tecplot
Master the View

Improved Performance of Large Data Visualization using Sub-Zone Load-On-Demand

White Paper

Published: January 2013

www.tecplot.com

Tecplot, Inc.
P.O. Box 52708
Bellevue, WA 98015

3535 Factoria Blvd. SE,
Ste 550
Bellevue, WA 98006

425.653.1200 *direct*
800.676.7568 *toll free*
425.653.9200 *fax*

info@tecplot.com

Contents

Improved Performance of Large Data Visualization using Sub-Zone Load-On-Demand

Scott T. Imlay[1] and Craig Mackey[2]
Tecplot Inc., Bellevue, WA, 98006

The size and number of datasets analyzed by post-processing and visualization tools is growing with Moore's law. Conversely, the disk-read data transfer rate is only doubling every 36 months and is destined to be the bottleneck for traditional post-processing architectures. To eliminate this bottleneck, a sub-zone load-on-demand visualization architecture has been developed which only loads the data needed to create the desired plot. The original dataset is subdivided into sub-zones of <256 cells or nodes and these subzones are indexed on the disk using interval trees for each variable. Loading the required data starts with an $O(log(n))$ query of the interval tree to determine which sub-zones should be loaded. The resulting visualization tool is faster and uses far less memory than the standard visualization package.

Nomenclature

α	=	angle of attack
β	=	yaw angle
a	=	cylinder diameter
C_p	=	pressure coefficient
M	=	Mach number
n	=	number of points in the full grid
Re	=	Reynolds number
t	=	time
τ	=	pseudo time
$\vec{v}$	=	velocity at a point in space
v_i	=	isosurface value of a variable
v_d	=	discriminant value of an interval tree root node or branch node
v_{min}^s	=	minimum value of variable in subzone s
v_{max}^s	=	maximum value of variable in subzone s
x, y, z	=	x-, y-, and z-coordinates
$\vec{x}$	=	(x, y, z) position in space

I. Introduction

THE application of computational fluid dynamics (CFD) in the aerospace design process has increased dramatically over the last decade. This is due, in large part, to the relentless and continuing growth of computer performance. In some cases the enhanced computer power is used to perform high-resolution CFD calculations to analyze the details of complicated unsteady flow fields around complex configurations. In other cases it is used to create a virtual wind-tunnel where hundreds or thousands of

[1] Chief Technology Officer, P.O. Box 52708, Bellevue, WA, Senior Member AIAA.
[2] Senior Research Engineer, Research, P.O. Box 52708, Bellevue, WA.

lower resolution CFD computations are performed to estimate the aerodynamic properties of a prospective configuration throughout its operating envelope. In either case, the total amount of data read during post processing is doubling every 18 months – in sync with Moore's law.

The data is generally stored on arrays of hard disk drives. Over the last two decades, the storage capacity of hard disks has grown in accordance with Kryder's law - doubled every 12 months. This is more than sufficient to keep up with the growth in dataset size. Unfortunately, the sustained rate at which data can be read from the hard disk is growing much slower – doubling every 36 months[1]. This is because the sustained data transfer rate grows with the lineal density of the magnetic dots on the hard disk while storage capacity grows with the areal density (roughly the square of the lineal density). While hard disk capacity is keeping up with dataset size, the speed at which we can read the data is not.

In the past, the primary bottleneck in visualization software performance was network speed. Over the last decade, the speed of Local Area Networks (LANs) has doubled every 2 years on average. It doesn't change that often, but upgrades tend to yield an order-of-magnitude increase in bandwidth (100Mb/s to 1Gb/s, for example). Likewise, Wide Area Network (WAN) performance is also doubling every 2 years, although it lags substantially behind LAN performance. The bandwidth for both LANs and WANs is growing more slowly than dataset size, but much faster than sustained disk-read data transfer rates.

Given these trends, a simple analysis of visualization system performance can be performed. Assume initial values of 100M cells in 2005, 100MB/s (1Gb/s) LAN in 2006, and 75 MB/s sustained read bandwidth for the hard-disk in 2006. The trends in time to load a large dataset is given in Figure 1. Note that the load-time ultimately becomes dominated by the hard-disk sustained read data transfer rate, with the cross-over date a function of the network type (bandwidth).

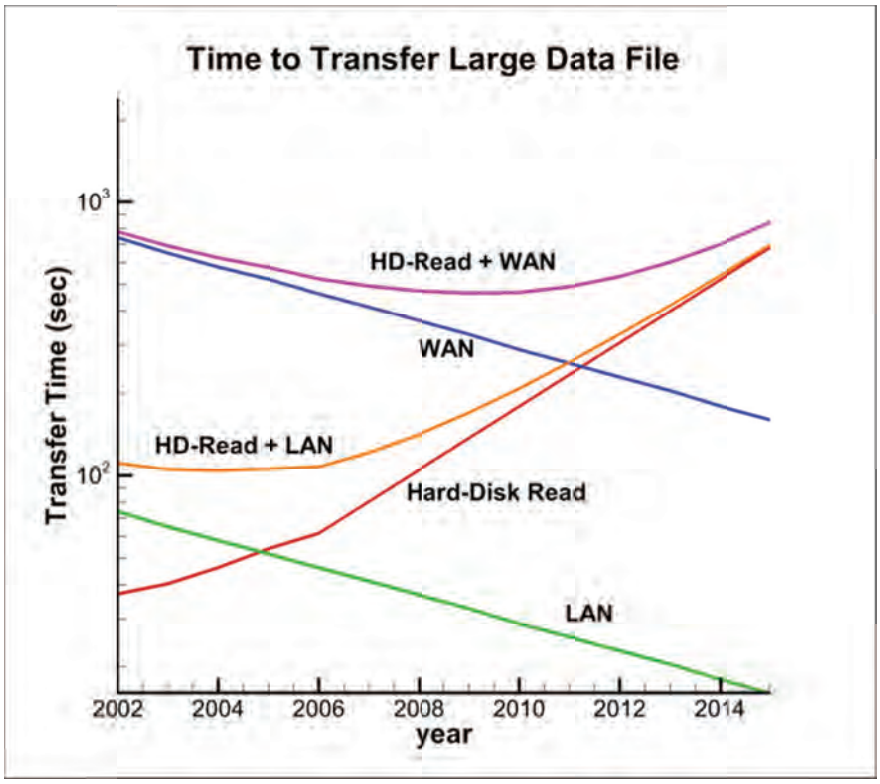

Figure 1. Time to transfer and load a large CFD dataset

These trends have a significant impact on the optimal visualization architecture. Traditional client-server architectures were designed to overcome network bandwidth constraints. In these architectures, the data is loaded on a remote computer with a high-bandwidth access to the data, important data

abstractions are extracted, and the geometry and data on these abstractions is transferred across the slow network to a local client. In this context, "abstractions" may include model geometry, slices, isosurfaces, streamlines, vortex cores, and any other one- or two-dimensional data extraction the user may desire. A modification of the client-server architecture is to render the plot remotely and transfer the image at video-like frame-rates. These client-server architectures do nothing to overcome the primary bottleneck, sustained disk-read data transfer rates (SDRDTR).

The current hardware-based solution to this problem is to increase the number of spindles (hard-disks) used in the parallel file system. If the number of spindles in the file system doubles every 3 years or so, the time to read a data file will remain constant. However, increasing the number of disks in the parallel file system is counter-intuitive, as the hard disk capacity will match the file size increases without adding disks. As such, that solution will likely meet with some resistance. Longer-term hardware-based solutions, such as solid-state disks (SSDs) are not yet economically viable for collections of large CFD datasets.

The software-based solution is to read less data. Generally, only a small percentage of the total dataset is needed to create the abstractions the user wishes to view, so this solution seems viable. To be sustainable, the percentage of the dataset loaded must decrease over time (halved every three to four years). This solution also has other benefits, like reduced memory requirements and reduced network bandwidth requirements. This is one architectural approach taken by Tecplot Inc. for large-data visualization.

In this paper, a new architecture will be described for visualizing large CFD datasets. It is based on loading subzones (spatially correlated sub-segments of the full dataset of less than 256 nodes or cells) on demand (only as needed). To support this algorithm, interval trees can be created to rapidly select the needed subzones. The architecture is sustainable: for slices and isosurfaces, the number of subzones loaded is approximately $O(n^{\frac{2}{3}})$ and for streamtraces it is approximately $O(n^{\frac{1}{3}})$. For finite-element data, the subzones are currently created using recursive orthogonal bisection. Speed and memory-usage measurements are included for 1 billion and 10 billion cell synthetic datasets.

II. Approach

A. Related Work

The current algorithms are largely based on the out-of-core techniques developed in the 1990's to visualize large datasets on computers with insufficient memory[2,3,4,5]. They divide the data into small chunks (as small as a cell) which are indexed by I/O trees of various types.

Our goal is different - to minimize the amount of data read from the disk – but related. The primary difference is that out-of-core techniques will off-load (or deallocate) data once it is no longer in use, while our software keeps the data loaded as long as memory is available. This is an advantage for interactive data exploration, where the same subzone may be used multiple times.

B. Subzone Creation

The current algorithm subdivides the grid to create blocks, or subzones, of less than 256 cells (or nodes) each, and loads only those subzones needed to create the desired visualization. There are separate subzones for the cells (node-maps) and nodes, which aren't necessarily co-located in space. The extent of each subzone of each variables is encoded in an interval tree which is used during visual abstraction generation to rapidly and efficiently choose the subzones to load (see following section). For node subzones, the extent (min-max) of the variables for all cells containing the node is encoded,

so node subzones can be loaded independently of cell subzones and required nodes are always available.

The goal is to create subzones which are nearly cubic in space (i.e. that have minimal extent of X, Y, and Z variables). In the current implementation, this is done using orthogonal recursive bisection (ORB)[7,8]. In ORB, the list of cells in the grid is first bisected about the median value of X, then each subdomain is bisected about a median value of Y, then each of those is divided about a median value of Z, and the process repeats until all subdomains (subzones) contain less than 256 cells. For cells, the X, Y, and Z values are cell centroids. In our implementation, the process is repeated for nodes (using nodal X, Y, and Z values). An example 2D ORB is shown in Figure 2.

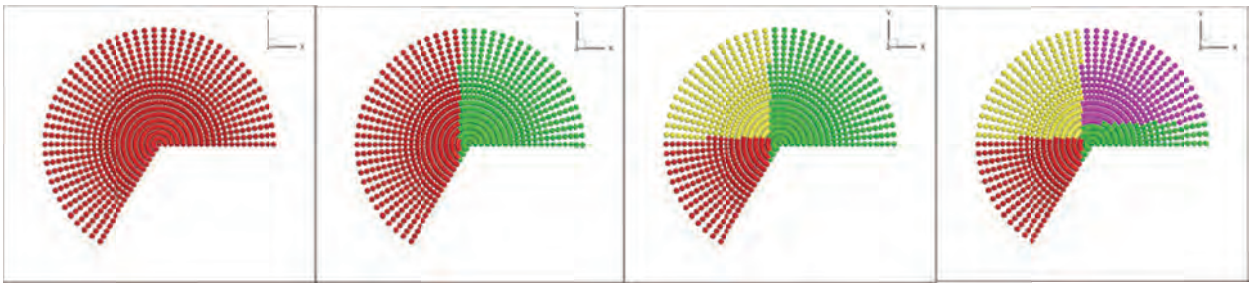

Figure 2. ORB for 2D set of nodes

The ORB is not the optimal algorithm for CFD domain decomposition, but it is more than adequate for our needs. CFD solutions require frequent communication between the processors that aren't required for most visualization algorithms. The ORB is simple, fast, and fairly easy to implement in out-of-core algorithms.

C. Slice and Isosurface Algorithm

For isosurfaces and slices (a subset of isosurfaces), the list of subzones required for the isosurface extraction is determined by querying two interval trees (one for the cells, the other for the nodes). An interval tree is a binary tree encoding the ranges of a variable for the subzones. A separate set of interval trees must be created for each of the grid coordinates (for slices) and each variable that might be used to create an isosurface.

Unfortunately, in tree terminology, the term "node" refers to component of the tree, a structure, that contains a value, a condition, or represents a separate data structure. Since this can be confused with grid nodes, the tree nodes will always be referred to as "root node", "branch node", or "leaf node".

A simple example of an interval tree is given in Figure 3. Each branch node of the interval tree contains: a discriminant value, a pointer to the branch (or leaf) nodes to the left, a pointer the branch (or leaf) nodes to the right, and two lists of subzones that contain the discriminant value in their range. The first list (AL in Figure 3) is sorted by minimum value of the subzone ranges. The second list (DH in Figure 3) is sorted by descending value of the maximum of the subzone ranges. The leaf nodes contain no sub-nodes.

Creating an isosurface requires loading all subzones that contain the isosurface value within their min/max range of the isosurface variable. This is done by querying the appropriate interval trees. The query algorithm is as follows:

1. Starting at the root node, compare the isosurface value v_i to the discriminant value, v_d
2. If $v_i \leq v_d$, search linearly through AL loading all subzones, s, with $v_i \geq v_{min}^s$, then repeat the process with the left node
3. If $v_i > v_d$, search linearly through DH loading all subzones, s, with $v_i \leq v_{min}^s$, then repeat the process with the right node

4. Repeat process until there are no more branch nodes to search

The computational complexity of the query is $O(\log(n))$, where n is the number of nodes, or cells, in the computational grid. Our subzone interval tree is similar, in concept, to the binary-blocked I/O interval tree of Chiang[3,4].

Once the necessary cell and node subzones are loaded, a standard isosurface extraction algorithm is used to extract the isosurface or slice. The current implementation uses a modified marching cubes for hexahedral elements, and marching tets for tetrahedral elements.

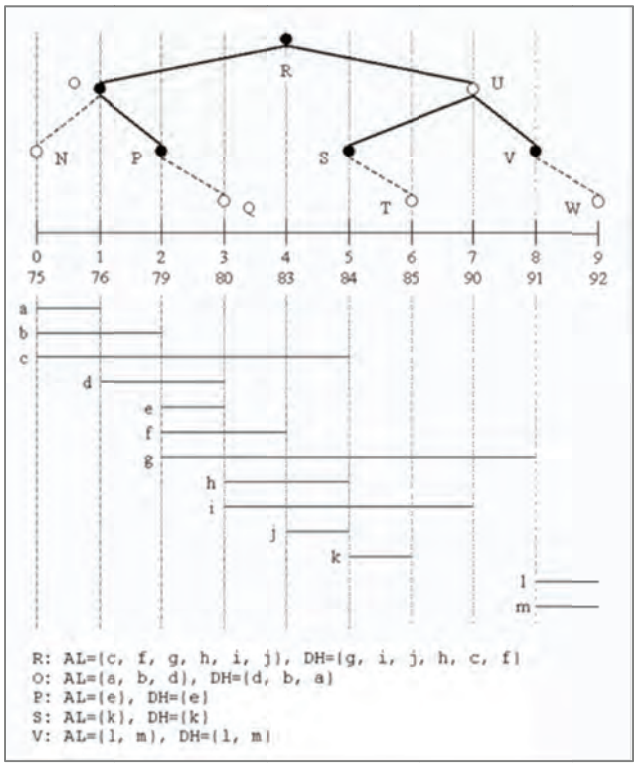

Figure 3. Interval tree

D. Streamlines, Particle Paths, and Streaklines

Streamlines, particle paths, and streamlines are all tangent curves to vector fields[9] that can be computed by integrating the following equation:

$$\frac{d}{d\tau}\,\vec{x}(\tau) = \vec{v}\big(\vec{x}(\tau)\big)$$

The subzone query and loading must occur sequentially as the streamline integration proceeds. Each query returns one or more subzones whose ranges contain the $\vec{x} = (x, y, z)$ position at which the velocity $\vec{v}$ must be evaluated. This is unlike the isosurfaces where all necessary subzones can be determined up front. More significantly, it requires a query on position, three scalar variables (x, y, z), instead of one scalar variable. The interval tree is designed for a query on one scalar variable.

There are alternative tree structures, such as k-d trees or oct-trees, that efficiently query on position. For the current work, we've chosen to use an interval tree on X and search linearly through the selected

zones to find those that satisfy the Y and Z criteria. This is not as efficient as using a k-d tree or oct-tree, but it doesn't require an additional tree structure to be stored on disk.

We solve the equation for the tangent field using a standard second-order two-step Runge Kutta method. Generally, it takes many steps of the Runge Kutta solver before the streamtrace exits a subzone and another subzone must be loaded.

III. Results

The performance and memory-usage of subzone load-on-demand was compared to the standard Tecplot 360 for slice, isosurface, and/or streamline generation in four test cases. The first case, in section A, is a fabricated dataset designed to measure the scaling of performance and memory usage as a function of dataset size, n. Dramatic improvements in performance were observed. The scaling of SZLoD performance and memory usage is $O(n^{\frac{2}{3}})$. The second case, in section B, is an animation of slices in a large, multi-block, structured-grid unsteady wind-turbine flow. The third case, in section C, is slice and streamline generation for a 187 million cell, unstructured-grid, generic transport configuration. The final case, in section D, is isosurface and slice generation for a 204 million cell unstructured-grid NASA Trapezoidal wing.

The first three cases were run on a Windows Vista 64-bit workstation having 32GB of memory and dual Intel Xeon E5404 4-core processors running at 2.00 GHz. The final case was run on a Windows 7 64-bit workstation having 48GB of memory and dual Intel Xeon E5520 4-core processors running at 2.26 GHz.

A. Synthetic Dataset

The first example is a dataset created for the purpose of testing. It is a cylindrical ordered-grid with a simple vortex velocity field and a scalar isosurface variable defined by the equation

$$p = x + x * y * z$$

Figure 4 shows the geometry, figure 5 shows the isosurface (at p = 0.4), and figure 6 shows the subzones used to render that isosurfaces in a 75 million node case.

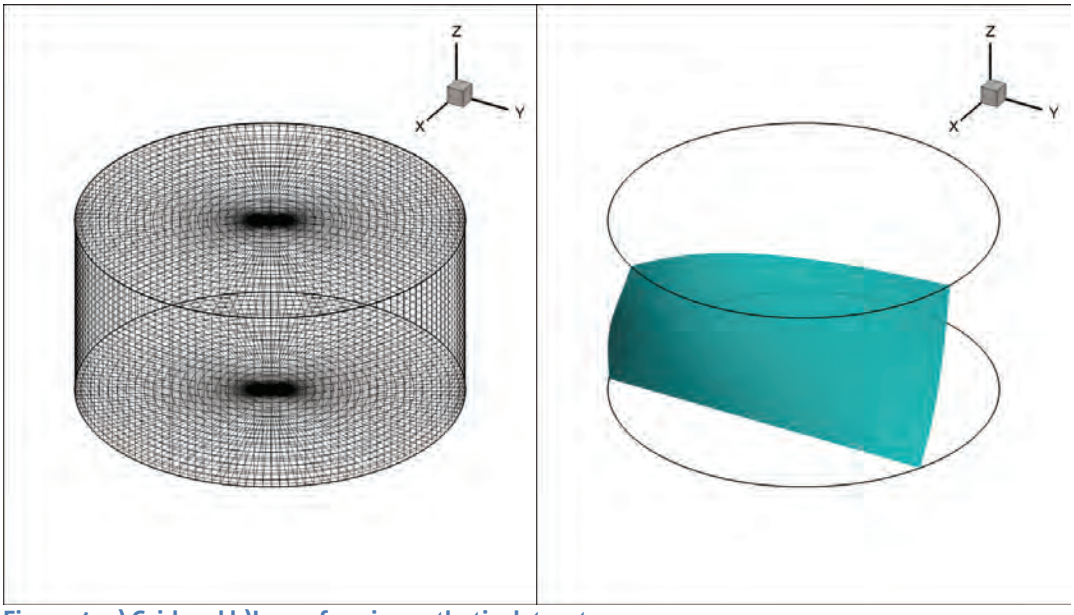

Figure 4. a) Grid and b)Isosurface in synthetic dataset

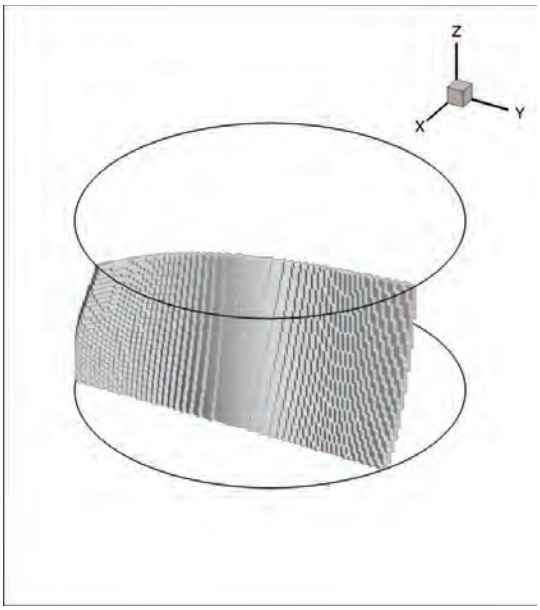

Figure 6. Sub-zones loaded to render isosurface

For the isosurfaces test, both Tecplot 360 and sub-zone load-on-demand were tested after a reboot for an uncached read test, and after several iterations of reading the same data for a cached read test. The timings are given in table 1 and compared in Figure 7.

Memory usage was monitored during the test, and the peak memory usage recorded as well as the memory usage upon task completion. Memory usages are given in table 2 and compared in Figure 8.

Note that the Tecplot 360 timings and memory usage scale with *n*, while the sub-zone load-on-demand times and memory usage are significantly lower and scale with $n^{2/3}$. Also of note is that sub-zone load-on demand completed the 1.2 billion case using a peak memory of 8.1GB while Tecplot 360 failed to complete the task, needing more than the available 32GB.

	Reading Uncached Data		Reading Cached Data	
Dataset Size	Tecplot 360	Sub-zone Load-on-Demand	Tecplot 360	Sub-zone Load-on-Demand
75M	35s	18s	8s	6s
150M	63s	30s	15s	9s
300M	126s	69s	29s	14s
600M	306s	138s	143	24s
1,200M	n/a	215s	n/a	41s

Table 1. Isosurface generation timing results

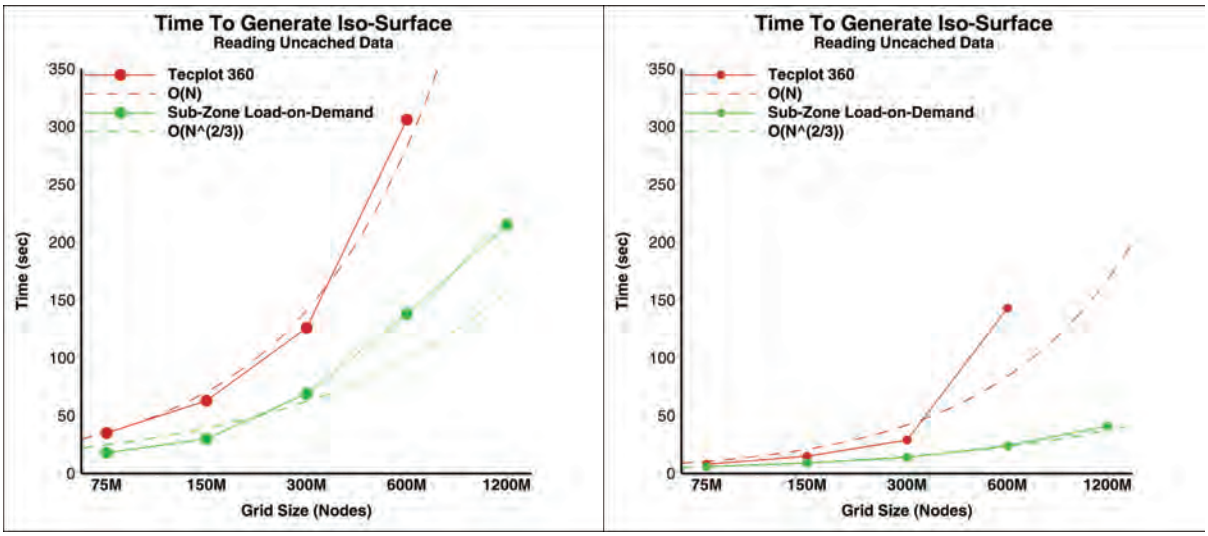

Figure 5. Isosurface timing results

	Peak Memory Used		Final State Memory Used	
Dataset Size	Tecplot 360	Sub-zone Load-on-Demand	Tecplot 360	Sub-zone Load-on-Demand
75M	2.7GB	0.92GB	2.2GB	0.46GB
150M	4.3GB	1.5GB	4.3GB	0.68GB
300M	11GB	2.3GB	8.4GB	1.0GB
600M	21GB	4.4GB	17GB	1.6GB
1,200M	n/a	8.1GB	n/a	2.5GB

Table 2. Isosurface generation memory-usage results

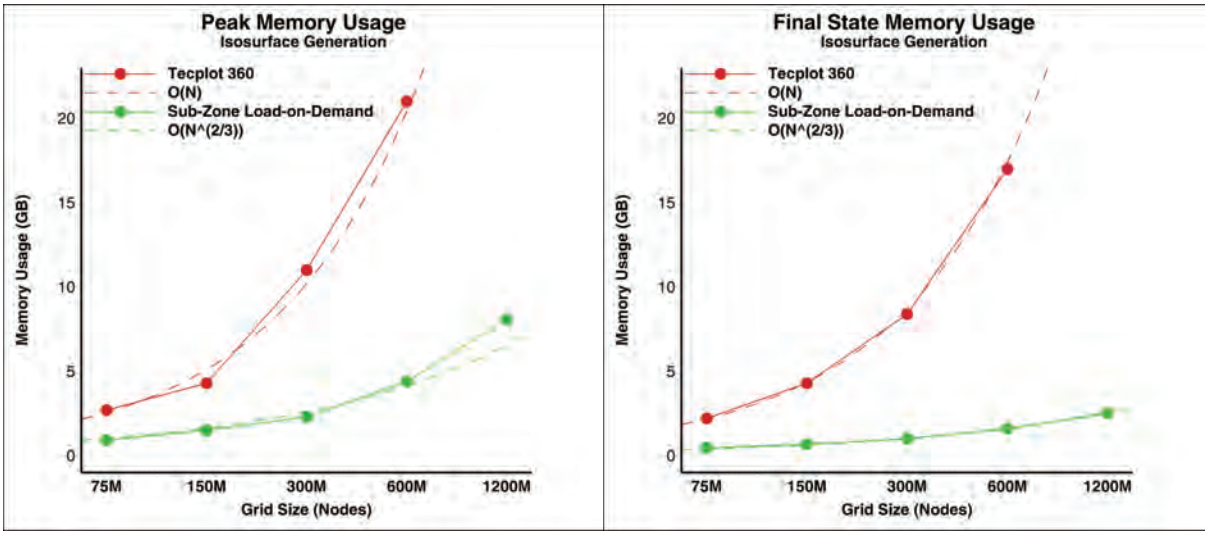

Figure 8. Isosurface memory-usage results

B. OVERFLOW Wind-Turbine Results

Modern industrial economies are heavily reliant on electricity and developing and maintaining a sufficient supply of affordable and reliable power generation capability is a priority of every nation. Fossil fuel based power generation capabilities have become less attractive for environmental reasons and the future expansion of nuclear power is uncertain following the recent Fukashima nuclear accident. For all of these reasons, "environmentally friendly" wind and solar power have become more attractive.

One of the largest issues with wind-turbines is noise. For this reason, Nelson, et. al., recently published a study that used a sequence of numerical methods to predict the noise pattern from wind-turbines[11]. The sequence starts with a solution of the Navier-Stokes equations for the unsteady flow-field around the NREL 10m diameter research wind turbine (figure 10). The equations were solved using the OVERFLOW 2.2 code with adaptive mesh refinement. The computational grid started with a 30.5 million point mesh (in 68 blocks) and over 7152 time steps grew to over 260 million points in more than 5600 blocks. The results were exported to plot files every 16 time steps for further analysis. In all, the plot files consumed roughly half a terabyte of disk space.

Figure 10. The NREL 10m Research Wind Turbine

The wind turbine results were animated with standard Tecplot 360 reading Plot3D files, with standard Tecplot 360 reading Tecplot plt files, and with subzone load-on-demand (SZLoD). The resulting times, per time step, to read, generate the slice, and render the image are showing in figure 11a. Near the beginning, when the grid is relatively small, SZLoD is only slightly faster than the standard Tecplot 360. However, as the number of grid points increases over time the advantage of SZLoD becomes apparent. At the end of the animation, SZLoD is more than a factor of 3 faster than the standard Tecplot 360.

The memory usage for the animation is shown in Figure 11b. SZLoD memory usage is nearly constant at just below 200MB. In comparison, the standard Tecplot memory usage grows proportionally with the number of grid points per time step. At the final time step, standard Tecplot 360 uses six times more memory than SZLoD.

The performance gains for the wind turbine animation are less dramatic than the performance gains for the other test cases. This is probably because the grid is composed of a large number of smaller zones. There are many operations that are linear in the number of zones, and the number of zones is growing with the grid size.

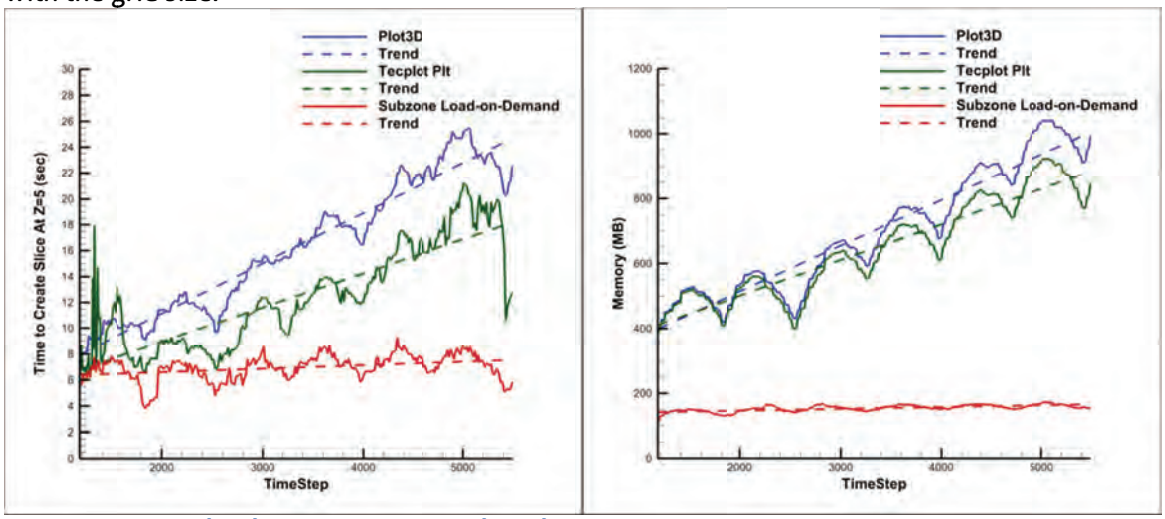

Figure 11. Timing (left) and memory usage (right) comparisons for the 10m NREL research wind turbine

C. Generic Transport Aircraft in Landing Configuration

The third example is subsonic flow past a generic transport aircraft in landing configuration, shown in Figure 12. The flaps, slats, and landing gear are down, and the aircraft is in a strong crosswind. The grid is composed of 187 million cells, with prisms near the wall and tetrahedrals away from the wall.

Subzone load-on-demand was tested by performing a sequence of slices along the x-axis from near the nose to near the tail. The timings include all operations from reading the data to rendering the image. The SZLoD timing for the slice as a function of position from nose to tail is show in Figure 13a. They vary from just over 1 second near the tail where there is a comparatively coarse grid, to just over 9 seconds at X=25, where the geometry is complex and there is a very fine grid. In comparison, the time to read the data, generate the slice, and render the images in standard Tecplot 360 was 139 seconds regardless of position. The ratio of standard Tecplot 360 time to SZLoD time is given in Figure 13b. SZLoD is from 20 to 120 times faster than standard Tecplot 360.

Peak memory usage by SZLoD, as a function of slice position, is shown in Figure 14a. It varies between 350MB and 750MB, depending on the grid refinement in the region being sliced. In contrast, the standard Tecplot 360 uses 16.2GB of memory. Figure 14b shows the ratio of memory requirements for standard Tecplot 360 and SZLoD. SZLoD uses 22 to 54 times less memory than the standard Tecplot 360.

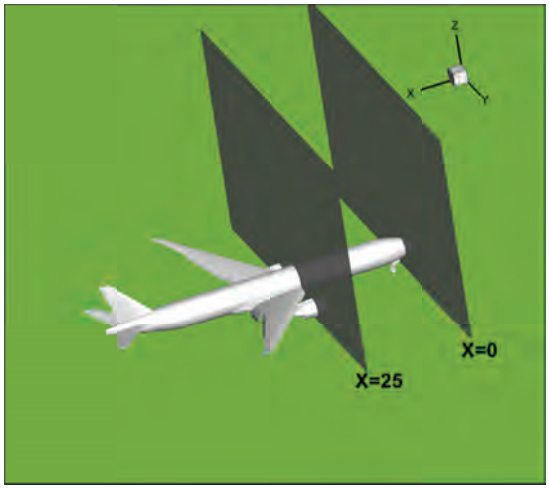

Figure 12. Generic transport aircraft with representative slices used for timing

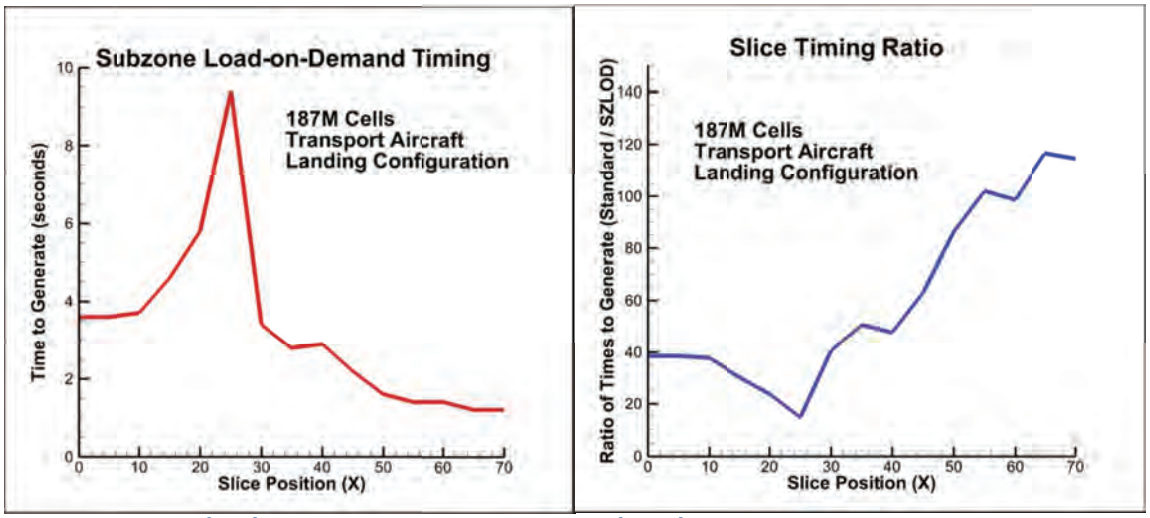

Figure 13. Timing (left) and performance improvement (right) for generic transport aircraft

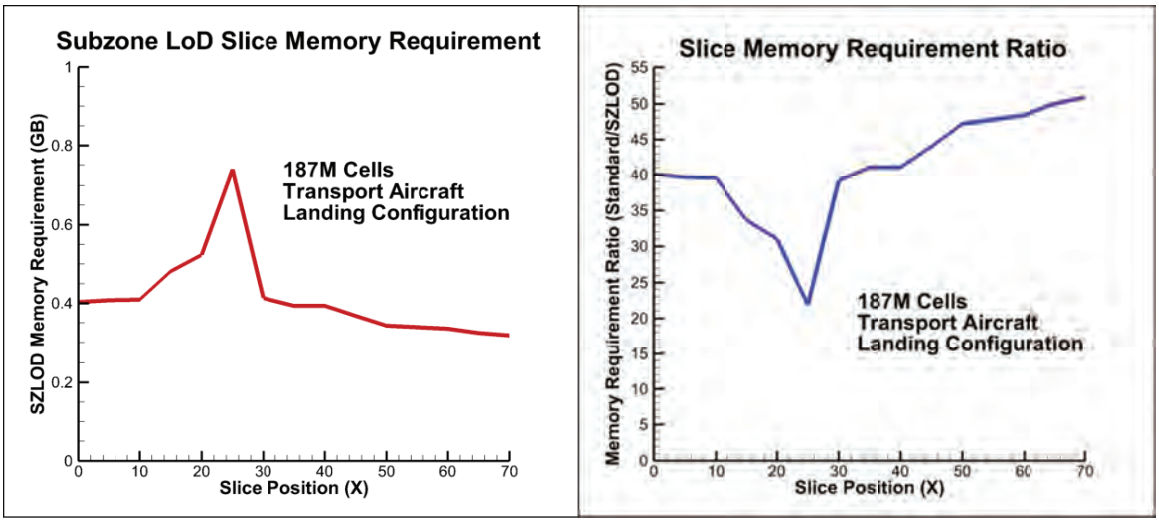

Figure 14. Slice memory requirements (left) and memory usage improvement (right) for generic transport aircraft

The use of subzone load-on-demand was also tested for the generation of a streamline in the generic transport dataset (Figure 15). In this case, the standard Tecplot 360 took 170 seconds to read the data, generate the streamline, and render the image. In the process, it utilized 16GB of memory. In contrast, SZLoD took 2.2 seconds and 1.3GB of memory to accomplish the same task.

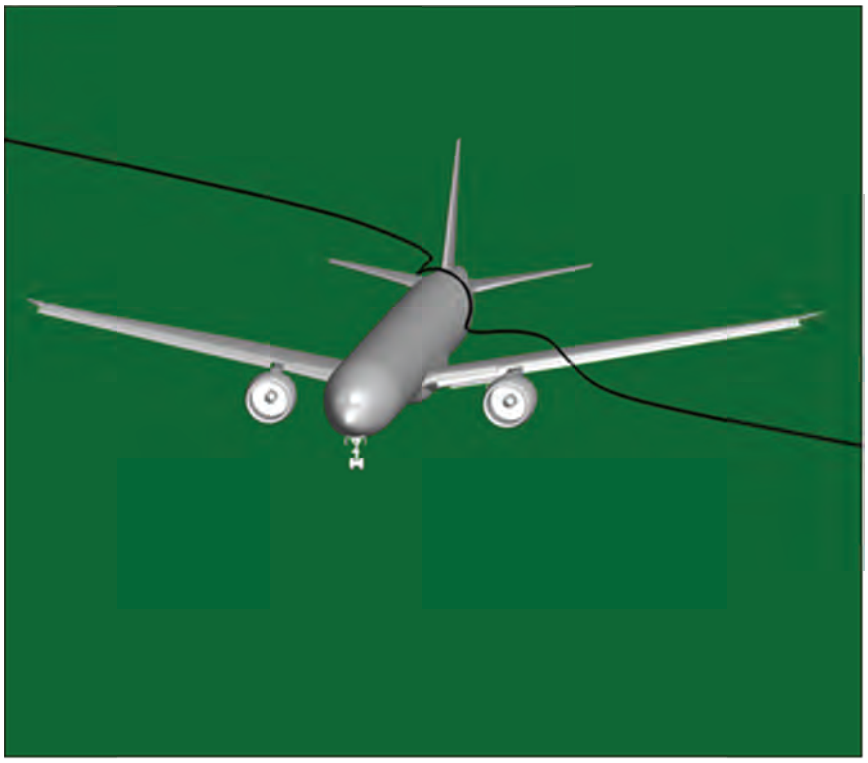

Figure 15. Streamline for generic transport aircraft

D. High Lift Prediction Workshop Unstructured Grid

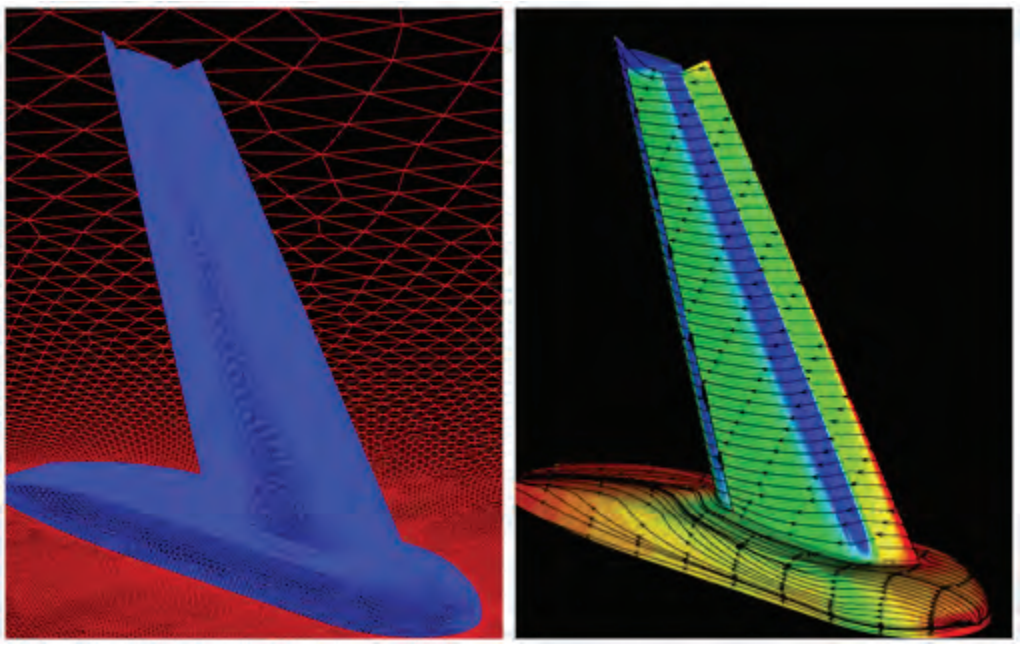

Figure 16. Grid and computed flow properties for the NASA Trapezoidal Wing configuration used in the High Lift Prediction Workshop (image from http://hiliftpw.larc.nasa.gov/Workshop1/geometries.html).

The fourth example is a finite-element grid calculation for the NASA Trapezoidal Wing configuration. This is the same configuration used at the 2010 High-Lift Prediction Workshop[13,14]. The grid consists of three Tecplot zones: containing, respectively, 122 million Brick elements in the viscous layer near the wall, 82 million Tetrahedral elements away from the wall, and 640 thousand Pyramid elements in the transition region between the first two zones. All together there are roughly 204 million cells and the zones share 76 million nodes where the field data is located.

This data was visualized using the prototype subzone load-on-demand add-on and the results were compared to an instrumented version of Tecplot. Timings and memory usage were compared for two cases, and isosurface of pressure coefficient at a value of -2.0, and a slice at Y = 100.

The isosurface of pressure coefficient is shown in Figure 17. It consists of 1.55 million surface elements near the leading edge, the slat, and the wing tip.

The timing measurements for this case are shown in the second column of Table 3. The standard Tecplot, with multi-threading, takes roughly 4 minutes to load the data, generate the isosurface, and render the image. The single-threaded Tecplot takes roughly 22 minutes – nearly six times as long as the multi-threaded Tecplot – to accomplish the same tasks. In contrast, subzone load-on-demand takes only 20.5 seconds to load the data, generate the isosurface, and render the image. This is 11.5 times faster than the standard Tecplot, and 65 times faster than the single-threaded Tecplot. Since the SZLoD prototype is single-threaded, we expect to gain substantial speed when we implement the multi-threaded version.

The memory-usage measurements for this isosurface are shown in the third column of Table 3. The standard and single-threaded Tecplot have a peak usage of 23GB of memory. In contrast, SZLoD uses 1.3GB of memory, a factor of 18 less than the standard Tecplot.

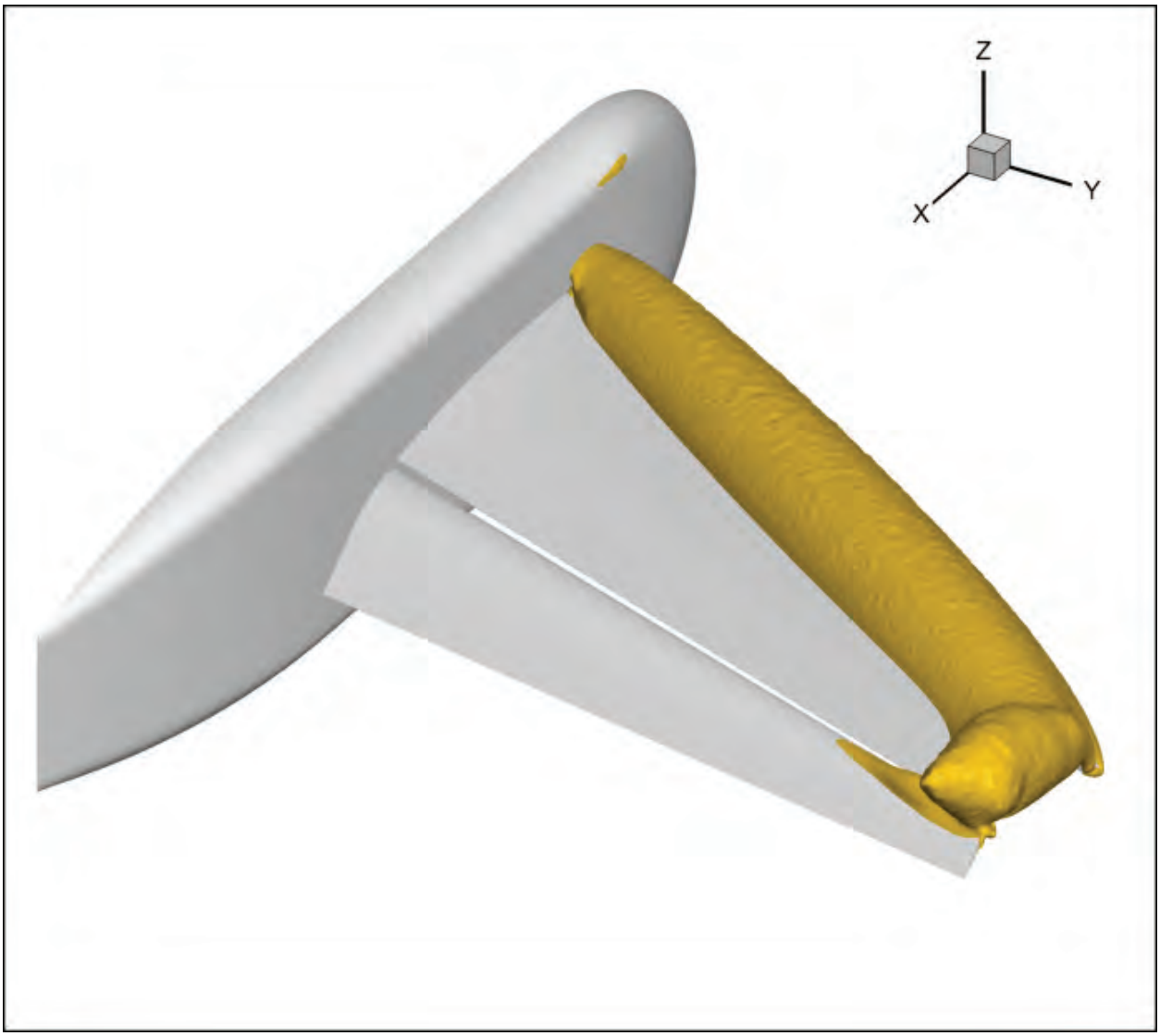

Figure 17. Isosurface of pressure coefficient at -2 for the NASA Trapezoidal Wing

Algorithm Used	Time (sec)	Peak Memory Usage (GB)
Standard Tecplot	234	23
Single-Threaded Tecplot	1330	22.8
Subzone Load-on-Demand	20.5	1.26

Table 3. Timing and peak memory usage for Cp = -2 isosurfaces of the NASA Trapezoidal Wing 205M cell dataset

The slice at Y=100 is shown in Figure 18. It located roughly 2/3 of the way from root to tip, and consists of 173 thousand surface elements.

The timing measurements for the slice are shown in the second column of Table 4. The standard Tecplot, with multi-threading, takes roughly 3.7 minutes to load the data, generate the slice, and render the image. The single-threaded Tecplot takes roughly 21 minutes – nearly six times as long as the multi-threaded Tecplot – to accomplish the same tasks. In contrast, subzone load-on-demand takes only 2.36 seconds to load the data, generate the isosurface, and render the image. This is 94 times faster

than the standard Tecplot, and 540 times faster than the single-threaded Tecplot. As with isosurface, we expect to gain substantial speed in SZLoD when we implement the multi-threaded version.

The memory-usage measurements for this slice are shown in the third column of Table 3. The standard and single-threaded Tecplot have a peak usage of 20.5GB of memory. In contrast, SZLoD uses 0.37GB of memory, a factor of 55 less than the standard Tecplot.

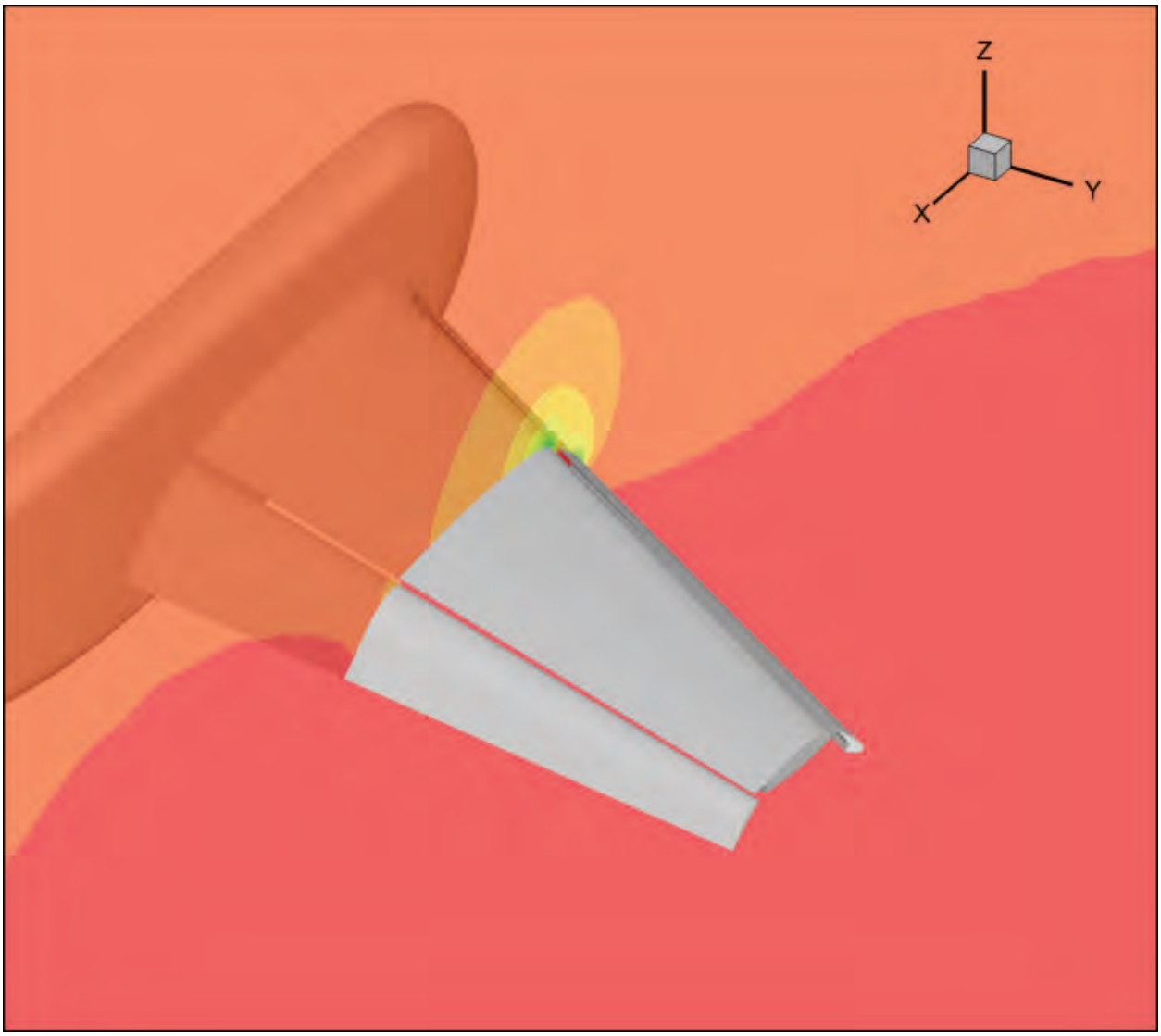

Figure 18. Slice at Y=100 for the NASA Trapezoidal Wing

Algorithm Used	Time (sec)	Peak Memory Usage (GB)
Standard Tecplot	222	20.5
Single-Threaded Tecplot	1279	20.4
Subzone Load-on-Demand	2.36	0.366

Table 4. Timing and peak memory usage for Y=100 slices of the NASA Trapezoidal Wing 205M cell dataset

IV. Conclusion

The sub-zone load-on-demand visualization architecture has demonstrated dramatic reductions in memory requirements and substantial reduction in time-to-first-image for isosurface, slice, and streamtrace calculations in a variety of structured and unstructured synthetic and CFD datasets. The memory requirement for large cases is 4 to 55 times less than the memory requirement for the standard Tecplot 360. Likewise, the time to first images is between 3 and 540 times faster than the standard Tecplot 360. In all cases, the largest unstructured-grid cases gave the greatest improvements.

The synthetic study demonstrated that the time to first image and memory requirements of subzone load-on-demand scales approximately with $O(n^{2/3})$, where n, is the number of grid cells. This result is significant because it helps close the widening gap between disk read performance, which doubles in performance every 36 months, and CPU performance, which doubles in speed every 18 months (with Moore's law).

The downside of subzone load-on-demand is that it requires a new file format to get the dramatic reductions in time to first image. Memory reductions, however, might be obtained with existing file formats if properly indexed.

Acknowledgments

The authors would like to thank Chris Nelson for providing the wind-turbine dataset.

References

[1]"Hitachi Global Storage Technologies," http://www.hitachigst.com/hdd/technolo/overview/storagetechchart.html.

[2]Moran, P.J., "Field Model: An Object-Oriented Data Model for Fields," NASA TR NAS-01-005, 2005.

[3]Chiang, Y.-J., ElSana, J., Lindstrom, P., Pajarolo, R., and Silva, C.T., "Out-of-Core Algorithms for Scientific Visualization and Computer Graphics," Tutorial Course Notes, IEEE Visualization 2003.

[4]Chiang, Y.-J., and Silva, C.T., "External Memory techniques for Isosurface Extraction in Scientific Visualization," *External Memory Algorithms and Visualization, DIMACS Series*, 50:247-277, 1999.

[5]Chiang, Y.-J., and Silva, C.T., "Interactive Out-Of-Core Isosurface Extraction," *IEEE Visualization 98*, 167-174, Oct. 1998.

[6]Ueng, S.-K., Sikorski, C., and Ma, K.-L., "Out-of-Core Streamline Visualization on Large Unstructured Meshes," *IEEE Transactions on Visualization and Computer Graphics*, 3(4):370-380, Oct. 1997.

[7]Fox, G. C. ``A review of automatic load balancing and decomposition methods for the hypercube,'' in M. Schultz, editor, Numerical Algorithms for Modern Parallel Computer Architectures, pages 63-76. Springer-Verlag, New York, 1988. Caltech Report C3P-385b.

[8]de Ronde, J.F., Schoneveld, A. and Sloot, P.M.A., "Load Balancing by Redundant Decomposition and Mapping,", *Future Generation Computer Systems*, 12(5):391-406, 1997.

[9]Weinkauf, T. and Theisel, H., "Streak Lines as Tangent Curves of a Derived Vector Field,", *IEEE Transactions on Visualization and Computer Graphics*, Vol. 16, Issue 2, Oct 2010.

[10]Moran, P.J., Henze, C., "Large Field Visualization With Demand-Driven Calculation," ieee_vis, pp.2, 10th IEEE Visualization 1999 (VIS '99), 1999.

[11]Nelson, C.C., Cain, A.B., Raman, G., Chan, T.C., Saunders, R.M., Noble, J., Engeln, R., Dougherty, R., Brentner, K.S., and Morris, P.J. "Numerical Studies of Wind Turbine Acoustics," AIAA 2012-0006, Jan. 2012.

[12]Moran, P.J., Henze, C., "Large Field Visualization With Demand-Driven Calculation," ieee_vis, pp.2, 10th IEEE Visualization 1999 (VIS '99), 1999.

[13]Slotnick, J.P., Hannon, J.A., and Chaffin, M., "Overview of the First AIAA High Lift Prediction Workshop," AIAA 2011-0862, Jan. 2011.

[14]Rumsey, C.L., Long, M., and Stuever, R.A., and Wayman, T.R.., "Summary of the First AIAA CFD High Lift Prediction Workshop," AIAA 2011-0939, Jan. 2011.