Octree Optimization

Al Globus
Computer Sciences Corporation
NASA Ames Research Center
3 January 1991

Abstract

A number of algorithms search large 3D arrays (computation space) for features of interest. Marching cubes isosurface generation described by Lorenson and Cline[1] is an example. The speed of these algorithms is dependent on the time necessary to find the features of interest in the data and to compute their graphic representation. Efficiently searching for these features is the topic of this paper.

I describe an optimizing search using octrees to divide computation space. When the tree is walked, information stored in the branch nodes is used to prune portions of computation space thus avoiding unnecessary memory references and tests for features of interest. This technique was implemented for marching cubes isosurface generation on computational fluid dynamics data. The code was then adapted to continuing particle traces in multiply zoned data sets when a trace leaves one zone and enters another. For multiple isosurfaces, numerical experiments indicate a factor of 3.8 - 9.0 overall performance increase, measured by stopwatch; and a factor of 3.9 - 9.9 speedup in calculation times as measured by the UNIX times()[2] utility. The overhead is a one time cost of 0.2 - 2.8 times the time to compute an average isosurface and O(n) space with a constant factor less than one. 'n' is the number of grid points.

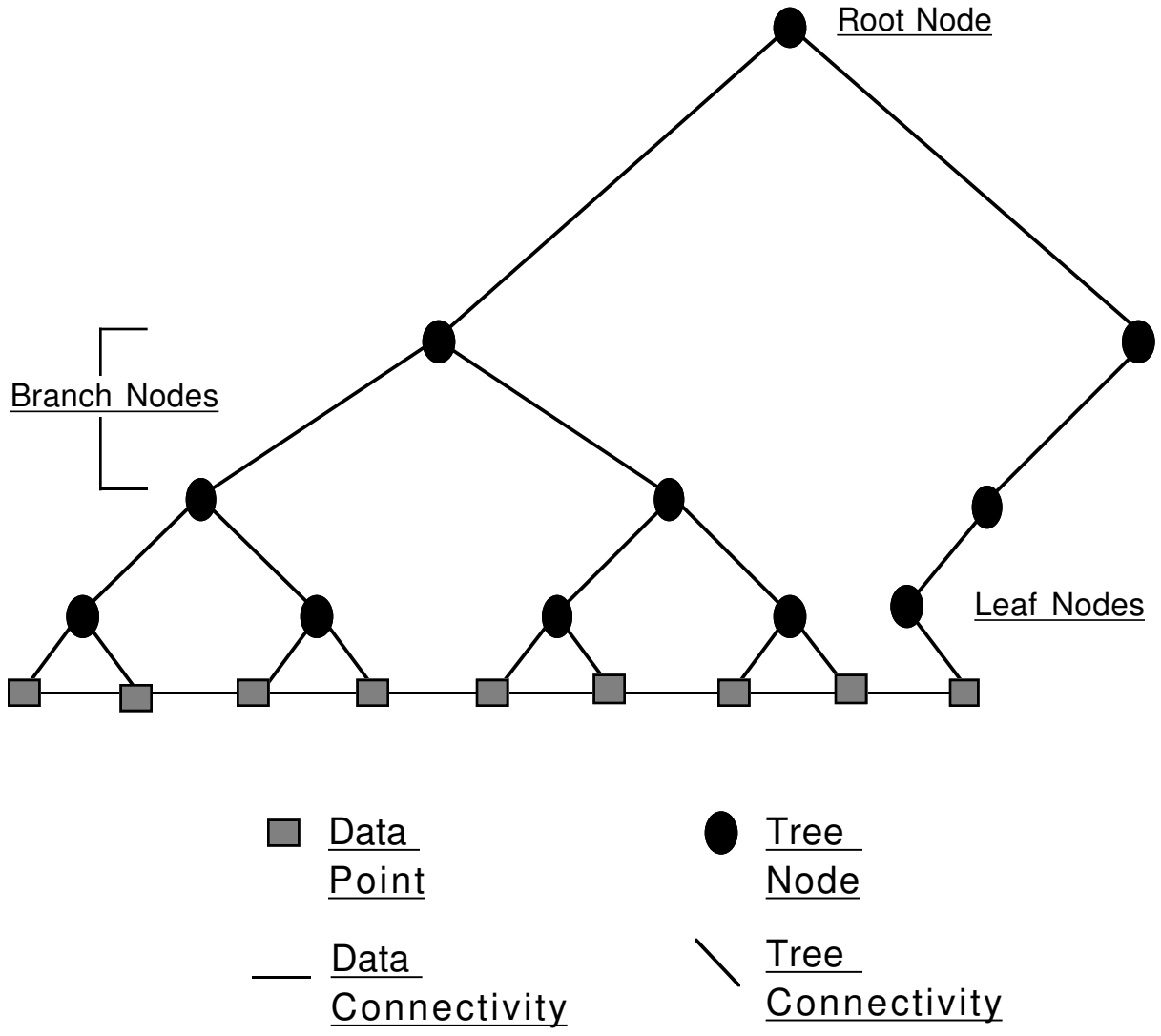
Introduction

Octree optimization can significantly speed up algorithms that repeatedly search a large computational space, e.g., a large 3D array of real numbers representing a scalar field. Octree optimization uses divide and conquer to prune the search space. The octree technique, originally developed for solid modeling, has been described by Jackins and Tanimoto[3] and Meagher[4]. In octree optimization, the root node represents all of computation space. It is recursively subdivided into eight children each representing approximately one-eighth of the space. Recursion ceases when each node represents less than a certain number of data points. Criteria for dividing computation space can be manipulated to produce trees with various properties, e.g., well balanced, desired shapes in computation space, etc. A small amount of information characterizing the data represented is stored in each node. The tree is then walked repeatedly with different parameters. The parameter is tested against the characterizing information stored in each node. If the test fails, the rest of the branch need not be examined, if it succeeds the tree walk continues. If a leaf node test succeeds, then the data represented is examined for features of interest and the appropriate computations performed.

For octree optimization to be effective there must be:

- Some means for the regular subdivision of computational space
- A fast, space efficient test to prune the search tree
- Features of interest that are somewhat sparse and clumpy. They must be sparse so that much of the space can be pruned away. They should be clumpy so that each branch taken yields a large number of feature 'hits'.
- Poor current performance.
- The need to apply the algorithm repeatedly over the same data with different parameters. Otherwise, the overhead of tree building is not worthwhile.

As an illustration, a binary optimization tree for one dimensional data might look something like this:



Previous work includes a paper by Woodwark[5] who used a similar technique in a different context. The kD tree search techniques developed by Bently[6] in the data base world are closely related. Wilhelm at the University of California at Santa Cruz has also implemented an octree based computation space optimization technique for marching cube isosurface generation [personal communication]. Glassner[7] uses

octrees in physical space to reduce the number of objects to test for ray trace intersections. Kerlick[8] describes a competitive technique using sorting to limit searches for isosurfaces generated using marching cubes. There is an analytic comparison of sorting versus octree optimization below.

Description of Implementation

CFD data are frequently visualized as scalar fields defined over generalized curvilinear grids[9]. Grids are represented as a three dimensional array of real numbers representing physical x, y, and z locations[9]. The integer space of indexes into the array is known as computation space, whereas the real space of the x, y and z locations is known as physical space. It is much faster to build optimization trees that divide computational space as opposed to physical space. The number of grid points in current studies is quite large, typically several hundred thousand to a million or more.

One interesting visualization is the display of isosurfaces of scalar fields. A second is function mapped cutting planes of arbitrary orientation. These can be generated by a program called ISOLEV[8] that uses the marching cubes algorithm. ISOLEV implements a second, faster surface algorithm called edge crossings. Here, a dot (single pixel) is drawn wherever the surface passes between two data points. The author has applied octree computation space optimization to ISOLEV. ISOLEV is a good candidate since:

- Computation space can be easily divided by cutting each of the three dimensions in half.
- The presence or absence of the surface can be readily detected by comparing the isovalue to the min and max of the portion of the scalar field represented by a node.
- Isosurfaces are frequently sparse and clumpy, and cutting planes are always sparse and somewhat clumpy. Isosurfaces can appear anywhere in the grid.
- ISOLEV's interactive performance was very poor.
- ISOLEV allows interactive modification of the isovalue or cutting plane location, as well as animated sweeps where the isovalue steps through all possible values. This causes the algorithm to be repeatedly executed over the same data.

Three operations are required for octree optimization implementation:

- Build the tree, necessary when the size of the computational space changes, i.e., new data is read in. Each node represents a space as 'cubic' as possible, i.e., the dimensions are as nearly equal to each other as possible.
- Set the min/max values in each node, necessary when the scalar field of interest changes or a new direction for a cutting plane is chosen. The min/max is used to test isovalues to see if the surface passes through the data represented by a node.
- Walk the tree, necessary when the value of the isosurface or location of the cutting plane changes. When a leaf node is reached and the surface is present, the marching cubes algorithm is executed on the sub-space represented.

The tree is built top down. The root node represents all of computation space. This is divided into eight (approximately) equal sub-spaces - usually by dividing each dimension by two - to produce the root's children. Oddly shaped subspaces are handled by an algorithm described by Globus[13]. This procedure is applied recursively to generate the whole tree. Recursion stops when a node represents less than 32 grid points.

To set the min/max values, the tree is walked and the min/maxes passed up from child to parent.

To walk the tree, a function is called with the following parameters:

- Tree node to walk.
- Isovalue.
- Function to execute once a leaf node is reached.
- Argument for the function containing a pointer to some necessary information.

A more general implementation would pass a boolean function and parameter instead of the isovalue. In ISOLEV's case, each node tests to see if the isovalue is between the min/max inclusive. If not, it returns. If so, branch nodes continue the walk and leaf nodes call the appropriate function.

Experiments

A set of computational experiments were run to determine if octree optimization of ISOLEV actually improves performance substantially. Data sets were taken from work performed at the Numerical Aerodynamic Simulation Division at NASA Ames Research Center.

Time is the dependent variable, the presence or absence of octree optimization the independent variable. Measures were taken by sweeping cutting planes and isosurface thresholds through each data set. Each sweep contained 20-21 steps. Stopwatch time was collected since it is the most important to the user. For stopwatch timed runs, each condition was repeated three times and the median or mean taken since uncontrolled system events can theoretically cause variations in the results. Actual results were very closely clustered.

In a separate experimental run, the UNIX times() utility was used to generate the:

- Total time to compute the surfaces.
- Minimum time to compute one of the surfaces.
- Maximum time to compute one of the surfaces.
- Time to build the tree, including setting the mins and maxes.
- Time to set the mins and maxes if the tree is already built.

Only user, not system, time is included in the comparison; although both were gathered. System time was generally very small.

Four kinds of surfaces were tested.

- Marching cubes isosurfaces, refered to as isosurface/solid.
- Dots drawn at edge crossings, refered to as isosurface/dots.
- Marching cubes cutting planes, refered to as x cut/solid.
- Dots drawn at cutting plane edge crossings, refered to as x cut/dots.

ISOLEV was implemented in C on a Silicon Graphics, Inc. IRIS 4D. All experiments were performed on an IRIS 4D 120 GTX. ISOLEV is part of the FAST (Flow Analysis Software Toolkit) CFD/visualization environment[10].

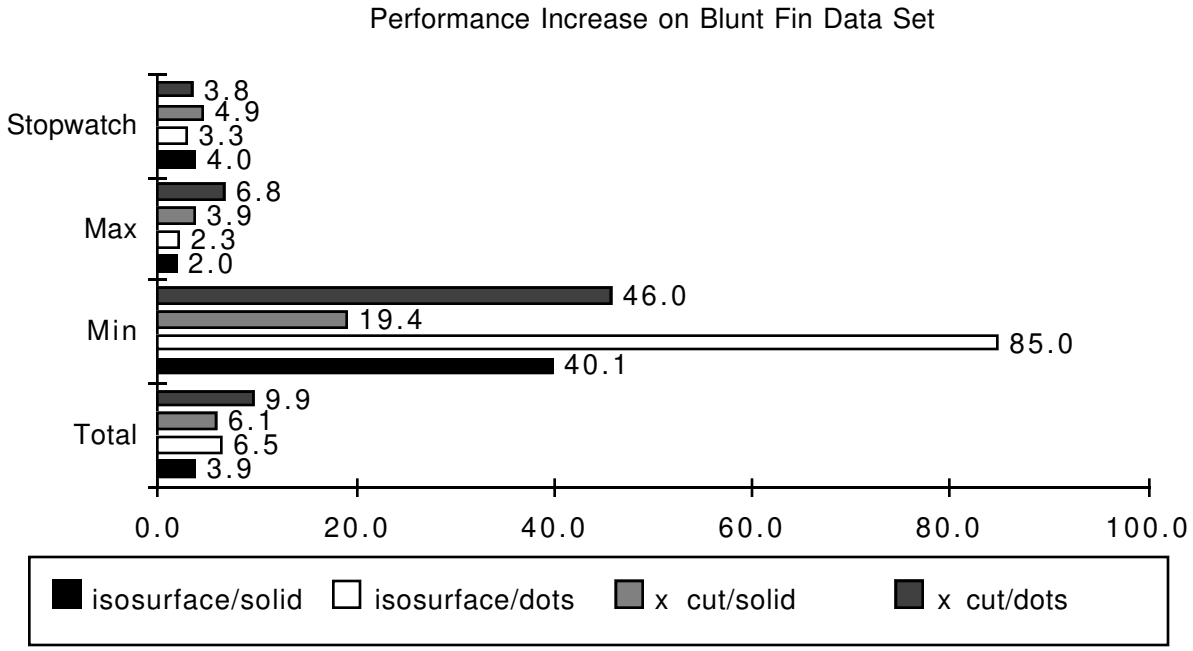
The IRIS hardware was:

- 2 16 MHZ IP5 Processors
- FPU: MIPS R2010A/R3010 VLSI Floating Point Chip Revision: 1.5
- CPU: MIPS R2000A/R3000 Processor Chip Revision: 1.6
- Data cache size: 64 Kbytes
- Instruction cache size: 64 Kbytes

- Main memory size: 16 Mbytes
- Interphase 3201 2-drive ESDI disk controller 0
- ESDI Disk drive: unit 0 on Interphase controller 0
- GT Graphics option installed
- Integral Ethernet controller
- Integral SCSI controller WD33C93
- Tape drive: unit 7 on SCSI controller 0: QIC 24

Results

The first chart shows the improvement factors gained by using octree optimization on the Blunt Fin[11] data set. The pressure field was used. This data set is 40x32x32 for a total of 40960 data points. The time ratio unoptimized/optimized is graphed. E.g., stopwatch time for the solid isosurface was four times faster with optimization.

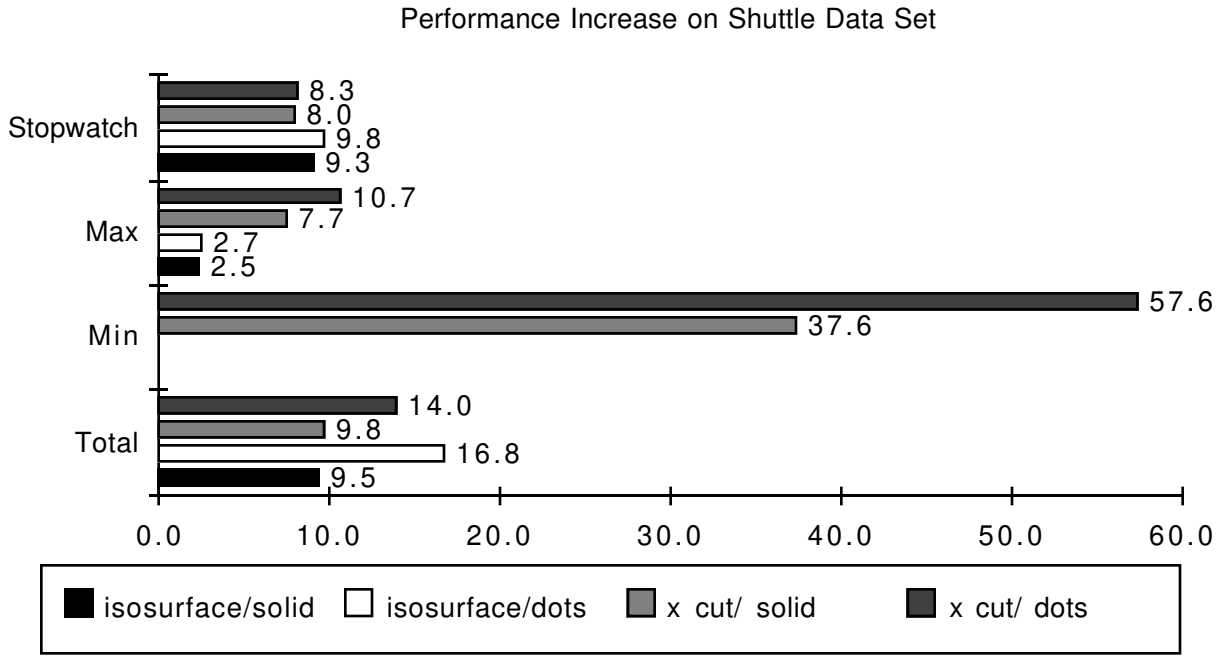


Note that improvements are very large for minimum times because most of the search space can be pruned away by the optimization. Note that maximum times are more nearly equal.

Stopwatch timed improvements are not as good since they include time to render the surfaces and the overhead introduced by FAST.

The next chart uses the same format to describe results using the shuttle data set[12]. Again the pressure

field was used. This data set is 80x63x45, for a total of 226,800 data points.

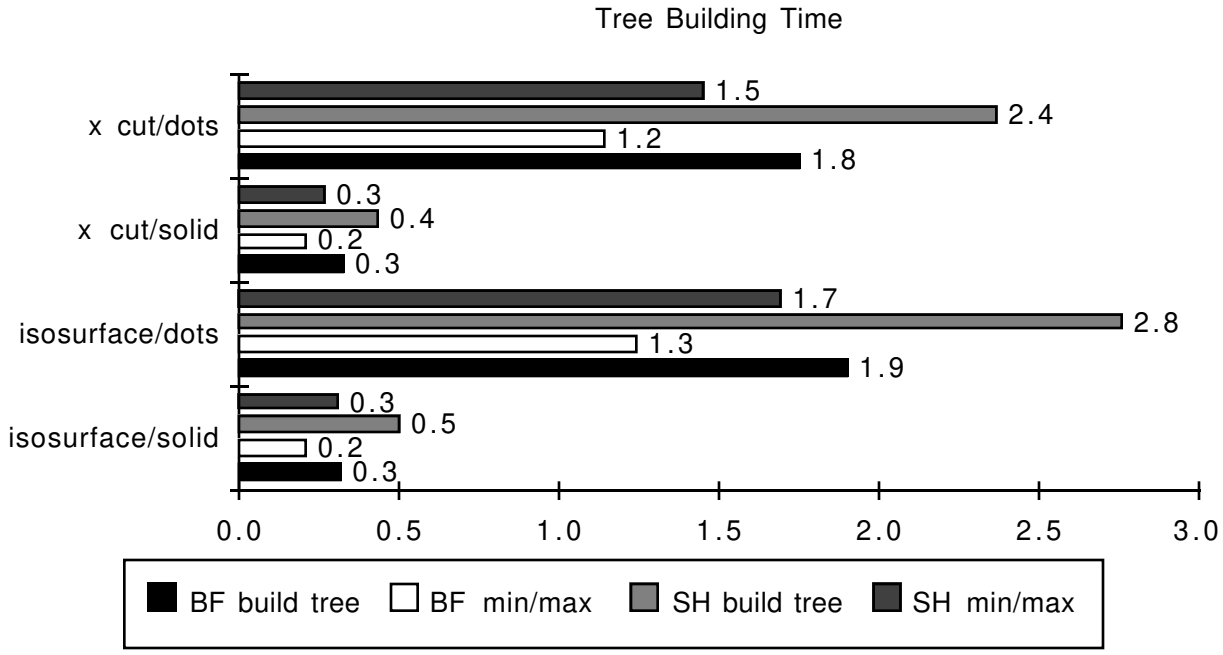


Two bars were left out. These are the minimum times for isosurfaces. The improvement factor was 588 for isosurface/solid and infinite for dots since the minimum dots time was so short it did not register on the UNIX time utility.

Improvements are greater since the data set size is much larger.

This next chart expresses tree building time as a multiple of the average time to generate one unoptimized isosurface (or cutting plane). E.g., building the tree (including generating min/maxes) for the

Blunt Fin took 0.3 times as long as unoptimized generation of an average isosurface.



BF means Blunt Fin. SH means shuttle.

Note that the time penalty for building optimization octrees is quite small. It is paid back within a few isosurface generation.

The raw data are published by Globus[13].

Discussion

Octree optimization should be useful if the interesting regions of a computational space are somewhat sparse and clumpy. Isosurfaces and function mapped cutting planes fit this description. There exist pathological isosurfaces that fill nearly the entire space where performance will actually degrade with optimization. Cutting planes, on the other hand, almost always pass through a smallish subset of computation space.

The octree code was modified to trace particles through multi-zoned grids where a trace can exit one computational space and enter another. Thus, to continue the trace one must find the cell where the trace entered the second grid. An octree optimization search using bounding box tests was used with excellent results. This will be described in more detail in a future paper.

Kerlick's sorting technique referenced above is a direct competitor to octree optimization, although application is limited to marching cubes isosurface generation. In Kerlick's technique, the computation cells are sorted by min/max and, as an isovalue sweep progresses, a list of current cells is maintained. This costs O(n) space, O(n log n) time for the sort plus time for list manipulations. By comparison, octree optimization is also O(n) in space but the constant factor is much smaller; and O(n/8 + n/64 ...) in

time for the build. Sorting for isosurface generation must only be done once, but cutting planes generate new scalar fields each time the orientation of the cutting plane is changed - which requires a new sort. Octree computation space optimization only requires that new min/maxes be regenerated when the scalar field changes, a much cheaper operation - O(n + n/8 + n/64 ...); and the constant is very small.

Design, implementation and testing of octree optimization for ISOLEV took about one person/week spread over about two weeks calendar time. To modify the code for particle tracing multi-zoned grids took a couple of hours.

Conclusion

Octree computation space optimization can significantly improve performance of algorithms that repeatedly search large computational spaces particularly if the interesting regions are sparse and clumpy. Furthermore, implementation is quick and easy.

Acknowledgements

I'd like to thank Dr. Val Watson and Dr. Tom Lasinski at NASA Ames Research Center for supporting this work. This paper was written under NASA contract NAS 2-12961.

References

1. W. E. Lorenson and H.E. Cline, "Marching Cubes: a High Resolution 3D Surface Construction Algorithm," *Computer Graphics*, Vol 21, No 4, July 1987, pp 163-169.

2. UNIX Programmer' Reference Manual, Volume 1 Section 2

3. C. L. Jackins and S. L. Tanimoto, "Octtrees and Their Use in Representing Three-Dimensional Objects," *Computer Graphics and Image Processing*, Vol. 14, No. 3, p 249-270

4. D. Meagher, "Geometric Modelling Using Octree Encoding, ", *Computer Graphics and Image Processing*, Vol. 19, No. 2, 1982, pp. 129-147.

5. J. R. Woodwark, "Techniques of spatial segmentation in solid modelling," *Spacial Data Processing Using Tesseral Methods*, Collected Papers from Tesseral Workshops 1 and 2, pp. 325-30. Sept. 1986 Swindown and Reading, England. Publ: Nat. Environ. Res. Council, Swindon, England

6. J. L. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching," *Association of Computing Machines,* Vol. 18, pp 509-517.

7. A. S. Glassner, "Space Subdivision for Fast Ray Tracing", *IEEE Computer Graphics and Applications*, Vol. 4, No. 10, pp. 15-22.

8. G. D. Kerlick, "ISOLEV: A Level Surface Cutting Plane Program for CFD Data," Report RNR-89-006, NAS Applied Research Office, MS T045-1, NASA Ames Research Center, Moffett Field, CA. 94035. This work was also published and presented at SPIE 1989.

9. P.P. Walatka, P.G. Buning, "PLOT3D User's Manual," NASA Technical Memorandum 101067,

NASA Ames Research Center.

10. G.Bancroft, F. Merritt, T. Plessel, P. Kelaita, R. McCabe, Al Globus, "FAST: A Multi-Processing Environment for Visualization of CFD", Visualization '90, IEEE Computer Society and ACM SIGGRAPH, October 23-26th, 1990 San Francisco, CA.

11. C. H. Hung, P.G. Buning, "Simulation of Blunt-Fin-Induced Shock-Wave and Turbulent Boundary-Layer Interaction", *J. Fluid Mech.,* 1985, Vol. 154, pp. 163-185.

12. Y. M. Rizk, S. Ben-Shmuel, "Computation of the Viscous Flow Around the Shuttle Orbiter at Low Supersonic Speeds", *AIAA 23rd Aerospace Sciences Meeting,* Jan. 14-17, 1985/Reno, Nevada, AIAA-85-0168.

13. A. Globus, "Octree Optimization", technical report *RNR-90-011,* NASA contract NAS 2-12961, Summer 1990, NASA Ames Research Center.