AIM

Association for Information and Image Management

1100 Wayne Avenue, Suite 1100
Silver Spring, Maryland 20910

301/587-8202

Centimeter

Inches

MANUFACTURED TO AIM STANDARDS
BY APPLIED IMAGE, INC.

1 of 1

UCRL-JC-110110
PREPRINT

RECEIVED
JUN 24 1993
OSTI

Visualization of High Resolution, Three-Dimensional, Nonlinear Finite Element Analyses

Mark A. Christon
Thomas E. Spelce

This paper was prepared for submittal to the
Visualization '92 Conference
Boston, Massachusetts
October 19-23, 1992

March 24, 1992

Lawrence
Livermore
National
Laboratory

This is a preprint of a paper intended for publication in a journal or proceedings. Since changes may be made before publication, this preprint is made available with the understanding that it will not be cited or reproduced without the permission of the author.

MASTER

DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

Visualization of High Resolution, Three-Dimensional, Nonlinear Finite Element Analyses *

Mark A. Christon
Thomas Spelce
Methods Development Group, L-122
Lawrence Livermore National Laboratory
P.O. Box 808
Livermore, California 94550

19 March 1992

Abstract

A two-pass surface extraction algorithm for adaptive finite element meshes is presented in the context of a visualization study for a particle impact and a turbine containment problem. The direct use of finite element data structures for the computation of external surfaces, surface normals, and derived physical quantities is discussed. An overview of the in-betweening algorithm which compensates for rigid body dynamics is presented with a brief discussion of a "direct-to-videodisk" animation strategy.

1 Introduction

Scientific Visualization is the computational process whereby numerical data is transformed into a visual form which augments the process of discovery and understanding [1]. Visualizing the data generated by finite element analyses presents many challenges due to the geometric complexity and the presence of multiple materials and element types. The data produced by a finite element analysis code can consist of time varying geometry, temperature, stress, strain and possibly mesh topology. Further, the mesh topology may be heterogeneous in two aspects: first, the topology may encompass beam, shell, and hexahedral elements (1-D, 2-D, and 3-D elements), and second, the topology is generally

unstructured. The finite element model may include special features such as sliding interfaces or material boundaries which must be tracked with time. Additionally, treating both Lagrangian and Eulerian grid based data in a single graphics application is complicated by the volume of data produced when high resolution meshes are employed for analyses with many time steps. This is especially true when considering the fact that the data produced is usually of a vector or tensor form (e.g., stress tensor, or a heat flux vector).

This paper focuses upon the application of surface rendering techniques for finite element data primarily from large scale Lagrangian simulations of the transient dynamic response of solids and structures. One of the key components to visualizing the data from this type of analysis code is the ability to rapidly perform surface extraction while maintaining both topological adaptivity in the finite element mesh, and volumetric information. Further, it is necessary to perform some type of data abstraction thereby reducing the order of the data from a second order tensor to a scalar field which is physically meaningful. Other key aspects of the visualization application include the ability to selectively display material components of the mesh, provide palette manipulation, map derived physical quantities into surface color, and perform in-betweening for animations. All of this must be done without inhibiting the ability to move randomly through large datasets viewing individual states in simulation time. The finite element analysis codes used as the data generators for the visual-

*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.

ization application discussed herein are DYNA3D [2] and NIKE3D [3].

This paper presents an algorithm for surface extraction which provides the ability to incrementally adapt to topological changes during the finite element simulation. This algorithm is presented in the context of visualizing several large scale finite element analyses which involve both rigid body dynamic motion and localized plastic deformation. In the demonstration calculations, it is possible for elements to be eliminated from the simulation based upon a maximum strain at failure criterion making it necessary to handle the arbitrary adaptive topology of the mesh. The process of data abstraction and "in-betweening" for this application are briefly presented, with results in the form of a video tape, and still frames for the simulation of the impact of a high velocity particle on idealized turbine stator blades, and failure containment in a turbine engine shroud.

2 Surface Extraction with Volumetric Adaptivity

In the finite element method, the physical domain of interest is approximated by a finite number of smaller sub-domains called elements. The elements typically used consist of eight node hexahedra (bricks), four node quadrilaterals, and two node beams. For the purposes of surface extraction and rendering, both beams and quadrilateral elements may be ignored for obvious reasons. However, the hexahedral elements require special attention because of the need to incrementally render only external surfaces and their associated data. As mentioned above, the solid and shell elements in DYNA3D can be subject to a strain failure criteria whereby the mesh topology will change with time in the simulation. This requires the ability to preserve volumetric data while permitting incremental surface extraction both during the interrogation process and the in-betweening process.

Unlike finite difference models which rely on logically regular grids, finite element models are topology based. Figure 1 illustrates the three element types and their associated local canonical topological node numbering schemes. Here, the local nodal numbering simply implies the use of pointers for indirect addressing using the global node numbers. Thus, the global node numbering for the mesh provides a means for relating nodes to elements. The

nodal connectivity data for a given element may be represented by C_i where $i = 1$ to N_{pe} , and N_{pe} is the number of nodes per element. Typically, finite element models group like elements together in order to simplify the connectivity (e.g., the connectivity is blocked into solid, shell and beam elements). In the case of the solid elements, C_i may be considered to be made up of six reduced sets of connectivity data which associates local nodal numbers with the faces of the hexahedral elements which we refer to as S_i where $i = 1$ to N_{nse} , and N_{nse} is the number of nodes per face. The nodal connectivity for S_i for the canonical local node numbering scheme in Figure 1 is shown in Table 1.

There are several surface extraction algorithms which may be applied to volume finite element data. The first is referred to as the assembled surface normal (ASN) algorithm [4]. In the ASN algorithm, surface connectivity (S_i) is used to construct normal vectors at the vertices of all solid elements allowing internal surfaces to be eliminated by noting that paired surfaces will generate vertex unit normals which essentially cancel out (Figure 2). Thus, by making use of the nodal numbering scheme embedded in the connectivity data (C_i), it is possible to assemble global unit normal vectors as:

$$\mathbf{e}_n = \sum_{i=1}^{N_{hex}} \sum_{j=1}^{N_{nse}} \mathbf{e}_{n_j}^{(i)} \quad (1)$$

where $\mathbf{e}_n$ is a vector of unit normals, the (i) superscript indicates the current element number, N_{hex} is the number of solid elements, and j ranges over the number of nodes per surface of the solid element and is contained in S_i . In Eq. 1, it is understood that the normal vector for each nodal point must be computed for each face to which it belongs in the solid element. After the global assembly has been

Table 1: Face Nodal Connectivity for the Hexahedral 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

performed, all nodal points which are interior to the body have a global unit normal whose components are zero. 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. Further, this approach proves far too computationally intensive requiring $24 * N_{hex}$ unit normal computations plus the global assembly operations for a given topology.

Another frequently used algorithm is the polygon-search/cancel (PSC) algorithm [5] which relies upon the nodal connectivity and avoids some of the problems associated with ASN. One difficulty with PSC is that it relies on a linear search which results in $O((6N_{hex})^2)$ 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 only one form of adaptivity found in finite element applications. It is from this point which we begin our description of the two pass surface extraction algorithm implemented herein.

Whereas the ASN and PSC algorithms essentially discard all volumetric data, it is necessary to retain volumetric information for the adaptive nature of the problems considered herein. By viewing the internal surfaces as paired between adjacent elements as in the ASN algorithm (Figure 2), it is possible to combine features of the ASN and PSC algorithms to efficiently construct the tables necessary to perform an incremental surface extraction for rendering purposes while maintaining volumetric adaptivity. Note that adaptivity in the context of beam and shell elements is essentially automatic given the fact that the analysis code produces the necessary data for the determination of element activity at each time step in the simulation.

It is inherent in finite element node numbering that there can be no free nodes (i.e., two node numbers with identical spatial coordinates) except where there are special model features such as sliding interfaces. Therefore, the key to incremental surface extraction with volumetric adaptivity is the identification of paired internal surfaces in the finite element mesh. This identification is considered to be the first pass in the two pass extraction algorithm discussed below. We define a hash function which will be used in a table of length $6N_{hex}$ for the hexahedral elements, i.e., the worst case possible is that all faces of the brick elements are exterior faces (a real possibility in the most general case).

$$H = \sum_{i=1}^{N_{nec}} w_i \bar{S}_i \quad (2)$$

Here $\bar{S}_i$ represents the surface connectivity in a sorted ascending order, and w_i is a nodal weight. 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 number of nodal points in the finite element mesh. Because of the locally ordered nodal connectivity used in Eq. 2, it is possible to define the nodal weighting as: $w_i = 10^{(i-1)}$. This choice in 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 [6]. However, the goal here is not to use the hash keys for direct addressing, 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.

As the hash values are computed and entered in the hash table, the associated element number and face number are also entered in the table. An indexed heap sort [7] (an $O(N \ln(N))$ algorithm), is used to generate a list of hash values in ascending order. Traversing this 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 ultimately adjacent to each other in the ordered hash list. During the traversal of the hash list, a master surface list which contains a count of surfaces which may be paired (i.e., back-to-back) for each hash value is accumulated to enable the detection of invalid hash function collisions. A maximum of two element and face numbers are included in the master surface list for later use in the second-pass volume adaptive surface extraction.

In the second pass of the surface extraction 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. At this point, the active shell and beam vertices and materials are concatenated to the polygon vertex pointer list. Initially, only unpaired surfaces from the solid elements are added to the master surface list.

As elements in the finite element simulation fail (according to the maximum strain at failure criteria) inactive elements are removed by setting an activity flag to zero. This activity flag is available in the graphics database, and it 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 3 illustrates the elimination of a single element revealing the previously hidden interior surfaces. The element number which has been identified for failure is used to locate and decrement the face count in the master surface list. Any faces in the master surface list with a face count of 1 are considered to be valid surfaces for display. The canonical local node numbering scheme for the solid elements ensures that polygon normals face outwards in the animation scene.

Vertex normals for the active polygons are computed in a two-step process where the first step involves the computation of vertex normals and the assembly of these normals into a global normal vector as:

$$\mathbf{e}_n = \sum_{i=1}^{N_s} \sum_{j=1}^{N_{s,i}} \mathbf{e}_{n_j}^{(i)} \quad (3)$$

where N_s is the number of active surfaces for rendering. During the assembly of the global normals, a count of polygons contributing to each nodal normal is accumulated. This allows an averaged normal at each polygon vertex to be computed. In order 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. This provides a fast mechanism for the detection of edges which exploits the existing finite element data structures already in place.

3 Data Abstraction

The implementation of the finite element visualization code is based upon Silicon Graphics GL library permitting up to eight lights in the animation scene. The surface rendering techniques used 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 (surface rendering) images can be generated interactively, with field variables such as time dependent effective stress or effective plastic strain optionally mapped onto the surface of the time dependent geometry. Multiple selectable color palettes are available for mapping the variation of field variables to surface color.

Mapping field variables onto surface color implies that a representative physically meaningful scalar field is available. For the analysis codes considered here, a stress tensor (σ_{ij}) and effective plastic strain (ϵ_p) are readily available. While the effective plastic strain is a scalar valued function, the stress is a symmetric tensor meaning that there are six unique values of stress available at each node in the mesh, or alternatively at each polygon vertex to be rendered. In the absence of a suitable glyph, the stress tensor may be reduced to an effective stress which is a scalar valued function as:

$$\sigma_{eff} = \left(\frac{3}{2} s_{ij} s_{ij} \right)^{\frac{1}{2}} \quad (4)$$

where

$$s_{ij} = \sigma_{ij} - \frac{1}{3} \delta_{ij} \sigma_{kk} \quad (5)$$

In Eq. 4 and Eq. 5, the indicial notation implies a summation on repeated indices, and δ_{ij} denotes the Kronecker delta.

In order to perform a table lookup or function evaluation for color specification, the scalar valued function must be appropriately scaled. Here, a scalar valued function ϕ is computed as:

$$\phi = \begin{cases} \frac{\sigma_{eff} - \min(\sigma_{eff})}{\max(\sigma_{eff}) - \min(\sigma_{eff})} & \text{Effective Stress} \\ \frac{\epsilon_p - \min(\epsilon_p)}{\max(\epsilon_p) - \min(\epsilon_p)} & \text{Eff. Plastic Strain} \end{cases} \quad (6)$$

In Eq. 6, the min/max values are global minimums and maximums over the entire simulation. The scalar valued function, ϕ , is used as the entry into either a color lookup table, or is passed to a color map function evaluation routine. The individual vertex colors are set based upon a user specified field cutoff parameter. Vertex colors inherit the specified material colors everywhere the field value (ϕ) is below the cutoff.

4 In-Betweening

The type of animation technique used for the finite element data considered herein is referred to as key-frame because it relies on "key frames" of data computed at discrete times levels by the analysis codes. In order to produce enough animation frames for video (30 frames per second of play time), it is necessary to interpolate between the key-frames from the analysis code. This process is often referred

to as "in-betweening", thus the application name "tweener". This section discusses the type of interpolation used for animating the key-frame finite element data, and the use of a "direct-to-videodisk" animation strategy.

For Lagrangian finite element calculations, the presence of rigid body dynamics can cause difficulties in the interpolation process. When rigid body dynamics are not important, a linear in-betweening is used where the time rate of change of spatial coordinates and field variables is computed based on key-frame data. As is typical with grid based transient solutions to systems of partial differential equations, the physical time step size varies during the simulation due to stability constraints. For the spatial coordinates, the following linear interpolation may be used with variable sized time steps between key frames:

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \dot{\mathbf{x}} \Delta t^* \quad (7)$$

where:

$$\dot{\mathbf{x}} = \left(\frac{\mathbf{x}^n - \mathbf{x}^{n-1}}{\Delta t} \right) \quad (8)$$

In Eq. 7, the subscript i identifies the animation frame, while the n superscript in Eq. 8 denotes the key-frame. For a given uniform animation time step size of dt , Δt^* may be computed as :

$$\Delta t^* = t_a - t^{n-1} \quad (9)$$

In Eq. 9, t_a is the accumulated animation time, and is computed as:

$$t_a = \sum_{i=1}^n dt \quad (10)$$

Inactive solid and shell elements are removed from the list of candidate elements for rendering when the interpolated time spans a simulated time level accounting for the topological adaptivity in the simulation. When rigid body dynamics is important (e.g., flight of an object before impact), it is necessary to modify the in-betweening algorithm. In this case, user input is required to identify objects by material in the finite element mesh for material specific conditional interpolation. The conditional interpolation must be activated and deactivated at the appropriate times in the animation in order to adequately capture the correct impact event. If this approach is not taken, artifacts can be produced when key-frames span an important event such as the impact of two bodies. In the case

of the particle - turbine stator blade simulation, the plastic deformation of the silver turbine blades and the associated effective stress (or effective plastic strain) field would appear prematurely in Figure 4 if the conditional interpolation were not applied correctly.

The fact that key-frames essentially span a step function in the data requires the user to preview both key-frames and interpolated frames in order to make a decision about when to re-activate global in-betweening and de-activate the conditional interpolation. In this application, two material conditional step function routines were implemented to handle both rigid body dynamics before impact, and after separation. Of course, the ideal situation would be when the key-frame data accurately captured the impact/separation events.

Typical "disk based" animation systems require the animator to first 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 the pure RGB color signals are converted to composite signals such as NTSC (National Television Standards Committee) [8]. In a disk based animation system, 1 minute of animation using 24-bit frames with a resolution of 640x480 pixels requires about 1.66 Gigabytes of disk (1.66 GB per minute of play time). By comparison, a video-disk holds approximately 24 minutes of video. At best, the disk based animations are batch processes, often requiring batch rendering, and consuming large amounts of disk resources.

The approach taken with the tweener 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 tweener dictated the use of a hardware rendering platform, and the ability to perform scan conversion of images to a format appropriate for recording (e.g. Beta, NTSC, PAL, etc.) on the fly. Therefore, the tweener is based on the Silicon Graphics platform and the Video Framer board.

The Video Framer provides a secondary low-resolution frame buffer for use during digital encoding and recording. Additionally, the Video Framer provides a V-LAN transmitter enabling the code for control of the video transport device to be independent of the actual video transport device. For example, the same coded instructions can be used with a frame accurate video tape deck instead of a

laser videodisk. The trade-off being made with the "direct-to-disk" animation system is that the hardware rendering process produces somewhat lower image quality in comparison to other rendering techniques (e.g., ray tracing), but 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.

The tweener enables the user to preview an animation, then directly record the animation from a window on the workstation to the video disk reducing the total time to produce an animation segment, as well as reducing the impact on computing resources. Key frames may be previewed individually, and optionally written to a 24-bit HDF file [9] for later use such as generation of 35mm slides. Further, the tweener provides the capability to preview still frames in an NTSC mode in order to allow the user to compensate for color changes and image degradation characteristic of NTSC video.

5 Results and Conclusions

The results presented in this section are the from a visualization study performed on several "proof-of-concept" problems using the newly implemented SAND (Slide surfaces with Adaptive New Definitions) technology in DYNA3D. DYNA3D is a nonlinear, time explicit, 3-D finite element code for solid and structural mechanics which was developed at Lawrence Livermore National Laboratory. The SAND technology was developed and implemented in DYNA3D by Robert G. Whirley and Bruce E. Englemann of the Methods Development Group at LLNL. SAND technology extends the capability of DYNA3D to include penetration problems not typically analyzed with a Lagrangian code. The images in Figures 4 and 5 illustrate the SAND technology as applied to the particle - turbine stator blade impact, and the turbine blade fragment containment problems. Each problem required approximately 10,000 solid elements ($O(10,000)$ nodes). The images are organized into three groups for each simulation and are separated according to the physical variable which has been mapped onto the geometry. The simulated time is on the order of milliseconds for both problems. For each simulation, the groups of images may be viewed in parallel so that the geometry (row #1), effective stress (row #2) and plastic strain (row #3) may be directly compared at a fixed simulation time.

In the images where effective plastic strain is

mapped onto the geometry, red indicates the maximum while blue signifies the minimum value of the strain. Similarly, where effective stress is mapped onto the geometry, red corresponds to the maximum value of stress, while blue denotes a minimum threshold value of stress. The first simulation demonstrates a particle traveling at high velocity and impacting idealized turbine stator blades. The sequence of images in Figure 4 clearly illustrates the local, short time hydrodynamic response near the particle coupled with the global, long time structural response of the blade. A symmetry plane is used to simplify the computation, and to illustrate the variation of effective stress and effective plastic strain through the volume of the plates along the symmetry plane.

The second simulation shown in Figure 5 demonstrates a turbine blade fragment striking a containment ring. Elapsed physical time in the simulations is approximately 1 millisecond. At the midpoint in the simulation, the blade fragment has not yet pierced the containment ring, but significant localized plastic deformation is apparent. At the end of the simulation, the turbine blade tip has nearly passed completely through the containment ring. A "shear plug" has been punched from the containment ring and is falling just below the turbine blade tip. The extreme plastic deformation (shown as red in row #3) remains highly localized, while the effective stress (row #2) is quite high everywhere in the containment ring.

The successful application of the two-pass surface extraction algorithm to this class of problems has been demonstrated for a simplified rendering/animation application. This algorithm permits the complex topology of finite element meshes with complex 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 proved invaluable in placing the interrogation and animation tool in the hands of the analysts performing the finite element simulations.

References

- [1] B. H. McCormick, T. A. Defanti, M. D. Brown, "Visualization in Scientific Computing," *Computer Graphics*, Vol. 21, No. 6, 1987.
- [2] R. G. Whirley, J. O. Hallquist, "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, May, 1991.

- [3] B. M. Maker, R. M. Ferencz, and J. O. Hallquist, "NIKE3D A Nonlinear, Implicit, Three-Dimensional Finite Element Code for Solid and Structural Mechanics - User's Manual," University of California, Lawrence Livermore National Laboratory, UCRL-MA-105268, January, 1991.
- [4] T. Belytschko, S. E. Law, "An Assembled Normal Algorithm for Interior Node Removal in Three-Dimensional Finite Element Meshes," *Engineering with Computers*, Vol. 1, pp. 55-60, 1985.
- [5] J. M. Winget, "Advanced Graphics for Finite Element Results Display," *Advanced Topics in Finite Element Analysis*, ASME PVP Vol. 1, pp. 53-57, 1988.
- [6] A. M. Tenenbaum, Y. Langsam, and M. J. Augenstein, *Data Structures Using C*, pp. 492-500, Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1990.
- [7] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes, The Art of Scientific Computing*, pp. 229-237, Cambridge University Press, New York, New York, 1987.
- [8] D. Winkler, "Video Technology for Computer Graphics," in *Siggraph '91 C17 Course Notes*, 1991.
- [9] National Center for Supercomputing Applications, *NCSA HDF Call Interfaces and Utilities*, 1989.

LIST OF FIGURES

Figure 1. Canonical Local Node Numbering in Computational Space for Brick, Shell and Beam Elements.

Figure 2. Internal Paired Surfaces Between Hexahedral Elements

Figure 3. Elimination of a Solid Element and the Resulting Exposure of New Surfaces for Rendering.

Figure 4. High Speed Particle Impact on Idealized Turbine Stator Blades (Row #1 - Geometry Only, Row #2 - Effective Stress, Row #3 - Plastic Strain).

Figure 5. Turbine Blade Fragment Striking a Containment Shroud (Row #1 - Geometry Only, Row #2 - Effective Stress, Row #3 - Plastic Strain).

Volume Element

Shell Element

Beam Element

**DATE
FILMED**

8 / 18 / 93

END