Conf-9401114--1

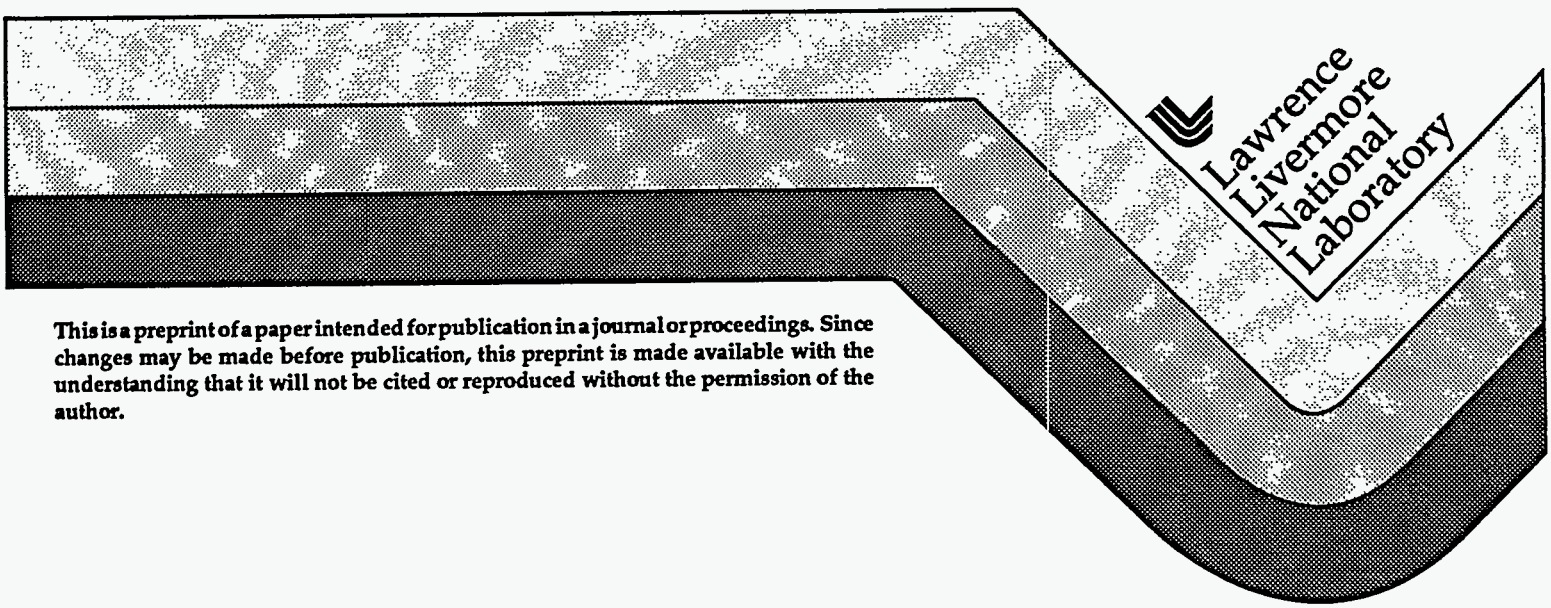
UCRL-JC-119879
PREPRINT

Visualization Methods for High-Resolution, Transient, 3-D, Finite Element Simulations

Mark A. Christon

This paper was prepared for submittal to the
International Workshop on Visualization
Paderborn, Germany
January 18–20, 1994

January 10, 1995



TS

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

1
Workshop on Visualization
D. Kröner and R. Rautmann (Eds)
1995 VSP/TEV

VISUALIZATION METHODS FOR HIGH-RESOLUTION, TRANSIENT, 3-D, FINITE ELEMENT SIMULATIONS

Mark A. Christon

Methods Development Group
Lawrence Livermore National Laboratory
P.O. Box 808
Livermore, California 94550

ABSTRACT

Scientific visualization is the process whereby numerical data is transformed into a visual form to augment the process of discovery and understanding. Visualizing the data generated by large-scale, transient, three-dimensional finite element simulations poses many challenges due to geometric complexity, the presence of multiple materials and multiple element types, and the inherent unstructured nature of the meshes. In this paper, the direct use of finite element data structures, nodal assembly procedures, and element interpolants for volumetric adaptive surface extraction, surface rendering, vector grids and particle tracing is discussed. A brief description of a "direct-to-disk" animation system is presented, and case studies which demonstrate the use of isosurfaces, vector plots, cutting planes, reference surfaces and particle tracing are then discussed in the context of several case studies for transient incompressible viscous flow, and acoustic fluid-structure interaction simulations. An overview of the implications of massively parallel computers on visualization is presented to highlight the issues in parallel visualization methodology, algorithms, data locality and the ultimate requirements for temporary and archival data storage and network bandwidth.

1. INTRODUCTION

The Methods Development Group (MDG) at Lawrence Livermore National Laboratory (LLNL) has been developing finite element simulation tools for over 15 years. These tools range from mesh generators to data visualizers and include the suite of codes shown in Table 1. The simulation tools include high-rate Lagrangian (DYNA2D, DYNA3D), low-rate and quasi-static Lagrangian (NIKE2D, NIKE3D), and purely Eulerian codes (TOPAZ2D, TOPAZ3D, and HYDRA). These simulation tools put demanding requirements on the visualization tools which must handle a broad variety of tasks ranging from simple 2-D temperature contour plots to 3-D, time-dependent particle traces in a flow simulation.

The power of visualization for complex transient simulations derives in part from the fact that approximately 50% of the brain's neurons are associated with vision, and that scientific visualization exercises this neurological machinery (McCormick, et al., 1987). In practice, the ultimate goal of visualization is to provide the engineer or scientist insight into the results produced by large scale scientific simulations. For transient simulations, effective visualization tools must rely upon the ability to interactively probe and interrogate data to the fullest extent.

The treatment of complex geometry, with unstructured and heterogeneous meshes poses many challenges for the design and implementation of effective visualization tools. Here, a heterogeneous mesh is defined to be a single mesh which contains 1-D, 2-D, and 3-D elements. The unstructured topology in a finite element mesh precludes the use of simple (i, j, k) indexing schemes for block-logical grids which are typically employed in finite difference or finite volume codes and their data visualizers. Indeed, finite element data visualizers must be robust in terms of the ability to treat arbitrarily complex geometry with inherently unstructured and even heterogeneous meshes.

In addition to geometrical and topological complexity, many finite element simulations use spatially varying material properties. For example, a typical time-dependent flow simulation with conjugate heat transfer would include a region of the mesh which is fluid and a region which is solid. Solid and struc-

[1] Work performed under the auspices of the U. S. Department of Energy by the Lawrence Livermore National Laboratory under contract No. W-7405-ENG-48.

Table 1: The Methods Development Group's suite of finite element simulations tools.

Finite Element Tools	2-D	3-D
Mesh Generation	MAZE	INGRID
Implicit Solid/Structural Mechanics	NIKE2D	NIKE3D
Explicit Solid/Structural Mechanics	DYNA2D	DYNA3D
Thermal Analysis	TOPAZ2D	TOPAZ3D
Acoustic Fluid-Structure Interaction	PING	PING
Incompressible, Viscous Flow	HYDRA	HYDRA
Scientific Visualization	ORION/GRIZ	GRIZ
Time History Processing	THUG	THUG

tural mechanics finite element models frequently include special features such as sliding interfaces and frequently use multiple material definitions.

In a Lagrangian framework, both contact and impact of multiple deformable bodies with complex material models and mesh adaptivity must be treated in an efficient manner in order for visualization to be useful. Further, many solid and structural problems are nonlinear in both the material and geometric sense, i.e., plastic deformation with large deformations and rotations are possible. Figure 1 shows a typical large-scale, transient, Lagrangian computation with both material and geometric nonlinearity in addition to sliding interfaces.

In contrast, time-dependent, Eulerian, finite element simulations for computational fluid dynamics (CFD) typically require high-resolution, unstructured meshes and frequently involve complex geometry. Unlike their Lagrangian counterpart, CFD meshes are homogeneous (i.e., a single element type) although usually of higher resolution. Figure 2 shows the geometric and topological complexity involved in a typical computational fluid dynamics simulation.

Both Eulerian and Lagrangian finite element simulations require the ability to abstract scalar, vector and tensor data for rendering purposes. However, one of the key components for visualizing time-dependent results is the ability to rapidly perform surface extraction for rendering purposes while maintaining both topological adaptivity in the finite element mesh and volumetric information for data abstraction and rendering. While there are many visualization requirements which are common to both Lagrangian and Eulerian transient finite element simulations, visualizing fluid-structure simulations requires the ability to view both structural and fluid variables and combine many aspects of Lagrangian and Eulerian rendering techniques.

Other key aspects of an effective finite element data visualizer include the ability to selectively display

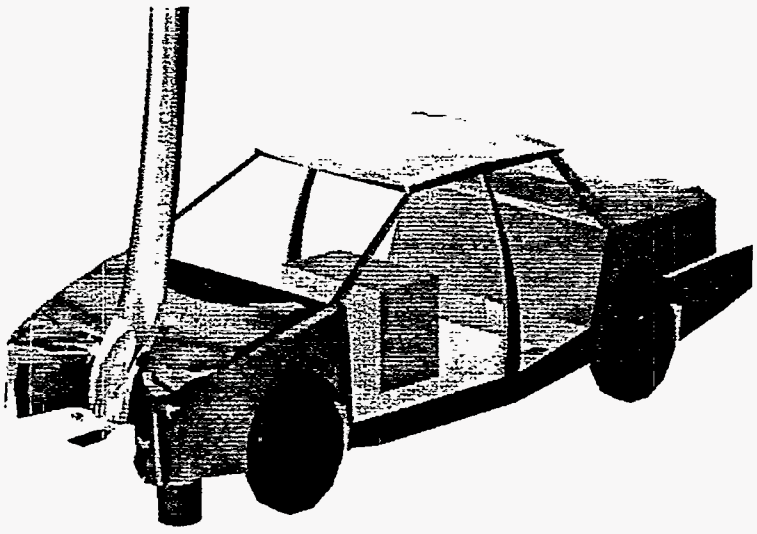


Fig. 1: DYNA3D impact simulation demonstrating contact surfaces.

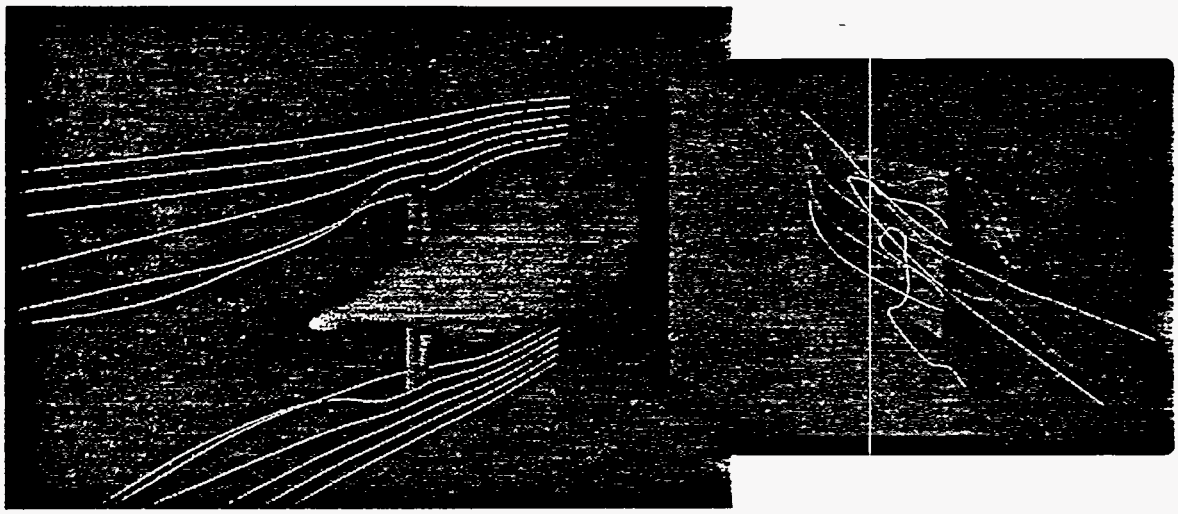


Fig. 2: Snapshot of particle traces in a time-dependent flow simulation.

material components of the mesh, provide palette manipulation, map derived physical quantities into color, and perform *in-betweening* for animations. This must be done without inhibiting the user's ability to move randomly through large datasets viewing individual states in simulation time.

Remark 1. in-betweening refers to the specialized interpolation procedures used in key-frame animation to generate the requisite graphics frames for playback of video at 30 frames per second.

This paper first presents an overview of the use of finite element data structures for visualization. In this section, the SEVA (Surface Extraction with Volumetric Adaptivity) algorithm for performing rapid surface extraction while permitting volumetric adaptivity is presented. In addition, this section outlines the computation of continuous vertex normals for rendering, the use of isosurfaces for cutting planes, and the calculation of vector grids and particle traces. A brief description of the video animation system in use by the Methods Development Group is described in section 3. Several visualization case studies are presented in section 4, and finally, some of the issues involved in visualizing the data produced by parallel simulations are discussed in section 5.

2. FINITE ELEMENT DATA STRUCTURES FOR VISUALIZATION

In the finite element method, the physical domain of interest is approximated by a finite number of smaller sub-domains referred to as elements. Nodes in a finite element mesh are attached to elements according to a canonical local numbering scheme. However, the global node and element numbering scheme can appear essentially random from a spatial point of view. Unlike finite difference grids which are logically regular, finite element meshes are topologically based.

In a finite element mesh, nodal connectivity data relates global node numbers to elements and is represented by C_{ij} where $i \in [1, N_{el}]$, $j \in [1, N_{npe}]$. Here, N_{el} is the number of elements, and N_{npe} is the number of nodes per element. The elements used in the MDG simulation codes consist of eight node hexahedra (hex elements), four node quadrilaterals (shell elements), and two node beams. Figure 3 shows these element types and their associated canonical local node numbering schemes. Typically, finite element codes group like elements together to simplify the connectivity and to avoid ragged data structures (e.g., the connectivity is blocked according to the hex, shell and beam elements).

For the hex elements, C_{ij} is comprised of six reduced sets of connectivity data which associate local nodal numbers with the faces of the hex elements. Thus, the reduced connectivity relates global node numbers to individual faces of elements. Here, the reduced connectivity is S_{ij} where $i \in [1, 6N_{hex}]$, $j \in [1, N_{nse}]$, and N_{nse} is the number of nodes per surface element. The canonical reduced nodal connectivity is shown in Table 2 for the hex element in Figure 3. The connectivity data plays a key role in performing rapid surface extraction.

Surface Extraction

For the purposes of surface extraction, both beams and quadrilateral elements may be ignored for obvious reasons. However, the hex elements require special attention because of the requirement for topological

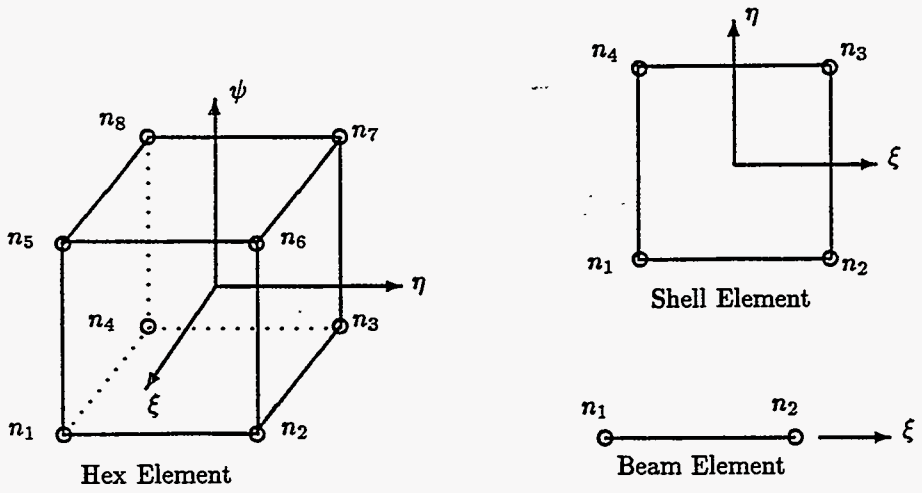


Fig. 3: Canonical local node numbering scheme in computational space for hex, shell, and beam elements ($-1 \leq \tilde{\xi} \leq 1$ where $\tilde{\xi} = (\xi, \eta, \psi)$).

adaptivity when rendering only external surfaces and the associated nodal point data. Topological adaptivity requires the ability to preserve volumetric data while permitting incremental surface extraction both during the interactive interrogation process for data exploration and the in-betweening process for animation. Before describing the volume adaptive surface extraction algorithm, a brief overview of two alternative schemes is presented.

The first surface extraction algorithm is referred to as the assembled surface normal (ASN) algorithm (Belytschko and Law, 1985). In the ASN algorithm, surface connectivity (S_{ij}) is used to construct normal vectors at the nodes of all surfaces of the hex elements. For example, at node n_1 on face 1, $\mathbf{n}_{n_1} = (\mathbf{x}_{n_2} - \mathbf{x}_{n_1}) \times (\mathbf{x}_{n_5} - \mathbf{x}_{n_1})$, where $\mathbf{n}$ is the normal vector. The use of S_{ij} here is simply to gather the appropriate global nodal coordinates required for the computation of the local nodal normals on each face. The local unit vector is computed as: $\vec{\mathbf{e}}_j = \mathbf{n}_j / \|\mathbf{n}_j\|$.

By making use of the global nodal numbering scheme embedded in the connectivity data (S_{ij}), the local normal vectors are assembled as (see Hughes, 1987 for an example of the FEM assembly procedure):

$$\mathbf{n} = \sum_{i=1}^{6N_{hex}} \sum_{j=1}^{N_{nsc}} S_{ij} \vec{\mathbf{e}}_j^{(i)} \tag{1}$$

where $\mathbf{n}$ is the assembled global normal vector, and $\vec{\mathbf{e}}_j^{(i)}$ is the local normal vector computed at each node of surface (i). In Eq. 1, S_{ij} scatters from a local (canonical) node number of an element face to a global node number during the finite element assembly procedure. Global unit normal vectors are computed as: $\vec{\mathbf{e}} = \mathbf{n}/\|\mathbf{n}\|$.

After the global assembly has been performed, all nodal points associated with surface elements which are interior to the mesh will have a global unit normal whose components are zero. Thus, using ASN,

Table 2: Face nodal connectivity for the hex element.

Face No.	Node-1	Node-2	Node-3	Node-4
1	n_1	n_2	n_6	n_5
2	n_2	n_3	n_7	n_6
3	n_4	n_8	n_7	n_3
4	n_1	n_5	n_8	n_4
5	n_1	n_4	n_3	n_2
6	n_5	n_6	n_7	n_8

internal surfaces are eliminated by noting that paired surfaces (see Figure 4) will generate vertex unit normals which essentially cancel during a nodal assembly process. In effect, the finite element assembly process in ASN identifies the external surface of the domain and produces unique, averaged normals on this surface.

While providing a robust surface extraction mechanism, the ASN algorithm fails to retain both volumetric and topological information which is central to any adaptive scheme. That is, ASN discards all interior data and retains only the polygons associated with the empty shell for rendering. Further, the ASN algorithm approach proves to be computationally intensive requiring $24N_{hex}$ unit normal computations plus the global assembly operations for a given 3-D mesh.

The second surface extraction algorithm is the polygon-search/cancel (PSC) algorithm (Winget, 1988) which also relies upon the node-to-element connectivity, C_{ij}, and avoids some of the floating point effort involved in the ASN algorithm. However, one difficulty with PSC is that it relies on a linear search which results in approximately $O((6N_{hex})^2)$ operations. This can be improved by using a hash table lookup, but requires chaining to resolve collisions in the hash table. Neither the ASN nor the PSC algorithm inherently deals with volumetric adaptivity which is an important type of adaptivity and is found in solid mechanics finite element codes like DYNA3D (Whirley and Englemann, 1993) (albeit, only one form of adaptivity).

It is possible to apply the ASN or PSC algorithm after each time step in which the mesh topology changes. However, this involves considerable computational overhead which can be avoided. The ASN and PSC algorithms form the starting point for the description of a two pass surface extraction algorithm that accommodates volumetric adaptivity.

By viewing internal surfaces as paired surfaces between adjacent elements (e.g., as shown in Figure 4 for the ASN algorithm), it is possible to combine features of the ASN and PSC algorithms to efficiently construct the data structures necessary to perform an incremental surface extraction while accounting for volumetric adaptivity. The incorporation of adaptivity for beam and shell elements is essentially automatic given the simplicity of these elements.

To begin, the node numbering scheme in the finite element method does not permit redundant nodes (i.e., two node numbers with identical spatial coordinates) except where there are special model features such as sliding interfaces or constrained nodal pairs. Therefore, the key to incremental surface extraction with volumetric adaptivity is the identification of paired internal surfaces in the mesh. This identification is considered to be the first pass in the two pass extraction algorithm which is referred to herein as the SEVA (Surface Extraction with Volumetric Adaptivity) algorithm.

For the discussion of the SEVA algorithm, a hash function is defined which will be used in a table of length $6N_{hex}$. Note that $6N_{hex}$ entries are required to account for the worst possible case in which all faces of the hex elements are exterior faces (a possibility in the most general case).

$$H_i = \sum_{j=1}^{Nnse} w_j \tilde{S}_{ij} \tag{2}$$

Here $\tilde{S}_{ij}$ is the local surface connectivity sorted in ascending order, and w_j is a nodal weight.

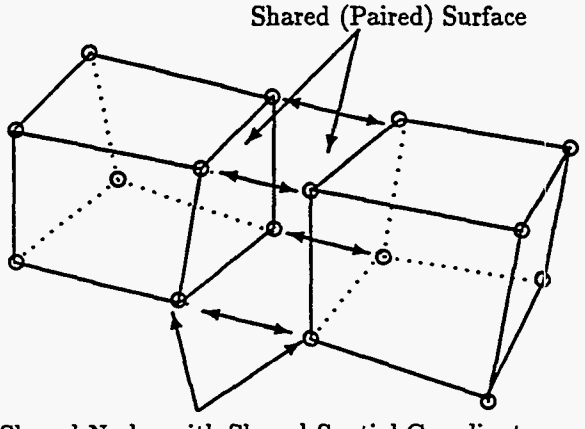


Fig. 4: Internal paired surfaces between hex elements.

The nodal numbering scheme in the finite element method ensures that there are no repeated nodal numbers, that is, $n_i \in [1, N_{np}]$ where N_{np} is the total number of nodal points in the mesh. Because of the locally ordered nodal connectivity used in Eq. 2, it is possible to define the nodal weighting as: $w_j = 10^{(j-1)}$. This choice of weights helps to maintain a relatively high degree of dispersion in the hash keys, but by no means ensures that each hash value will be unique (Tenenbaum, et al., 1990). However, the goal here is not to use the hash keys for direct address calculation, but to identify paired surfaces in the volume. Therefore, the hash keys are entered into the hash table in sequential order as they are computed.

During the construction of the hash table, the element and face number associated with each hash key are also entered in the table. An indexed heap sort (an $O(N \ln(N)$ algorithm (Press, et al.,1987)) is used to generate a list of hash values in ascending order. By traversing the sorted list of hash values, it is possible to identify any paired surfaces in the volume because such surfaces will yield identical hash function values which are adjacent to each other in the sorted hash list. Note that hash keys for shell elements could be included in the surface extraction process to handle the situation where shell elements lie on top of hex elements.

Traversing the hash table, a master surface list which identifies paired surfaces (i.e., surfaces which are back-to-back) is constructed. For each hash value a maximum of two element and two face numbers are included with a surface counter in the master surface list for later use in the second-pass, volume adaptive, surface extraction. The construction of the hash table, the heap sort and subsequent construction of the master surface list in the first pass of the algorithm is only performed once. The master surface list contains all the necessary data for subsequent adaptive surface extraction computations.

In the second pass of the SEVA algorithm, the master surface list is used to generate a polygon vertex pointer list which includes the global node numbers for the polygon vertices, and an element material number. Initially, only unpaired surfaces, i.e., external surface elements (with a surface count of 1), from the master surface list are included in the polygon list. Any active shell elements are then concatenated onto the polygon vertex pointer list. The vertex pointer is then used to gather the appropriate vertex coordinates and normals for rendering.

One of the key applications for SEVA is in problems which require volumetric adaptivity with sliding interfaces which may be found in DYNA3D simulations (Christon and Spelce, 1992, Whirley and Englemann, 1993). In this application elements may fail at a sliding interface and subsequently be removed from the computation based upon a maximum strain failure criteria. An activity vector is included in the graphics database for this specific case and is used to remove inactive surfaces from the polygon pointer list, as well as, to split paired surfaces where failed elements have been removed. Figure 5 illustrates the situation when an element has become inactive revealing the previously hidden interior surfaces.

Splitting paired surfaces is accomplished by reducing the surface count in the master surface list for all faces of an element which has failed. Any surfaces in the master surface list with a count of 1, i.e., occurring only one time, are considered to be valid surfaces for rendering purposes. The canonical local node numbering scheme for the hex elements ensures that freshly exposed polygon normals face outwards in the animation scene at all times which guarantees consistent lighting calculations during rendering.

The SEVA algorithm was initially implemented in a finite element data visualizer used for interpolating between discrete states of simulation results for video animation (Christon and Spelce, 1992). The process of temporal interpolation is often referred to as "in-betweening", and the original visualization code was called the TWEENER. A modified version of the SEVA algorithm is currently in use in GRIZ (Dovey and Spelce, 1993).

Vertex Normals for Rendering

Vertex normals are required for lighting and shading computations during the rendering process. In order to represent smooth surfaces, spatially varying, but continuous surface normals are necessary. Vertex normals may be computed using the following two-step process. First local nodal normals are computed for all active surfaces, and then they are assembled into a global normal vector. The assembly is similar to that used in the ASN algorithm, but occurs only for those nodes attached to active surface elements in the master surface list. Using the polygon vertex list, P_{ij},

$$\mathbf{n} = \sum_{i=1}^{N_{srf}} \sum_{j=1}^{N_{nsc}} P_{ij} \vec{\mathbf{e}}_j^{(i)} \tag{3}$$

where N_{srf} is the number of active surfaces for rendering. Here, the vertex list is used to scatter the appropriate local nodal normals to the global nodes.

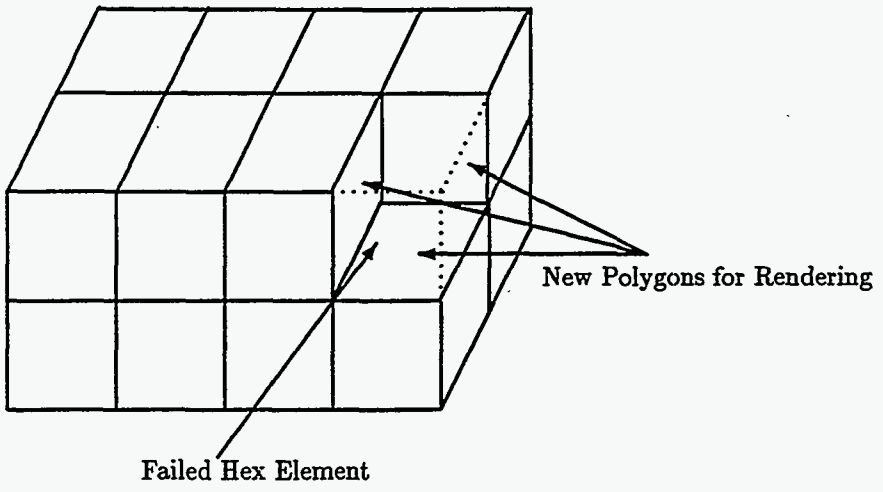


Fig. 5: Elimination of a hex element and the resulting exposure of new surfaces for rendering.

During the assembly of the global normals, the number of polygons contributing to each nodal normal is accumulated and stored. This allows average normals to be computed on the external surface of the domain by simply dividing the nodal normal by the accumulated number of polygons contributing to a given nodal normal. To account for "sharp" corners, a simple heuristic based upon the dot product of a vertex normal and its corresponding globally averaged normal is employed to decide when to use the averaged normal, and when to use the local normal.

Isosurfaces and Cutplanes for Volumetric Interrogation

Isosurfaces and cutplanes are commonly used visualization techniques for volumetric interrogation of 3-D data. An isosurface is the three dimensional equivalent of a single contour line because it spatially maps out the presence of a specific scalar value in three-dimensions. A modified version of the well-known marching cubes algorithm (Lorensen and Cline, 1987) is used to generate the polygonal surface based upon the element local nodal values of a specified scalar quantity. Note that the scalar quantity may be an abstracted value such as helicity, velocity magnitude or effective stress.

Isosurfaces provide a global view of the variation of a single quantity in the spatial domain. However, it is often of interest to cut a physical domain in a specific way in order to visualize the variation of a specific variable on a given plane. The cutplane geometry and field quantities are obtained using the isosurface algorithm with a simple modification. Rather than applying the isosurface algorithm to a scalar quantity such as pressure, the normal distance to the cutplane is used.

The configuration of a cutplane is simply defined by a point on the plane and a unit normal vector. The normal distance of all nodes from the cutplane is computed and supplied to the isosurface algorithm. By computing an isosurface corresponding to a field variable which is zero, the cutplane is generated. During the isosurface computation, the scalar physical quantity of interest is interpolated at the triangle vertices in the isosurface. The vertex values are then color mapped and rendered on the cutplane.

Vector Grids

Although typically used for CFD data, the display of instantaneous velocity distributions is also useful for Lagrangian computations where rigid body motion is of interest. However, the rendering of nodal vectors on graded and unstructured finite element meshes is problematic because often the areas of interest are obscured by the density of vectors in highly refined areas of the mesh.

While it is possible to conditionally render vectors, for example, at every other node, generally the results are inadequate. However, the finite element method provides a direct mechanism for evaluating data in element interiors and forms the basis for computing overlaid vector grids which can have a grid point density or spatial dimensionality different from the computational grid.

One of the key aspects of using an interpolated, overlaid grid is that the interpolation is performed using the same shape functions that are used in the simulation code. The variation of the field variables is known and continuous everywhere in a given element. This is a clear advantage over interpolation schemes for finite difference and finite volume grids which introduce errors above and beyond the normal discretization errors from the simulation.

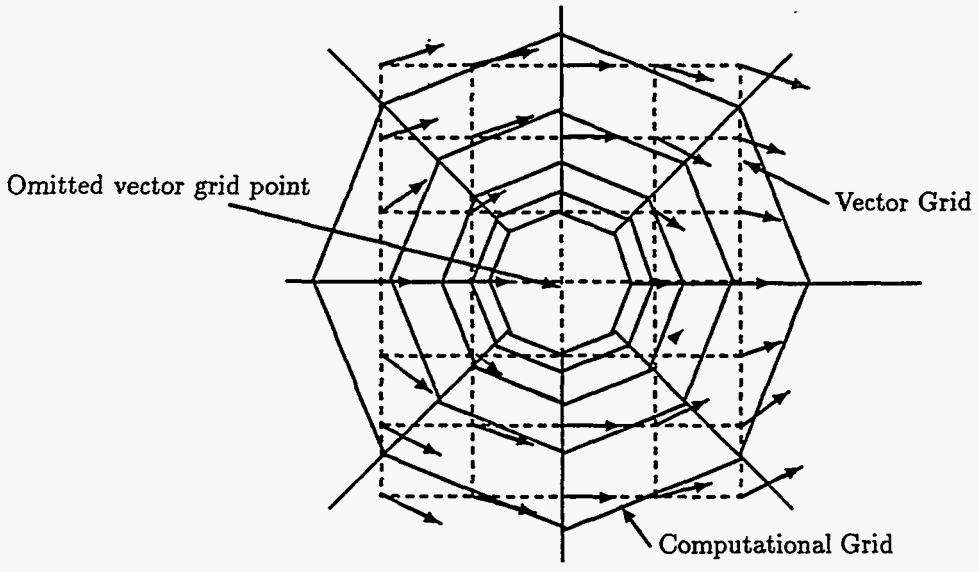


Fig. 6: 5x7 Vector grid and vector plot on a portion of a 2-D mesh with a hole.

For the computation of a vector grid, the expansion of both the nodal coordinates and velocities in Eq.'s 4-5 are required. Here, $\bar{\mathbf{x}}$ represents known (i.e., prescribed) spatial coordinates of the vector grid, $\bar{\mathbf{u}}$ is the velocity to be interpolated at the given spatial coordinate, $\tilde{\xi} = (\xi, \eta, \psi)$, and $\mathbf{x}_i$ and $\mathbf{u}_i$ are the known nodal coordinates and velocities.

$$\bar{\mathbf{x}} = \sum_{i=1}^{Nnpe} N_i(\tilde{\xi})\mathbf{x}_i \tag{4}$$

$$\bar{\mathbf{u}} = \sum_{i=1}^{Nnpe} N_i(\tilde{\xi})\mathbf{u}_i \tag{5}$$

In two dimensions, the computation of the vector grid requires the specification of a bounding box and the corresponding size of the vector grid, e.g. a $5x7$ grid overlaid on a portion of a 2-D domain as shown in Figure 6. Once the prescribed vector grid coordinates are known, Eq. 4 must be solved for $\tilde{\xi}$ which then may be used to evaluate Eq. 5 and sample the velocity field at each vector grid coordinate.

For the two-dimensional case, Eq. 4 may be expanded and written in terms of the unknown natural coordinates (ξ, η) as in Eq. 6. The coefficients in Eq. 6 are derived from the known nodal coordinates as shown in Eq. 7.

$$\begin{aligned} \alpha_1\xi + \alpha_2\eta + \alpha_3\xi\eta &= b_1 \\ \gamma_1\xi + \gamma_2\eta + \gamma_3\xi\eta &= b_2 \end{aligned} \tag{6}$$

$$\begin{aligned} \alpha_1 &= -x_1 + x_2 + x_3 - x_4, \quad \gamma_1 = -y_1 + y_2 + y_3 - y_4 \\ \alpha_2 &= -x_1 - x_2 + x_3 + x_4, \quad \gamma_2 = -y_1 - y_2 + y_3 + y_4 \\ \alpha_3 &= \ x_1 - x_2 + x_3 - x_4, \quad \gamma_3 = \ y_1 - y_2 + y_3 - y_4 \\ b_1 &= 4\bar{x} - (\textstyle\sum_{i=1}^{N_{npe}} x_i), \qquad b_2 = 4\bar{y} - (\textstyle\sum_{i=1}^{N_{npe}} y_i) \end{aligned} \tag{7}$$

The solution of Eq. 6 for ξ and η is achieved by first solving the quadratic equation in Eq. 8 for η, and then substituting the result in Eq. 9 to compute ξ. After computing (ξ, η), the shape functions, N_i, may be evaluated, and the velocity vector at $\bar{\mathbf{x}}$ evaluted.

$$\gamma_2\alpha_3\eta^2 + (\gamma_2\alpha_1 + \gamma_3 - \gamma_1\alpha_2 - \alpha_3 b_2)\eta + (\gamma_1 b_1 - \alpha_1 b_2) = 0 \tag{8}$$

$$\xi = (b_1 - \alpha_2\eta)/(\alpha_1 + \alpha_3\eta) \tag{9}$$

In three dimensions, the inverse computation for the natural coordinates is somewhat more involved. To begin, the following system must be solved for (ξ, η, ψ).

$$\begin{aligned}
\alpha_1\xi + \alpha_2\eta + \alpha_3\psi + \alpha_4\eta\psi + \alpha_5\xi\psi + \alpha_6\xi\eta + \alpha_7\xi\eta\psi &= b_1 \\
\beta_1\xi + \beta_2\eta + \beta_3\psi + \beta_4\eta\psi + \beta_5\xi\psi + \beta_6\xi\eta + \beta_7\xi\eta\psi &= b_2 \\
\gamma_1\xi + \gamma_2\eta + \gamma_3\psi + \gamma_4\eta\psi + \gamma_5\xi\psi + \gamma_6\xi\eta + \gamma_7\xi\eta\psi &= b_3
\end{aligned} \tag{10}$$

$$\begin{aligned}
\alpha_1 &= -x_1 + x_2 + x_3 - x_4 - x_5 + x_6 + x_7 - x_8, & \alpha_2 &= -x_1 - x_2 + x_3 + x_4 - x_5 - x_6 + x_7 + x_8 \\
\alpha_3 &= -x_1 - x_2 - x_3 - x_4 + x_5 + x_6 + x_7 + x_8, & \alpha_4 &= x_1 + x_2 - x_3 - x_4 - x_5 - x_6 + x_7 + x_8 \\
\alpha_5 &= x_1 - x_2 - x_3 + x_4 - x_5 + x_6 + x_7 - x_8, & \alpha_6 &= x_1 - x_2 + x_3 - x_4 + x_5 - x_6 + x_7 - x_8 \\
\alpha_7 &= -x_1 + x_2 - x_3 + x_4 + x_5 - x_6 + x_7 - x_8, & b_1 &= 8\bar{x} - \left(\textstyle\sum_{i=1}^{N_{npe}} x_i\right)
\end{aligned} \tag{11}$$

The β and γ coefficients may be computed by substituting the y and z grid and nodal coordinates in Eq. 11.

Eq. 10 may be written as $f(\tilde{\xi}) = 0$ for the purpose of applying a Newton iteration. The solution to Eq. 10 then proceeds by using Cramer's rule to first solve the linear problem in Eq. 12 for an initial estimate for $\tilde{\xi}$. This estimate for $\tilde{\xi}$ is then used as a starting point in a Newton iteration using Eq.'s 13-14.

$$\begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 \\ \beta_1 & \beta_2 & \beta_3 \\ \gamma_1 & \gamma_2 & \gamma_3 \end{bmatrix} \begin{Bmatrix} \xi \\ \eta \\ \psi \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_2 \\ b_3 \end{Bmatrix} \tag{12}$$

$$[A]\{\Delta\tilde{\xi}\} = \{f(\tilde{\xi})\} \tag{13}$$

where

$$[A] = \frac{\partial f(\tilde{\xi})}{\partial \tilde{\xi}} \tag{14}$$

Newton's method delivers quadratic convergence and is guaranteed to converge if the initial estimate for $\tilde{\xi}$ is in the neighborhood of the solution. Eq. 12 provides a reasonable estimate for $\tilde{\xi}$ and results in a solution in only 3–4 iterations with a residual of 10^{-4} (i.e., $\|f(\tilde{\xi})\| \leq 10^{-4}$). This is a relatively inexpensive computation on a per vector grid point basis.

In both the 2-D and 3-D case, element bounding boxes are used to determine if a grid point is resident in a candidate element. The computation of natural coordinates which are not bounded by ± 1 signals that the next candidate element should be used for the current vector grid point.

Particle Traces

Another commonly used technique for flow visualization is particle tracing. In this work, particle tracing refers to the process of tracing out the path which a particle would follow when subjected to a time dependent velocity field. The realization of particle tracing can take the form of continuous traces of particle paths, or snapshots of particles emitted at a fixed frequency which are referred to as "time lines". Here, the emphasis is on the former.

Continuous particle traces can be constructed by computing a sequence of points $(\mathbf{x}^0, \mathbf{x}^1, ..., \mathbf{x}^n)$ by integrating the following system of ordinary differential equations numerically.

$$\dot{\mathbf{x}} = \mathbf{u}(\mathbf{x}, t) \tag{15}$$

The initial conditions for Eq. 15 are given in the form of a *particle rake*, i.e., the particle rake provides the initial set of coordinates at $t = 0$ for the subsequent particle tracing. The integration of Eq. 15 is performed using a second-order accurate Runge-Kutta integrator. There are many time integration algorithms suitable for the system in Eq. 15. However, second-order Runge-Kutta provides a reasonable trade-off between accuracy and interactivity.

Figure 7 shows a simple schematic of a particle path being traced through several elements in the boundary layer near a cylinder wall. As illustrated in the schematic, the particle trace passes through

elements which may be numbered in essentially a random order, e.g., elements I and J in Figure 7 need not be numbered contiguously.

The algorithm used to compute particle traces is as follows. First, a particle rake is specified as the seed point (initial conditions) for the integration of Eq. 15. A time step is taken holding the current velocity field fixed, and new coordinates for all particle positions are evaluated. Using the new known particle positions the current natural coordinates ($\tilde{\xi}$) are computed according to the vector grid algorithm outlined above. The particle velocities are then evaluated at the new time level. Note that a new velocity field may be read from disk if necessary for in-betweening during this step. Using the current set of particle natural coordinates, the new velocity field is sampled, and the next time integration step is taken. This sequence continues until a user specified time or until the end of the simulated time.

During the computation of the natural coordinates, the fact that the natural coordinates are bounded is used to establish criteria for moving to an adjacent element. If a natural coordinate exceeds its bounds ($-1 \leq \tilde{\xi} \leq 1$) during the solution of Eq. 10, then the natural coordinate is used to determine which face of the element the particle has traversed. For example, $\xi \geq 1$ signals that the particle has crossed face 1 of the element (refer to Figure 3 and Table 2). The element number and face number are used to first compute a hash value. Then this hash value is used to identify the paired surfaces in the master surface list and their corresponding element numbers. Knowing the current element number, the adjacent element number is retrieved from the master surface list and used for subsequent natural coordinate computations.

As pointed out in Hultquist (1994), the use of element adjacency can speed up the process of particle tracing. However, the *spatial coherence* which Hultquist relies upon can not be implemented in a simple (i, j, k) form for a finite element mesh. Further, specifics about geometry can not be used without sacrificing the ability to treat arbitrarily complex geometry.

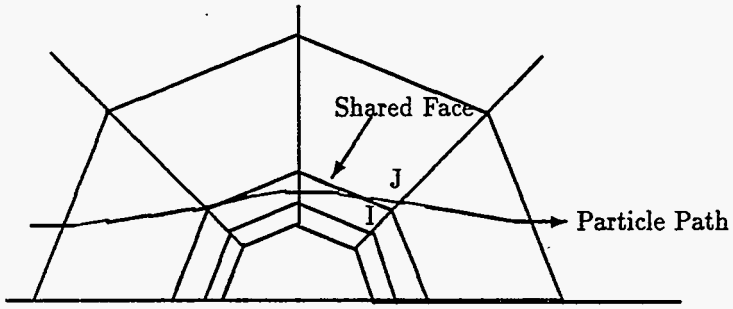


Fig. 7: Schematic of particle tracing.

Data Abstraction

The surface rendering techniques employed in both the TWEENER and in GRIZ allow the specification of user defined material properties such as specularity, diffusivity, and color, as well as the color and position of up to eight light sources in a simplified scene description. Both vector (wire frame) and solid images (i.e., surface rendering) can be generated interactively, with field variables such as time-dependent pressure or effective stress optionally mapped onto the surface of the time-varying geometry.

Mapping field variables onto surface color implies that a physically meaningful scalar field is available which can adequately represent the results. To perform a table lookup or function evaluation for vertex color specification, the derived scalar valued field variable must be appropriately scaled. The scalar valued variable, ϕ, is computed as:

$$\phi = \left[\frac{\theta - \theta_{min}}{\theta_{max} - \theta_{min}} \right] \tag{16}$$

In Eq. 16, θ_{min} and θ_{max} are global minimums and maximums over the entire simulation, i.e., both spatially and temporally. The scalar valued function, ϕ, is normalized (i.e., $0 \leq \phi \leq 1$) for use as the entry into either a color lookup table, or for a color map function evaluation procedure. Here, θ can be taken to represent the scalar valued function of interest, e.g., vorticity magnitude, velocity magnitude, helicity, effective stress, etc.

3. Video Animation System

The type of animation technique used for the finite element data considered herein is referred to as key-frame animation because it relies on "key frames" of data computed at discrete time levels by the simulation code. In order to produce enough animation frames for video (30 frames per second of video), it is necessary to interpolate between the key-frames produced by the analysis code. This section briefly discusses the type of interpolation used for animating the key-frame data, and the use of a "direct-to-videodisk" animation strategy.

In-Betweening and Animation Strategy

The in-betweening process for Eulerian key-frame animations relies on the linear interpolation of data at discrete time intervals. However, Lagrangian finite element datasets can require special treatment when rigid body dynamics and contact/impact of deformable bodies are of interest.

As an example, great care must be taken when applying an in-betweening algorithm to the flight of an object before impact. In this case, user input is required to identify objects by material in the mesh for material specific conditional interpolation. The conditional interpolation must be activated and de-activated at the appropriate times in the animation to capture the impact event adequately. If this approach is not taken, artifacts can be produced when key-frames span an important event such as the impact of two bodies. The alternative is to store as many time states as possible at times just before and after the impact event, since it may be practically impossible to place a key-frame precisely at the time of impact. When rigid body dynamics are not important, a simple linear in-betweening is sufficient for both the spatial coordinates and the corresponding field variables.

In the in-betweening process, "disk-based" animation systems require the animator to render and store images on rotating disk, only to later re-display, scan convert and record the images on video tape or videodisk. The scan conversion step is also referred to as digital encoding, and is the process whereby RGB color signals are converted to composite signals such as NTSC (National Television Standards Committee) (Winkler, 1991). In a disk based animation system, 60 seconds of animation using 24-bit frames with a resolution of 640x480 pixels requires about 1.66 Gigabytes of disk (i.e., 1.66 GB per minute of video play time). By comparison, a videodisk holds approximately 24 minutes of video.

In the disk-based model, animations are batch processes at best, often requiring batch rendering, and consuming large amounts of disk resources. The approach adopted in MDG was to avoid the use of rotating disk for animation frames by recording the frames directly to a laser videodisk. The desired interactivity of the visualization application dictated the use of a hardware rendering platform, and the ability to perform scan conversion to a format appropriate for recording video (e.g. NTSC or PAL) "on the fly".

For this purpose, the Silicon Graphics Video Framer was chosen because it provides a secondary low-resolution frame buffer for use during digital encoding and recording. Additionally, the Video Framer provides a V-LAN (Video Local Area Network) transmitter enabling the code for control of the video transport device to be essentially independent of the actual video transport device. For example, the same coded instructions can be used with a frame accurate video tape recorder instead of a laser videodisk. Figure 8 shows the MDG video equipment, workstation, and preview monitors for the direct-to-disk animation strategy.

The trade-off with the "direct-to-disk" animation system is that the hardware rendering process can produce somewhat lower image quality in comparison to other rendering techniques (e.g., ray tracing). However, it offers the ability for the analyst or researcher to immediately preview dynamic processes both interactively on the workstation, and via playback from the videodisk. In addition, key frames may be previewed individually, and optionally written to a 24-bit HDF file (NCSA, 1989) for the generation of 35mm slides or viewgraphs. Alternatively, key frames may be viewed in an NTSC mode to permit the user to compensate for color changes and image degradation characteristic of NTSC video. When compared to disk-based animation schemes, this approach reduces the total time required to produce an animation segment and the impact on computing resources.

Temporal Aliasing

The accurate transformation of temporally discrete key-frames into a continuous video segment requires some knowledge of the time scales for the physics involved in the simulation. A failure to consider the important time scales in a simulation can result in misleading behavior in the video. For example, aliasing occurs when the Nyquist sampling theorem for the key frames is violated.

A good example to illustrate this problem is vortex shedding behind a circular cylinder. The Nyquist

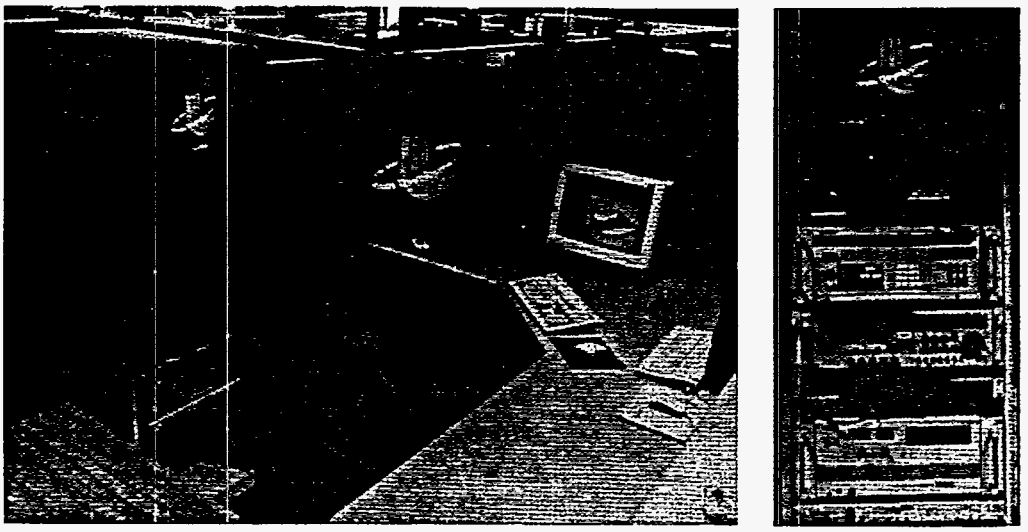


Fig. 8: MDG Video Animation System.

theorem states that a sampling frequency of at least twice the highest frequency should be used. For a Reynolds number of 100, the Strouhal number (non-dimensional frequency) is 0.21. For the case when N_{per} samples per period are made, and the simulation is carried out for τ time units, the number of key-frames which must be stored can be computed as:

$$N_{samp} = \tau N_{per} St \tag{17}$$

Considering the case of laminar vortex shedding, experience suggests that a sampling rate of about $10 - 20$ samples per period is required to accurately represent the temporal behavior. Thus, for the vortex shedding example with $\tau = 100$ time units, $St = 0.21$, and $N_{per} = 20$, $N_{samp} = 420$. In this example, a temporal undersampling can result in vortices moving upstream in the video when they should be convected downstream. Of course during a non-periodic transient, the appropriate sampling rate is difficult to estimate. However, a good rule of thumb is to sample at a rate of at least 10 key-frames per time constant. For example, the time constant could be the time for an elastic longitudinal wave to propagate 1 characteristic length in a solid. For periodic wave-like phenomena, the characteristic grid frequency, i.e., the highest frequency which the grid can adequately resolve, may be used to estimate time scales of interest.

4. Visualization Case Studies

In this section, several examples are presented to illustrate the application of the visualization algorithms described in the previous sections. The first problem considered is time-dependent vortex shedding for flow past a circular cylinder at $Re_D = 100$ based upon the cylinder diameter.

Figure 9a shows a snapshot of the stream function, while 9b shows the instantaneous vorticity. In both cases, the colormap was specifically designed to emphasize zero crossings and clearly discriminate between positive and negative values of stream function and vorticity in order to reveal the Karman vortex street downstream of the cylinder. The effect of the colormap on the stream function is to provide pseudo-contours which emphasize stream function values which are close to the stagnation stream function value.

In Figure 9b, the colormap helps to discriminate regions of positive and negative vorticity. The dark colored regions of vorticity show fluid which is rotating counter clockwise, while the light colored regions are rotating clockwise. The separation in color near the zero-vorticity value helps to discriminate the attached secondary vortex in the near-wake from the upper and lower vortices which are being shed.

A $20x20$ vector grid is shown overlaid on the finite element mesh in Figure 9c. This vector grid demonstrates the use of a uniform vector grid with a non-uniform mesh and reveals the counter clockwise recirculation zone of the attached eddy in the near wake of the cylinder. With a vector grid, elements may be sub-sampled or super-sampled during the generation of vector plots in regions where the mesh is graded.

While field plots of vorticity and velocity vectors are useful, a complete synthesis of transient results can not be made quantitative without the use of time history data. Figure 9d shows a velocity time history plot for a grid point on the centerline of the wake downstream of the cylinder. The time history data in this plot is required to extract the frequency content of the velocity field, i.e., in order to obtain a Strouhal number.

The second case involves the flow over a flat plate with a circular post attached. Figure 10a shows instantaneous pressure isosurfaces at $\tau = 400$ time units for $Re_D = 100$. The isosurfaces are raised slightly above the plate revealing the stagnation point at the leading edge of the plate. Similarly, a stagnation point can be observed in the isosurface attached to the upstream side of the post. The sculptured isosurface in the downstream pressure field corresponds to the regions of high and low pressure associated with the regions of high and low vorticity.

Figure 10b shows the instantaneous vertical component of vorticity at $\tau = 400$ time units. The dark isosurfaces indicate regions where the vertical vorticity component is positive, and the lighter isosurfaces show regions where the vorticity is negative. Along the top symmetry plane, the vorticity isosurfaces provide contour lines which are very similar to the vorticity snapshot shown in Figure 9b.

While the vertical vorticity component appears to be somewhat two-dimensional, the particle traces in Figure 10c clearly reveal the three-dimensional nature of the flow field in the wake of the cylinder. The particle trace snapshot at $\tau = 100$ time units is based upon the particle paths taken by 4 particles released

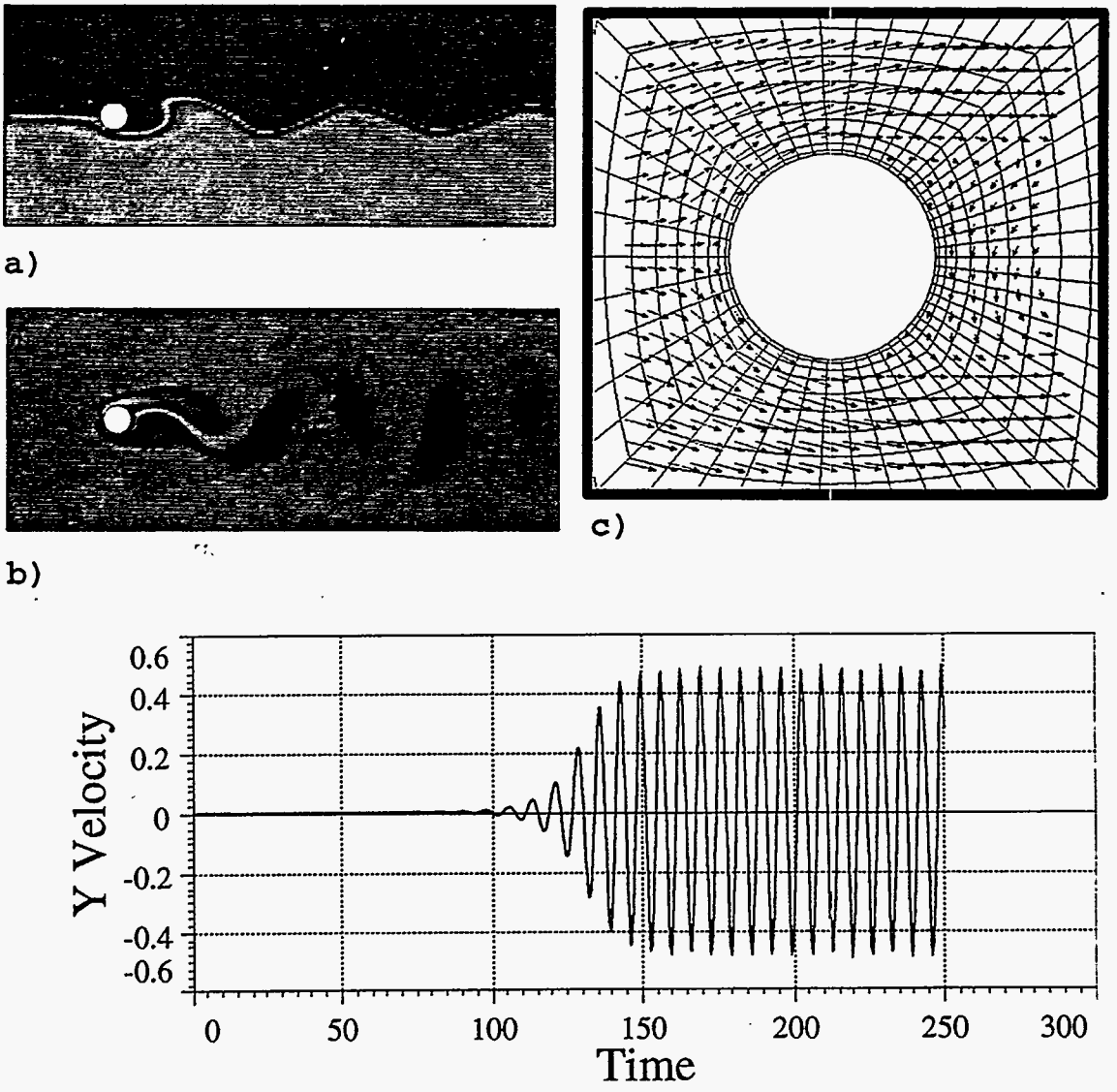


Fig. 9: 2-D Vortex shedding with $Re_D = 100$. a) Stream Function, b) Vorticity, c) 20x20 Vector Grid, d) Time History Plot.

at $\tau = 10$ just upstream of the post. The particle paths show that there is a strong upward component of velocity during the quasi-steady period ($10 \leq \tau \leq 100$) during which two symmetric vortices stand just downstream of the post. That is, the standing vortices in this case are the three-dimensional counterpart of the standing vortices for the two-dimensional vortex shedding problem.

In Figure 10, all of the snapshots make use of a reference surface, i.e., a list of polygons which is used to define the no-slip surfaces in the flow problem. In Figure 10c, the pressure has been color mapped onto the reference surface to provide a visual cue by illustrating the upstream stagnation point and the downstream separation points. Thus, the reference surface is used not only to highlight geometric aspects, but to give visual hints on the flow direction to the observer.

Figure 11 shows pressure isosurfaces in the fluid surrounding an elastic streamline body in an acoustic fluid-structure interaction problem. In this image, the scaled displacements are shown on the inset elastic body with the fluid pressure color mapped onto the surface (a color-mapped reference surface). A horizontal cutplane is used with isosurfaces and the edge definitions of the mesh in order to emphasize both spatial length scales and to highlight the orientation of the computational domain. This example shows the importance of combining visualization techniques for fluid-structure problems (i.e., reference surfaces, cutplanes, isosurfaces, edges, and exaggerated Lagrangian displacements).

In Figure 12, a simulation which made use of a form of volumetric adaptivity is shown. The top row of images show snapshots of the geometry only, while the second row shows effective stress, and the third row shows effective plastic strain. A vertical plane of symmetry is used to permit visualization of

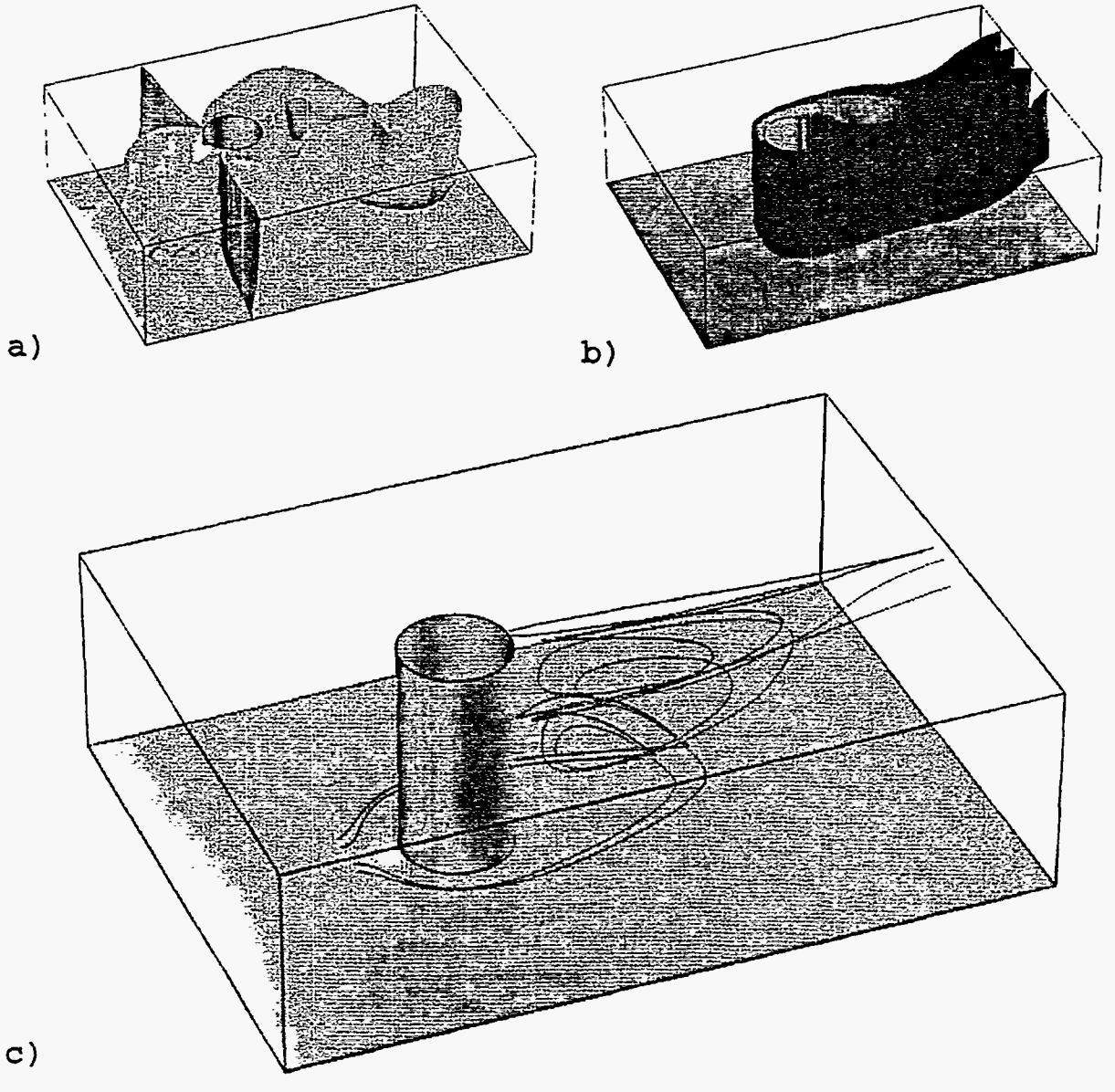


Fig. 10: Vortex shedding with a post and plate $Re_D = 100$. a) Pressure Isosurfaces at $\tau = 400$, b) Z-vorticity Isosurface at $\tau = 400$, c) Particle traces for $10 \leq \tau \leq 100$.

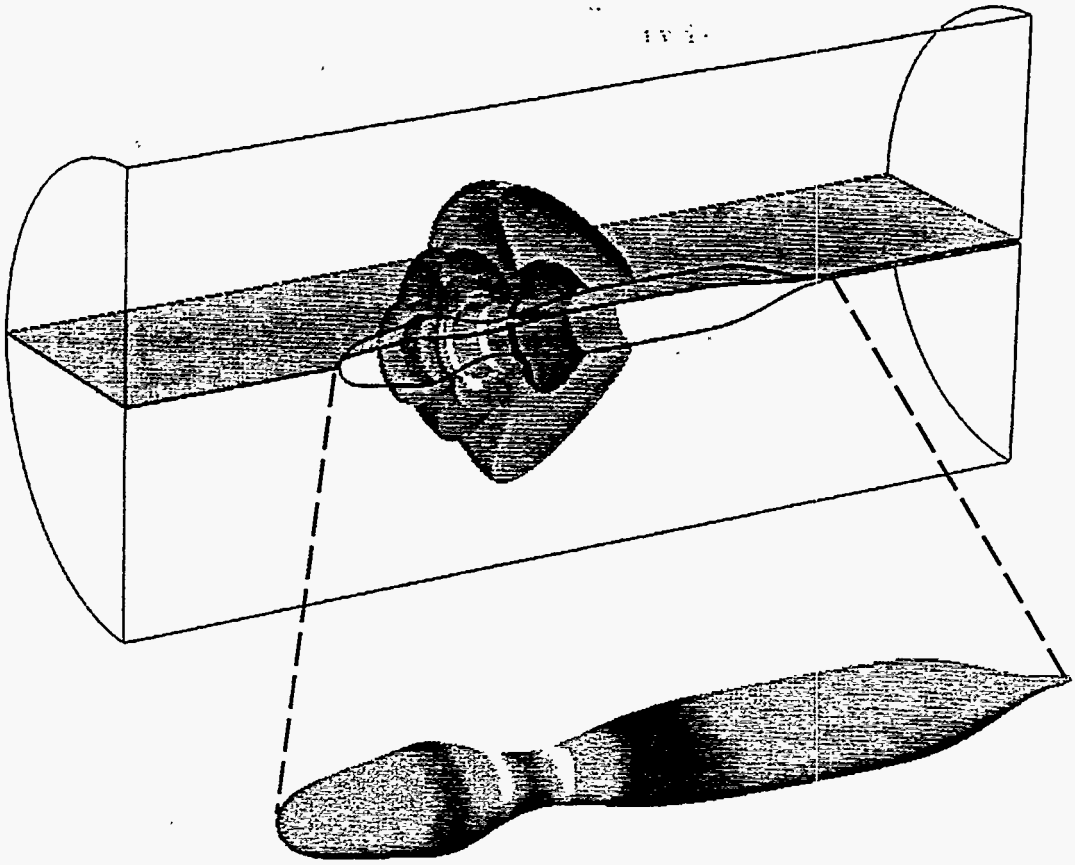


Fig. 11: Perturbation pressure isosurfaces and shell displacements (displacement scale factor 3000).

the volumetric adaptive region where the projectile impacts the plates. For the bottom row of images, thresholding was applied to cutoff the color mapping at 10% effective plastic strain. This technique is useful when the variable being color mapped is somewhat spatially localized. This example demonstrates the applicability of the SEVA algorithm for large-scale, nonlinear continuum mechanics computations where sliding interfaces and adaptivity are important.

5. Parallel Visualization Issues

Scientific visualization requirements for grand challenge problems, i.e., problems with 10^6 to 10^9 grid points, pose many difficulties in light of the current generation of parallel computers. In the model of analysis whereby the process of mesh generation, simulation, and visualization may be repeated several times as part of a computational investigation, the implicit assumption is that there will be sufficient disk space available to hold the required data for a subsequent visualization study.

In a visualization study, derived quantities such as vorticity, wall shear stress, stress components, effective stress, etc. are required for data abstraction. It is important to note that the investigator may not know a priori that visualization of a specific quantity, e.g., effective stress, is even required until after some initial interrogation of the data has been performed. This requires two things: first, the original primitive data must be available for the computation of the derived variables, and second, there must be sufficient compute power to perform the necessary operations for the visualization process. While this is usually the case for traditional CRAY supercomputers (and the problem size which they can handle), when considering the increased number of grid points which will be used in simulations on massively parallel computers (MPCs), the available disk space on these machines is far from adequate.

Consider the example of an acoustic fluid-structure interaction simulation where a highly discretized fluid region is required. PING (Christon, 1993) output is usually in the form of 3 pressure components: the incident pressure, the elastic-scattered pressure, and the total perturbation pressure (neglecting the smaller number of structural displacements, velocities and accelerations). Assuming for the moment that 32-bit floating point values are sufficient to represent the data, and neglecting the data defining how nodes are connected to elements, the number of states to faithfully represent temporal wave-like phenomenon in the pressure field as set by the Nyquist theorem can be estimated in terms of the grid frequency.

Table 5: PING - State Disk Storage Requirements for a $10^6 - 10^9$ grid Point Problem.

No. of Grid Points	No. of States	Storage per State [MBytes]	Total Storage [GBytes]
10^6	500	12	6
10^7	1000	120	120
10^8	2000	1200	2400
10^9	4000	12000	48000

The term "grid frequency" refers to the highest discrete frequency which can be resolved by a given mesh. The Nyquist theorem states that a sampling frequency twice the highest frequency of interest must be used to avoid temporal aliasing. This translates into the number of states of data which must be saved if an accurate animation of wave propagation is to be created. For the 256 processor Meiko CS-2 at LLNL, upwards of 10^6 node simulations will be tractable in terms of the necessary FLOP rate and available memory per processor. Table 5 shows estimates of the storage requirements for PING data based on calculations with $10^6 - 10^9$ grid points.

In Table 5, the number of states is based upon a sampling rate of 20 states per temporal cycle, and the storage per state is based upon storing 3 32-bit floating point values per grid point. Clearly, if visualization is to occur after the simulation, and permit the computation of derived results, then 6 GBytes of disk space will be required for a 10^6 grid point mesh while 48 TBytes would be necessary for a 10^9 grid point computation. The disk requirements for a 10^9 grid point problem may make the data management difficult and possibly intractable. These estimates for disk storage account only for the output state datasets and do not include the disk storage required for the mesh, or the time history datasets.

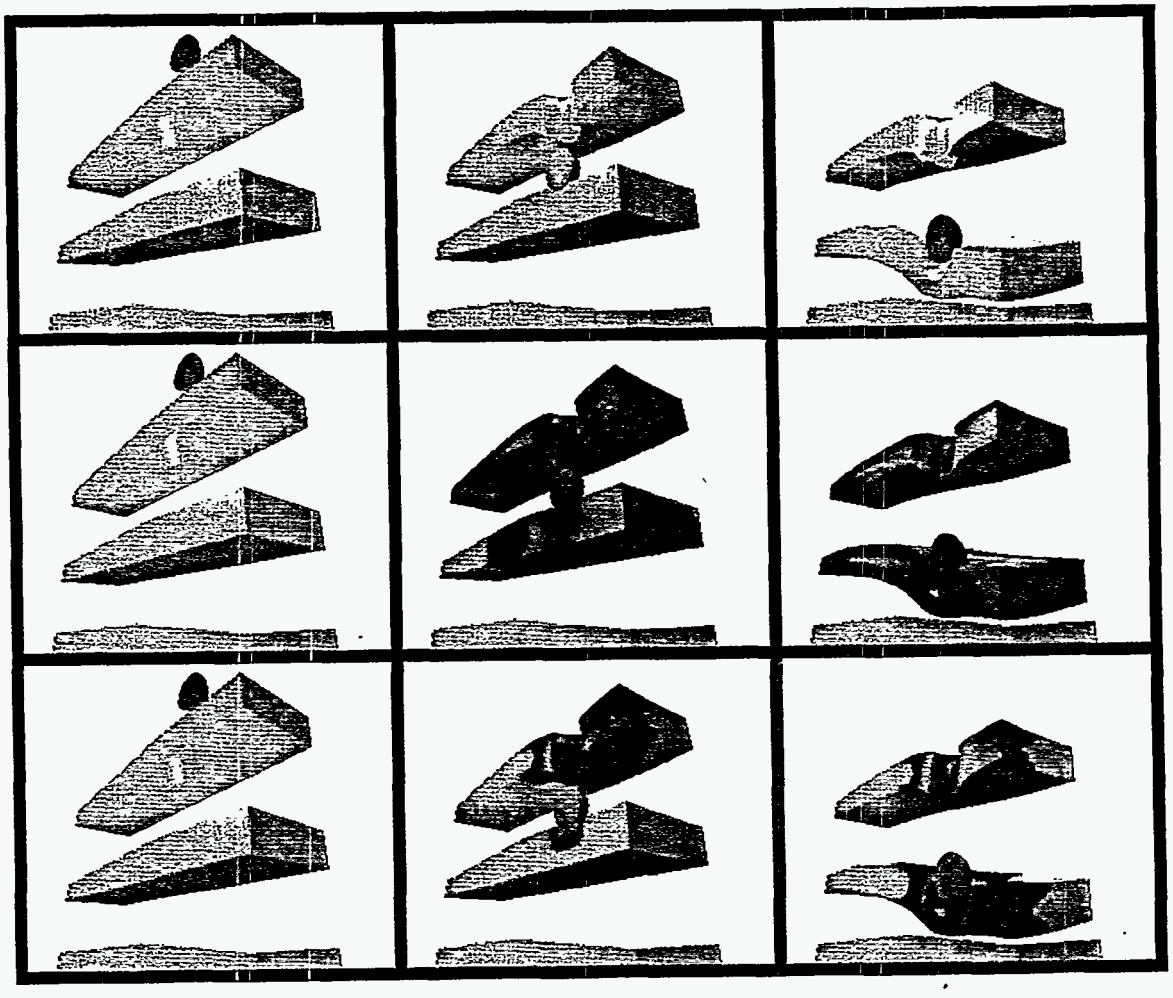


Fig. 12: Snapshots of particle-plate impact showing displacements (top), effective stress (middle), and plastic strain (bottom).

Of course, this estimate of required disk space assumes that the temporal behavior is of enough interest to prompt the investigator to look at transient phenomena. This estimate also assumes that the visualization study will make use of the high quality volume data which was computed on the MPC. For example, isosurfaces of the pressure field in the volume of the discrete domain will be visualized, as opposed to essentially gutting the entire volume of data in favor of performing visualization studies on the external boundary of the domain.

The reason for the interest in the interior volume data stems from several things. First, it is assumed that the MPC simulation was performed to obtain all the data, and discarding the interior volume of data would be a gross waste of both time and money in terms of the computer resources expended for the simulation. Second, visualizing boundary data is usually not interesting, especially when the boundary conditions are well known and a specified part of the problem. The interesting and unknown results are typically in the volume of data for which the computation was performed. Thus, "dataset gutting" does not appeal to the practical user who is interested in solving real problems. In fact, the idea of gutting datasets clearly falls in the category of ways to say nothing with scientific visualization (Globus and Raible, 1992).

Another alternative for MPC visualization is the idea of co-processing as proposed by Haimes (1994). The basic concept is that because it will not be possible to store the 10 – 100 GBytes of data from the simulation, the visualization must be done on the fly while the simulation is running. Of course, this forces the computational part of the visualization software to run on the MPC concurrently with the simulation. One difficulty with this approach is that the user must select the derived variable to be visualized at the time the simulation is run. Because the data is essentially viewed and discarded, any change in the derived variable requires the simulation and visualization to be re-run. Clearly for a grand challenge problem which requires all the processors of the MPC, re-running problems just because the user wishes to view velocity magnitude instead of a vorticity component is impractical. Such a paradigm shift for analysis and visualization is in vogue, but the model is only suitable for the hero model of supercomputing where a single user may absorb virtually all of the MPC and network resources for an important computation (Nielsen, 1991).

While the hero model may be workable under certain circumstances, the use of MPC's for production, general purpose computing will necessitate the intelligent use of both computer and network resources. The idea of graphical extracts (Globus, 1992) for visualization of transient computational fluid dynamics problems makes use of the ability to communicate graphical primitives to high performance graphics workstations for rendering. This approach requires a distributed visualization application where the I/O and compute intensive portion of the visualization application runs on the MPC, and only primitives such as polygons and derived vertex data are communicated to the workstation. In this visualization model, the amount of data to be moved to the workstation is reduced to a workable level and ultimately avoids discarding volumetric data. Further, this model for MPC visualization permits the user to perform geometric transformations (i.e., rotate, translate, scale), as well as operations such as color palette manipulation interactively on the workstation without being impeded by interactive response on the MPC.

In the distributed, parallel implementation of a visualization tool at LLNL, it is assumed that the processor-local disk space of the Meiko CS-2 will be used. Figure 13 shows a schematic for a parallel, distributed visualization tool. While this is currently limited to approximately 1/2 GByte per processor, the advantages of the processor-local disk space are significant. The processor-local disk is key because the re-assembly of sequential graphics datasets is a sequential bottleneck in a parallel FEM application. Further, the visualization of a grand challenge problem will require parallel computations. If a sequential database were constructed, then this data will ultimately have to be re-partitioned and re-distributed for the visualization study.

Further, not even a super-workstation will handle the sequential dataset for a problem with greater than 10^6 grid points. Many graphics operations such as the computation of isosurfaces, cutting planes, and vector plots, are embarrassingly parallel. Thus if each processor writes a stand-alone graphics state dataset on the processor local disk, there is no sequential bottleneck for the analysis code, and the visualization process can easily exploit the distributed state data. For example, using processor local stand-alone state datasets, the user may wish to first interrogate results on an interesting subdomain before attempting a global view of the data.

The network based MPC visualization model relies upon the communication of graphical primitives via networks. Current ethernet speeds are sufficient to support point-to-point communication of approximately 10,000 polygons per second (assuming a polygon consists of a set of 4 vertex coordinates, 4 vertex normals, and 1 scalar derived quantity per vertex). Thus, the one difficulty is with the network load when many processors are trying to communicate with a single workstation. Interactivity may ultimately

dictate the use of Gigabit networks for parallel distributed visualization in a production environment where an MPC is serving a group of users rather than an individual in the hero model. Ultimately, the network visualization model requires that that there be sufficient disk space available on the MPC to hold the graphics datasets from a transient simulation, and that the network bandwidth must be sufficient to permit the rapid communication of a relatively large number of polygons from multiple processors of the MPC to the workstation.

6. Summary

The successful application of the two-pass surface extraction algorithm (SEVA) to high resolution finite element simulation data has been demonstrated for a broad class of rendering/animation applications. This algorithm permits the complex topology of finite element meshes with arbitrary geometries to be handled efficiently without discarding volumetric data which is crucial to adaptive problems. Further, the implementation of the "direct-to-disk" animation process has proven invaluable in placing an interrogation and animation tool in the hands of the analysts performing the finite element simulations. This tool enables analysts to gain new insight into the voluminous data generated by large scale simulations which ultimately increases the value of such simulations.

The issues regarding the use of parallel supercomputers for both grand challenge problems and routine production computing are somewhat involved and are related in general to the need for additional disk space, high bandwidth networking, continued scientific visualization software development, and efficient input/output (I/O). Although the emphasis herein has been upon the Meiko CS-2, similar issues are valid for the INTEL Paragon, the IBM SP-2 and CRAY T3D. The issues which will hamper the progress of FEM applications and visualization on MPC's are itemized below.

- The processor-local disk space of the Meiko CS-2 is one of its key architectural advantages over competing parallel supercomputers. For both production and grand challenge problems, sufficient disk and archival storage (ranging from O(10) GBytes to O(100) TBytes) is necessary to fully utilize the Meiko. Using the processor-local disk is key to having effective parallel distributed visualization tools.
- Network infrastructure in the form of Gigabit networks for distributed visualization as well as for archival storage will be required for production, general purpose supercomputing.
- The current I/O options for unstructured grid applications pose sequential bottlenecks for parallel FEM codes. Constructing sequential graphics databases for grand challenge problems will not be feasible in the limit of machine capacity problems.

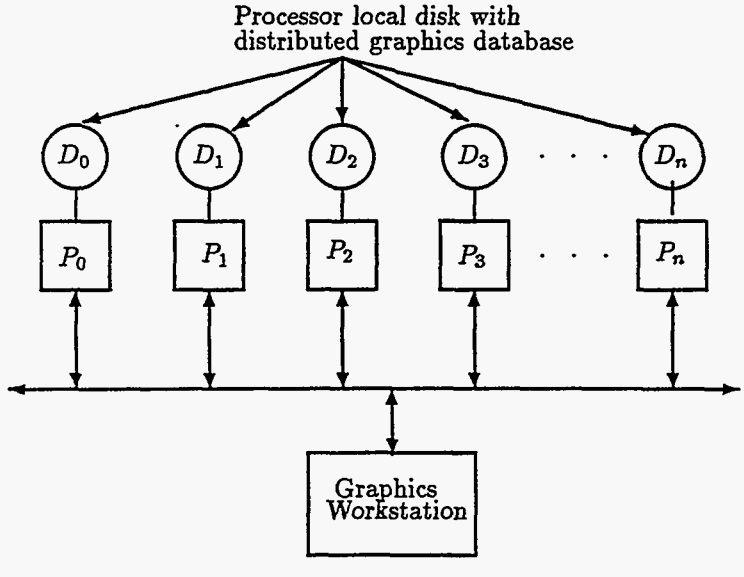


Fig. 13: Network based distributed visualization model.

REFERENCES

McCormick, B. H., T. A. Defanti, and M. D. Brown (1987). "Visualization in Scientific Computing," *Computer Graphics*, Vol. 21, No. 6.

Belytschko, T. and S. E. Law (1985). "An Assembled Normal Algorithm for Interior Node Removal in Three-Dimensional Finite Element Meshes," *Engineering with Computers*, Vol. 1, pp. 55–60.

Hughes, T. J. R. (1987). *The Finite Element Method*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, pp. 42–44.

Winget, J. M. (1988). "Advanced Graphics for Finite Element Results Display," *Advanced Topics in Finite Element Analysis*, ASME PVP Vol. 1, pp. 53–57.

Whirley, R. G. and B. E. Englemann (1993). "DYNA3D: A Nonlinear, Explicit, Three-Dimensional Finite Element Code for Solid and Structural Mechanics - User Manual," University of California, Lawrence Livermore National Laboratory, UCRL-MA-107254 Rev. 1.

Tenenbaum, A. M., Y. Langsam and M. J. Augenstein (1990). *Data Structures Using C*, Prentice Hall, Inc., Englewood Cliffs, New Jersey, pp. 492–500.

Press, W. H., B. P. Flannery, S. A. Teukolsky and W. T. Vetterling (1987). *Numerical Recipes, The Art of Scientific Computing*, Cambridge University Press, New York, New York. pp. 229–237,

Christon, M. A. and T. Spelce (1992). "Visualization of High Resolution, Three-Dimensional, Nonlinear Finite Element Analyses," proc. *IEEE Visualization '92*, pp. 321–331.

Dovey D. J. and T. E. Spelce (1993). "GRIZ Finite Element Analysis Results Visualization for Unstructured Grids," Draft manual - version 2.

Lorensen, W. E. and H. E. Cline (1987). "Marching Cubes: A High Resolution 3-D Surface Construction Algorithm," *Computer Graphics*, Vol. 21, No. 4, pp. 44–50.

Hultquist, J. (1994). "Improving the Performance of Particle Tracing in Curvilinear Grids," AIAA 94-0324.

Winkler, D. (1991). "Video Technology for Computer Graphics," *Siggraph '91 C17 Course Notes.*

National Center for Supercomputing Applications (1989). *NCSA HDF Call Interfaces and Utilities.*

Christon, M. A., S. J. Wineman, G. L. Goudreau, and J. D. Foch (1993). "A Mixed Time Integration Method for Large Scale Acoustic Fluid-Structure Interaction," Lawrence Livermore National Laboratory, UCRL-JC-117820.

Globus, A. and E. Raible (1992). "13 Ways to Say Nothing with Scientific Visualization," NASA Ames Research Center, Report RNR-92-006.

Haimes, R. (1994). "pV3: A Distributed System for Large-Scale Unsteady CFD Visualization," AIAA 94-0321.

Nielsen, D. E. Jr. (1991). "General Purpose Parallel Supercomputing," UCRL-ID-108228, Lawrence Livermore National Laboratory.

Globus, A. (1992). "A Software Model for Visualization of Time Dependent 3-D Computational Fluid Dynamics Results," NASA Ames Research Center, Report RNR-92-031.