Compression of Finite-Element Node Maps with Subzone 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 subzone load-on-demand (SZL) visualization architecture has been developed which only loads the data needed to create the desired plot. The sub-division of the dataset into subzones with 256 cells or nodes allows the node map of finite-element data to be compressed by more than a factor of two. The compressed node maps allow random access for each cell. The resulting file is 37% to 55% smaller than the original datasets and loading is faster.

Nomenclature

N_k^c. = node number at corner *k* of cell *c* of a cell subzone
S_n = referenced node subzone *n* of cell subzone
O_k^c = offset into referenced node subzones for corner *k* of cell *c* of a cell subzone
n_k^c = index within the node subzone for corner *k* of cell *c* of a cell subzone

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

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 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 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 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 are 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. Assuming 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 are 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).

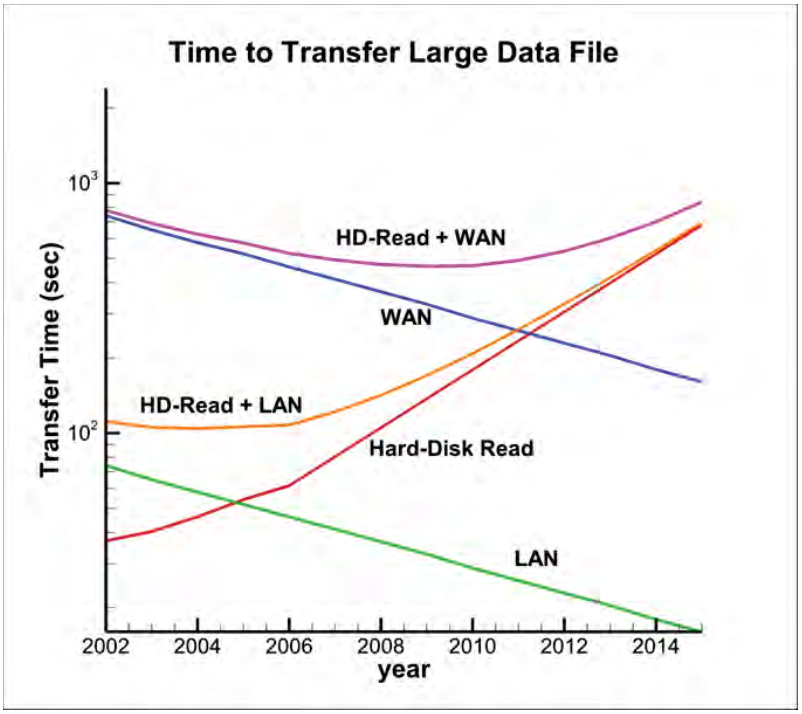

Figure 1. Time to transfer and load a large CFD dataset

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 compute 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, iso-surfaces, 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 new 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 a previous paper[11], a new architecture was described for visualizing large CFD datasets. It was based on loading subzones (spatially correlated sub-segments of the full dataset of up to 256 nodes or cells) on demand (only as needed). To support this algorithm, trees can be created to rapidly select the needed subzones. The architecture is sustainable: for slices and iso-surfaces, the number of subzones loaded is approximately $O(n^{\frac{2}{3}})$ and for streamtraces it is approximately $O(n^{\frac{1}{3}})$.

In this paper, a node-map compression scheme is described for subzone load-on-demand data. In compressed form, the node-maps for each cell may still be accessed randomly, so the new data structures may be used inside the application to reduce memory usage. The compression is based on the fact that the cells in each cell subzone tend to use nodes from relatively few node subzones.

II. Approach

A. Related Work

Many techniques have been proposed for compression of finite-element node maps. Most involve a technique like the cut border machine[17], where cells are added progressively by starting with an existing cell face and specifying the variables of the new node(s), or a progressive grid scheme like implant sprays[16], where nodes are expanded into edges or tetrahedra. These schemes contain additional compression steps and are streaming in nature. To decompress the file you need to, in general, stream through a large portion of the file. These techniques yield dramatic reductions in node-map size but they are not applicable to the subzone load-on-demand grid where we wish to load in small portions of the file independently. Also, it is ideal if the compression scheme also allows random access to the node map for each cell so that it can be used as the data structure in memory.

B. Compression Algorithm

The new node-map compression algorithm utilizes the structures of subzones. In subzone load-on-demand the cell subzones are created independently of the node subzones. Cell subzones (mostly) contain 256 cells and node subzones

(mostly) contain 256 nodes. There are frequently one or two smaller subzones to handle the common cases where the number of cells and/or nodes is not divisible by 256.

The node-map is the list of nodes at the corners of the cells. For tetrahedra there are four node numbers listed per cell and for hexahedra there are eight node numbers listed per cells. The node numbers listed in the node maps are usually 32-bit numbers for large data sets less than two billion cells, but with subzone load-on-demand it makes more sense to use a (node-subzone-number, offset-within-subzone) pair to specify the nodes. The offset is an 8-bit unsigned integer (256 nodes per node subzone) and the subzone number is a 32-bit integer.

Node-map compression is possible because the node map for each cell subzone references nodes from far less than the four billion subzones that could be specified by a 32-bit integer. Figure 2 contains two histograms showing the number of node subzones referenced by cell subzones for large aircraft grids (183M cells and 204M cells respectively). Notice that for the generic transport 98% of the cell subzones reference 16 or less node subzones and the remaining 2% of the cell subzones reference more than 16 but less than 257 node subzones. Likewise for the NASA Trapezoidal Wing, 69% of the cell subzones reference 16 or less node subzones and the remaining 31% reference more than 16 but less than 257 node subzones.

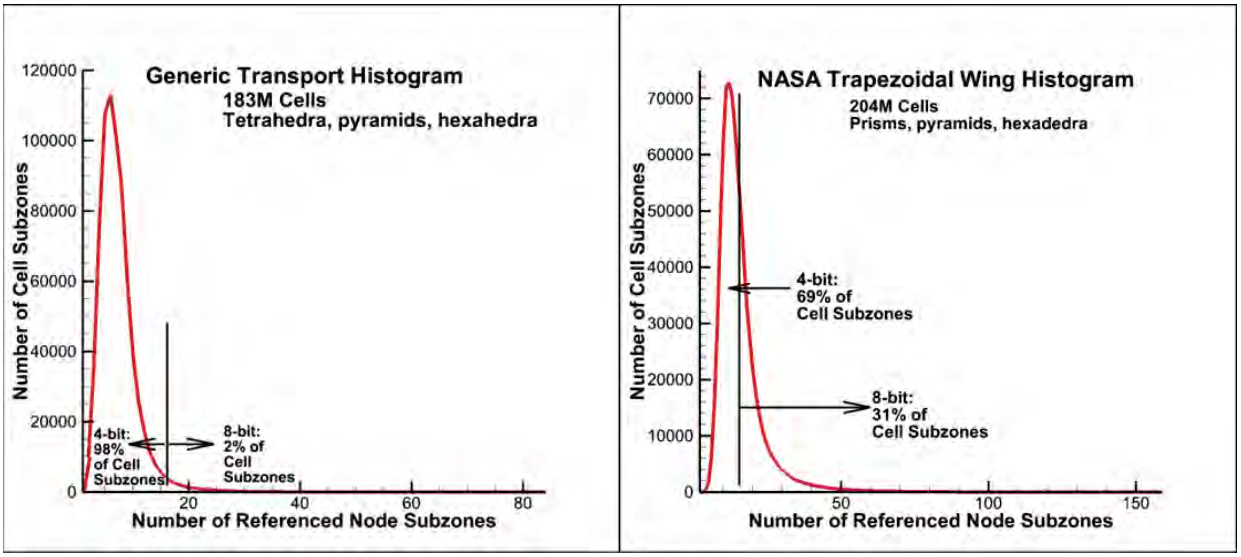

Figure 2. Histograms for number of cell subzones versus number of referenced node subzones. The cases are both finite-element with mixtures of tetrahedral, pyramid, prism, and hexahedral elements.

To compress the list of 32-bit integers for the node subzone numbers

$$N_1^1, N_2^1, N_3^1, N_4^1, N_5^1, N_6^1, N_7^1, N_8^1$$
$$N_1^2, N_2^2, N_3^2, N_4^2, N_5^2, N_6^2, N_7^2, N_8^2$$
$$\ldots$$
$$N_1^{256}, N_2^{256}, N_3^{256}, N_4^{256}, N_5^{256}, N_6^{256}, N_7^{256}, N_8^{256}$$

is replaced by a list of variable precision integers containing offset, O_{corner}^{cell}, into a list of referenced node subzones, S_n, and a list of 8-bit integers containing the index, n_{corner}^{cell}, within that node subzone.

$$S_1, S_2, \ldots$$
$$O_1^1, O_2^1, O_3^1, O_4^1, O_5^1, O_6^1, O_7^1, O_8^1$$
$$O_1^2, O_2^2, O_3^2, O_4^2, O_5^2, O_6^2, O_7^2, O_8^2$$
$$\ldots$$
$$O_1^{256}, O_2^{256}, O_3^{256}, O_4^{256}, O_5^{256}, O_6^{256}, O_7^{256}, O_8^{256}$$
$$n_1^1, n_2^1, n_3^1, n_4^1, n_5^1, n_6^1, n_7^1, n_8^1$$

$$n_1^2, n_2^2, n_3^2, n_4^2, n_5^2, n_6^2, n_7^2, n_8^2$$
...

$$n_1^{256}, n_2^{256}, n_3^{256}, n_4^{256}, n_5^{256}, n_6^{256}, n_7^{256}, n_8^{256}$$

The precision of O_{corner}^{cell} can be 4-bit (up to 16 node subzones referenced), 8-bit (17 to 256 node subzones referenced), or 16-bit (257 to 65,536 node subzones referenced). An offset of greater than 16-bit will never be needed because the maximum number of references is 256 cells times 8 nodes per cell.

For the cases shown in figure 2, and the other cases we've examined, the vast majority of offsets can be either 4-bit or 8-bit unsigned integers, resulting in a substantial (greater than 75%) reduction in the size of the node lists. This reduction is offset somewhat by the addition of the second list of referenced node subzones. Still, the final node map is roughly 40% the size of the uncompressed node map.

It would be possible to reduce the size of the node map further by using additional gradations in the size of the encoded offsets. In the optimal case, the offsets for each cell subzone would be encoded in the minimum number of bits needed for the size of the referenced node subzone list. For example, a cell subzone that referenced only 2 node subzones could have 1-bit offsets into the referenced node subzone list. Likewise, a cell subzone that references 19 cell subzones could have 5-bit offsets into the referenced node subzone list. Using this optimal encoding would reduce the size of the node-maps by roughly another 10%. However, decoding the offsets is more difficult and may be slower.

Note that this compression is still random access. It doesn't require any streaming decompression before it is used to determine the nodes of a specified cell. It can therefore be used directly in the analysis code data structure to reduce memory requirements. When implemented in the subzone load-on-demand version of the Tecplot 360 visualization software, the reduction in required memory was consistent with reduction in file size. More specific number are given for specific cases in the Results section.

III. Results

The compression was tested on two synthetic datasets and three CFD datasets. The synthetic datasets are cylindrical IJK-ordered data converted to tetrahedral and hexahedral elements. The CFD datasets are all mixed-elements. They include a pair of helicopters flying in formation, a generic transport aircraft in landing configuration and the NASA trapezoidal wing dataset in the High Lift Prediction workshop[11,12]. More details on the test cases are given in subsections A through E. Timing tests were run on a Windows Vista 64-bit workstation having 32GB of memory and dual Intel Xeon E5404 4-core processors. All data was stored on the local disk drive and the computer was restarted between tests to clear any disk cache.

Data Set	Cells	Nodes	PLT size	SZPLT size Uncompressed	SZPLT size Compressed	File Compression
Synthetic Tetrahedra	30.0M	5.12M	563MB	567MB	269MB	53%
Synthetic Hexahedra	30.0M	30.4M	1.45GB	1.46GB	885MB	39%
Helicopters	8.29M	1.94M	296MB	298MB	134MB	55%
Generic Transport	183M	44.0M	6.57GB	6.60GB	2.97GB	55%
Trap Wing (half)	204M	228M	10.2GB	10.3GB	6.48GB	37%

Table 1. File compression results

Data Set	Cells	Nodes	Nodemap Uncompressed	Nodemap Compressed	Nodemap Compression
Synthetic Tetrahedra	30.0M	5.12M	485M	187M	61%
Synthetic Hexahedra	30.0M	30.4M	974M	398M	59%
Helicopters	8.29M	1.94M	267M	103M	61%
Generic Transport	183M	44.0M	5.90G	2.27G	62%
Trap Wing (half)	204M	228M	6.65G	2.83G	57%

Table 2. Node map compression results

Tables 1 and 2 show the file and node-map compression results. For file compression only four variables (the minimum number) were included in the file. Additional variables would decrease the effective compression.

As expected, the uncompressed subzone load-on-demand (szplt) file is slightly larger than the standard Tecplot (plt) file. This is because of the additional header information and indexing required to track subzones. After compression, the subzone load-on-demand (szplt) file is between 37% and 55% smaller than the standard Tecplot (plt) file.

Table 2 examines just the node-map section of the file – the only section where compression is occurring. The node-map is compressed between 57% and 61%.

Data Set	PLT	SzUncomp	SzComp	Plt/SzComp	SzUn/SzComp
Synthetic Tetrahedra	00:15.4	00:07.7	00:07.0	2.20	1.10
Synthetic Hexahedra	00:27.6	00:10.0	00:08.2	3.39	1.22
Helicopters	00:10.1	00:03.1	00:03.5	2.94	0.91
Generic Transport	02:56.0	00:19.4	00:17.3	10.18	1.12
Trap Wing (half)	08:14.9	00:20.8	00:18.5	26.78	1.13

Table 3. Time to load data and generate slice.

Data Set	PLT	SzUncomp	SzComp	Plt/SzComp	SzUn/SzComp
Synthetic Tetrahedra	00:18.2	00:02.0	00:02.0	9.27	1.02
Synthetic Hexahedra	00:15.7	00:04.9	00:04.0	3.94	1.22
Helicopters	00:25.6	00:15.1	00:15.5	1.65	0.97
Generic Transport	03:28.0	00:13.7	00:12.3	16.91	1.11
Trap Wing (half)	09:24.1	01:03.4	00:50.0	11.29	1.27

Table 4. Time to load data and generate iso-surface.

Tables 3 and 4 show the impact on performance of the subzone load-on-demand and compression. In Table 3, the time to load data and generate a slice are compared for all five cases using standard Tecplot (plt), uncompressed subzone load-on-demand (szplt), and compressed subzone load-on-demand (szplt). The compressed subzone load-on-demand (szplt) is substantially faster than the standard Tecplot (plt) for all case – between 2.2 times faster and 26.78 times faster. Likewise, the compressed subzone load-on-demand is faster than the uncompressed subzone load-on-demand in all cases but one. The results in Table 4 are similar. Speedups relative to standard Tecplot data vary from 1.65 times faster to 16.91 times faster. Speedups relative to uncompressed subzone load-on-demand data vary from 3% slower to 27% faster.

The speed-up with compression is because less data must be loaded. However, that benefit is reduced by the need for additional processing. In most cases the benefit of loading less data outweighs the cost of the additional processing. In one case, the Helicopters, the opposite appears to be true. However, compression always reduces file size.

A. Synthetic Tetrahedra

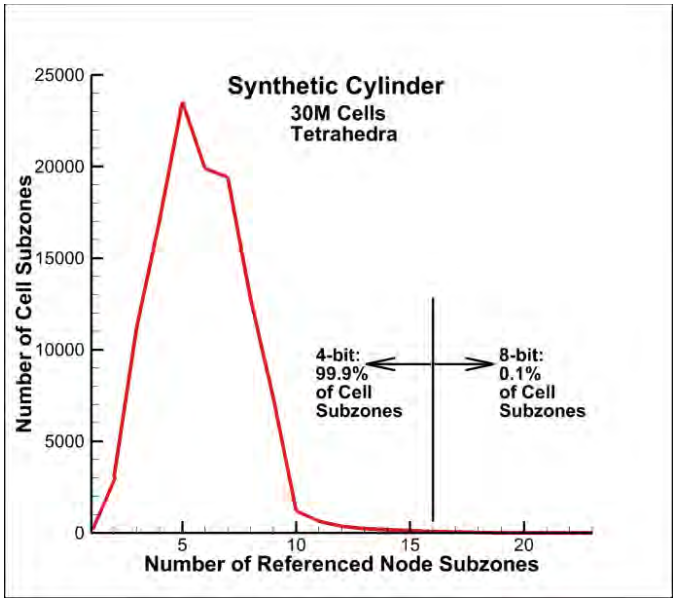

Figure 3. Histogram of number of referenced node subzones for the Synthetic Tetrahedra.

The histogram for the synthetic cylinder with Tetrahedra is shown in Figure 3. The vast majority, 99.9%, of the cell subzones reference 16 or fewer node subzones and use 4-bit encoding. As a result, the compression scheme is very effective, with the node-maps being compressed by 61%. The file, with four variables, is compressed by 53%.

Two test cases, time to first image for a slice and an iso-surface, were performed with standard Tecplot plt files, the uncompressed subzone load-on-demand szplt files, and the compressed subzone load-on-demand szplt files. The results are show in tables 3 and 4. The time-to-first-image a factor of 2.2 and 9.27 times less with compressed subzone load-on-demand than with the standard Tecplot files. Relative to the uncompressed subzone load-on-demand, the timing with compression is 2% to 10% faster.

B. Synthetic Hexahedra

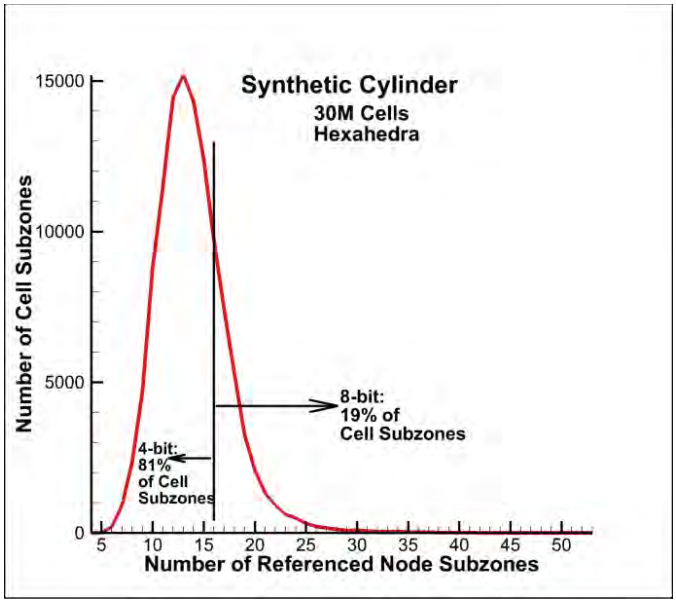

Figure 4. Histogram of number of referenced node subzones for the Synthetic Hexahedra

The histogram for the synthetic cylinder with hexahedra is shown in Figure 4. The majority, 81%, of the cell subzones reference 16 or fewer node subzones and use 4-bit encoding. As a result, the compression scheme is very effective, with the node-maps being compressed by 59%. The file, with four variables, is compressed by 39%. The file compression is less effective than with the Tetrahedra because the number of nodes is much larger, and the uncompressed portion of the file (the coordinates and field data) scales with the number of nodes.

Two test cases, time to first image for a slice and an iso-surface, were performed with standard Tecplot plt files, the uncompressed subzone load-on-demand szplt files, and the compressed subzone load-on-demand szplt files. The results are show in tables 3 and 4. The time-to-first-image a factor of 3.39 and 3.94 times less with compressed subzone load-on-demand than with the standard Tecplot files. Relative to the uncompressed subzone load-on-demand, the timing with compression was 22% faster in both cases.

C. Helicopters

The third example is subsonic flow past a pair of helicopters flying in formation, as shown in Figure 5. The grid is composed of 8.29 million cells and 1.94 million nodes, with prisms near the wall and tetrahedral away from the wall. There are separate overset grids for each helicopter and the node-map is represented as a hexahedral node-map (8 nodes per cell) with repeated node numbers for tetrahedra and prisms. The initial file size was 298MB, of which 267MB is node-map.

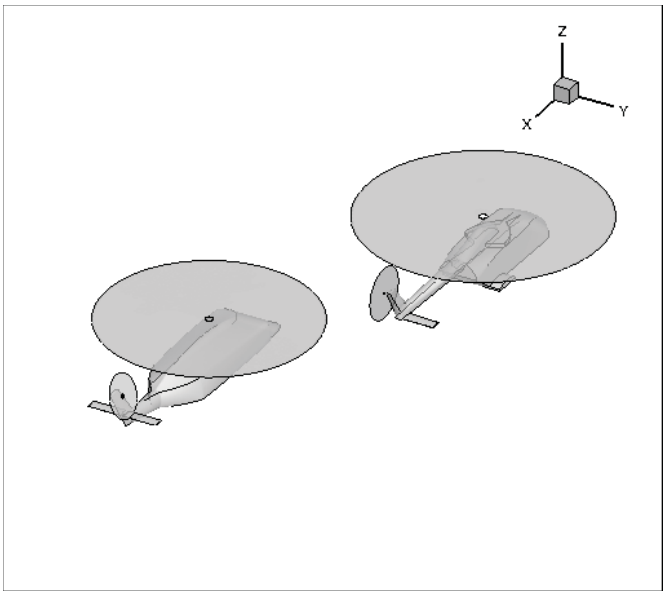

Figure 5. Helicopters flying in formation.

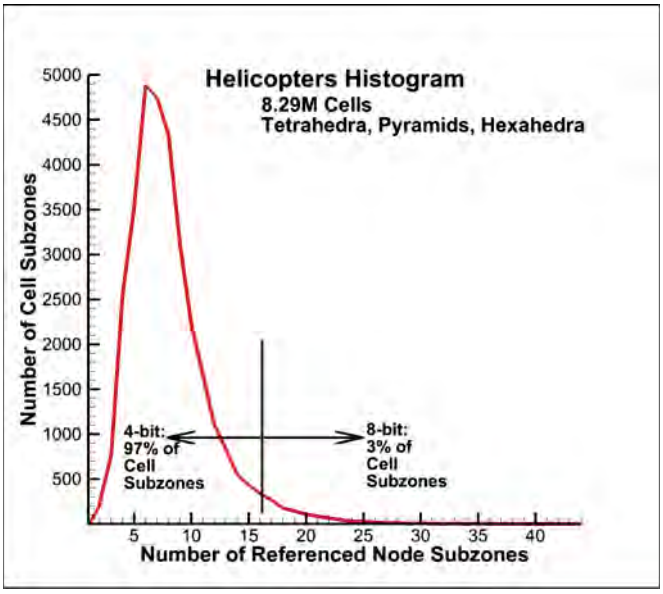

Figure 6. Histogram of number of referenced node subzones for helicopters

The histogram for the helicopters case is shown in Figure 6. The vast majority, 97%, of the cell subzones reference 16 or fewer node subzones and use 4-bit encoding. As a result, the compression scheme is very effective. The compression yielded a 61% reduction in the size of the node map from 297MB to 103MB, which resulted in a 55% reduction in the file size from 298MB to 134MB. The compressed file was between 3% and 9% slower to load and generate a plot than using subzone load-on-demand without compression. It is 1.65 to 2.94 times faster than loading data and generating the plot using the standard Tecplot (plt) file.

D. Generic Transport Aircraft in Landing Configuration

The fourth example is subsonic flow past a generic transport aircraft in landing configuration, shown in Figure 3. The flaps, slats, and landing gear are down, and the aircraft is in a strong crosswind. The grid is composed of 183 million cells and 44 million nodes, with hexahedra near the wall, tetrahedra in the far field, and pyramids in the transition area. The mixed cell type node map is represented as a hexahedral node-map (8 nodes per cell) with repeated node numbers for tetrahedral and pyramids. The file has 4 variables at each node. The initial file size was 6.60GB, of which 5.90GB was node map.

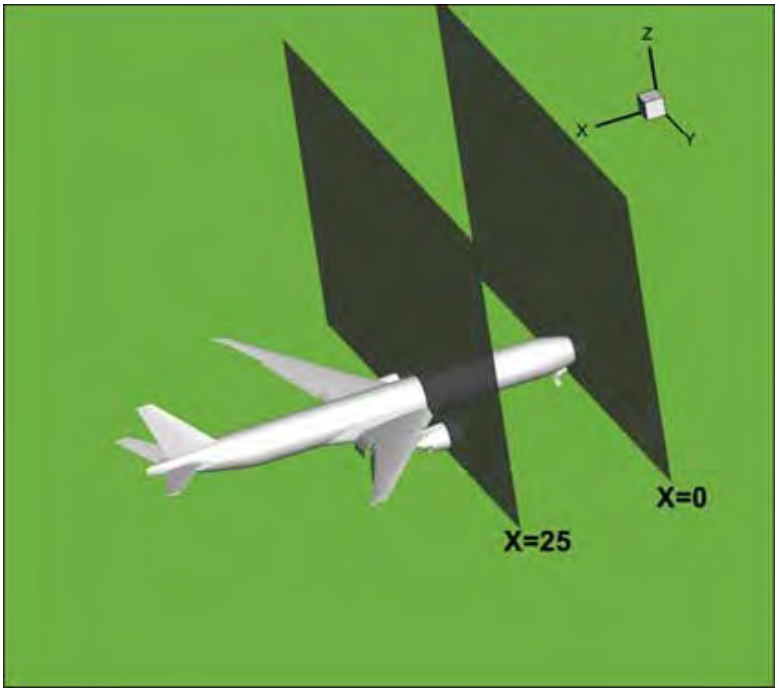

Figure 7. Generic transport aircraft in landing configuration and two slice planes.

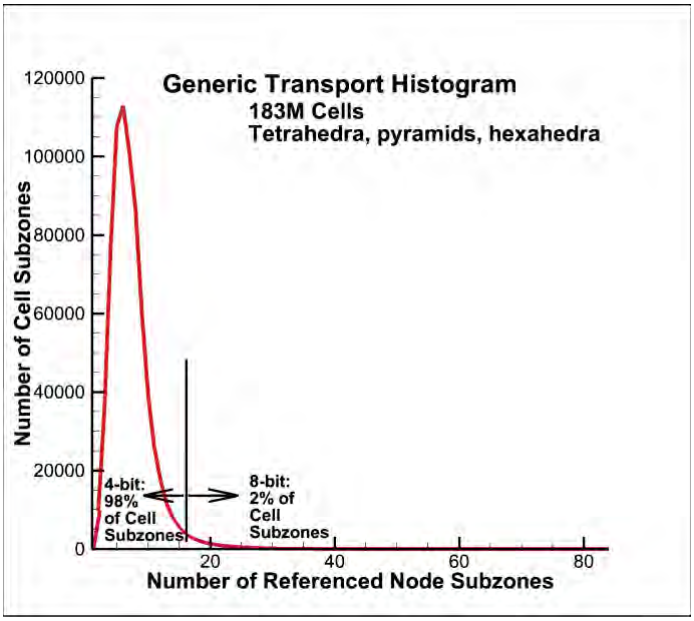

Figure 8. Histogram of number of referenced node subzones for generic transport

The histogram for the generic transport case is shown in Figure 8. The vast majority, 98%, of the cell subzones reference 16 or fewer node subzones and use 4-bit encoding. As a result, the compression scheme is very effective. The compression yielded a 62% reduction in the size of the node map from 5.90GB to 2.27GB, which resulted in a 44% reduction in the file size from 6.60GB to 2.27GB. The compressed file was also 11% to 12% faster to load and generate a plot than using subzone load-on-demand without compression. It is 10.18 to 16.91 times faster than loading data and generating the plot using the standard Tecplot (plt) file.

E. High Lift Prediction Workshop Unstructured Grid

The fifth example is the NASA trapezoidal wing from the first High Lift Prediction Workshop[5,6] . The data set has 254 million cell and 76 million nodes, with prisms near the wall, tetrahedra in the far-field, and some pyramids in the transitional area. The geometry, shown in figure 4, is a half body with wing flaps and slats extended. The initial file size is 10.3GB, of which 6.65GB is node map.

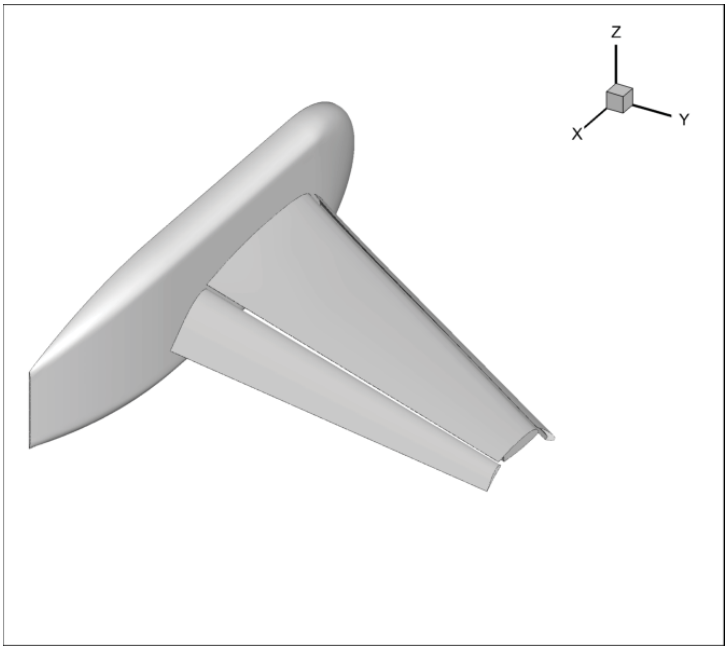

Figure 9. Geometry for the NASA Trapezoidal Wing configuration used in the High Lift Prediction Workshop.

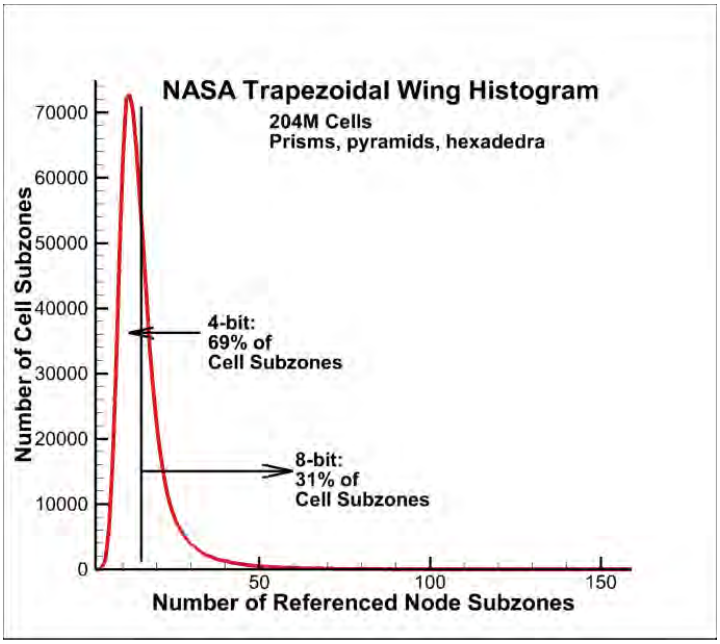

Figure 10. Histogram of number of referenced node subzones for NASA Trapezoidal Wing

The compression yields a 47% reduction in the size of the node map from 6.53GB to 3.49GB, which resulted in a 30% reduction in the file size from 10.19GB to 7.14GB. The compressed file was also faster to load using subzone load-on-demand.

The histogram for the trapezoidal wing case is shown in Figure 10. The majority, 69%, of the cell subzones reference 16 or fewer node subzones and use 4-bit encoding. As a result, the compression scheme is very effective. The compression yielded a 57% reduction in the size of the node map from 6.65GB to 2.83GB, which resulted in a 37% reduction in the file size from 10.3GB to 6.48GB. The compressed file was also 13% to 27% faster to load and generate a plot than using subzone load-on-demand without compression. It is 11.29 to 26.78 times faster than loading data and generating the plot using the standard Tecplot (plt) file.

IV. Conclusion

The node map for the subzone load-on-demand (SZL) visualization architecture has demonstrated significant reductions in file size and memory requirements, as well as substantial reductions in time-to-first-image for slice and iso-surface and streamtrace calculations for large CFD data sets.

V. Further Work

Going forward, we would like to pursue additional bit-size encoding for the offsets to referenced node subzones other than 4, 8 and 16. The bit sizes that appear to be the most promising are 3-bit and 5-bit. Calculations shows that these encodings could save an addition 2-5% off the uncompressed node-map size, and up to 10% off the current compressed sizes. Unfortunately, both of these encodings require a more complicated across-byte packing that may impede performance. 2-bit subzone offsets would not have this problem, but calculations show that the results would be minor (a 1% addition compression).

There is also the question of optimal subzone size. The current size of 256 fits nicely into a byte, but smaller sizes could use the remaining bits for referenced subzone offset encoding.

Also, we would like to extend the variable-sized bit-encoding into the memory storage of the data. Currently, the file representation is converted to 16-bit encoding for all subzones. This conversion removes overhead from data access, but at upfront computation and addition memory us. Storing the data in memory in the same configuration as the file may reduce time to first image and memory consumption, and would perhaps even improve speed due to better processor cache hits on the smaller data structures.

We are also looking into combining our method with other methods such as tetrahedral strip approaches. This would significantly increase the compression of the node-map but it would no longer be possible to randomly access the node map for each cell.

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]Imlay, S.T., and Mackey, C.M., "Improved Performance of Large Data Visualization using Subzone Load-On-Demand," AIAA 2011-1161, Jan. 2013.

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

[13]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.

[14]Isenburg, M. "Compressing Polygon Mesh Connectivity with Degree Duality Prediction," *Graphics Interface*, 2002.

[15]Kronrod, B. and Gotsman, C., "Efficient Coding of Nontriangular Mesh Connecitvity," *Graphical Method,* 63, pp 263-275, 2001.

[16]Pajarola, R. Rossignac, J. Szymczak, A., "Implant Sprays: Compression of Progressive Tetrahedral Mesh Connectivity," *Proceedings of IEEE Visualization 99*, 1999.

[17]Gumhold, S, Guthe, S, and Strasser, W., "Tetrahedral Mesh Compression with the Cut-Border Machine," *Proceedings of IEEE Visualization 99*, 1999.