

# Accelerating Large Data Analysis By Exploiting Regularities

David Ellsworth\*

Advanced Management Technologies Incorporated  
NASA Ames Research Center  
ellsworth@nas.nasa.gov

Patrick J. Moran†

NASA Ames Research Center  
patrick.j.moran@nasa.gov

## Abstract

We present techniques for discovering and exploiting regularity in large curvilinear data sets. The data can be based on a single mesh or a mesh composed of multiple submeshes (also known as *zones*). Multi-zone data are typical in Computational Fluid Dynamics (CFD) simulations. Regularities include axis-aligned rectilinear and cylindrical meshes as well as cases where one zone is equivalent to a rigid-body transformation of another. Our algorithms can also discover rigid-body motion of meshes in time-series data. Next, we describe a data model where we can utilize the results from the discovery process in order to accelerate large data visualizations. Where possible, we replace general curvilinear zones with rectilinear or cylindrical zones. In rigid-body motion cases we replace a time-series of meshes with a *transformed mesh* object where a reference mesh is dynamically transformed based on a given time value in order to satisfy geometry requests, on demand. The data model enables us to make these substitutions and dynamic transformations transparently with respect to the visualization algorithms. We present results with large data sets where we combine our mesh replacement and transformation techniques with out-of-core paging in order to achieve analysis speedups ranging from 1.5 to 2.

**CR Categories:** E. Data (large); I.1.3 Languages and Systems, Evaluation strategies; I.3.8 Computer Graphics Applications

**Keywords:** regularity finding, data models, object-oriented, C++, templates, scientific visualization, paging, demand-driven evaluation.

## 1 Introduction

In scientific data set meshes there are often geometric and topological regularities. By *regularities* we mean patterns that can be exploited by visualization systems in order to accelerate the analysis process, or to enable the analysis of much larger data than would be otherwise feasible. Regular meshes—meshes that can be defined as the Cartesian product of regularly sampled intervals—may be the example that first comes to mind when one thinks of regularities, but there are many others. Rectilinear (the Cartesian product of intervals that are not necessarily evenly sampled) and cylindrical meshes all possess regularities. Regularities may also be found

in meshes composed of multiple submeshes, also known as *zones*. For example, a domain may be covered with overlapping curvilinear meshes in expected regions-of-interest but utilize simpler regular or cylindrical meshes in the remainder of the domain, such as in “free stream” regions. Unstructured meshes—i.e., meshes that do not have the uniform topological regularity of structured meshes—may also potentially possess some regularities. Here we restrict ourselves to single and multi-zone structured objects.

The regularities described above suggest two categories of acceleration opportunities. First, we can construct meshes without explicitly storing the coordinates for every vertex. This is a potential savings both in disk space and main memory usage. There is also a savings at mesh construction time since little disk I/O is required. In essence the regularity provides significant compression opportunities. This compression is for the most part lossless, though there may be some slight perturbations due to working with floating-point numbers. Second, there are straightforward point location and interpolation acceleration techniques that apply to meshes with regularities. With a carefully designed data model we can provide these advantages in a manner transparent to the data analysis algorithms operating on model objects.

Time-series data tend to act as a multiplier with respect to the opportunities described above. Savings in disk or memory usage are multiplied by hundreds of time steps. Performance savings due to more efficient point location and interpolation routines are also magnified. Time series data provide a further opportunity: in many cases the mesh at time step  $t + 1$  is the same as in time step  $t$ , except for a rigid-body transformation. With multi-zone meshes we may have a mix of static and moving submeshes. Given the appropriate transformations, we can load a mesh once and then apply the transformations on demand for successive time steps. Again, a carefully designed data model can apply these techniques in a manner transparent to the analysis algorithms.

One impediment to exploiting regularity is that essential data about the data—metadata—are not always explicitly available to analysis tools. For example, file formats such as PLOT3D [Walatka et al. 1992] save all structured meshes using the most general type: curvilinear. The file format does not provide a general means for storing metadata, thus regularity information tends to get separated from the raw data. Clearly, one option is to ask the original scientist for the metadata, but sometimes that option is not available. Even if the scientist is available, the necessary metadata may not be readily accessible. The original meshes may be the product of automated mesh generation or adaptive simulation tools. Such tools may not be instrumented to output regularity information in a form that is easily used by analysis systems. A second impediment to exploiting regularity is that even with the prerequisite metadata, techniques such as mesh substitution and dynamic transformation may require significant changes to one’s visualization algorithm implementations if the data model does not insulate the analysis algorithms from how the data are provided. The effort required to modify the analysis software may be more than one is willing to make.

In this article we present techniques that enable regularity discovery and exploitation in large curvilinear data sets. In the following section we review related work in large data analysis strategies

\*Mail Stop T27A-1, Moffett Field, CA 94035

†Mail Stop T27A-2, Moffett Field, CA 94035

IEEE Visualization 2003,  
October 19-24, 2003, Seattle, Washington, USA  
0-7803-8120-3/03/\$17.00 ©2003 IEEE

and data models. Next we describe the regularity-finding algorithm in detail, followed by an overview of the data model. Following our overview of the regularity discovery algorithms and data model, we present results from three large data sets. Finally, we conclude with a summary of what has been accomplished so far and some thoughts on future work.

## 2 Related Work

### 2.1 Large Data Strategies

Large data visualization has been an active area of research in recent years. Most strategies depend on avoiding the need to load the whole data set into memory at one time. UFAT [Lane 1994] used an approach to particle tracing in time-series data where only a small working set of time steps were loaded at once. UFAT also supported a small number of other visualization techniques. Cox and Ellsworth [1997] describe a more general out-of-core paging system that exploited the fact that many visualization algorithms exhibit both spatial and temporal locality in data access. An alternative to paging is the streaming approach described by Law *et al.* [1999].

There are also numerous efforts that are specific to particular visualization techniques. For example, there have been many algorithms proposed that accelerate isosurface computation by creating an index beforehand that enables an isosurface algorithm to only load data in the neighborhood of the surface [Chiang and Silva 1997; Sutton and Hansen 1999; Itoh and Koyamada 1995; Chiang *et al.* 1998; Shen 1998]. An octree-based approach to limiting the amount of data loaded for streamline computation was described by Ueng *et al.* [1997]. Shen *et al.* [1999a] describe a Time-Space Partitioning (TSP) tree approach to volume rendering time-series data.

### 2.2 Data Models

The importance of a well-designed data model has been recognized early on in the visualization community, and there have been a number of efforts to develop a general design with a strong, formal foundation. One of the earliest was the fiber bundle model by Butler and Pendley [Butler and Pendley 1989]. Their model was inspired by the mathematical abstraction of the same name. Fiber bundles have proven to be somewhat difficult to implement in their pure form, though the concepts have inspired several follow-on efforts. The original fiber bundle abstractions did not provide a convenient means of accessing the underlying discretization (mesh) of a data set. This was a problem since many visualization algorithms operate by iterating over various types of cells of the mesh.

One system in particular that has been influenced by fiber bundle concepts is OpenDX (formerly IBM Data Explorer [Lucas *et al.* 1992; Abram and Treinish 1995]). Beginning with Haber *et al.* [1991], the fiber bundle model was adapted into a model that would support a general-purpose data-flow visualization system. OpenDX can handle both vertex-centered and cell-centered fields.

Another field modeling effort was Field Encapsulation Library (FEL) [Moran *et al.* 2000]. FEL excelled with multi-zone curvilinear grids. FEL differed from most other modeling efforts in that it defined separate class hierarchies for meshes and fields, rather than a single combined object type. FEL introduced fundamental design features that enabled the library to operate efficiently with extremely large data sets, including a consistent demand-driven evaluation model [Moran and Henze 1999] and the integration of demand-paging techniques [Cox and Ellsworth 1997]. FEL assumed that all objects were in  $\mathbf{R}^3$  physical space and vertex-centered.

The Visualization Toolkit (VTK) [Schroeder *et al.* 1997], like OpenDX, is an open source visualization system with a fairly general data model. The VTK data model uses an extended concept of cells, including such primitives as polylines and triangle strips as cell types. Recent extensions [Law *et al.* 1999] have focused on enabling the data model (and ultimately the whole system) to handle large data via streaming. Like FEL, VTK utilizes a demand-driven evaluation strategy.

VisAD [Hibbard *et al.* 1994; Hibbard 1998] is a relatively general, object-oriented model for numerical data. The user can construct data objects with a style similar to expressing mathematical functions. In contrast to the models described previously, VisAD is implemented in Java. The VisAD model is quite flexible, though the performance of the Java implementation makes it less suitable for very large data. The VisAD model does put more effort into the inclusion of metadata—data about data—than most other designs.

## 3 Regularity Finding

The first step to exploiting regularities in large data sets is to find the regularities. The regularities are found using a discovery algorithm which is run as a preprocessing step before any visualizations are computed. The algorithm recognizes three types of zones. It recognizes regular meshes, which include meshes with both regular and irregular spacing; axis-aligned cylindrical meshes; and transformed meshes, ones that are transformed versions of other zones. The high-level algorithm tries the regular mesh discovery algorithm; if that fails, it tries the cylindrical algorithm, and finally it runs the transformed mesh algorithm. For each of these zone types, the algorithm first attempts to recover the parameters that describe the regular, cylindrical, or transformed zone. If the parameters are recovered successfully, all of the vertices in the zone are checked to ensure that they match. If either step fails, the next type is checked. The complete check is time consuming, but is required since we want to guarantee a close match: one that models every vertex correctly. A final step recognizes integer per-vertex classification values known as IBLANKs (used frequently with multi-zone PLOT3D data [Walatka *et al.* 1992]) that either do not vary over time or are shared between time steps.

### 3.1 Errors

Unfortunately, floating-point errors prevent exact matching between the modeled and actual zone. These errors come from the inexact nature of floating-point calculations [Press *et al.* 1998; Pizer and Wallace 1983]. The errors come from three sources. One source is floating-point errors introduced when the vertices in the input files are originally computed. They may not be exactly equal to, for example, a cylinder if the calculation was not done carefully or was done using single precision. Or, errors can be introduced when a rotated zone is created from another existing zone and the rotation matrix used is not exactly as intended due to floating-point errors. A second source of error is part of the discovery algorithm. The calculations to recover the modeled zone's parameters may not recover the parameters exactly due to floating-point error. The third source of error occurs during the verification step, where floating-point error can cause the modeled vertices to not be computed exactly even if the model parameters are exact.

These error sources mean that the modeled and actual vertices can only be compared to a given error tolerance. This error tolerance should be given as an absolute error tolerance because recognizing zones using relative errors (errors expressed as a magnitude of the values being compared) would require loose error bounds. The loose relative error bounds are needed for vertices that have a component that is near zero. For example, if the 2D point (10, 0.0001) is rotated by 20 degrees using single precision math, the

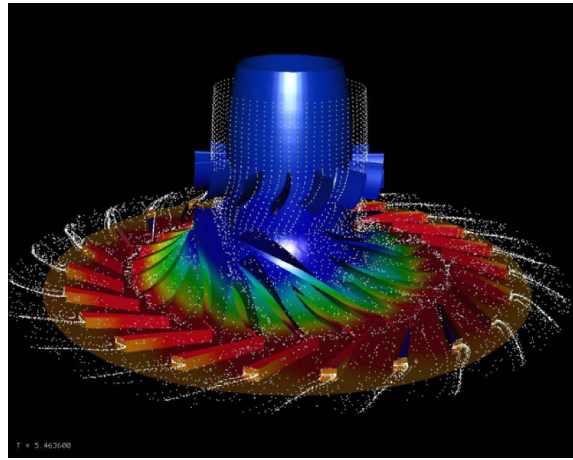
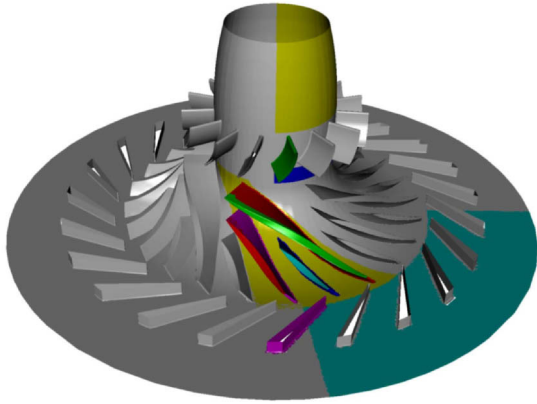


Figure 1: Two views of the same turbo-pump data set: to the left we highlight a subset of the 114 zones in the data set. Using the subset of zones as reference meshes, we can use transformed meshes to represent the remainder of the geometry. To the right is a visualization of the number of turbine rotors and stators are relatively prime. While it is not obvious without an extreme close up, the pressure and thus the color mapping on each blade is not exactly the same, thus we cannot simply replicate the graphics primitives from a subset of the blades in order to produce the full visualization.

resulting point is (9.3968915, 3.4202952). When the transformed point is rotated by -20 degrees, the result is not the original point but (9.9999990, 0.0001001517). The relative error in the  $y$  coordinate is noticeably large, 0.1%, while the absolute error in both coordinates is small. Because of this, while we have the user specify a relative error for convenience, the actual algorithm uses an absolute error. The absolute error is calculated by multiplying the relative error by the size of the zone's bounding box.

### 3.2 Recognizing Regular Zones

Recognizing regular zones is fairly simple. First, each of the computational coordinate axes  $i$ ,  $j$ , and  $k$  is examined to see whether only one physical coordinate changes when each axis is traversed, and that the physical coordinate that changes represents each of the three physical axes. This is done by checking the vertices with  $i = j = 0$ ,  $k = j = 0$ , and  $i = k = 0$ , not the entire mesh. If the computational axis checks succeed, then the mesh spacing is extracted from the vertices along the computational axes, and then all of the vertices are checked.

### 3.3 Recognizing Cylindrical Zones

Cylindrical zones are recognized by first determining which computational axis corresponds to the length of the cylinder, which corresponds to the radius (from the middle of the cylinder to the outside), and which axis corresponds to the rotation of the cylinder; these axes are called the *length*, *radius*, and *theta* axes, respectively. The theta axis direction is determined by examining the three sets of edges starting at one of the corners. Edges along the theta axis will not have collinear edges, while the others will. If any degenerate edges are found, the process restarts at another corner.

The next step is to assign the radius and theta axes to the two remaining computational axes. This is done by noting that adjacent edges along the rotation of the cylinder and going down the length of the cylinder are parallel, while adjacent edges going in and out of the cylinder (that are on the same face or slice of the cylinder) are not parallel. Then the algorithm computes the center of the cylinder by intersecting two radial lines. To reduce numerical error, these lines are chosen to be about 90 degrees apart and to extend the full radius of the cylinder. The final discovery step is to find the mesh

intervals for the length, radial, and theta axes. The final verification step checks that each vertex matches the cylindrical model of the mesh.

### 3.4 Recognizing Transformed Zones

The algorithm will find a match between the *current* zone in the current file, and a *matching* zone. If the current file is the only file, or the first file in the time series, the algorithm only considers zones in that file. In fact, it will only consider zones encountered before the current zone in the file since doing so will consider all pairs of zones. Otherwise, the current file is part of a time series, and the algorithm will iterate over all of the zones in the series's first file, considering in turn each of the zones as the current matching zone. The algorithm stops once a match is found. The recovered transformation is a general  $3 \times 3$  transformation matrix that includes rotations, scales, and shears and a separate translation vector. However, the algorithm is optimized for finding rotations around a single physical coordinate axis, possibly combined with a translation.

The transformation is recovered using four main steps: a dimensions check, a direct solution step, an error minimization step, and a heuristic cleaning step. The dimensions check simply verifies that the dimensions of the current and matching zones are the same. The direct solution step gets a good solution of the 12 values that specify the  $3 \times 3$  matrix and the translation vector by solving a system of linear equations. The error minimization step refines the solution by minimizing the errors between a set of vertices from the current and matching zones. The cleaning step applies heuristics that recognize rotations and reduces the residual error.

The direct solution step uses the fact that the zone-to-zone correspondence between four non-collinear vertices with the same  $i, j, k$  locations in each mesh is sufficient to determine the transformation, provided of course that the zones are actually transformed versions of each other. The transformation can be determined by starting with the transformation equations for a set of vertices,  $\mathbf{m}_i = \mathbf{A}\mathbf{c}_i + \mathbf{t}$ , where  $\mathbf{m}_i$  is a vertex from the matching zone,  $\mathbf{c}_i$  is a vertex from the current zone,  $\mathbf{A}$  is the  $3 \times 3$  transformation matrix, and  $\mathbf{t}$  is the translation vector. The  $\mathbf{m}_i$  and  $\mathbf{c}_i$  vertices share the same locations in the zone. The direct solution step generates a set of 12 linear equations defined by the equation above and the four selected vertex pairs, and then solves the equations for  $\mathbf{A}$  and  $\mathbf{t}$ .

The four vertices used to recover the transformation are found using a greedy algorithm that tries to place them far apart physically. Using widely-spaced vertices avoids a problem seen with closely-spaced vertices: vertices near each other can have coordinates with many digits in common, which would limit the precision of the recovered transformation. The first vertex is chosen to be the zone's computational origin. The second vertex is chosen from a set made up of a  $9 \times 9 \times 9$  lattice of vertices spaced evenly computationally through the mesh, and is the one from the set that creates the longest line from the origin. The third vertex is the vertex from the set that, given the first two vertices, would create the triangle with the largest area. The fourth vertex is the one from the set that creates the tetrahedron with the largest volume given the first three vertices. The fixed  $9 \times 9 \times 9$  lattice was chosen to limit the time for large meshes, and was the smallest lattice that found tetrahedra which gave good results for our data sets.

Given the four vertices, the algorithm solves the system of equations using first LU decomposition and then iterative improvement [Press et al. 1998] to reduce computation errors. Then, an error minimization step further reduces the error using Polak-Rebiere conjugate gradient minimization [Press et al. 1998]. This step is used because the transformation defined by the four pairs of vertices might not be exactly the same as the transformation applied to the zone due to floating-point errors created when the pairs of vertices were calculated. The minimization step minimizes the distance between a large set of vertices from the current and transformed modeled zone. The set of vertices is the unique vertices in the  $9 \times 9 \times 9$  lattice that were considered as candidates for the four vertex pairs used in the direct solution step. This lattice size was chosen because it would reuse points from the previous step and because it had enough vertices to reasonably sample the mesh. We have found that the error minimization step is only important with transformations that are not a simple rotation around an axis plus a translation because the heuristic cleaning step can sufficiently reduce the error in those cases.

The final heuristic cleaning step tries to recognize rotations around a coordinate axis and reduce the error. For each coordinate axis in turn, the transformation matrix is examined to see whether it looks like a rotation around that axis. A rotation matrix should have a single value near one, four values near zero, two cosine values nearly equal to each other, and two sine values that have opposite signs and nearly the same magnitude. The location of the one, zero, cosine, and sine values within the matrix vary according to the rotation axis. In addition, the cosine and sine values must correspond to nearly the same angle. If the matrix is a rotation matrix, the rotation angle is calculated and the matrix is computed from scratch to further reduce numeric errors.

The final part of the cleaning is to remove any small translation values. Here small is defined as values smaller than one LSB in the single precision floating-point value of the size of that zone's bounding box. This step is used because, in practice, the previous calculations do not recover a zero translation when no translation was used. Not deleting these small translations would noticeably affect vertices with components near zero, if the transformation was indeed zero. On the other hand, for most of the vertices with larger components, the resulting vertices would be the same whether or not these small translations are included since the translation makes no difference in the final floating-point value.

Once the transformation has been recovered, all of the zone's vertices are checked to see whether they have been indeed transformed from the other zone's vertices. If so, the algorithm reports the transformation and quits; otherwise, the algorithm checks the remaining zones.

### 3.5 Recognizing Duplicate IBLANK Data

Our experimental data sets are in the PLOT3D [Walatka et al. 1992] format. PLOT3D data can include what are known as IBLANKS: associated with each vertex in the mesh is a 32-bit integer that serves two purposes. First, it can flag whether the field data associated with the vertex is valid (1) or invalid (0). For example, the Rotor data (see Figure 2) has regular meshes with vertices that are located within the rotor blade; those vertices have 0 IBLANKs. The second function of IBLANKs is to indicate mesh overlaps in multi-zone data, using negative integers. The overlap information is crucial for efficient point location in multi-zone data.

In time-varying data sets, the IBLANKs for a given zone may or may not vary over time. The IBLANKs for a given zone may be unique for each time step, or they may not vary at all. Or, there may be a small set of IBLANK values that are used over the time series. We have seen each of these cases for different zones of the Turbo data set, described below. Identifying zones that have a small set of IBLANK values is useful because our out-of-core system caches data in a memory pool. If only a small set of IBLANK values are used, then it is likely that the IBLANKs from a previous time step are still in the memory pool, and do not need to be read from disk.

Our algorithm finds zones that have identical IBLANKs. It compares the IBLANKs of every zone of every time step with the IBLANKs of all the other zones and time steps. The algorithm performs the comparisons efficiently by using a two pass algorithm. In the first pass, a hash value is computed for each zone of each time step. The hash value is the Cyclical Redundancy Check (CRC) value computed over the IBLANK data. Hashes of zones with identical IBLANKs will be the same, and otherwise will nearly always be different. We use the same CRC-32 algorithm [Peterson and E. J. Weldon 1972] that is used for Ethernet packets. Once the hash values are computed, the algorithm sorts the zones by hash value and identifies matching zones (ones having the same CRC value and dimensions). The final step is to verify that the zones are indeed identical by comparing the IBLANK values.

## 4 Data Model

The data model used for our experiments is called *Field Model (FM)*. Like its predecessor FEL [Moran et al. 2000], *FM* is object-oriented, written in templated C++, with out-of-core paging functionality [Cox and Ellsworth 1997] and a consistent, demand-driven evaluation philosophy [Moran and Henze 1999]. Both FEL and *FM* provide data access via `at_cell` calls. *FM* goes further in terms of generality: *FM* can handle mesh and field objects in spaces other than  $\mathbf{R}^3$  as well as "cell-centered" data. Like FEL, *FM* strives to provide flexibility while still maintaining performance. *FM* also provides optimized classes for particular mesh types, such as the regular and cylindrical meshes used in our experiments. A broader overview of *FM* can be found in a previous technical report [Moran 2001]. *Field Model* is an Open Source project [*Field Model* 2003].

Transformed meshes in *FM* are constructed with a reference mesh and a transformation  $T$ .  $T$  defines the transformation from the native coordinate system of the reference mesh to the new coordinate system emulated by the transformed object. For example, a transformed mesh could apply a 30 degree rotation about the X-axis to its reference mesh. Requests for coordinates are forwarded through the transformed mesh to the reference object, and the transformation  $T$  is applied to the reference mesh results. Point location requests are satisfied by applying the inverse transformation  $T^{-1}$  to the query point in order to place it in the native coordinate system of the reference mesh, and then calling the point location routine of the reference mesh. The inverse transformation is applied just once per point location request; the cost of applying the transformation is usually small compared to the cost of point location.

*Field Model* is organized as a set of modules. The central module, *FM*, provides standard classes shared by all modules. Additional modules provide functionality specific to various file formats and data standards. The PLOT3D [Walatka et al. 1992] module is of particular interest here since our experimental data sets are in that format.

## 5 Results

We have implemented the regularity discovery algorithm using *FM* [Moran 2001]. The results for three sample large data sets indicate that the regularities provide opportunities to reduce mesh storage requirements by factors of 7.5 to 600. We have also implemented a visualization system that exploits most of the regularities that were discovered, and measured its performance when computing some simple visualizations of the three data sets. The timings show that by exploiting the regularities we can reduce the visualization computation time by 35 to 50%.

### 5.1 Experiment Overview

We used three fairly large curvilinear CFD simulations as example data sets. One data set is static and two vary with time. Table 1 shows statistics about the data sets; more details are below. The visualization for each data set contains scalar-mapped surfaces showing the object that was simulated along with streaklines for the time-varying data sets and streamlines for the steady data set. A fourth-order Runge-Kutta algorithm implementation was used for the particle advection. The visualization algorithms were provided by the VisTech library [Shen et al. 1999b].

The visualizations were computed as a batch process using a single processor of a Dell Precision 530 workstation with two 2 GHz Pentium Xeon processors, 4 GiB of memory, and with the Rotor and Delta data stored on three striped 73 GB 10000 RPM SCSI disks. The Turbo data was stored on a remote file server since it was too large to fit on local disk. It was accessed using the custom remote access protocol described in [Ellsworth 2001], but without the multi-threading features described in that paper. The file server had two 1 GHz Pentium III processors, 2 GiB of memory, and eight 120 GB 5400 RPM IDE disks combined into one logical volume using software RAID-5. The two systems were connected with Gigabit Ethernet. In order to provide consistent results, the disk cache on each system was flushed before each run by allocating nearly all of the system's memory and then randomly reading a portion of a large file.

Most of the grid and solution data was loaded on demand using the out-of-core paging system [Cox and Ellsworth 1997]. This gives better performance compared to completely loading each time step since only a small fraction of the data is used to compute the visualization. However, we completely loaded the first grid file of the time series to improve performance because the data would be used repeatedly as a reference for transformed zones, and completely-loaded data can be accessed with less overhead compared to data that is loaded on demand. The paging system's cache was set large enough to allow a time step's working set to easily fit.

Two of the data sets are PLOT3D multi-zone files, with IBLANKs [Walatka et al. 1992]. Since the IBLANK data cannot be modeled as part of a regular, cylindrical, or transformed zone, they must be read from the files. We use two techniques to reduce the amount of IBLANK data that must be read. The first technique takes advantage of the fact that most of the IBLANKs have a value of 1, indicating valid data. When the IBLANK data come from out-of-core paged files, we losslessly compress the all-1 regions on a block-by-block basis. If all of the IBLANKs in a  $8 \times 8 \times 8$  data block are equal to 1, that block is not stored in the paged file, and that fact is recorded in the file. When that block is needed during

Statistic	Rotor	Delta	Turbo
Zones	210	1	24
Vertices (millions)	57.7	0.69	19.5
Number of Time Steps	1	600	300
Data Set Size (GiB)	1.94	12.3	196.8

Table 1: Data set statistics.

Statistic	Rotor	Delta		Turbo	
		first file	files 2-600	first file	files 2-300
Regular Zones	206	0	0	0	0
Regular Vertices	97.6%	0%	0%	0%	0%
Cylindrical Zones	0	0	0	2	2
Cylindrical Vertices	0%	0%	0%	12.4%	12.4%
Transformed Zones	0	0	1	12	22
Transformed Vertices	0%	0%	100%	45.8%	87.6%
Unmatched Zones	4	1	0	10	0
Unmatched Vertices	2.4%	100%	0%	41.8%	0%
IBLANK Compression	55.1%	—	—	—	43.8%
IBLANK Duplication	—	—	—	—	82.9%
IBLANK Comp. & Dupl.	55.1%	—	—	—	91.3%

Table 2: Results from the discovery algorithm.

Statistic	Rotor (MiB)	Delta (MiB)	Turbo (GiB)
Original Mesh Size	883	4710	87.5
Original Mesh Read	56.9	1880	16.1
Optimized Mesh Size	117	7.85	2.19
Optimized Mesh Read	15.7	7.85	1.35
Solution Size	1104	7870	110
Solution Read	33.4	2470	16.3

Table 3: Sizes and amount read for the three data sets. Includes statistics for the original mesh and solution files, and for the optimized versions of the meshes.

the visualization run, the block of 1 values is not read from disk; instead, the internal block pointer is set to point to a constant block filled with 1's. We have explored compressing constant blocks with values other than 1, but the additional compression was not significant.

The second IBLANK reduction technique exploits the duplicate IBLANK zones found during the regularity discovery process (see Section 3.5). We minimize the amount of IBLANK data read from disk by creating data structures to use the first time step's IBLANK data for the matching data in all subsequent time steps. This reduces the amount of data that is read because the out-of-core algorithm caches file data in a memory pool. The IBLANK data from the earlier time step is likely to still be in the pool when working on a later time step.

### 5.2 Results Overview

Table 2 shows the results of the regularity discovery algorithm. It shows the number of zones matched for each regularity type as well as the fraction of vertices in those zones. The time-varying data sets have separate columns for subsequent time steps since those time steps can reference data from the first time step. The last three lines show the results for optimizing the IBLANK data. "IBLANK Compression" gives the percentage of IBLANKs that fall into all-1 blocks, and "IBLANK Duplication" gives the percentage of IBLANKs that are redundant because a zone in an earlier time step has the same values. The last line gives the percentage of

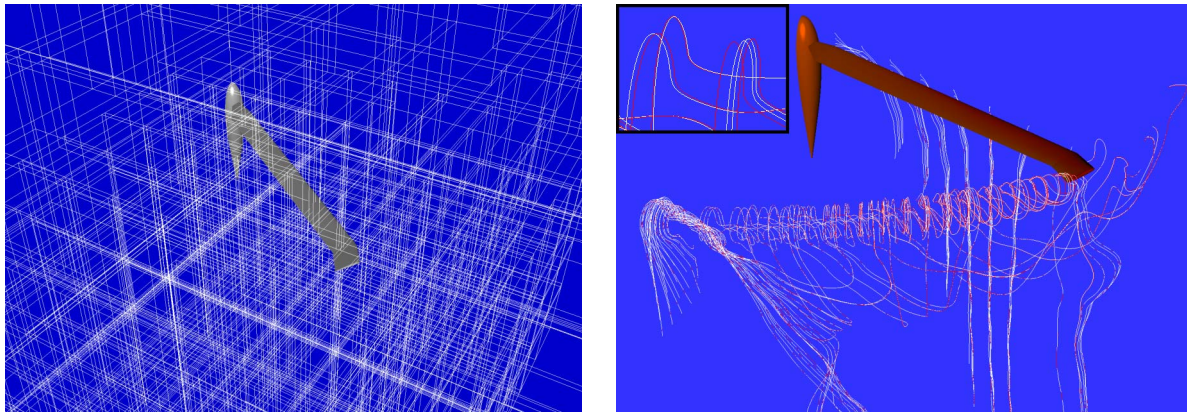


Figure 2: The Rotor data set: The data consist of 210 zones, 206 of which are regular meshes; the remaining 4 are curvilinear and surround the blade, blade ends, and hub. The regular zones were detected and replaced with optimized objects in order to accelerate the visualization computation. In the image to the left the bounding boxes of the regular mesh zones are visible; to the right the visualization displays the rotor surface color-mapped with pressure and streamlines. The white streamlines were computed with the original zones, the red with the optimized regular zones. Note that the red and white streamlines are very close, but as is apparent in the inset close-up, not exactly the same.

IBLANKs that do not need to be stored when both techniques are used.

When the regularity discovery algorithm is run on a Dell Precision 530 Workstation on a single 2 GHz processor, the discovery algorithm takes, per mesh file, 35 seconds for Rotor, 1 second for Delta, and 60 seconds for Turbo. The regularities were matched with tolerances of  $1 \times 10^{-8}$ ,  $1 \times 10^{-5}$ , and  $2 \times 10^{-6}$  for the three data sets, respectively. These tolerances mean that the vertices have an absolute error of less than the tolerance multiplied by the size of the zone's bounding box. The tolerance with Rotor was small because calculating vertices for regular grids does not accumulate much floating-point error. The tolerance for Delta was large because it was hard to recognize that the smallest rotation, 0.0046 degrees, was actually a rotation and not a general transformation.

Table 3 shows how the amount of mesh data that is potentially accessed is reduced by using the optimized meshes, and shows the extent of the actual reduction in the amount of data that is read. Since an out-of-core paging algorithm is used, only a small fraction of the original mesh, optimized mesh, and solution are read because the paging algorithm only reads blocks on demand, as needed by the visualization algorithms. The original data numbers do not use either type of IBLANK compression.

### 5.3 Rotor

The first data set, Rotor, is a steady-state rotorcraft simulation [Strawn and Djomehri 2002] (see Figure 2). The mesh has 210 zones, of which only the 4 immediately surrounding the rotor require curvilinear representation, while the rest can be represented by regular mesh objects. By replacing the regular zones with *FM* analytic mesh objects, at most only 2.4% of the vertices may need to be read during visualization (see Table 2). Furthermore, about half of the IBLANKs are in all-1 blocks, thus the optimized mesh size is only 117 MiB, 13% of the original mesh (see Table 3). With the optimized mesh objects, the visualization calculation reads a much smaller amount of data from disk: 15.7 MiB instead of 56.9 MiB. The decrease in the amount of geometry and IBLANK data loaded contributes to the visualization calculation time decrease from 40.8 seconds using the original data to 26.5 seconds, a reduction of 35%. This reduction is smaller than the decrease in mesh data read (from 56.9 MiB to 15.7 MiB, a reduction of 74%) because the amount of solution data read, 33.4 MiB, does not change.

We ran the rotor visualization a second time without out-of-core

objects to distinguish between the performance gains due to reading less data from disk and the gains due to the optimizations provided by the regular mesh replacement objects. Comparing the execution times after the data were completely loaded in memory, we found a 12% speed up with the replacement zones. This confirms that replacement regular meshes provide significantly faster point location and interpolation routines, but the overall gain is limited by the factors that did not change between the two runs: IBLANK access, field data access, and visualization algorithm calculation.

Figure 2 illustrates a case where we see a difference in the visualization due to the replacement of the original curvilinear meshes with regular meshes. The vertex geometry errors due to our replacement meshes were very small: at most  $10^{-8}$  times the size of the bounding box of the zone that contains the vertex. When rendering surfaces of transformed objects, the perturbations in the coordinates were not perceptible in the visualization. We did observe minor differences in the computed stream and streak lines, as seen in Figure 2. These differences occur because the particle integration accumulates the small vertex errors as the particle is advanced. The errors are only noticeable after thousands of integration steps, and the difference is still not very large. In practice, stream and streak line advection tends to be sensitive to many things: the choice of integration algorithm, the accuracy of the interpolation techniques, and so on. Users of advection visualization techniques are accustomed to using them to answer qualitative questions about the flow rather than specific "does a massless particle advect exactly from *A* to *B*?". Thus we concluded that the perturbations were acceptable. We saw similarly small changes in the locations of particles in the visualizations of the other two data sets.

### 5.4 Delta

The second data set, Delta [Chaderjian and Schiff 1996], is the classic rolling delta wing data used to test large data visualization, and has appeared in numerous previous visualization studies. Figure 3 provides a representative image from the visualization. Mesh motion in the original data was represented by 600 mesh files, even though the motion was simply rigid body rotation of a single curvilinear mesh. Our regularity detection algorithm successfully recovered the rotations. Given the rotations, we replaced the 600 time steps with a dynamically transformed mesh object based on the first time step. This replacement effectively gave us a compression ratio of 600 to 1. The total amount of geometry data actually read

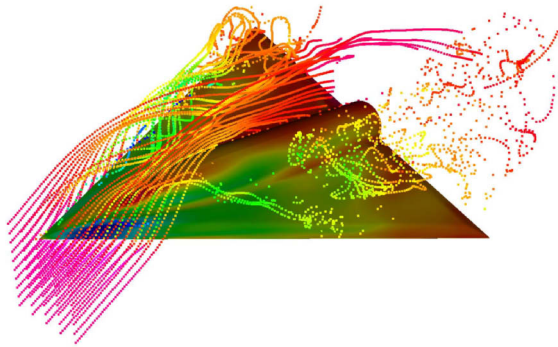


Figure 3: A snapshot from the visualization of the Delta data set. The wing surfaces and particles are color mapped with pressure.

decreased by a factor of 240 (from 1880 with out-of-core paging of the original time-series geometry to 7.85 MiB with the dynamically transformed object). Due to this reduction in I/O, the time to compute the visualization decreased from 32.7 to 17.7 minutes, a reduction of 46%. Like the Rotor data set, the reduction in time is limited by the unchanged large amount of solution data that must be read, as shown in Table 3.

## 5.5 Turbo

The third data set, Turbo, is a turbo-pump simulation. It is the most challenging of the three data sets because it is both multi-zoned and time-varying. We regret that restrictions placed on the data prevent us from showing images of this data set, but it has some similarities to the turbo-pump data set shown in Figure 1. The discovery algorithm finds that the first zone has several blades that are identical except for rotation, and that there are two cylindrical zones; see Table 2. It also finds that all of the non-cylindrical zones in subsequent time steps are transformed versions of zones in the first time step. Finally, the algorithm finds that the IBLANK data can be reduced significantly by using both all-1 block elimination, which would reduce the amount of IBLANK data by 43.8%, and by sharing IBLANK data across time steps, which would result in a 82.9% savings. Taken together, the amount of IBLANK data can be reduced by 91.3%.

Our optimized replacement mesh reads the first time step completely so that the overhead of the demand paging code is avoided, at the cost of additional memory usage. One of the two cylindrical meshes is replaced with an analytic mesh. The second cylindrical mesh uses curvilinear data because the mesh rotates over time. Most zones in subsequent time steps use *FM* objects that transform data from the corresponding zone in the first time step. The exception is the single non-rotating cylindrical zone, which is modeled analytically.

The replacement meshes reduce amount of mesh data that can potentially be read from the original 87.4 GiB to 2.19 GiB, a 40 to 1 reduction (see Table 3). The amount of mesh data that is actually read decreases from 16.1 GiB (for out-of-core paging of the original time-series data) to 1.35 GiB, a 12 to 1 reduction. The latter reduction ratio is smaller than the first because the reduced data is used by computations for multiple time steps, which in turn causes the demand paging code to load the union of the data needed by the time step computations.

When the mesh data were read from the original paged files, the computation required 233 minutes. The time was cut in half, to 116 minutes, when the cylindrical and transformed zones were used. Like the other data sets, the reduction in time is much less than the

Result	Streaklines	Surfaces	Both
Original Mesh Size	87.5	87.5	87.5
Original Mesh Read	4.80	12.5	16.1
Optimized Mesh Size	2.19	2.19	2.19
Optimized Mesh Read	0.454	1.32	1.35
Solution Size	110	110	110
Solution Read	1.00	15.6	16.3

Table 4: Size and amount read for three visualizations of the Turbo data set, in GiB.

Mesh Type	Streaklines	Surfaces	Both
Original mesh	94.5	149	233
Optimized mesh	20.3	96	116
Time reduction	79%	35%	50%

Table 5: Visualization computation times for three visualizations of the Turbo data set, in minutes.

reduction in mesh data due to the large unchangeable amount of required solution data.

There are further regularity opportunities with the Turbo data which have not been exploited yet. For simplicity in the initial tests each blade was represented by a transformed version of itself from the initial time step. Further reductions could be achieved by taking advantage of the fact that a single blade in the initial data could be used as the reference both for itself and for other blades. Creating transformed meshes, each with its own transformation but based on the same reference, would further slightly reduce the amount of geometry data that would have to be accessed, and would noticeably reduce the amount of data held in main memory (by about 100 MiB).

## 5.6 Effect of Visualization Type

The amount of analysis speedup depends on the data access patterns of the visualization techniques. As an example, we ran three different variations of the Turbo data set visualization: one having only the rake of streaklines, a second having only the pressure-mapped surfaces of the body, and a third having both the streaklines and surfaces, as before. The timing results are shown in Table 5. The streaklines visualization calculation was over four times faster when using the optimized mesh instead of the original mesh, but the surfaces-only visualization calculation took only 35% less time with the optimized mesh. The results when both streaklines and surfaces were calculated were in the middle.

The variation in the visualization speedups result from the variation in the amounts of data that are loaded (see Table 4). Statistics from runs not shown here indicate that the surface visualization run did not benefit from any all-1 IBLANK block compression, while the streaklines run had significant compression. This is because the surfaces were largely on zone boundaries, while the streakline particles were located throughout the zones. Vertices on zone boundaries often have negative IBLANK values indicating overlap with another zone. In addition, the streaklines visualization is more geometry and IBLANK intensive because point location sometimes requires searching over many zones. Our optimizations tend to most benefit such algorithms. Surface visualization in contrast requires no point location.

## 6 Conclusion

We have presented a discovery algorithm for identifying regularities in large, curvilinear data sets. We then show how we can ex-

plot these regularities within a data model that provides optimized regular and cylindrical meshes as well as dynamically transformed meshes. Using our techniques we were able to reduce the visualization times by 35%, 46% and 50% for the Rotor, Delta, and Turbo data sets, respectively. Mesh storage requirements decreased, respectively, by factors of 7.5, 600, and 40. We expect that the same techniques could be applied to many other data, in particular other time-series data sets. Our data model makes it relatively easy to experiment with alternative “virtual mesh” objects since we can substitute for original mesh objects in our visualizations without modifying our visualization algorithms.

In the future we expect that the performance gains for data sets with IBLANKs could be further improved. Allocating 32 bits per vertex for the amount of information IBLANKs contain is known to be not particularly efficient. Using more sophisticated compression techniques, such as those proposed by Hultquist [1994], we anticipate that the performance of multi-zone data visualizations could be significantly improved.

A complete system for exploiting regularities would have 3 components: a regularity detector, a data model where we can seamlessly substitute optimized objects for the original curvilinear objects, and a metadata model that would contain the detection results and direct the construction of the optimized data objects. In this paper we have described the first two. Without the third, translating detection results into the code to construct optimized objects required manual intervention. We have taken the first steps toward a metadata model prototype. We expect that with the third component it will be relatively easy to experiment with applying our automatic optimization techniques to a variety of data sets.

## Acknowledgments

We could like to thank Roger Strawn, Neal Chaderjian, and Cetin Kiris for providing the data sets utilized in our experiments. This work was funded by the NASA Computing, Information, and Communications Technology (CICT) Program, partially via NASA contract DTTS59-99-D-00437/A61812D. Finally, we would like to thank VA Software for their ongoing support of the Open Source [Open Source 2003] software movement, and SourceForge in particular.

## References

ABRAM, G., AND TREINISH, L. 1995. An extended data-flow architecture for a data analysis and visualization. In *Proceedings of Visualization '95*, IEEE Computer Society Press, 263–270.

BUTLER, D. M., AND PENDLEY, M. H. 1989. A visualization model based on the mathematics of fiber bundles. *Computers in Physics* 3, 5 (Sep/Oct), 45–51.

CHADERJIAN, N. M., AND SCHIFF, L. B. 1996. Numerical simulation of forced and free-to-roll delta-wing oscillations. *Journal of Aircraft* 33, 1 (Jan/Feb), 93–99.

CHIANG, Y., AND SILVA, C. T. 1997. I/O optimal isosurface extraction. In *IEEE Visualization '97*, R. Yagel and H. Hagen, Eds., IEEE, 293–300.

CHIANG, Y.-J., SILVA, C. T., AND SCHROEDER, W. J. 1998. Interactive out-of-core isosurface extraction. *IEEE Visualization '98* (October), 167–174. ISBN 0-8186-9176-X.

COX, M., AND ELLSWORTH, D. 1997. Application-controlled demand paging for out-of-core visualization. In *Proceedings of Visualization '97*, IEEE Computer Society Press, 235–244.

ELLSWORTH, D. 2001. Accelerating demand paging for local and remote out-of-core visualization. Tech. Rep. NAS-01-004, NAS Division, NASA Ames Research Center, June.

*Field Model*, 2003. <http://field-model.sourceforge.net>.

HABER, R., LUCAS, B., AND COLLINS, N. 1991. A data model for scientific visualization with provisions for regular and irregular grids. In *Proceedings of Visualization '91*, 298–305.

HIBBARD, W., DYER, C., AND PAUL, B. 1994. A lattice data model for data display. In *Proceedings of Visualization '94*, 310–317.

HIBBARD, W. 1998. VisAD: Connecting people to computations and people to people. *Computer Graphics* 32, 3.

HULTQUIST, J. P. M. 1994. Improving the performance of particle tracing of curvilinear grids. Tech. rep., National Aeronautics and Space Administration. RNR-94-009.

ITOH, T., AND KOYAMADA, K. 1995. Automatic isosurface propagation using an extrema graph and sorted boundart cell lists. *IEEE Transactions on Visualization and Computer Graphics* 1, 4 (December), 319–327.

LANE, D. 1994. UFAT: A particle tracer for time-dependent flow fields. In *Proceedings of Visualization '94*, IEEE Computer Society Press, 257–264.

LAW, C., MARTIN, K., SCHROEDER, W., AND TEMKIN, J. 1999. A multi-threaded streaming pipeline architecture for large structured data sets. In *Proceedings of Visualization '99*, 225–232.

LUCAS, B., ET AL. 1992. An architecture for a scientific visualization system. In *Proceedings of Visualization '92*, IEEE Computer Society Press, 107–114.

MORAN, P., AND HENZE, C. 1999. Large data visualization with demand-driven calculation. In *Proceedings of Visualization '99*, IEEE Computer Society Press, 27–33.

MORAN, P., HENZE, C., AND ELLSWORTH, D. 2000. The FEL 2.2 user guide. Tech. rep., National Aeronautics and Space Administration. NAS-00-002.

MORAN, P. 2001. Field model: An object-oriented data model for fields. Tech. rep., National Aeronautics and Space Administration. NAS-01-005.

OPEN SOURCE, 2003. <http://www.opensource.org>.

PETERSON, W. W., AND E. J. WELDON, J. 1972. *Error Correcting Codes*. MIT Press, Cambridge, MA.

PIZER, S. M., AND WALLACE, V. L. 1983. *To Compute Numerically*. Little, Brown, and Company, Boston, MA.

PRESS, W. H., FLANNERY, B. P., TEUKOLSKY, S. A., AND VETTERLING, W. T. 1998. *Numerical Recipes in C*. Cambridge University Press, Cambridge, United Kingdom.

SCHROEDER, W., MARTIN, K., AND LORENSEN, B. 1997. *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*, second ed. Prentice-Hall Inc., New Jersey.

SHEN, H.-W., CHIANG, L.-J., AND MA, K.-L. 1999. A fast volume rendering algorithm for time-varying fields using a time-space partitioning (TSP) tree. In *Proceedings of Visualization '99*, IEEE Computer Society Press, 371–377.

SHEN, H.-W., SANDSTROM, T., KENWRIGHT, D., AND CHIANG, L.-J. 1999. *VisTech Library User and Programmer Guide*. National Aeronautics and Space Administration.

SHEN, H.-W. 1998. Isosurface extraction in time-varying fields using a temporal hierarchical index tree. In *Proceedings of Visualization '98*, IEEE Computer Society Press, 159–166.

STRAWN, R., AND DJOMEHRI, M. J. 2002. Computational modeling of hovering rotor and wake aerodynamics. *AIAA Journal of Aircraft* 39, 5 (September–October), 786–793.

SUTTON, P. M., AND HANSEN, C. D. 1999. Isosurface extraction in time-varying fields using a temporal branch-on-need tree (T-BON). *IEEE Visualization '99* (October), 147–154.

UENG, S., SIKORSKI, C., AND MA, K. 1997. Out-of-core streamline visualization on large unstructured meshes. *IEEE Transactions on Visualization and Computer Graphics* 3, 4 (December), 370–380.

WALATKA, P., BUNING, P., PIERCE, L., AND ELSON, P. 1992. *PLOT3D User's Manual*. National Aeronautics and Space Administration, July. NASA Technical Memorandum 101067.