of transonic, unsteady flowfields about wings and wing-bodies include aeroelastic effects.

Unsteady Viscous Flow Over 3-D Rectangular Cavity

N. Duane Melson
NASA Langley Research Center

http://www.icase.edu/workshops/vtvd/datasets/cavity.html

An iterative-implicit thin-layer Navier-Stokes solver with multigrid acceleration was used to calculate the self-excited unsteady laminar flow over and in a rectangular shallow cavity in a flat plate. A two-block structured grid was used for the calculation.

PARTICIPANTS

Marc Abrams
Department of Computer Science
VPI & SU
Blacksburg, VA 24061-0106
U.S.A.
(540) 231-8457
abrams@vt.edu

Mary Adams
Mail Stop 128
NASA Langley Research Center
Hampton, VA 23681-0001
U.S.A.
(804) 864-2314
m.s.adams@larc.nasa.gov

Rick Angelini
U. S. Army Research Laboratory
AMSRL-SC-CC
High Performance Computing Division
Aberdeen Proving Ground, MD 21005-5067
U.S.A.
(410) 278-6266
angel@arl.mil

David Banks
Department of Computer Science
Mississippi State University
Mississippi State, MS 39762
U.S.A.
(601) 325-2756
banks@cs.msstate.edu

R. Daniel Bergeron
Department of Computer Science
University of New Hampshire
Kingsbury Hall
Durham, NH 03824
U.S.A.
(603) 862-3778
rdb@cs.unh.edu

Daryl Bonhaus
Mail Stop 128
NASA Langley Research Center
Hampton, VA 23681-0001
U.S.A.
(804) 864-2293
d.l.bonhaus@larc.nasa.gov

John T. Bowen
Mail Stop 125
NASA Langley Research Center
Hampton, VA 23681-0001
U.S.A.
(804) 864-6725
j.t.bowen@larc.nasa.gov

Steve Bryson
Computer Sciences Corporation
Mail Stop T-27A-1
NASA Ames Research Center
Moffett Field, CA 94035-1000
U.S.A.
(415) 604-4524
bryson@nas.nasa.gov

Judy Busby
David Taylor Research Center
Code 542
CDNSWC DTRC
Bethesda, MD 20084
U.S.A.
(301) 227-3274
busby@dawgs.dt.navy.mil

Rod Coleman
David Taylor Model Basin
Code 542
NSWC/Carderock
Bethesda, MD 20084
U.S.A.
(301) 227-1930
coleman@oasys.dt.navy.mil

Robert Z. Conrow
ITT Federal Services Corporation
Mail Stop AP220
P. O. Box 5728
Vandenberg Air Force Base, CA 93437
U.S.A.
(805) 734-8232
zeb@tecnetl.jcte.jcs.mil

Tom Crockett
ICASE
Mail Stop 132C
NASA Langley Research Center
Hampton, VA 23681-0001
U.S.A.
(804) 864-2182
tom@icase.edu

Eric Davies
Barrodale Computing Services
Suite 8 - 1560 Church Avenue
Victoria, B.C.
Canada V8P 2H1
ejdavies@barrodale.com

David E. Edwards
United Technologies Research Center
411 Silver Lane
Mail Stop 129-13
East Hartford, CT 06108
U.S.A.
(203) 727-7518
edwardde@utrc.utc.com

John Edwards
Mail Stop 242
NASA Langley Research Center
Hampton, VA 23681-0001
U.S.A.
(804) 864-2273
j.w.edwards@larc.nasa.gov

Gordon Erlebacher
ICASE
Mail Stop 132C
NASA Langley Research Center
Hampton, VA 23681-0001
U.S.A.
(804) 864-6781
erlebach@icase.edu

Charles Fenno
Mail Stop 463
NASA Langley Research Center
Hampton, VA 23681-0001
U.S.A.
(804) 864-3579
c.c.fenno@larc.nasa.gov

Michael Gerald-Yamasaki
Mail Stop T27A-1
NASA Ames Research Center
Moffett Field, CA 94035-1000
U.S.A.
(415) 604-4412
yamo@nas.nasa.gov

Roger Hahn
Century Computing, Inc.
8101 Sandy Spring Road
Laurel, MD 20707
U.S.A.
(301) 953-3330
rhahn@cen.com

Robert Haimes
Massachusetts Institute of Technology
Room 37-467
77 Massachusetts Avenue
Cambridge, MA 02139
U.S.A.
(617) 253-7518
haimes@orville.mit.edu

Jennifer Hare
U. S. Army Research Laboratory
AMSRL-SC-CC
High Performance Computing Division
Aberdeen Proving Ground, MD 21005-5067
U.S.A.
(410) 278-9149
jen@arl.mil

Lambertus Hesselink
Department of Aeronautics and Astronautics
Stanford University
Durand Building - Room 353
Stanford, CA 94305-4035
U.S.A.
(415) 723-4850
bert@kaos.stanford.edu

Nilan Karunaratne
ICASE
Mail Stop 132C
NASA Langley Research Center
Hampton, VA 23681-0001
U.S.A.
(804) 864-2173
karunnc@icase.edu

Patricia Kerr
Mail Stop 125
NASA Langley Research Center
Hampton, VA 23681-0001
U.S.A.
(804) 864-5782
p.a.kerr@larc.nasa.gov

William Kleb
Mail Stop 408A
NASA Langley Research Center
Hampton, VA 23681-0001
U.S.A.
(804) 864-4364
w.l.kleb@larc.nasa.gov

Steve Kramer
Intelligent Light
1099 Wall Street West
Suite 387
Lyndhurst, NJ 07071
U.S.A.
(201) 460-4700
kramer@ilight.com

Don Krieger
Children's Hospital of Pittsburgh
University of Pittsburgh, Room 3699
3705 Fifth Avenue
Pittsburgh, PA 15213
U.S.A.
(412) 692-5093
don@neuronet.pitt.edu

Jay Lambiotte
Mail Stop 125
NASA Langley Research Center
Hampton, VA 23681-0001
U.S.A.
(804) 864-5794
j.j.lambiotte@larc.nasa.gov

David Lane
Computer Sciences Corporation
NASA Ames Research Center
Mail Stop T27A-2
Moffett Field, CA 94035
U.S.A.
(415) 604-4375
lane@nas.nasa.gov

Yu-Tai Lee
David Taylor Model Basin
Code 542
Naval Surface Warfare Center
Bethesda, MD 20084-5000
U.S.A.
(301) 227-1328
ylee@oasys.dt.navy.mil

Stephen Legensky
Intelligent Light
1099 Wall Street West
Suite 387
Lyndhurst, NJ 07071
U.S.A.
(201) 460-4700
sml@ilight.com

David Leone
Sterling Software
303 Twin Dolphin Drive
Suite 510
Redwood City, CA 94065-1417
U.S.A.
(415) 802-7100
daveleone@sterling.com

Kwan-Liu Ma
ICASE
Mail Stop 132C
NASA Langley Research Center
Hampton, VA 23681-0001
U.S.A.
(804) 864-2195
kma@icase.edu

Wayne Mastin
Mail Stop 125
NASA Langley Research Center
Hampton, VA 23681-0001
U.S.A.
(804) 864-5781
cwm@geolab.larc.nasa.gov

Anup Mathur
Department of Computer Science
VPI & SU
Blacksburg, VA 24061-0106
U.S.A.
(540) 953-2800, ext. 3038
mathur@csgrad.cs.vt.edu

Larry Matthias
Lockheed Engineering & Sciences Company
144 Research Drive
Hampton, VA 23666
U.S.A.
(804) 766-9726
l.e.matthias@larc.nasa.gov

William Mattson
U. S. Army Research Laboratory
AMSRL-SC-CC
High Performance Computing Division
Aberdeen Proving Ground, MD 21005-5067
U.S.A.
(410) 278-7502
wmattson@arl.mil

Nelson Max
Lawrence Livermore National Laboratory
Mail Stop L-301
7000 East Avenue
Livermore, CA 94550
U.S.A.
(510) 422-4074
max2@llnl.gov

Piyush Mehrotra
ICASE
Mail Stop 132C
NASA Langley Research Center
Hampton, VA 23681-0001
U.S.A.
(804) 864-2188
pm@icase.edu

N. Duane Melson
Mail Stop 128
NASA Langley Research Center
Hampton, VA 23681-0001
U.S.A.
(804) 864-2222
n.d.melson@larc.nasa.gov

Robert Moorhead
ERC
Mississippi State University
P. O. Box 6176
Mississippi State, MS 39762
U.S.A.
(601) 325-2850
rjm@erc.msstate.edu

Amar Mukherjee
Department of Computer Science
University of Central Florida
Orlando, FL 32816
U.S.A.
(407) 823-2763
amar@cs.ucf.edu

Richard Muntz
Department of Computer Science
University of California - Los Angeles
3732 Boelter Hall
Los Angeles, CA 90095-1596
U.S.A.
(310) 825-3546
muntz@cs.ucla.edu

David Myrick
Computer Sciences Corporation
Mail Stop 157D
NASA Langley Research Center
Hampton, VA 23681-0001
U.S.A.
(804) 864-9230
myrick@magician.larc.nasa.gov

Robert Nowak
Mail Stop 408A
NASA Langley Research Center
Hampton, VA 23681-0001
U.S.A.
(804) 864-1391
r.j.nowak@larc.nasa.gov

Matthew O'Keefe
Department of Electrical Engineering
University of Minnesota
4-174, EE/CSci Building
Minneapolis, MN 55455
U.S.A.
(612) 625-6306
okeefe@mountains.ee.umn.edu

Tom Palmer
CEI, Inc.
P. O. Box 14306
Research Triangle Park, NC 27709
U.S.A.
(919) 481-4301
palmer@ceintl.com

Alan Pope
Mail Stop 152
NASA Langley Research Center
Hampton, VA 23681-0001
U.S.A.
(804) 864-6642
a.t.pope@larc.nasa.gov

Mary-Anne Posenau
Mail Stop 125
NASA Langley Research Center
Hampton, VA 23681-0001
U.S.A.
(804) 864-6717
m.a.k.posenau@larc.nasa.gov

Raghu Ramakrishnan
Department of Computer Science
University of Wisconsin - Madison
1210 W. Dayton
Madison, WI 53706
U.S.A.
(608) 262-9759
raghu@cs.wisc.edu

Daniel A. Reed
Department of Computer Science
University of Illinois
1304 W. Springfield Avenue
Urbana, IL 61801
U.S.A.
(217) 333-3807
reed@cs.uiuc.edu

Wei-Min Ren
Siemens Automotive
615 Bland Boulevard
Newport News, VA 23602
U.S.A.
(804) 875-7454

Randy Ribler
Department of Computer Science
VPI & SU
Blacksburg, VA 24061-0106
U.S.A.
(540) 231-6931
ribler@csgrad.cs.vt.edu

Vincent Roland
Mail Stop 125
NASA Langley Research Center
Hampton, VA 23681-0001
U.S.A.
(804) 864-6721
v.r.roland@larc.nasa.gov

Kevin Russo
Naval Research Laboratory
Code 5580
4555 Overlook Avenue
Washington, DC 20375-5337
U.S.A.
(202) 767-3879
russo@ait.nrl.navy.mil

James R. Schiess
Mail Stop 125
NASA Langley Research Center
Hampton, VA 23681-0001
U.S.A.
(804) 864-6718
j.r.schiess@larc.nasa.gov

Richard Schwartz
Vigyan, Inc.
Mail Stop 493
NASA Langley Research Center
Hampton, VA 23681-0001
U.S.A.
(804) 864-4594
r.s.schwartz@larc.nasa.gov

Will Scullin
Department of Computer Science
University of Illinois
1304 W. Springfield Avenue
Urbana, IL 61801
U.S.A.
(217) 333-1515
scullin@cs.uiuc.edu

Kurt Severance
Mail Stop 125
NASA Langley Research Center
Hampton, VA 23681-0001
U.S.A.
(804) 864-6715
k.severance@larc.nasa.gov

Robert Smith
Mail Stop 125
NASA Langley Research Center
Hampton, VA 23681-0001
U.S.A.
(804) 864-5774
bobs@bobsun.larc.nasa.agov

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE January 1996	3. REPORT TYPE AND DATES COVERED Conference Publication

4. TITLE AND SUBTITLE
ICASE/LaRC Symposium on Visualizing Time-Varying Data

5. FUNDING NUMBERS
C NAS1-19480
505-90-52-01

6. AUTHOR(S)
D. C. Banks, T. W. Crockett, and K. Stacy, editors

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NASA Langley Research Center
Hampton, VA 23681-0001

8. PERFORMING ORGANIZATION REPORT NUMBER
L-17555

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001
and
Institute for Computer Applications in Science and Engineering (ICASE)
Hampton, VA 23681-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA CP-3321

11. SUPPLEMENTARY NOTES
Langley Technical Monitor: Dennis M. Bushnell
Final Report

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified–Unlimited

Subject Category 60,61

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
Time-varying datasets present difficult problems for both analysis and visualization. For example, the data may be terabytes in size, distributed across mass storage systems at several sites, with time scales ranging from femtoseconds to eons. In response to these challenges, ICASE and NASA Langley Research Center, in cooperation with ACM SIGGRAPH, organized the first symposium on Visualizing Time-Varying Data. The purpose was to bring the producers of time-varying data together with visualization specialists to assess open issues in the field, to present new solutions, and to encourage collaborative problem-solving.

These proceedings contain the peer-reviewed papers which were presented at the Symposium. They cover a broad range of topics, from methods for modeling and compressing data to systems for visualizing CFD simulations and the World Wide Web traffic. Because the subject matter is inherently dynamic, a paper proceedings cannot adequately convey all aspects of the work. The accompanying video proceedings provide additional context for several of the papers.

14. SUBJECT TERMS
Visualization; Time-varying data

15. NUMBER OF PAGES
131

16. PRICE CODE
A07

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

Kathryn Stacy
Mail Stop 125
NASA Langley Research Center
Hampton, VA 23681-0001
U.S.A.
(804) 864-6719
k.stacy@larc.nasa.gov

James Stegeman
Mail Stop 5-11
NASA Lewis Research Center
Cleveland, OH 44135
U.S.A.
(216) 433-3389
stegeman@lerc.nasa.gov

Joe Thompson
Engineering Research Center for
Computational Field Simulation
Mississippi State University
Box 6176
Mississippi State, MS 39762
U.S.A.
(601) 325-7299
joe@erc.msstate.edu

Ronald Ungewitter
Rockwell International
6633 Canoga Avenue
P. O. Box 7922
Canoga Park, CA 91309-7922
U.S.A.
(818) 586-0531
r.j.ungewitter@rdyne.rockwell.com

Mary Vickerman
Mail Stop 142-5
NASA Lewis Research Center
21000 Brookpark Road
Cleveland, OH 44135
U.S.A.
(216) 433-5067
vickerman@lerc.nasa.gov

William von Ofenheim
Mail Stop 125
NASA Langley Research Center
Hampton, VA 23681-0001
U.S.A.
(804) 864-6712
w.h.c.vonofenheim@larc.nasa.gov

Thomas Whittaker
Sterling Software
MS T27A-2
NAS/NASA Ames Research Center
Moffett Field, CA 94037
U.S.A.
(415) 604-4451
tomw@nas.nasa.gov

Pak Wong
Department of Computer Science
University of New Hampshire
Kingsbury Hall
Durham, NH 03824
U.S.A.
(603) 862-3782
pcw@cs.unh.edu