Internet-based, out-of-core flow visualization

John Tourtellott[*], Gene S. Greger
RayTech Computing Incorporated

ABSTRACT

The cost of visualizing computational fluid dynamics (CFD) and other flow field data sets is increasing rapidly due to ever-increasing grid sizes that constantly strain platform memory capacity and bandwidth. To address this problem of "big data", techniques have been developed in two areas: *out-of-core* visualization, which exploits the fact that most flow visualizations require a very sparse traversal of the data set, and *remote* visualization, in which images are rendered by large-scale computing systems and transmitted via network to desktop systems. A new method, which combines out-of-core and remote techniques, offers a potentially significant improvement in both scalability and cost. By incorporating new techniques for spatial partitioning, data prediction, and explicit memory management, this new method enables desktop computing applications to selectively read the contents of massive data sets from remote servers connected by local or wide area networks. Initial testing has shown that local memory usage is nearly independent of data set size, overcoming the key limitation of prior out-of-core methods. By performing the visualization computations and graphics rendering on the local/desktop platform, the new method also provides a significant improvement in the price-performance ratio compared to current remote visualization methods.

Keywords: computational fluid dynamics, out-of-core visualization, network graphics, particle tracing, remote visualization, spatial prediction, storage management, visualization

1. INTRODUCTION

Interactive visualization of large computational grids is important to many disciplines in science and engineering. Flow visualization is used to graphically display data sets that depict the dynamic behavior of liquids and gases. These data sets are typically produced by computational fluid dynamics (CFD) simulation in order to provide technical insight in fields such as aerospace, turbomachinery, and combustion system design. Typical CFD data sets are organized as three-dimensional (3-D) sampled grids, with each grid coordinate storing physical properties such as spatial position, pressure, temperature, and velocity. Grids come in a variety of configurations depending upon topology and geometry. This paper focuses primarily on structured or curvilinear grids having regular topology and irregular geometry; however, the same concepts can be readily applied to other configurations, such as rectilinear grids (regular topology and regular geometry) or unstructured grids (irregular topology and irregular geometry).

There are many different ways to visualize CFD data sets. Sampled points can be displayed as glyphs, with data set properties such as temperature, pressure, and velocity mapped to glyph characteristics such as size, color, and orientation. Cut planes display a two-dimensional (2-D) area that intersects the space and is color-mapped to a selected scalar property. Advection-based methods, such as streamlines and particle tracing, display the hypothetical path that massless particles traverse over time when injected into the flow field. The path is computed by interpolation and numerical integration of the velocity vector field. Isosurfaces display contiguous 2-D surfaces intersecting the space at a constant (user specified) property value. Other flow visualization methods include volume rendering and line integral convolution.

With continual, aggressive advances in large-scale computing systems, numerical methods such as CFD are being applied to problems on an increasingly larger scale. Present-day supercomputers can process grids with tens of millions, and in some cases, hundreds of millions of vertices. When unsteady flow solutions are computed for hundreds of time steps, this can easily result in data sets that are hundreds of GB in size. Visualizing these data sets is not possible using mainstream (i.e., desktop) computing platforms, but instead requires large-scale graphics computing systems. Methods to deal with the problem of big data are desired not only reduce the visualization costs associated with CFD, but also to increase the number of potential users and collaborators that can take advantage of large-scale numerical simulation.

[*] email johnt@raytechcomputing.com; voice 518-783-1722; fax 800-878-4110; RayTech Computing Incorporated, 5 Herbert Drive, Latham, NY 12110-3819

2. RELATED WORK

2.1 Out-of-core visualization

Out-of-core methods, in which only a portion of the data set is stored in local memory, are often effective since many flow visualization techniques operate on sparse, localized data. A point sample and corresponding glyph, for example, require only a single grid cell to compute and display the local data set properties. Early implementations of out-of-core flow visualization relied on the computing platform's internal paging mechanisms to load grid and solution data into virtual memory. This often caused considerable thrashing, however, and the resulting poor performance led researchers to develop application-controlled data segmentation. In unsteady flow visualization, for example, Lane[1] implemented software to sequentially load and release grid/solution data for each step as (simulated) time proceeds. Cox and Ellsworth[2] improved upon this by preprocessing the data for each time step to produce segments of user-specified size. During visualization, these segments are loaded as needed by the application software, and released by the platform operating system's virtual memory software. Cox and Ellsworth also rearranged the data set file format for improved spatial coherence. When tested with four data sets of size varying between 20-7430 MB, streakline/streamline traversal showed that 9-23 percent of the total file contents were loaded into memory when generating the images.

More aggressive approaches described by Legensky[3] and Ueng et al[4] augment the application-controlled data segmentation approach by constructing meta data (via preprocessing) to relate spatial position to file position. In both cases, the primary goal of improving throughput (frame rate) was achieved, with Legensky's method demonstrating a 5X speedup compared to a highly optimized software implementation without meta data, and Ueng showing a two orders of magnitude speedup compared to software using only operating system virtual memory. In both cases, local memory usage was less than 10 percent of overall data set size; however, further research is needed because data set sizes continue to grow at a much faster rate than platform memory capacity and bandwidth. Most importantly, new algorithms are needed in which performance is not a function of data set size, so that out-of-core techniques can be utilized as data sets continue to grow in scale.

Most of the published research in out-of-core visualization has focused on local methods, in which the data set files are stored in local disk space. In general, the same concepts can be applied to remote out-of-core visualization, in which the data set files are stored on network servers. Although network-based file servers generally provide economic benefits at the enterprise level, individual desktop users may incur lower application performance due to networking overhead. Figure 1 plots the results of three tests done by Cox and Ellsworth using their out-of-core software with data sets stored on a local network server. The horizontal axis plots the ratio of data set size to platform physical memory size, and the vertical axis plots the ratio of remote versus local frame rates. These results indicate that the penalty for remote out-of-core visualization is relatively small, and more importantly, that local and remote methods approach parity as the data set size increases with respect to platform memory. In some cases, remote visualization might run faster than local visualization because the higher speed of network disk drives could more than compensate for network delay.

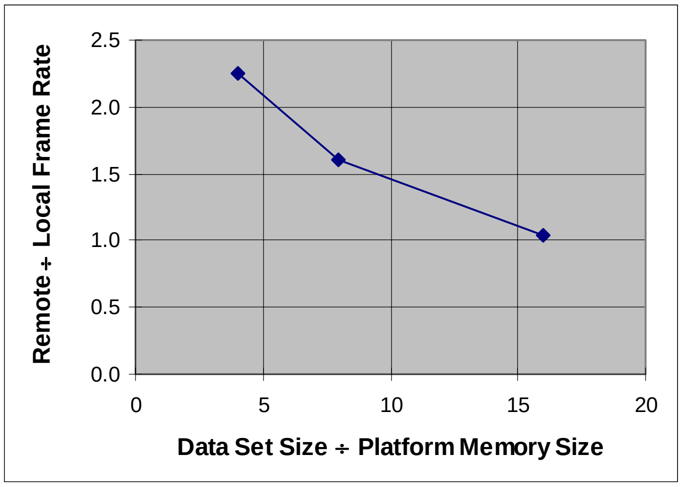


Figure 1. Remote versus local out-of-core visualization (data from Cox and Ellsworth[2])

2.2 Remote visualization

An alternative approach to visualizing big data sets is to perform the computations and graphics rendering in large-scale graphics platforms that are connected directly to the simulation computers. Friesen and Tarman[5], for example, described a system at Sandia National Laboratories that uses a shared asynchronous transfer mode (ATM) network to connect a visualization server in New Mexico with end users in California. The system, depicted schematically in Figure 2, performs visualization computations and graphics rendering in the server and transmits compressed image data via ATM to the remote location. Control messages are sent back over an Internet Protocol (IP) network to the visualization server. Using JPEG image compression, the system was judged to provide good image quality while using 24 Mbps bandwidth to transmit 1280 x 1024 frames (24 bits per pixel) at 30 Hz. A commercial system from SGI provides similar functionality over local and/or wide area IP networks[6].

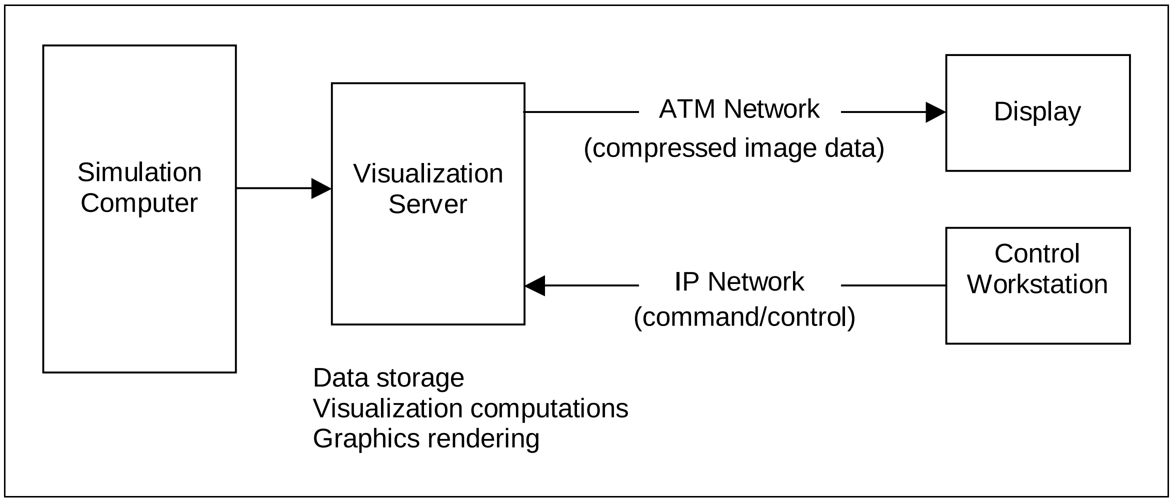


Figure 2. Remote visualization system

The advantages and appeal of remote visualization will become even more significant as major advances are made to the global information infrastructure. Ongoing initiatives such as the Next Generation Internet and Internet2 are developing end-to-end capabilities that will run 1000 times faster than today's Internet. Quality of Service (QoS) protocols will enable users to reserve bandwidth and predictable latency times for demanding applications such as video conferencing and remote telepresence[7]. These new capabilities will enable scientific and engineering collaboration to take place on an unprecedented level. Although current remote visualization systems can be readily adapted to use these new Internet technologies, by performing the visualization computations and graphics rendering in the server, these systems severely underutilize the CPU and graphics resources commonly available in desktop platforms. Incorporating these operations into the desktop will significantly reduce server platform costs.

3. NEW ALGORITHM FOR REMOTE OUT-OF-CORE VISUALIZATION

The primary motivation for a new algorithm is to extend the benefits of remote visualization to out-of-core systems. The basic strategy is to enable desktop computing applications to selectively read the contents of large data sets from servers that are connected by local or wide area networks. To provide a context for outlining key components of our algorithm, Figure 3 shows a functional model of the client-side (desktop) processes involved. In the client-side platform, the *visualization computation* process carries out standard algorithms such as constructing isosurfaces, tracing particles, and/or computing streamlines from flow field data. The *graphics rendering* process handles rasterization and drawing of graphics primitives (polygons, lines, textures) created by the visualization computation, as well as user interaction functions. These two components – visualization computation and graphics rendering – are common to all scientific visualization applications.

To efficiently retrieve data on a just-in-time basis, the system model includes a *spatial prediction* component, which monitors the visualization computations and computes the spatial regions where data set contents are required next. This information is then used by a *spatial-index mapping* component to determine the corresponding data set elements (e.g., grid vertices) that are needed. These results are, in turn, used by the *memory mapping* component to retrieve those needed segments that are not currently stored in local (reserved) memory. The three new components – spatial prediction, spatial-index mapping, and memory mapping – are described in the next three subsections.

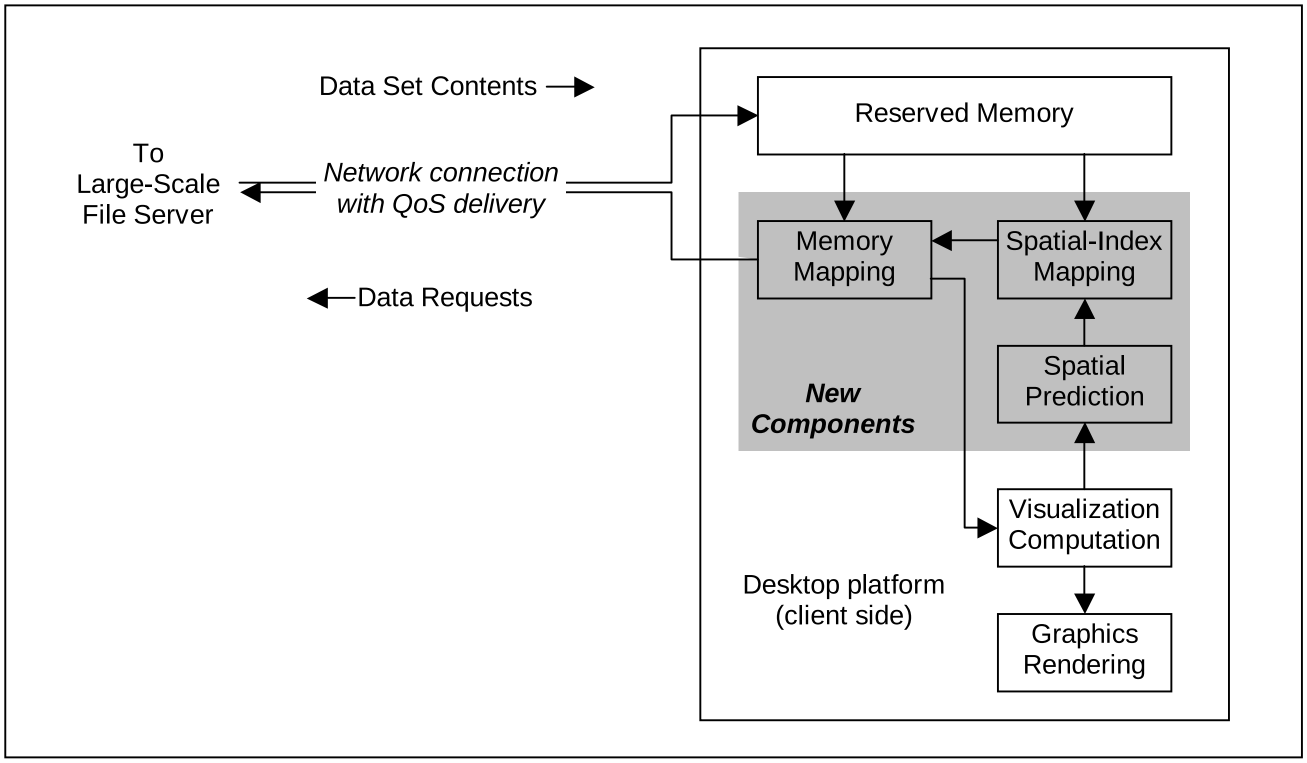
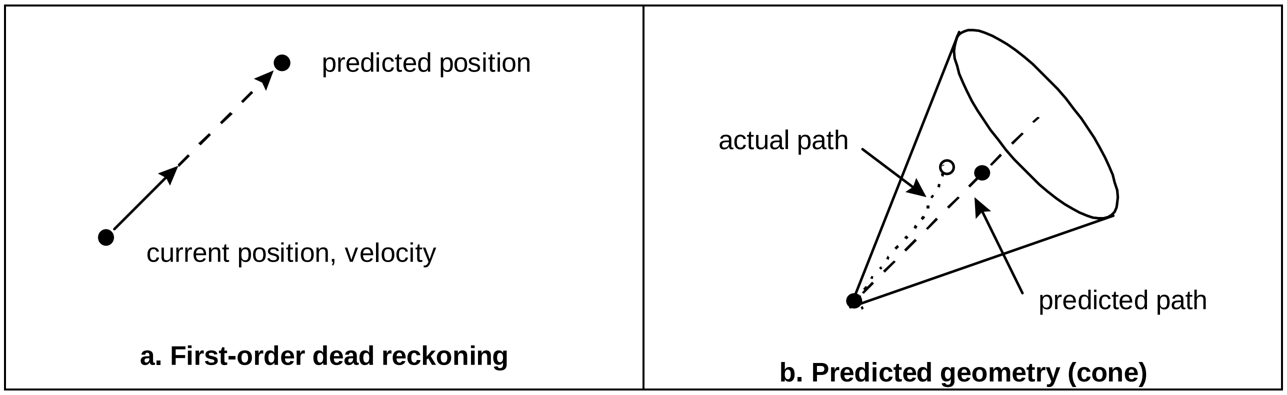


Figure 3. Remote out-of-core visualization (functional model)

3.1 Spatial prediction

The computational algorithms used in spatial prediction are specific to each visualization technique, and must work without using grid or solution data (since those data have not, in general, been retrieved from the server). For advection-based methods, such as particle tracing and streamlines, the particle velocity vector can be used to predict future position via dead reckoning as shown in Figure 4a. From the predicted particle path, a radially symmetric, 3-D envelope can be constructed to encapsulate the region where the actual particle will traverse with a high degree of certainty. In Figure 4b, the dead reckoning path is surrounded by a cone to reflect the fact that uncertainty increases with distance from the current position.



predicted position
current position, velocity
a. First-order dead reckoning
actual path
predicted path
b. Predicted geometry (cone)

Figure 4. Spatial prediction for particle tracing

3.2 Spatial-index mapping

After the spatial regions are computed, an effective meta data structure is needed to provide sorting and searching of data set contents based on spatial position. This data structure, which is computed by preprocessing of flow field data, must have several important characteristics:

- *Nonuniform partition size*: Since grid density varies over many orders of magnitude for typical CFD data sets, the meta data structure should accommodate regions of variable size.
- *Organized by cell (versus point) index*: For a given spatial envelope, a subset of flow field data is needed for interpolating velocity and other properties in that region. If the meta data provides cell indices, then the complete set of point indices can be derived from them, including some points which lie outside the spatial envelope.
- *Support efficient searches*: Searches by spatial envelope must be accomplished with low processing overhead.
- *Easily partitioned*: Because the size of our meta data structure grows roughly in proportion to the size of the flow field data set, it must not consume all of the client platform memory resources. Although partitioning the meta data does not eliminate file bloat, its impact on the client-side platform is manageable.

Given these requirements, an octree data structure was determined to be the most suitable framework for spatial partitioning. The octree hierarchy divides space into regular, rectangular volumes, with each node pointing to a list of indices referring back to the grid and solution files. A simple example is illustrated in Figure 5. The root node of the octree represents the bounding box for the overall flow field, with each recursive level beneath it representing a 2:1 subdivision in each coordinate direction.

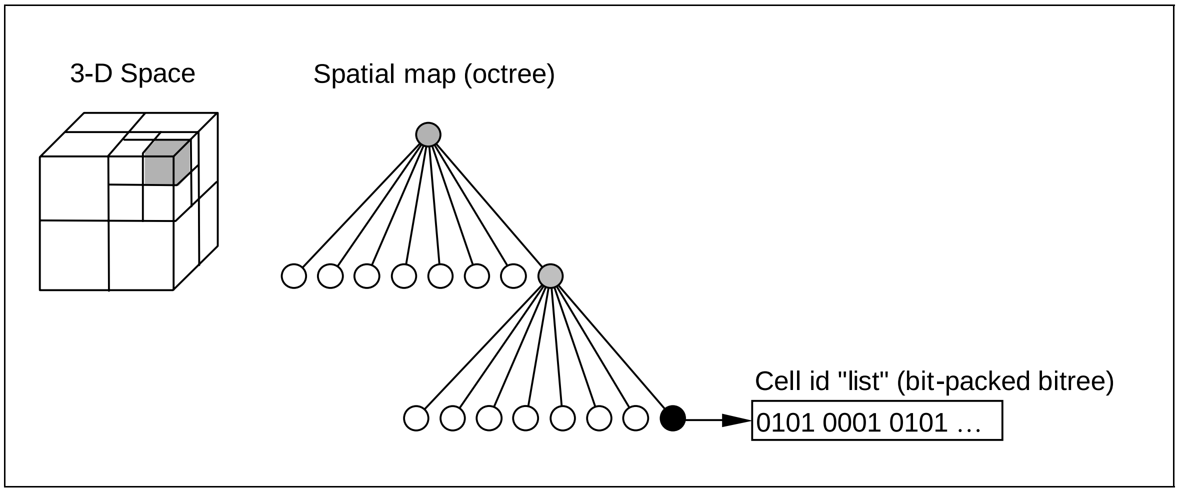


Figure 5. Octree data structure

Each node in the octree stores a "list" of the cell indices that intersect the corresponding region in flow field space. Each cell in the flow field grid is identified by a unique integer id derived from its three coordinate (i,j,k) triplet. The cell ids are not stored in a traditional linked list at each octree node but instead a binary interval tree, or bitree as described by Meagher[8], is used. This approach has two advantages: (1) the bitree can be bit-packed for very efficient memory utilization, and (2) it enables very efficient cell id searching via the memory mapping scheme (see Section 3.3).

In practice, the spatial index map is precomputed for each data set. The processing is derived from the spatial partitioning algorithms developed in the 1980s for accelerating ray tracing[9]. In ray tracing, each geometric object in the 3-D scene is first mapped into a meta data structure which sorts objects by spatial occupancy. For CFD visualization, the geometric "objects" to be partitioned are the tetrahedral and/or hexahedral cells making up the flow field data set. When partitioning objects, a key implementation issue involves selecting the spatial resolution that optimizes the tradeoff between performance (spatial efficiency) and memory efficiency. Conventional methods typically subdivide each geometric object to the same spatial resolution, resulting in very large numbers of voxels for large objects, and consequently very large meta data structures. The approach taken in our system sets the spatial resolution independently for each object, with the octree leaf node level selected to "match" the size of the object bounding box. This ensures that each cell in the data set is partitioned into (a maximum of) eight octree nodes. Compared to methods that use a fixed leaf node level, this approach provides a significant reduction in the size of the spatial-index mapping data structure.

3.3 Memory mapping

Our algorithm includes an explicit memory management strategy in order to optimize the retrieval of data set contents from the remote server, while avoiding the thrashing problems experienced with systems that rely solely on operating system virtual memory. A key component of this strategy is a second meta data structure (in addition to the spatial-index map described in Section 3.2) that provides dynamic sorting and searching of the grid and solution file segments stored in local memory. A bitree data structure is used so that Boolean/set operations, which are needed to eliminate redundancy in the data retrieval requests, can be performed very efficiently. The bitree data structure is updated each visualization frame, as data set points are retrieved from the server and stored in local memory. The bitree is used for two main functions:

- Translating data set indices into memory addresses for the visualization process. For a given grid point with index triplet (i,j,k), the memory mapping software traverses the bitree data structure to obtain the memory location for the desired data, and returns a pointer to the visualization software.
- Pruning lists of spatially predicted points. Before retrieving a list of data set points from the remote server, the memory mapping software first traverses the bitree data structure and removes from the list any/all points that are already stored in local memory.

Because the bitree data structure provides a sorted hierarchy, these functions are carried out with high efficiency, conserving both time and platform processing power.

4. IMPLEMENTATION

To evaluate technical feasibility, a software prototype based on our remote out-of-core algorithm was designed and implemented. Key elements of the system are shown in Figure 6. As the figure implies, the system comprises two software executables. On the right-hand side, a client process performs standard particle tracing calculations and display, using a virtual data set to provide flow field coordinates and velocity data. The client data set is virtual in the sense that it does not store the entire data set in local memory, but retrieves selected points from the server on both predicted and as-needed bases. The client software also includes a data predictor that, using dead reckoning and precomputed meta data, determines which data set contents should be prefetched for future visualization frames. These data are retrieved through a TCP/IP connection from the server process, which, for this implementation, stores the entire data set (PLOT3D file) in memory. When data requests are received from the client process, the server extracts the designated point data and assembles them into a reply message. Key components in the prototype system are described in the following subsections.

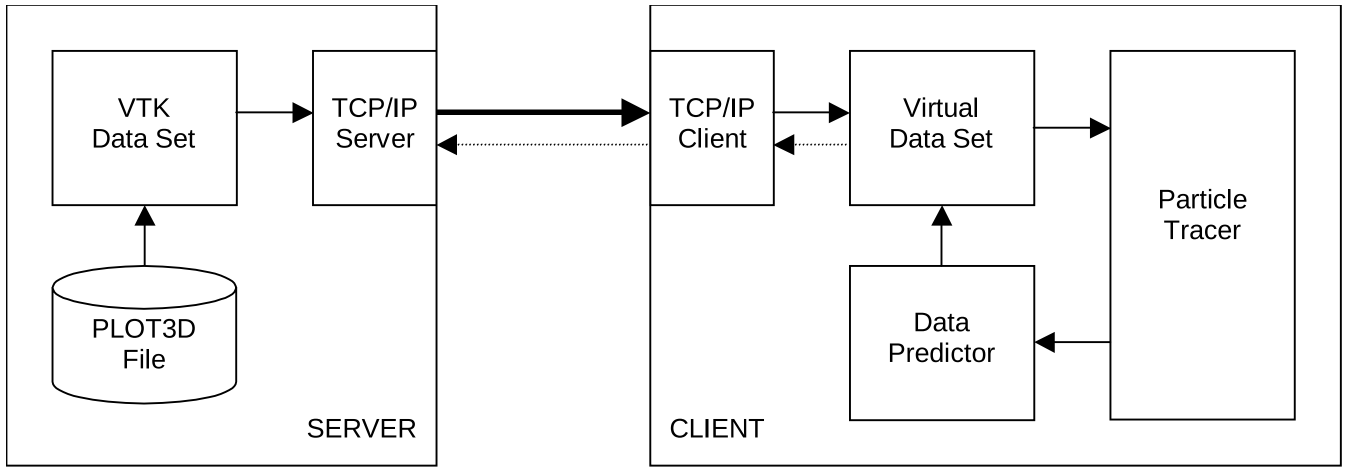
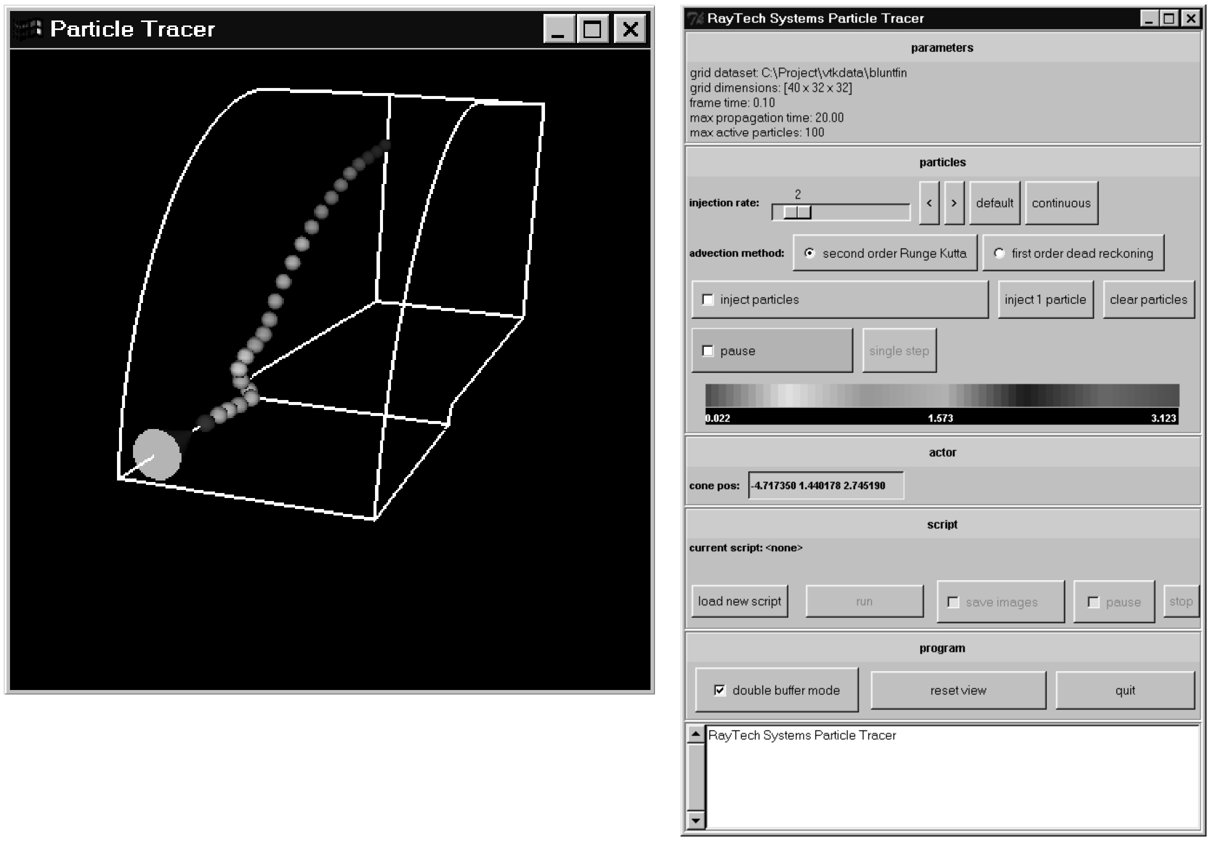


Figure 6. Software prototype (top level block diagram)

4.1 Particle tracer

The starting point for this software was an in-core particle tracer we developed using the Visualization Toolkit (VTK)[10] C++ class library. Screen capture images from the particle tracer are shown in Figure 7. A cone is used to control the location for particle injection. The cone can be interactively dragged to a desired position, and, upon user command, particles are generated at that position and traverse via advection through the flow field. Intra-cell properties are interpolated using a modified Newton's method and numerical integration is done using second-order Runge-Kutta. Each particle is displayed as a sphere that is colored according to the magnitude of the local velocity vector. Development environments and run time executables were implemented for both SGI (IRIX) and Win32 computing platforms.

For out-of-core operation, the particle tracer source code was modified to deal with situations when advection causes a particle to traverse into a cell for which data (for one or more of its vertices) are not currently stored in local memory. When that occurs, the particle tracer notifies the memory mapping software to retrieve the missing data, and then proceeds to the next particle and performs its advection processing. After completing a first pass of all particles, the particle tracer checks that the requested data set contents have been retrieved, and resumes processing any stalled particles.



grid dataset: C:\Project\vtkdata\bluntfin
grid dimensions: [40 x 32 x 32]
frame time: 0.10
max propagation time: 20.00
max active particles: 100

injection rate: 2
advection method: [✓ second order Runge Kutta] first order dead reckoning
☐ inject particles
☐ pause
cone pos: -4.717350 1.440178 2.745190
current script: <none>
☐ save images
☐ pause
✓ double buffer mode

RayTech Systems Particle Tracer

Figure 7. Screen capture images from particle tracer

4.2 Data predictor

The software for predicting which data set contents to retrieve each frame was implemented in three main parts: (1) software to compute spatial geometry; (2) software to traverse the spatially-organized meta data and extract cell ids; and (3) software to efficiently merge cell id data and convert to point ids.

To compute/predict spatial geometry, the software applies dead reckoning to each particle being traced through the flow field. The particle velocity vector, already computed by the advection calculations, is used in conjunction with a parameter specifying the desired lookahead time to compute a predicted path. (In practice, the lookahead parameter could be set to match the expected and/or actual network latency time.) The predicted path is then used as the centerline of a cone defining the predicted geometry, as described in Section 3.1 (Figure 4). For computational efficiency, a bounding pyramid approximation, defined by five first-order surfaces, is used in lieu of the second-order cone geometry.

To traverse the spatially-organized meta data and extract cell ids, the second part of the software converts each pyramid into a constructive solid geometry (CSG) form, by intersecting five half-spaces corresponding to the five pyramid surfaces. Using Meagher's[11] method based on simple arithmetic, the software traverses the octree and marks all nodes intersecting the CSG object, after which it traverses all marked nodes to extract cell id lists, which have been encoded in a compressed, bit -packed format.

In the final step, the individual cell id lists are merged via simple Boolean union operations and the combined list is converted to point ids before being passed to the virtual data set for retrieval from the data set server.

4.3 Virtual data set

The virtual data set provides the storage for data set contents – which, for the prototype system, consist of position and velocity coordinates – retrieved from the server. At program initialization, the virtual data set reserves a block of platform memory as a floating-point array. As point id lists are generated by the data predictor software, they are requested from the server and the retrieved data are stored in the reserved memory. A bitree data structure records the location of each data set point so that it can provide address translation for the visualization processing. The bitree data structure is also used to prune point id lists generated by spatial prediction, in order to only retrieve those points that are not already stored in reserved memory.

4.4 Client-server interface

The communications software between client and server processes is implemented in TCP/IP using standard socket interfaces. On the SGI platform, software classes for client and server processes were implemented using Berkeley (BSD) sockets. The client class has methods for connecting/disconnecting to servers and sending/receiving data using the streaming socket (TCP) protocol, and the server class provides methods for listening for host requests and sending/receiving data. On the PC platform, equivalent client and server classes were implemented using the Windows Socket programming interface. To ensure cross-platform compatibility, data transfers between client and server are formatted using the eXternal Data Representation (XDR) standard.

5. TEST

To test the prototype software, the NASA Blunt Fin[12] and LOX Post[13] data sets were used. To assess the scalability of our out-of-core algorithm, several resolutions of data sets were used. From the Blunt Fin data set, which has 40x32x32 grid points, we generated a double-sampled version (2X) with 79x63x63 grid points and a triple-sampled version (3X) with 118x94x94 grid points. The sizes of the grid plus solution files for these three data sets are 1.3 MB, 10.0 MB, and 33.4 MB, respectively, providing more than an order of magnitude in scale.

Computer software was also developed to preprocess the data sets and construct spatially-organized meta data (spatial-index maps) based on the algorithm outlined in Section 3. The proof-of-concept software was tested on three different computing platforms, all networked to a 10-Mbps Ethernet hub:

- SGI Indigo^2 with 200-MHz MIPS 4400 CPU, High-Impact graphics, running Irix 6.5.4m
- Desktop PC with 450-MHz AMD K6-III CPU, ATI Rage Fury graphics, running Windows 98
- Notebook PC with 600-MHz Intel Pentium III CPU, ATI Rage Mobility graphics, running Windows 98 SE.

To quantitatively evaluate performance, a list of 3-D positions (xyz coordinates) was used to provide injection points for individual particles. For each entry in the list, the software first clears the virtual data set memory, then injects one particle at the designated position, and advects the particle until it exits grid space. The data captured and recorded for each particle includes the:

- Number of image frames rendered
- Number of grid cells traversed
- Number of data set points (xyz and velocity coordinates) retrieved from the server.

For this last item – the number of data set points retrieved – separate counts were made for those points retrieved on a spatially predicted basis versus those points retrieved on an ad hoc basis (due to stall conditions, i.e., when a point is needed outside the predicted regions).

6. RESULTS

6.1 Meta data

Table 1 lists several properties of the four data sets and corresponding meta data that were generated by the spatial-index mapping/preprocessing software. In all four cases, the size of the meta data was between 18 and 22% of the data set size. Compared to some out-of-core systems where file bloat can exceed 100%, this approach represents a significant improvement.

Table 1. Data sets used for testing

Name	Points	Size (KB)	Meta Data (KB)	Percentage
Blunt Fin 1X	40x32x32	1,312	247	18.8%
Blunt Fin 2X	79x63x63	10,035	2,116	21.1%
Blunt Fin 3X	118x94x94	33,366	7,120	21.6%
LOX Post	38x76x38	3,951	738	18.7%

6.2 Total memory usage

To determine memory usage, we used 1000 points distributed throughout the Blunt Fin data set as particle injection points. After injecting and advecting all 1000 particles, the total number of points retrieved from the server for the Blunt Fin 1X, 2X, and 3X data sets were 2.1 million, 5.3 million, and 8.5 million, respectively. A plot of these results as a function of data set size is shown in Figure 8. For reference, the dashed line in the plot represents the theoretical case where memory consumed is proportional to data set size, which is how out-of-core visualization systems typically have been evaluated in the past. Figure 8 shows that our algorithm uses significantly less memory.

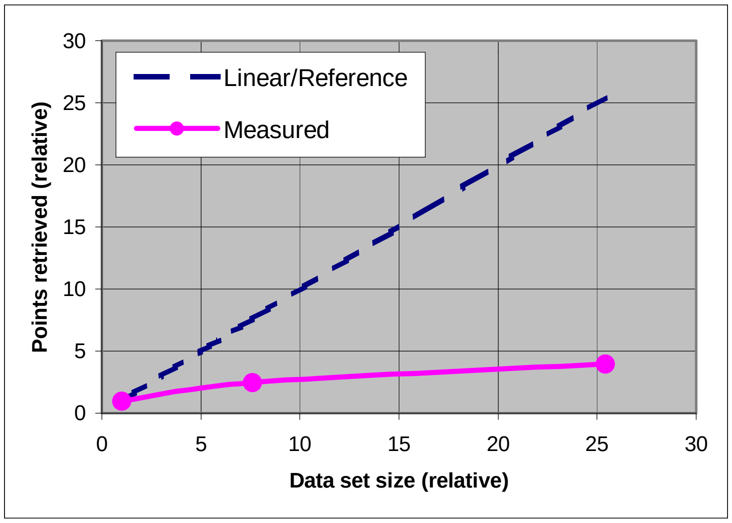


Figure 8. Total memory usage

6.3 Normalized memory usage

A better measure of memory performance should take into account the increased grid density of the 2X and 3X data sets. When particle tracing these data sets, more (and smaller) grid cells are traversed so that, as a result, more grid points should be retrieved by the client software. To provide an "apples-to-apples" comparison, the data in Figure 8 have been normalized by taking the ratio of the number of points retrieved from the server divided by the number of grid cells traversed by advection; i.e.,

$$\frac{Number\ Of\ \ Data\ Set\ Points\ Retrieved}{Number\ Of\ \ Grid\ Cells\ Traversed}$$

The smaller this ratio is for a given test sequence, the better the out-of-core system is at retrieving the minimal data required for a particular visualization. Conversely, the larger this ratio is, the more data that is retrieved, and therefore, the more memory resources are consumed. For the 1X, 2X, and 3X Blunt Fin data sets in Figure 8, the normalized ratios were found to be 55.4, 71.3, and 75.6, respectively. These results are plotted (in relative scale) as a line graph in Figure 9, along with a bar graph showing data set size. Figure 9 provides clear evidence that the remote out-of-core algorithm has uncoupled memory usage from data set size.

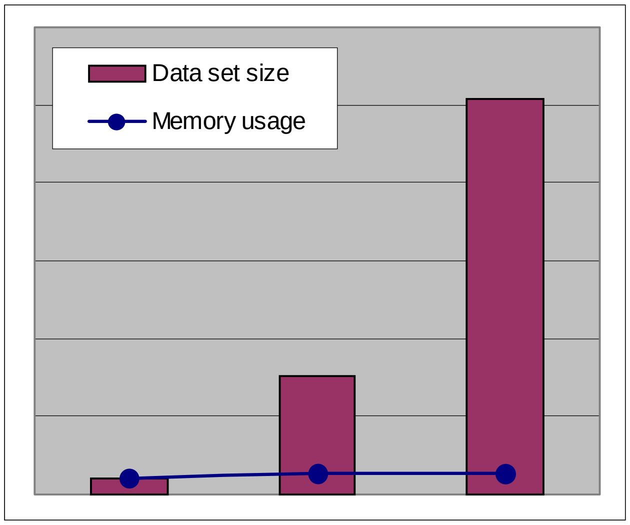


Figure 9. Data set size and normalized memory usage

6.4 Predicted versus ad hoc data retrieval

Another important factor in the success of our remote out-of-core algorithm is the premise that spatial prediction can be used to identify and request data set contents several frames before they are needed. If not, data starvation will occur due to network latency, and as a result, the client visualization software will stall. In the particle tracing tests used to generate Figures 8 and 9, a total of 83,005 data requests were made between the client and server processes. Overall, 97.1% of the transactions were based on spatial prediction and 2.9% were due to stall conditions. (Individual results for the Blunt Fin 1X, 2X, and 3X data sets were very consistent, at 97.4, 97.2, and 96.9%, respectively.) The impact of stall conditions is, therefore, relatively small.

6.5 In-core versus out-of-core throughput

As with any virtual memory system, the proof-of-concept software consumes platform resources/overhead in providing and managing virtual data for the application. Although throughput was not an explicit goal in developing the current software, a comparison of in-core versus out-of-core frame rates can provide an initial estimate of the computational and networking overhead required for the out-of-core algorithm. Using the Blunt Fin 1X data set, throughput measurements were obtained from a 1070-frame test sequence that injects approximately 150 particles that, in turn, traverse several thousand grid cells overall. The results for several platform configurations are shown in Table 2. These results indicate that the combined computational and networking overhead reduces throughput by a factor of 3X-4X.

Table 2. Throughput measurements

Client	Server	Frames per second	
		in-core	out-of-core
SGI Indigo²	localhost	22.5	5.4
	Notebook PC		7.9
Notebook PC	localhost	30.2	10.4
	Desktop PC		8.0
Desktop PC	localhost	27.7	10.2
	Notebook PC		7.9

7. CONCLUSIONS AND FUTURE WORK

The work to date has established that a remote out-of-core system for flow visualization is technically feasible. The implementation and demonstration of working, proof-of-concept software proves that all of the enabling technologies can be implemented and integrated into a practical visualization system. The strategy of spatially predicting and prefetching data set contents is effective in delivering data to the visualization software on a just-in-time basis. In our tests, stall conditions occurred less than 3% of the time. The design of the system meta data was successful in capturing the spatial profile of each data set in a relatively small data structure that can be efficiently traversed for visualization. Testing showed that local/client memory usage is not proportional to data set size (Figure 9), thus overcoming the fundamental limitation of current out-of-core systems. This suggests that the visualization of big data sets can be effectively brought to desktop computing platforms, enabling the entire technical community to use these data sets, instead of a relatively small number of users with mainframe-scale visualization resources.

To scale the prototype software for larger and more general data sets, several additional capabilities are needed. Data replacement software is needed to remove selected memory contents from the client platform as visualization proceeds. The spatially-organized meta data can be used to remove points on a spatial basis, which we expect will provide better memory usage than the more common least-recently-used (LRU) strategy. Additional preprocessing software is needed to partition the meta data into separate, relocatable segments which can be retrieved from server-to-client on an as-needed basis (the current software stores the entire meta data in client memory). The server software should also be optimized to selectively load data set contents from disk to memory, providing more effective use of these resources.

Further extensions to the prototype software are planned to enable visualization of data sets with unstructured grids as well as time-varying data sets. To support unstructured grids, new client software is needed to retrieve and store connection data in addition to point coordinates and velocity. As a result, data retrieval for unstructured grids will take two steps, with the retrieval of connection data first, followed by the retrieval of point (vertex) data. For time-varying data sets, client and server software must be upgraded to retrieve and manage the data set contents in temporal sequence. For each time step, two virtual data sets are stored in local memory, corresponding to the time steps just prior to and just after the current frame time.

ACKNOWLEDGEMENTS

This research was partially funded through the Small Business Innovation Research (SBIR) program at NASA Ames Research Center, Moffett Field, CA. The authors also wish to thank Dr. Charles Law of Kitware, Inc., for his assistance testing and debugging the prototype software.

REFERENCES

1. D. Lane, "UFAT – A Particle Tracer for Time-Dependent Flow Fields," *Proceedings of Visualization '94*, pp. 257-264, Washington, DC, Oct 17 – 21, 1994.
2. M. Cox and D. Ellsworth, "Application-Controlled Demand Paging for Out-of-Core Visualization," *Proceedings of Visualization '97*, pp. 235-244, Phoenix, AZ, Oct 19 – 24, 1997.
3. S. Legensky, "Recent Advances in Unsteady Flow Visualization," *AIAA 13th Computational Fluids Dynamics Conference*, pp 17-22, Snowmass, CO, Jun 29 – Jul 2, 1997.

4. S.-K Ueng, C. Sikorski, and K.-L. Ma, “Out-of-Core Streamline Visualization on Large Unstructured Meshes,” *IEEE Transactions of Visualization and Graphics*, Vol. 3, No. 4, pp. 370-380, Oct – Dec, 1997.
5. J. A. Friesen and T. D. Tarman, “Remote High-Performance Visualization and Collaboration,” *IEEE Computer Graphics & Applications*, Vol. 20, No. 4, pp. 45-49, Jul – Aug, 2000.
6. C. Ohazama, “OpenGL Vizserver,” http://www.sgi.com/Products/PDF/2597.pdf, available through http://www.sgi.com/software/vizserver/.
7. B. Teitelbaum and T. Hanss, “QoS Requirements for Internet2,” *First Internet2 Joint Applications/Engineering QoS Workshop Proceedings*, http://www.internet2.edu/qos/may98Workshop/html/papers.html.
8. D. J. Meagher, “A New Mathematics for Solids Processing,” *Computer Graphics World*, Vol. 7, No. 10, pp. 75-88, Oct 1984.
9. E. Freniere and J. Tourtellott, “A Brief History of Ray Tracing,” *Lens Design, Illumination, and Optomechanical Modeling (SPIE 3130)*, pp. 170-178, San Diego, CA, Jul 29 – 30, 1997.
10. W. Schroeder, K. Martin, and W. Lorenson , *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*, Prentice-Hall, Upper Saddle River, NJ, 1998.
11. D. J. Meagher, “Geometric Modeling Using Octree Encoding,” *Computer Graphics and Image Processing*, Vol. 19, pp. 129 – 147, 1982.
12. C.M. Hung and P.G. Buning, “Blunt Fin,” http://www.nas.nasa.gov/Research/Datasets/Hung/index.shtml.
13. S. E. Rogers, D. Kwak, and U. Kaul, “Liquid Oxygen Post,” http://www.nas.nasa.gov/Research/Datasets/Rogers/index.shtml.