A Software Model for Visualization of Large Unsteady 3-D CFD Results

Al Globus, Computer Sciences Corporation[1]
globus@nas.nasa.gov

Abstract

The largest unsteady 3-D computational fluid dynamics (CFD) results produced today exceed 100 gigabytes. Visualizing these large data sets stresses computational resources, requiring users and visualization systems to make trade-offs between time, space, and flexibility. Current CFD visualization systems make trade-offs which severely limit visualization of the largest unsteady 3-D CFD results. We propose a software model that breaks visualization processing into several stages: generation of solutions, extracts, graphic objects, scenes, and movies (explained below). Discarding data from early stages saves storage. Postponing decisions to later stages improves flexibility and speed. We also propose storing these data in a persistent object database in order to separate the logical and physical representations of the data and provide a consistent, clean interface between visualization software and data from all stages. Meta-data, such as performance information, may also be stored to support intelligent time vs. space vs. flexibility trade-offs. Ultimately, more optimal trade-offs expand the visualizations possible within constant hardware and wall clock time limits. This paper will describe, explain, and motivate the proposed model. In principle, the model should be of value in disciplines other than CFD.

I. Introduction

CFD scientists use flow solver software to generate vector and scalar fields describing fluid flow. These fields are called the solution. The solution is discretely sampled on spatial grids with enough sample points, called nodes, to capture the complex fluid physics. Capturing ever more detailed physics requires more and more nodes, increasing storage requirements. Unsteady solutions consist of a series of single-time solutions, called time steps. Adding the time dimension dramatically increases solution size.

1. This work is supported through NASA contract NAS2-12961.

CFD results are traditionally among the largest numerical data sets visualized. The largest 3-D unsteady data sets recently generated at NAS[1] range from 5,600 to 162,000 megabytes (see table 1).

Table 1: Some Unsteady CFD Simulations at NAS

Data Set	Grid Nodes	Time Steps	Disk Read Time[a] (sec/time step)	Total Size (megabytes)
Smith91	2,800,000	100	15.6	5,600
GB92	1,400,000	400	7.8	11,200
Atwood92	1,600,000	500	8.9	16,000
Chawla92	900,000	9,000	5.0	162,000

a. At 3.6 megabytes/second, the best performance achieved on workstations at NAS [Lam92].

Many visualization systems have been successfully used to examine steady-state CFD solutions [Bancroft90, Buning85, Legensky90, SGI92, Upson89, Wavefront91]. However, visualization of 3-D unsteady solutions is difficult because existing systems make performance trade-offs that assume much smaller quantities of data; *i.e.*, that the solution and grid fit in RAM. Unfortunately, no current computer has enough RAM to hold the solutions in table 1. When faced with data quantity problems, it is tempting to wait for hardware improvements. However, simulation codes typically use all of available memory for a single time-step, and then save hundreds or thousands of timesteps on disk. Since CFD codes usually have access to at least as much memory as visualization systems, the largest unsteady CFD results will not fit in RAM until the physics cannot be improved by adding grid nodes. This will not occur for quite some time.

1. The NAS (Numerical Aerodynamic Simulation) Systems Division at NASA Ames Research Center is a leading supercomputer facility dedicated to the study of computational aerosciences, particularly CFD [Bailey86].

The proposed visualization software model allows trade-offs to be customized to the visualization problem at hand; thus avoiding unnecessary data motion and processing. This should help make visualization of large unsteady solutions feasible. Figure 1 is a schematic of the model. A detailed description is found in section III.

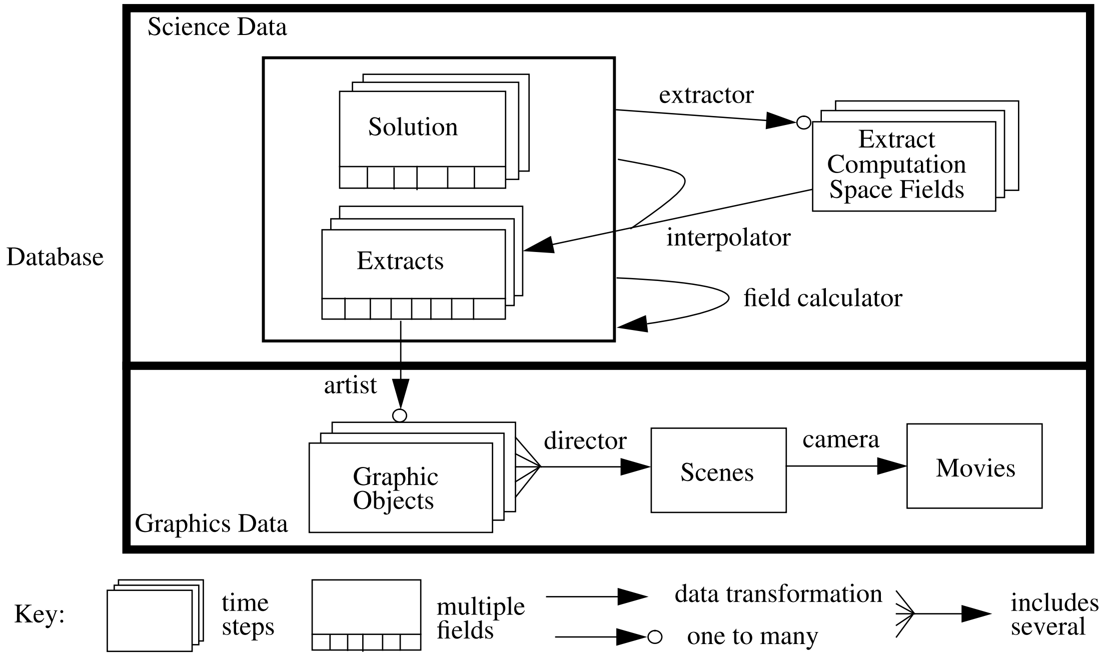


Figure 1: Proposed Model

There are several key points explained in section III:

- Extracts, *i.e.*, scientific data, are calculated from solutions rather than graphics as is usually the case. Extracts are further processed to generate the graphics.
- Computation space fields on extracts are stored to postpone decisions about the fields to visualize and to avoid redundant extraction steps.
- All data are collected into a persistent object database, along with performance data. The database maintains a clean interface between the logical schema and the physical representation on disk.

Most, although not all, of the elements proposed in this paper appear in the visualization literature or existing visualization systems, as indicated by the references in sections II and III. Some have been adopted by visualization system developers working with the author since the first draft of this paper appeared two years ago. The main contributions of this paper are in assembling the elements important to unsteady flow visualization into a coherent software model, carefully examining the time vs. space vs. flexibility trade-offs involved, and identifying weaknesses in existing approaches. We hope that this model will help developers create visualization systems that meet the challenges of unsteady flow visualization.

Overview

Section II reviews previous work. Section III describes the model. Section IV briefly discusses miscellaneous issues. Section V explores future possibilities. Section VI is a brief summary. Appendix A is a glossary of the terminology used in this paper. Appendix B discusses visualization techniques in the model's terms. Appendix C contains data relevant to space vs. time trade-offs.

II. Previous Work

Previous work can be divided into two categories: 1. papers on visualization models and, 2., visualization systems designed for unsteady flow. Visualization systems always have some underlying model, although it may not be explicit. Some of the literature straddles these two categories.

Visualization Models

Upson89 describes the data flow model of visualization implemented and popularized by AVS and other visualization systems. In this model, data flows between independent modules that implement pure functions. Most implementations have a visual programming 'language' to construct networks of modules. For unsteady flow visualization, data flow systems are difficult to use because looping is problematic and data management facilities are weak. This is pointed out in Lang91. Lang91 argues that a database is a key element in visualization systems and the data flow model has no concept of a database.

Lucas92 describes an implementation of the data flow model with the interesting property that the data are cached in what is effectively a database. In this system (the IBM Data Explorer), the data flow description is used as a program for a process that automatically makes decisions about what data to keep in RAM.

Smith89 proposed unsteady flow visualization by taking subsets of the data and visualizing the subsets rather than the entire data set. Smith, *et. al.*, write "... in many cases the storage of the solution set versus time at a few well-chosen cross-sections (*e.g.*, two-coordinate slices through the three-coordinate space) can capture the critical flow features." This is a limited special case of the extract concept presented in section III.

Butler89b and Haber91 developed a model of scientific visualization data. As Haber91 says, "Butler and Pendley observed that the mathematical notion of *fiber bundles* provides a useful abstraction for an object-oriented scientific data model of considerable generality [Butler89a, Butler89b]. This paper [Haber91] specializes and extends this idea to incorporate localized, piecewise field descriptions." Unfortunately, fiber bundles are difficult to describe and are not well understood within the visualization community. For these reasons, we do not use the fiber bundle model in this paper.

Campbell89 discusses the application of AI techniques to handling large-scale scientific databases. The focus in this paper is on sensor data rather than numerical experiments.

Two workshops on data models for scientific visualization deserve mention. The SIGGRAPH '90 Workshop Report on Data Structures and Access Software for Scientific Visualization [Treinish90] discusses a complex data model including calibration data, lab notes, data error characteristics, etc. The Visualization '91 Workshop Report on Scientific Visualization Environments [Butler91] proposes a scientific visualization reference model. Some of the ideas presented here were touched on at this workshop.

Visualization Systems

Hibbard90 describes the VIS-5D system designed for interactive visualization of unsteady 3-D data. VIS-5D uses a five dimensional array to store data in RAM.

Haimes91 describes the Visual3 software package used to visualize unsteady 3-D flows, particularly on unstructured grids. Performance on large 3-D unsteady solutions is currently inadequate.

To improve performance, processing was distributed over networked workstations and parts of our model (extracts) incorporated [Haimes94].

Dickenson91 presents excellent time control facilities, at least on small data sets.

Bryson91 uses virtual reality techniques to examine time-dependent vector fields, although data sets must fit in RAM. Bryson92 distributes processing between a workstation and a mini-supercomputer to increase RAM, and therefore problem, size.

Yamasaki92 distributes computation between supercomputer and workstation to visualize unsteady flow. Gerald-Yamasaki has made significant contributions to this paper and has used some of the ideas presented here in recent work.

Schroeder92 describes an object oriented visualization system where animation is an integral part of every object's capabilities.

Lane93 implements particle paths and streaklines in the Unsteady Flow Analysis Toolkit (UFAT) visualization system. Some ideas from this paper have been incorporated into UFAT.

Finally, an earlier version of this report appeared as an internal technical report [Globus92b] that was distributed to NAS visualization 'customers' over the Internet and via the World Wide Web.

III. The Model

The most generic visualization system processes raw data to generate images. Hopefully, these images illuminate the content of the raw data. We start with a diagram of a generic visualization system.

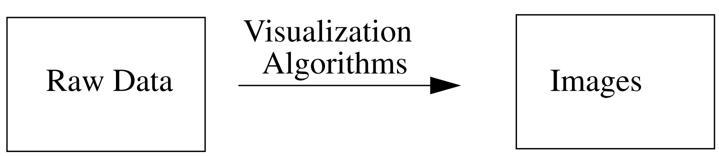


Figure 2: Visualization Systems Convert Raw Data into Images

Current CFD Visualization Systems

The next step in presenting the model is to describe current CFD visualization systems as a variant of the generic visualization model. Additional elements are then added one at a time, with explanations following a diagram, until the model is complete.

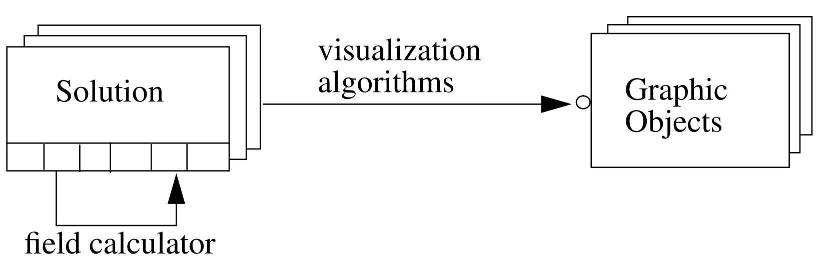


Figure 3: Typical CFD Visualization System

For CFD, the raw data to be visualized are solutions of partial differential equations modelling fluid flow. Remember that these solutions are represented by scalar and vector fields sampled on a time-space domain. At NAS, these fields are typically density, momentum and energy. Scientists frequently wish to examine additional fields derived from the scalar and vector fields solvers produce; so most CFD visualization systems use a field calculator to generate additional derived fields, such as pressure or vorticity, from solver output. These field calculations generally store derived fields in newly allocated memory. This trades storage and preprocessing time to increase speed when derived fields are interactively explored.

Images are very inflexible, *i.e.*, the viewpoint cannot be changed, so most visualization systems used in CFD create intermediate results we call *graphic objects*.[1] Graphic objects are usually converted to images by workstation hardware. A single time step of a graphic object is a set of vertices defining a set of surfaces, curves, and/or points combined with rendering information such as vertex colors and normal vectors. Graphic objects greatly increase flexibility because the viewpoint can be changed. Furthermore, graphic objects usually, but not always, require less space than images. The cost of using graphic objects is the time to display successive time steps. Even the fastest workstation hardware cannot often compete with a 30 frame per second video tape of a complex scene.

Extracts

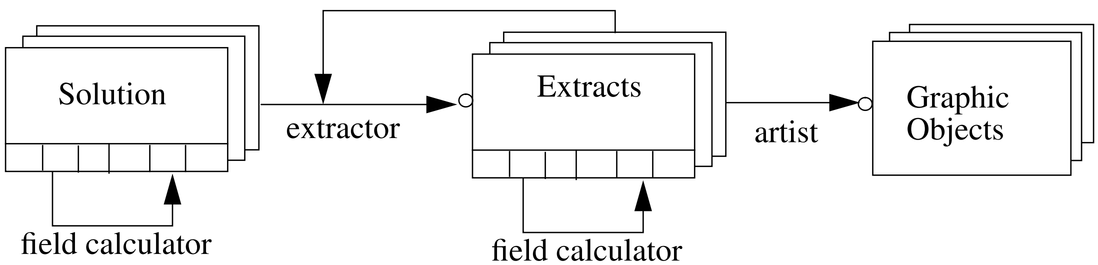


Figure 4: Extracts

An *extract* is a set of fields defined over a subdomain of the solution domain. To understand extracts, note that generating graphic objects from solutions can be divided into two steps: 1. extracting surface, curve, and/or point locations and associated field values to be visualized (the extractor step), and 2. converting field values to graphical properties (the artist step). The results of step 1 (locations and field values) are extracts. Extracts are much like solutions in that they are scalar, vector, and/or tensor fields on a (sub)domain. Examples of extract domains include isosurfaces, cut planes, and particles traces. Each solution time step usually has a corresponding extract time step. See appendix B for a discussion of common visualization algorithms as extract producers.

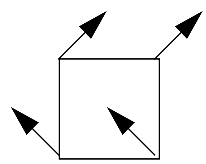
1. The terms *display list*, *geometry*, and *geom* are approximate synonyms for graphic object.

Figure 5 provides an example of converting a solution to an extract, calculating a derived field, and drawing a graphic object.

0,0,0 1,7,3,4,5	1,0,0 1,2,3,4,5	2,0,0 1,2,6,4,5	Grid and Solution x,y,z q1, q2, q3, q4, q5
0,1,0 1,2,3,4,5	1,1,0 1,9,3,4,5	2,1,0 1,2,3,0,5	
0,2,0 1,2,0,4,5	1,2,0 2,2,3,4,5	2,2,0 1,9,3,4,5	

1,1,0 1,9,3,4,5	2,1,0 1,2,3,0,5	Subset extract with grid and full solution
1,2,0 2,2,3,4,5	2,2,0 1,9,3,4,5	

1,1,0	1,1,0	Derived vector field
-1,1,0	-1,1,0	



Graphic object
(grid lines and arrows)

Figure 5: Solution - Extract - Graphic Object

Comparing extracts to graphic objects:

- Graphic object vertices are the equivalent of extract nodes. Both have a spatial location (x,y,z). However, extract nodes are associated with scientific data such as scalar, vector, and tensors while graphic object vertices are associated with rendering information such as colors and normal vectors.
- Each graphic object line segment usually has a corresponding extract grid edge.
- Each graphic object polygon usually has a corresponding extract grid cell.
- Each graphic object time step has a corresponding extract object time step.

Consider the trade-offs between storing extracts and graphic objects as separate objects vs. only storing graphic objects:

- Extracts require roughly the same storage as graphic objects. Consider each extract node and equivalent graphic object vertex:
  - Both generally contain a physical location (x,y,z).
  - For color mapped scalar fields, a scalar value requires the same space as eight bits each of red, green, blue, and alpha.
  - For vector plots, a vector requires the same space as the endpoint of a line representing the vector.
- The extractor step usually (but not always) requires much more time than rendering graphic objects (factors of 8 - 2124, see appendix C). For one thing, extraction is generally done with standard CPUs whereas very fast special purpose graphics hardware is

available for rendering. More important is that extraction algorithms must sometimes access much or all of each 3-D solution time step (consider isosurface extraction), whereas graphic object rendering accesses only surfaces, lines, and/or points.

- Extracts can be quickly converted to different graphic representations (see previous bullet). Consider a scalar field, say pressure, calculated on a streamline. This extract can be rendered as a color mapped curve, a tube with diameter as a function of scalar value, a ribbon with ribbon orientation controlled by the scalar field, etc.
- Extracts can be processed to produce new extracts. For example, streamlines[1] on a cutplane can be calculated from a velocity field on the cutplane, but not from a set of normals.
- Graphic objects may be implemented using extract data directly, or as optimized graphic data structures into which converted extract data are copied. Using extract data directly conserves space by a factor of about two, but can sometimes require substantially longer rendering times (factors of up to 4 in appendix B's data).

Consider calculating derived fields on the entire solution vs. on an extract. Although the solution domain has three spatial dimensions plus time, the spatial domain of most extracts and their corresponding graphic objects is 0, 1, or 2 dimensional (points, curves, or surfaces) plus time. It is inefficient to calculate and store a field over the entire 4-D solution domain if the field is to be viewed only on lower dimensional graphic objects. Thus, derived field calculations should be limited to extracts whenever possible. In order to calculate some derived fields on extracts, it is necessary to find the derivatives of solver output fields on extract domains.

The following trade-offs require attention when comparing extracts to solutions.

- If the solution can be discarded once extracts are calculated, storage requirements are greatly reduced. In our investigation, individual extracts vary from 0.3% - 6.7% of solution size, although these figures depend on many assumptions (see appendix C). On the other hand, the flexibility to examine the solution domain outside of the chosen extracts is lost completely.
- If field calculation can be limited to a few extract domains, computation time is substantially reduced. Extracts are much smaller than solutions and most derived field computations are 0(data-size). When derived field calculations are limited to extracts, one cannot quickly examine derived fields outside of those extracts.

Note that choosing the right extracts is critical to successful visualization, much as choosing the right instrumentation is crucial to the success of a wind-tunnel experiment.

1. A streamline is a particle trace in a steady velocity field.

Computation Space Fields

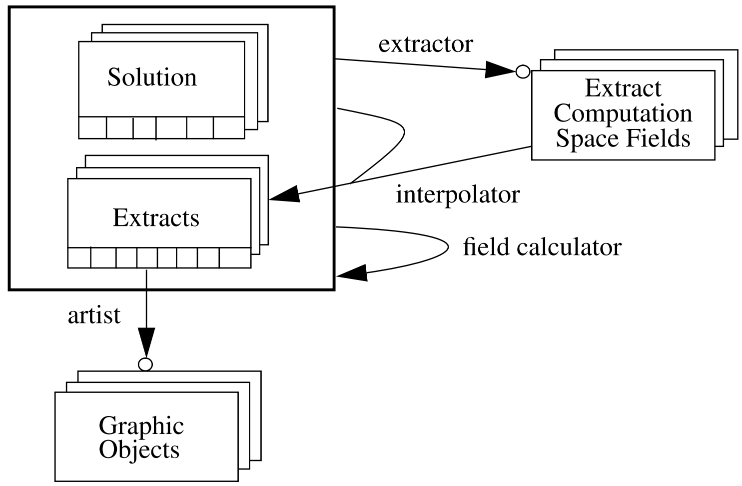


Figure 6: Computation Space Extracts

Extraction may be divided into two steps: 1. finding extract nodes in computation space, and 2. interpolating fields using computation space positions to index into field data. The results of step 1 are a computation space field.

An extract's computation space field is a set of computation space positions on an extract's domain. A computation space position is the memory offsets and interpolation coefficients used to interpolate solutions or derived fields at a point. In a sense, computation space is the way a field's data is laid out in memory, *e.g.*, in a three dimensional array. For 3-D curvilinear grids, the formal definition of computation space is:

$$X(\xi, \eta, \zeta) = x, y, z \qquad\qquad \xi(x, y, z) = \xi, \eta, \zeta$$

where $X(\xi, \eta, \zeta)$ maps a regular grid onto physical space and $\xi(x, y, z)$ is the inverse mapping.

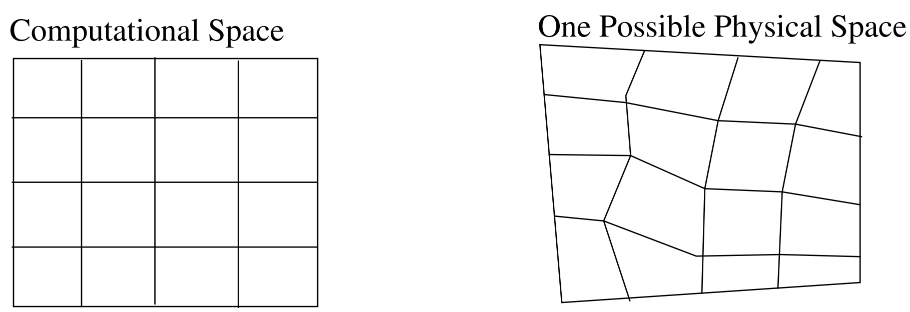


Figure 7: Computational vs. Physical Space

Thus, for 3-D curvilinear grids, a computation space position can be expressed as three real numbers. The integer part of each number indexes into a 3-D array. The fractional parts are the interpolation values for each dimension.

Most visualization algorithms find computation space positions on the fly, use them to interpolate the fields desired, and discard them. The model suggests that computation space positions be stored as fields on extract domains for later reuse. Storing computation space positions involves the following trade-offs:

- Storage requirements increase. Remember that for 3-D curvilinear grids, computation space positions are three floating point values. This price must be paid for each extract node.
- Additional fields can be interpolated later without rerunning the extractor. This increases flexibility since one is not limited to the fields calculated when the extractor is run.
- Visualization software is simpler and potentially faster. If all extractors create computation space extracts, then the same interpolation code can be used for all extractor output. This interpolation code is embarrassingly parallel because it does the identical operation on each computation space position. Thus, vectorization and parallelization are easy.

Note that even the physical location of extract nodes may be calculated by using computation space positions to index into the grid [Haimes92].

Animation

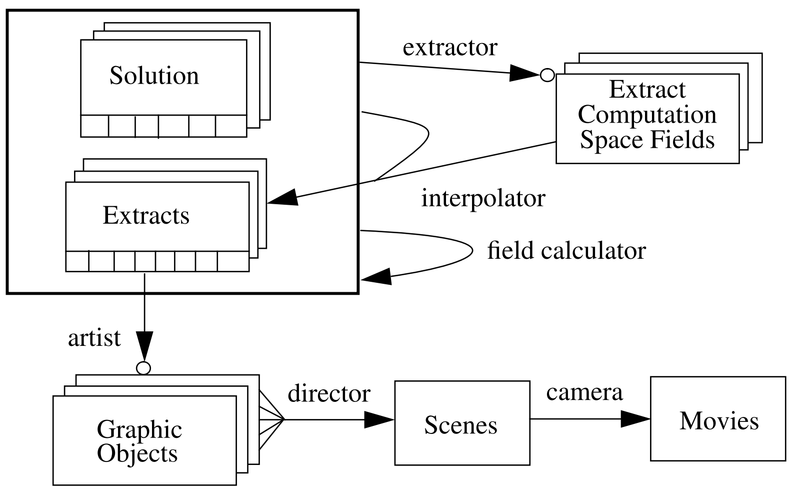


Figure 8: Animation Facilities

Although animation is not a focus of this paper, any unsteady flow visualization system needs animation facilities. A minimal animation data type is a sequence of images, called a *movie*. Like images, movies have no viewpoint control. To provide viewpoint control, the model also provides *scenes*. A scene is a set of graphic objects evolving through time and a *navigation*. A navigation is the path a virtual camera takes through time and space to create a movie, i.e., an ordered set of viewpoints and viewing directions. Scenes are put together by a *director*. A movie is created by ‘turning on’ a virtual *camera* and running a scene.

The terms artist, director, and camera are obviously borrowed. An artist chooses a visual representation for an idea (extract). A director chooses props and actors (graphic objects) and camera angles (navigation). Cameras record movies.

Scenes generally (although not always) require much less storage than movies, and scenes often playback more slowly. Movies stored on video tape can circumvent the storage problem, although video quality is seldom as good as that provided by the computer screen.

Persistent Object Database

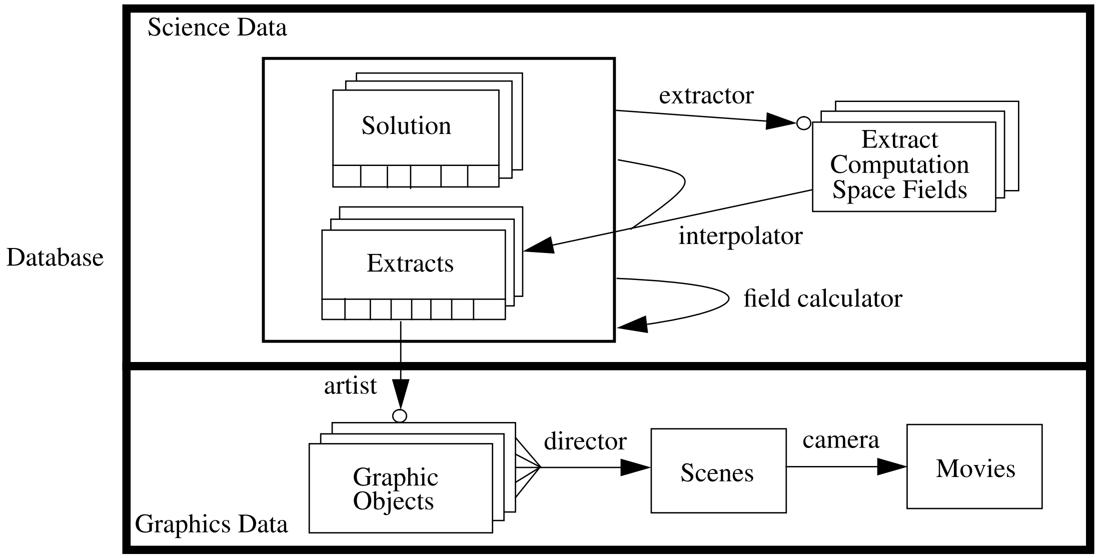


Figure 9: Persistent Object Database

Each solution, extract, graphic object, scene, navigation, and movie is an object. These disparate objects need a coherent organization and should be persistent. Furthermore, it is often useful to distinguish between the logical organization of the data (logical schema) and the physical representation on disk (physical schema). These facilities can be provided by collecting all objects into a persistent, possibly distributed, object database. Note that objects may be classified as either science or graphics data.

The database should be designed to provide:

- *Separation of logical and physical schemas.* This simplifies visualization algorithms and, sometimes, can dramatically improve performance. Visualization algorithms using a logical schema can, without change, access data physically laid out in many formats, with (usually) some loss of performance. On the other hand, working with massively parallel processors, [Seamons94] has shown that storing multi-dimensional arrays in single processor ‘chunks’ can improve IO performance by an order of magnitude. A logical interface to the arrays allows simulation and visualization software to remain unchanged, even thought the data are not arranged in traditional FORTRAN (or C) order.
- *Unity and completeness.* At a minimum, all objects may be accessed, created, deleted, and modified through at least one common interface. Special purpose interfaces can provide more optimal access to different types of data.
- *Performance metrics, i.e.*, object storage requirements and generation time. These are essential to making informed time vs. space vs. flexibility trade-offs. For example, the storage required for an isosurface not currently being rendered vs. the time to re-calculate the isosurface from the solution; or the flexibility lost when a solution is discarded after extracts are calculated vs. the storage required by the solution.
- *Persistence, e.g.*, lossless save and restore state. This allows visualization work sessions to be restarted, moved between machines, and shared with others just as word processor

documents are. Note that splitting and merging of databases is a valuable feature, since portions of a database can be shared and reused.

- *Object history.* A description of how each object is created. This is useful to conveniently implement repetitive operations; for example, calculating an isosurface for many time steps. Object history can help a system make time vs. space trade-offs when saving state; *e.g.*, choosing whether to save a cutplane to disk or recalculate it when restoring state.
- *An archive.* The database can be pruned to a bare minimum of storage supporting (or invalidating) an experiment's hypothesis, and stored on suitable media along with journal articles, solver source code, etc. As large simulations can consume months of a scientist's career, it makes sense to keep a substantial, well organized, portable archive.

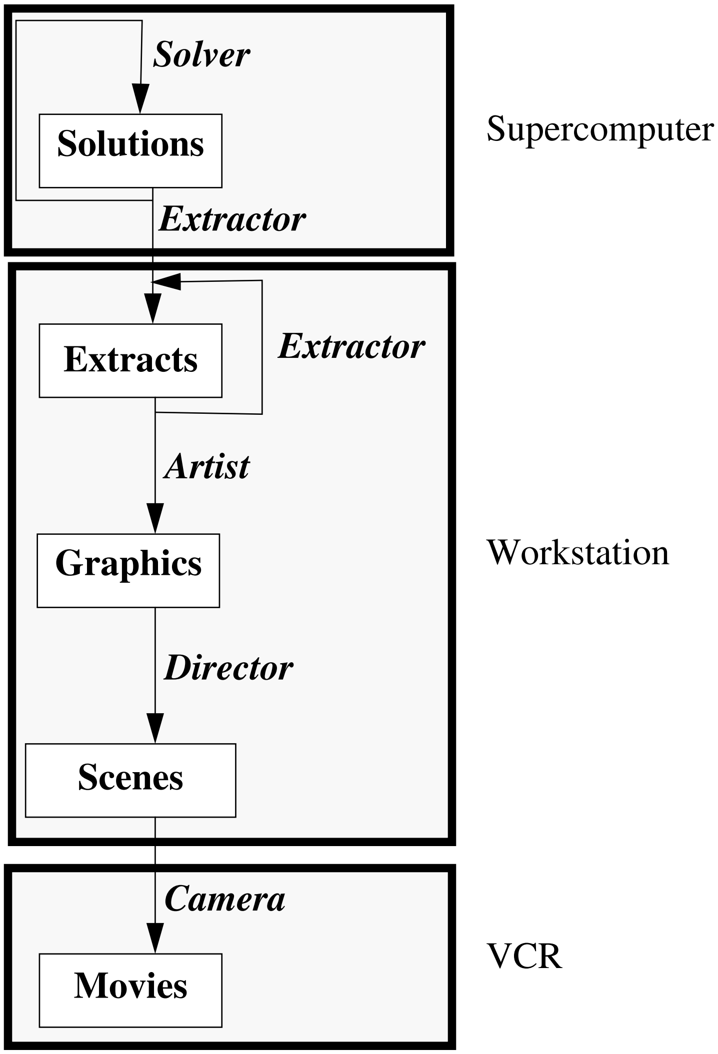
To understand a unsteady flow, it is often useful to watch a scene over and over, with small changes in the point of view. To do this rapidly requires that graphic object time steps be stored in the database rather than recomputed from the solution. Storing extract time steps also allows relatively fast changes in graphic representation.

In general, the database aspect of visualization is not well supported by current systems, with the notable exception of [Lang91]. Data flow systems such as AVS and SGI Explorer are particularly weak in database support [Globus92a]. This is a serious deficiency.

IV. Discussion

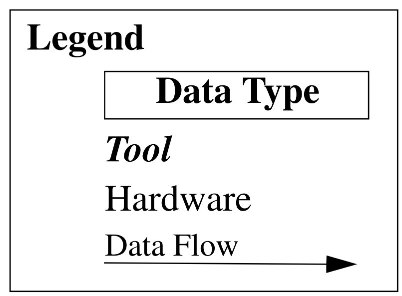
The most popular visualization system model today is the data flow model used by AVS [Upson89], SGI Explorer [SGI92], IBM Data Explorer [Lucas92], and others. Clearly, the data flow model can accommodate breaking visualization processing into extracting computation space fields, interpolating extract physical space fields, and then generating graphic objects. Each of these steps can be implemented as a set of modules. Implementing a database as described in this paper within the data flow model is much more difficult, requiring major rethinking by system developers.

Although the model is very flexible and can be used in many ways, it was developed to address the situation where the solution is calculated on a supercomputer, visualized on a workstation, and there is not enough disk space to store all solution time steps. An approach to this problem using the model is illustrated in figure 10.



The supercomputer computes solution time steps and calculates extracts extracted directly from the solution. Solution time steps are buffered on disk to give maximum time to find the right extracts. Due to storage limitations, however, eventually solution time steps must be archived to tape or deleted.

The workstation calculates extracts extracted from other extracts, converts extracts to a variety of graphic objects, and creates scenes.



Video tape captures scenes as sequences of images, i.e., movies.

Figure 10: One Use of the Model

This paper has focused on very large unsteady CFD solutions. These data are characterized by a set of 3-D steady-state solutions indexed by time. Many disciplines produce data with similar characteristics and might benefit from this model. Furthermore, the fact that the index is *time* is not crucial to the model. In a more complex situation, one can imagine numerical experiments with two indexes, *e.g.*, Reynolds number and time. In this case, the model should be even more useful, but database management and director tools become more complex.

Visualization is used not only to understand results, but also to debug solvers [Tuchman91]. The system characteristics required for unsteady flow visualization are similar to those required for debugging since the solution is converging and is thus unsteady. For debugging, discarding solution time steps is much less objectionable than when visualizing results, so extracts are more useful.

V. Future Work

The author worked with unsteady CFD visualization system developers such as R. Haimes [Haimes91], M. Gerald-Yamasaki [Yamasaki92], D. Lane [Lane93] and the FAST team [Bancroft90] to help integrate the model into working systems. In general, the extract concept has been fairly well accepted, but developers have yet to embrace the database concept.

The model assumes that users examine the performance data in the object database to make time vs. space trade-offs. Once sufficient experience is gained, it may be possible to develop intelligent software assistants [Miceli92] to automate the trade-offs based on available resources and user goals.

For the most part, this paper has assumed that users choose the extracts to calculate. However, imagine smart sensors that examine solution time steps to find features and generate extracts appropriately. For example, setting an isosurface threshold at the center of the steepest gradient of a scalar field, or creating three orthogonal cutting planes at vector field stationary points. This concept, contributed by Val Watson, is a rich area for future research.

VI. Summary

We have presented a software model intended as a guide to solving performance and flexibility problems encountered when visualizing the largest unsteady volumetric CFD results. The model consists of a database of solutions, extracts, graphic objects, scenes, and movies. The most important parts of the model are:

- Calculating extracts rather than graphic objects from solutions.
- Dividing extract computations into finding computation space fields and interpolating physical space fields.
- Collecting all data into a persistent object database separating the logical schema from the physical representation of data.

Although most visualization systems implement parts of the model and can be analyzed in the model's terms, there is no complete implementation to prove the concept.

We hope to see the model used to guide developers as visualization systems are developed or modified to examine large unsteady data sets. Developers should recognize that all portions of the model are important. We believe that leaving out portions will lead to loss of flexibility and reduced performance.

Appendix A: Glossary

The **artist** creates graphic objects from extracts.

An **animation** is a picture that changes over time.

The **camera** produces movies by recording the sequence of images produced by a scene.

A persistent object **database** unifies access to all objects, maintains object histories and performance metrics, and separates the logical schema from the data's physical representation.

The **director** produces scenes by choosing a set of graphic objects to animate and the path the camera navigates.

An **extractor** creates extracts from solutions and/or other extracts.

Extracts are a subset of a solution's nodes or a resampling of the solution domain. In each case, an extract may extend over some or all time steps.

Graphic objects are sets of vertices with rendering information.

Movies are sequences of images, e.g., video tapes.

A **navigation** is a sequence of viewpoints and view direction used to animate a set of graphic objects in a scene. Navigations may be canned or interactive.

A **scene** is a set of graphic objects and a navigation. A navigation is a viewpoint and viewing direction evolving over time.

A **solution** is a set of sampled scalar and vector fields evolving in time in a volume. Fields are sampled at each node of a computational grid at each of many time steps. Grids may evolve with time. Solution time steps are generated from initial conditions or previous time steps by solvers.

Viewpoint is the location of the viewer, or camera.

View direction is the direction a camera is looking.

Appendix B: Visualization Techniques Producing Extracts

Most common visualization techniques can be easily modified to produce extracts rather than graphic objects. Table 2 summarizes the characteristics of common visualization techniques applied to fields on curvilinear grids. Fields are not discussed because any field may be calculated on any extract, although some make more sense than others. Table 5 gives performance data for some of these extracts.

Table 2: Visualization Techniques

Visualization Technique	Grid Type Produced[a]	Description
Grid Plane	2-D curvilinear	A two dimensional slice of a 3-D field created by holding one index constant and allowing the other two to vary over their whole range. A set of grid planes over time is similar to a 3-D data set and may be examined with some of the same techniques.
Grid Volume	3-D curvilinear	A three dimensional portion of a 3-D field created by limiting the number of solution grid nodes included, e.g., by limiting the extent of each dimension.
Surface Grid	2-D curvilinear	Covers the surface of the vehicle or hardware investigated.
Cut Plane	2-D unstructured	An arbitrarily oriented plane through the data. If one uses marching cubes to generate the cutplane [Kerlick89], the plane is represented by an unstructured grid of triangles.

Table 2: Visualization Techniques

Visualization Technique	Grid Type Produced[a]	Description
Isosurface	2-D unstructured	The surface where a scalar field has a constant value. Isosurfaces do not always need to have field values calculated for each vertex since the spatial information alone is frequently of great interest.
Tangent Curve	1-D curvilinear	A curve that is everywhere tangent to a vector field. Functions on tangent curves are frequently used to control graphic representation, e.g., stream polygons [Schroeder91].
Particle Trace	0-D	The path a massless particle takes when released into a vector field. Note that in a steady vector field, a particle trace is identical to a tangent curve.
Streaklines	0-D	A set of points generated by continuously emitting particles from seed points. Each time step new grid nodes are created at the seed point.
Tangent Surface [Hultquist90, 92b]	2-D curvilinear or unstructured	A surface that is everywhere tangent to a vector field.
Pixel Plane Ray Casting	2-D regular	An image created by casting rays from the eye-point through each pixel into the data volume. Information collected along the ray is deposited in the pixel. Thus, a 2-D field in the viewing plane is produced.
X-Rays [Darmofal91]	2-D	An image created by casting rays from an emitter to a detector. Rays process information along their path and deposit a scalar field on the detector.
Vector Field Topology [Helman90]	0-D, 1-D, 2-D	The zeros (stationary or critical points) in a vector field and the stable and unstable manifolds passing through them. Thus, at each time step there is a set of points (0-D), a set of tangent curves (1-D) and tangent surfaces (2-D). The eigenvalues and eigenvectors of the derivative tensor at each zero may be thought of as a field on the stationary point grid. A derived field may only be desired on parts of the grid (*e.g.*, the zeros but not the manifolds).

a. The type of grid at a single time step assuming a 3D curvilinear solution.

Appendix C: Storage and CPU Time Data

Extract Sizes

Imagine that three grid planes, three large marching cubes surfaces (isosurfaces or cutting planes), and a particle cloud containing 20,000 particles are extracted from various size 3D solutions on curvilinear grids. Assume two scalar and one vector fields are on each extract. Table 3 shows the relative sizes of the extracts and the solution. Note that the extract/solution size ratio decreases as the solution size increases because solutions are three dimensional and extracts are 2, 1, or 0 dimensional.

Table 3: Relationship Between Solution and Extract Time Step Storage Requirements

Solution Grid (nodes)	Solution (megabytes)	Combined Extracts (megabytes)	Extract as a Percent of Solution
500,000	10	5.2	52%
1,000,000	20	8.0	40%
2,000,000	40	12.5	31%
3,000,000	60	16.3	27%
4,000,000	80	19.7	25%
5,000,000	100	22.8	23%
10,000,000	200	36.0	18%

Note that a user can control the memory needed by extracts by choosing the number and type of extracts to calculate. Consider a solution on a 3,000,000 node solution on a curvilinear grid. Table 4 shows storage requirements for various extracts. Note that the marching cubes surfaces consume the most storage:

Table 4: Space Consumed by Different Extracts

Measure	Grid Plane	Small[a] Marching Cubes Surface	Large Marching Cube Surface	10,000 Particles
Size (megabytes)	0.3 / 0.5[b]	0.8 / 1.0	3.3 / 4.0	0.2 / 0.3
Percent of Solution Size	0.5% / 0.8%	1.4% / 1.7%	5.6% / 6.7%	0.3% / 0.4%

a. Marching cubes surfaces can vary greatly in size depending on the number of cells intersected.
b. The first number assumes one scalar field on the extract's grid (x,y,z plus scalar). The second number assumes a vector field on the extract's grid (x,y,z plus 3 vector).

Extractor CPU Time

Table 5 provides performance metrics for the extracts and related graphic objects in table 2. Many simplifying assumptions were made to generate this table, some of which are described in the footnotes.

Table 5: Extract Performance[a] Characteristics

Extract	Number in a Scene[b]	Grid Nodes (x1000)	Grid Size (Mbytes)	Scalar/Vector Field Size per Time Step[c] (Mbytes)	Extract Calc/Render CPU Time per Time Step (seconds)	Graphic Object Render CPU Time per Time Step (seconds)
Grid Plane	5	111	1.78	0.44/1.33	NA/7.94	2.43
Cut Plane	5	265	9.38	1.06/3.18	66/26.46	6.61
Isosurface	3	214	7.55	0.86/2.57	53/20.24	5.69
Tangent Curve	100	222	2.66	0.89/2.66	131/0.72	0.55
Vector Field Topology	0D: 88 1D: 265	0D: 0.088 1D: 148	0D: 0 1D: 1.78	0D:0/0 1D:0.59/1.78	0D: 406/0.53 1D: 1041/1.15	0D: 0.49 1D: 0.75
Total	NA	960	23	3.84/11.53	1697/57	16.51
Total w/o topology	NA	811	21	3.25/9.74	250/55	15
Solution[d]	1	1,000	16	4/12/20[e]	150[f]	NA

a. These data were taken using FAST on a 226,800 node data set [Rizk85] using an SGI 320 VGX with 64 Mbytes of memory. All numbers were linearly extrapolated to simulate a 1,000,000 point data set. Elapsed times were measured with a stop watch. Rendering times are the average of 100 trials.
b. This column is the number of each type of extract that might be desired for a visualization. This number, although somewhat arbitrary, is necessary to calculate the values in the performance columns.
c. Assuming one scalar and one vector field for each extract.
d. Added for reference.
e. 20 Mbytes for a typical 5 value (density, momentum vector, energy) CFD solution
f. Solver seconds per visualization time step assuming 10 solver time steps per visualization time step. Speeds are consistent with typical NAS solvers on a Cray YMP.

Acknowledgments

E. Raible suggested the term ‘extract’. R. Haimes, J. Hultquist, B. Klein, T. Lasinski, D. Lane, K. Miceli, E. Miya, A. Vaziri, and, particularly, M. Gerald-Yamasaki reviewed drafts of this paper and provided many helpful comments, some of which lead to wholesale revision. This work is supported through NASA contract NAS2-12961.

References

[Atwood92] C. A. Atwood and W. R. Van Dalsem, “Flowfield Simulation about the SOFIA Airborne Observatory,” *30th AIAA Aerospace Sciences Meeting and Exhibit*, January 1992, Reno, Nevada, AIAA 92-0656.

[Bailey86] F. R. Bailey, “Status and Projections of the NAS Program,” *Symposium on Future Directions of Computational Mechanics*, ASME, Anaheim, CA, 7-12 December, 1986.

[Bancroft90] G. Bancroft, F. Merritt, T. Plessel, P. Kelaita, R. McCabe, and A. Globus, “FAST: A Multi-Processing Environment for Visualization of CFD,” *Proceedings Visualization ‘90*, IEEE Computer Society/ACM SIGGRAPH, San Francisco (1990).

[Bryson91] S. Bryson and C. Levit, “The Virtual Windtunnel: An Environment for the Exploration of Three-Dimensional Unsteady Flows,” *Proceedings \ Visualization ‘91*, IEEE Computer Society/ACM SIGGRAPH, 22-25 October, 1991, San Diego, California.

[Bryson92] S. Bryson and M. Gerald-Yamasaki, “The Distributed Virtual Windtunnel”, *Proceedings Supercomputing ‘92*, IEEE Computer Society/ACM, November, 1992.

[Buning85] P. G. Buning and J. L. Steger, “Graphics and Flow Visualization in Computational Fluid Dynamics,” AIAA-85-1507-CP, AIAA 7th Computational Fluid Dynamics Conference, 15-17 July, 1985, Cincinnati, OH.

[Butler89a] D. M. Butler and M. H. Pendley, “The Visualization Management System Approach to Visualization in Scientific Computing,” *Computers in Physics*, Vol. 3, pp. 40-44, 1989.

[Butler89b] D. M. Butler and M. H. Pendley, “A Visualization Model Based on the Mathematics of Fiber Bundles,” *Computers in Physics*, Vol. 3, pp. 45-51, 1989.

[Butler91] D. M. Butler, C. Hansen, organizers, “Visualization ‘91 Workshop Report: Scientific Visualization Environments,” IEEE/ACM SIGGRAPH Visualization ‘91, 22-25 October, 1991, San Diego, California.

[Campbell89] W. J. Campbell, N. M. Short, L. A. Treinish, “Adding Intelligence to Scientific Data Management,” *Computers in Physics*, May/June 1989.

[Chawla92] K. Chawla and W. R. Van Dalsem, “Numerical Simulation of STOL Operations Using Thrust Reversers,” AIAA 92-4254, AIAA Aircraft Design Systems Meeting, 24-26 August, 1992, Hilton Head, SC.

[Dickenson91] R. R. Dickenson, “Interactive Visualization of Transient Fields,” *Proceedings SPIE/IS&T Symposium on Electronic Imaging Science and Technology*, 24 February - 1 March 1991, San Jose, California, USA.

[Globus92a] A. Globus, “Perspectives on the IRIS Explorer Visualization Environment,” NASA Ames Research Center, NAS Systems Division, Applied Research Branch technical report RNR-92-021, May 1992.

[Globus92b] A. Globus, “A Software Model for Visualization of Time Dependent 3-D Computational Fluid Dynamics Results,” NASA Ames Research Center, NAS Systems Division, Applied Research Branch technical report RNR-92-031, November 1992.

[GB92] K. Gundy-Burlet, personal communication.

[Haber91] R. B. Haber, B. Lucas, N. Collins, “A Data Model for Scientific Visualization with Provisions for Regular and Irregular Grids,” *Proceedings Visualization ‘91*, IEEE Computer Society/

ACM SIGGRAPH, San Diego, CA, pp. 298-305, 1991.

[Haimes91] R. Haimes and M. Giles, “VISUAL3: Interactive Unsteady Unstructured 3D Visualization,” 29th AIAA Aerospace Sciences Meeting and Exhibit, January 1991, Reno, Nevada, AIAA 91-0794.

[Haimes92] R. Haimes, “Techniques for Interactive and Interrogative Scientific Volumetric Visualization,” personal communication.

[Haimes94] R. Haimes, “pV3: A Distributed System for Large-Scale Unsteady CFD Visualization,” 32th AIAA Aerospace Sciences Meeting and Exhibit, AIAA Paper 94-0321, Reno, NV, Jan. 1994

[Hibbard90] B. Hibbard and D. Santek, “The VIS-5D System for Easy Interactive Visualization,” *Proceedings Visualization ‘90*, IEEE Computer Society/ACM SIGGRAPH, San Francisco, 1990.

[Kerlick89] G. D. Kerlick, “ISOLEV: A Level Surface Cutting Plane Program for CFD Data,” NASA Ames Research Center, NAS Systems Division, Applied Research Branch technical report RNR-89-006, March 1992.

[Lam92] T. L. Lam, “Improving File System Performance by Striping,” NASA Ames Research Center, NAS Systems Division, Systems Development Branch technical report RND-92-014, April 1992.

[Lane93] D. A. Lane, “Visualization of Time-Dependent Flow Fields,” *Proceedings Visualization ‘91*, IEEE Computer Society/ACM SIGGRAPH, San Jose, CA October 1993.

[Lang91] U. Lang, R. Lang, and R. Ruehle, “Integration of Visualization and Scientific Calculation in a Software System,” *Proceedings Visualization ‘91*, IEEE Computer Society/ACM SIGGRAPH, 22-25 October, 1991, San Diego, California.

[Legensky90] S. M. Legensky, “Interactive Investigation of Fluid Mechanics Data Sets,” *Proceedings Visualization ‘90*, IEEE Computer Society/ACM SIGGRAPH, San Francisco, 1990.

[Lucas92] B. Lucas, G. D. Abram, N. S. Collins, D. A. Epstein, D. L. Gresh, K. P. McAuliffe, “An Architecture for a Scientific Visualization System,” *Proceedings Visualization ‘92*, IEEE Computer Society/ACM SIGGRAPH, October, 1992, Boston, MA.

[Miceli92] K. D. Miceli, “A Framework for the Design of Effective Graphics for Scientific Visualization,” NASA Ames Research Center, NAS Systems Division, Applied Research Branch technical report RNR-92-035, December 1992.

[Schroeder91] W. J. Schroeder, C. R. Volpe, and W. E. Lorensen, “*The Stream Polygon*: A Technique for 3D Vector Field Visualization,” *Proceedings Visualization ‘91*, IEEE Computer Society/ACM SIGGRAPH, 22-25 October, 1991, San Diego, California.

[Schroeder92] W. J. Schroeder, W. E. Lorenson, “VISAGE: An Object-Oriented Scientific Visualization System,” *Proceedings Visualization ‘92*, IEEE Computer Society/ACM SIGGRAPH, October, 1992, Boston, MA, pp. 219-226.

[Seamons94] K. E. Seamons and M. Winslett, “An Efficient Abstract Interface for Multidimensional Array I/O”, *Proceedings of Supercomputing ’94*, Washington D.C., November 1994.

[SGI92] *IRIS Explorer User’s Guide*, Silicon Graphics Computer Systems, Document 007-1369-010, 1992.

[Smith89] M. H. Smith, W. R. Van Dalsem, R. C. Dougherty, and P. G. Buning, “Analysis and

Visualization of Complex Unsteady Three-Dimensional Flows," 27th AIAA Aerospace Sciences Meeting and Exhibit, January 1989, Reno, Nevada, AIAA 91-0794.

[Smith91] M. Smith, K. Chawla, and W. Van Dalsem, "Numerical Simulation of a Complete STOVL Aircraft in Ground Effect," AIAA-91-3293, AIAA 9th Applied Aerodynamics Conference, 23-25 September 1991, Baltimore, MD.

[Treinish90] L. A. Treinish, "SIGGRAPH '90 Workshop Report on Data Structures and Access Software for Scientific Visualization," *Computer Graphics*, Vol. 25, Number 2, April 1991, pp. 104-116.

[Tuchman91] A. Tuchman, D. Jablonowski, G. Cybenko, "Run-time Visualization of Program Data," *Proceedings Visualization '91*, IEEE Computer Society/ACM SIGGRAPH, 22-25 October, 1991, San Diego, California, pp. 255-261.

[Upson89] C. Upson, T. Faulhaber, D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, A. van Dam, "The Application Visualization System: A Computational Environment for Scientific Visualization," *IEEE Computer Graphics and Applications*, July 1989, pp. 30-41.

[Wavefront91] *The Data Visualizer User's Guide*. Wavefront Technologies, Inc.

[Yamasaki92] M. J. Gerald-Yamasaki, "Interactive and Cooperative Visualization of Unsteady Fluid Flow," NASA Ames Research Center, NAS Systems Division, Applied Research Branch technical report RNR-92-018, March 1992.