Accelerating Large Data Analysis By Exploiting Regularities

Patrick J. Moran*
NASA Ames Research Center
patrick.j.moran@nasa.gov

David Ellsworth†
Advanced Management Technologies Incorporated
NASA Ames Research Center
ellswort@nas.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 to 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 significant speed-ups in analysis.

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 straight-forward 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 provides 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 [31] 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 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.

*Mail Stop T27A-2, Moffett Field, CA 94035
†Mail Stop T27A-1, Moffett Field, CA 94035

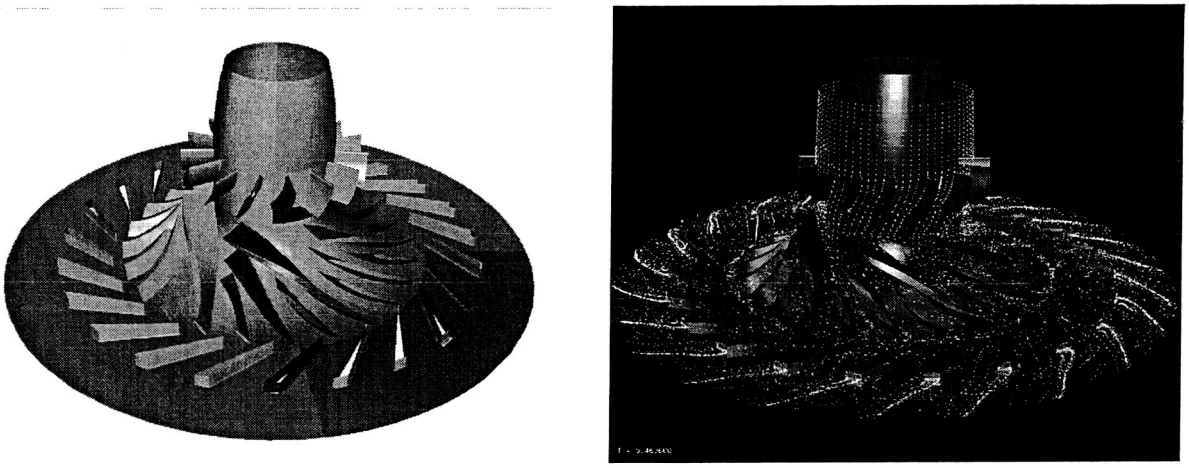

Figure 1: Two views of the same turbo-pump data set: to the left we highlight a subset of the 114 zones in the data set. Using the subset of zones as reference meshes, we can use transformed meshes to represent the remainder of the geometry. To the right is a visualization of the same data set. The number 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.

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 [15] uses 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 [6] describe a more general out-of-core paging system that exploited the fact that many visualization algorithms exhibit spatial and temporal locality in data access. An alternative to paging is the streaming approach described by Law *et al* [16].

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 [4, 29, 13, 5, 24]. An octree-based approach to limiting the amount of data loaded for streamline computation was described by Ueng *et al* [30]. Shen *et al* [25] describe a Time-Space Partitioning (TSP) tree approach 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 [2]. Their model was inspired 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 to access 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[17, 1]). Beginning with Haber et al [9], 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) [20]. FEL excelled with the 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 with far larger data sets, including a consistent demand-driven evaluation model [19] and the integration of demand-paging techniques [6]. FEL assumed that all objects were in $\mathbf{R}^3$ physical space and vertex-centered.

The Visualization Toolkit (vtk) [23], 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 [16] have focused on enabling the data model (and thus the whole system) to handle large data via streaming. Like FEL, vtk utilizes a demand-driven evaluation strategy.

VisAD [11, 10] 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 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 includes 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 runs the transformed mesh algorithm. For each of these zone types, the al-

gorithm first attempts to recover the parameters that describe the transformed or procedural 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.

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 [22, 21]. The errors come from three sources. One source is floating point errors introduced when the the vertices in the input files are 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 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 are 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 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 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 for recognizing transformed zones can either find zones that are transformed versions of ones in the same file, or zones found in a second reference file. The reference file is typically the first file in the time series. When looking at zones from a single file, the algorithm only considers lower-numbered zones, while it considers all of the zones from the reference file. The algorithm will find a match between the *current* zone in the current file, and a *matching* zone from either the current or first file, as appropriate. The recovered transformation is a general transformation: it is a a general 3×3 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 3×3 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 location 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 3×3 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 9×9×9 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.

Given the four vertices, the algorithm solves the system of equations using first LU decomposition and then iterative improvement [22] to reduce computation errors. Then, an error minimiza-

tion step further reduces the error using Polak-Rebiere conjugate gradient minimization [22]. 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 $9\times9\times9$ lattice that were considered as candidates for the four vertex pairs used in the direct solution step.

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 larger vertices, 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.

4 Data Model

The data model used for the experiments is called *Field Model* (*FM*). Like its predecessor FEL [20], *FM* is object-oriented, written in templated C++, with out-of-core paging functionality [6] and a consistent, demand-driven evaluation philosophy [19]. 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 [18]. *Field Model* is as an Open Source project [8].

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.

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)	1.61	12.3	187.2

Table 1: Data set statistics.

Result	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%	43.8%
Original Data Required	13.0%	100%	0%	45.4%	14.1%

Table 2: Data set statistics and results from the discovery algorithm. The last line shows the fraction of the original data that might need to be loaded as a fraction of the original data size, and includes the IBLANK portion (see text for details).

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 [31] module is of particular interest here since our experimental data sets are in that format. A PLOT3D data can include what are known as IBLANK's: 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, in the Rotor data (see Figure 2) vertices falling withing the rotor blade from regular meshes have their IBLANK values set to 0. The second function of IBLANK's 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.

5 Regularity Discovery Results

The regularity discovery algorithm has been implemented using *FM* [18]. The results from running the algorithm show that considerable data savings are possible, and are summarized in Table 2. One data set is static, two vary with time. All of the data sets are fairly large curvilinear CFD simulations; see Table 1.

Two of the data sets are PLOT3D multi-zone files, with IBLANK's [31]. Since the IBLANK data cannot be modeled as part of a regular, cylindrical, or transformed zone, they must be read from the files. However, most of the IBLANK's have a value of 1, indicating valid data. When the IBLANK data comes from out-of-core paged files, we losslessly compress the all-1 regions on a block-by-block basis. If all of the IBLANK's in a $8\times8\times8$ 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 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.

The first data set, Rotor, is a steady-state rotorcraft simulation. Most of the vertices are in regular grids because it simplifies the computation and also eases running the CFD algorithm on multiple processors, as shown in Figure 2. Only a few zones near the rotor

blade contain curvilinear data, so that only 2.4% of the vertices need to be read during visualization. Since about half of the IBLANK's can be compressed, at most 13% of the original grid file would need to be read.

The second data set, Delta, is a simulation of a delta wing rolling left and right, and is shown in Figure 4. This is a single zone data set without any regular or cylindrical zones. This means that all of the vertices from the first file may need to be read. However, vertices from subsequent time steps can be be computed by rotating vertices from the first time step. Since this data set does not have IBLANK information, only the first time step is needed, which results in a 600 to 1 compression ratio over the entire time series.

The third data set, Turbo, is a turbo-pump simulation. We regret that we cannot show 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 there are several blades that are identical except for rotation, and that there are two cylindrical zones. Since only the first blade's geometry needs to be read, and the cylindrical zones do not need to be read, only 41.8% of the vertices need to be read. The IBLANK data can be compressed by 43.8% (leaving 56.2%), which means that, at most, 45.5% of the first file's data may need to be read.

Because portions of the turbo-pump are either static or rotate over time, subsequent timesteps do not need any vertex data to be read from the mesh files. Instead, either vertices from the first file (possibly rotated) or vertices generated from a cylinder model can be used to produce visualizations. Only the incompressible portions of the IBLANK data may need to be read from the mesh file, reducing the data requirements to 14.1% of the original file data. This is a large reduction in data size since there are currently 300 files in this data set. The original mesh files have a total size of 87 GB, while the compressed IBLANK data has only 8.5 GB — a savings of over 90%.

When the algorithm is run on a Dell Precision 530 Workstation on a single 2 GHz processor, the discovery algorithm takes 1 second per Delta mesh file, and 35 seconds per Rotor or Turbo mesh file.

6 Visualization Computation Results

While the savings of disk and main memory space can be important, increasing the performance of the visualization computation is also important. We have implemented a visualization system that exploits most of the regularities that were discussed in the previous section, and measured its performance when computing a simple visualization of each data set. Each visualization contains scalar-mapped surfaces showing the object being simulated along with streaklines for the time-varying data sets, and with streamlines for the steady data set. The visualization algorithms were provided by the VisTech library [27].

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 GB of memory, and with the Rotor and Delta data stored on three striped 73 GB 10000 RPM SSCI 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 [7], but without the multi-threading features described in that paper. The file server had two 1 GHz Pentium III processors, 2 GB 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 [6]. 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.

6.1 Rotor

The largest reduction in time was observed with the Rotor data set (see Figure 2). The time decreased from 47 seconds using the original data to 22 seconds using the replacement regular meshes. This large time reduction occurred because there was a reduction in both data loading and computation. The data loading reduction occurred since the regular mesh vertex data (which represented 97.6% of the total number of vertices) did not need to be loaded. The computation reduction occurred because it is much simpler to perform point location and interpolation in regular meshes than curvilinear meshes.

Figure 3 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 0.0001% of the size of the overall mesh. 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 3. 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.

6.2 Delta

The rolling delta wing data set [3] is a classic in large data visualization and has appeared in numerous previous visualization studies [14, 26, 24, 25, 19]. Mesh motion in the original data was represented by 600 mesh files, even though the motion is simply rigid body rotation. Here we obviate the need for reading all the steps through the use of a a dynamically transformed mesh. Using the transformed object the time to compute the visualization (shown in Figure 4) decreased from 31.3 to 18.3 minutes, a reduction of 42%. This reduction is larger than for the Turbo data set described below because the Delta data set does require IBLANK information; disk seeks for mesh data after the first time step were completely eliminated.

6.3 Turbo

The turbo-pump data set is the most challenging of the three, since it is both multi-zoned and time-varying. When the mesh data was read from the original paged files, the computation required 198 minutes. The time decreased to 162 minutes when the cylindrical and transformed zones were used, a reduction of 18%. The reduction is less than one might expect, given that only 14% of the data from the original mesh files are needed after the first time step (see Table 2). The initial indications are that the reduced time was due to the reduction of data loaded from the remote file server. The time reduction was only 18% since both runs loaded the same amount of solution and IBLANK data, and because that data was larger than

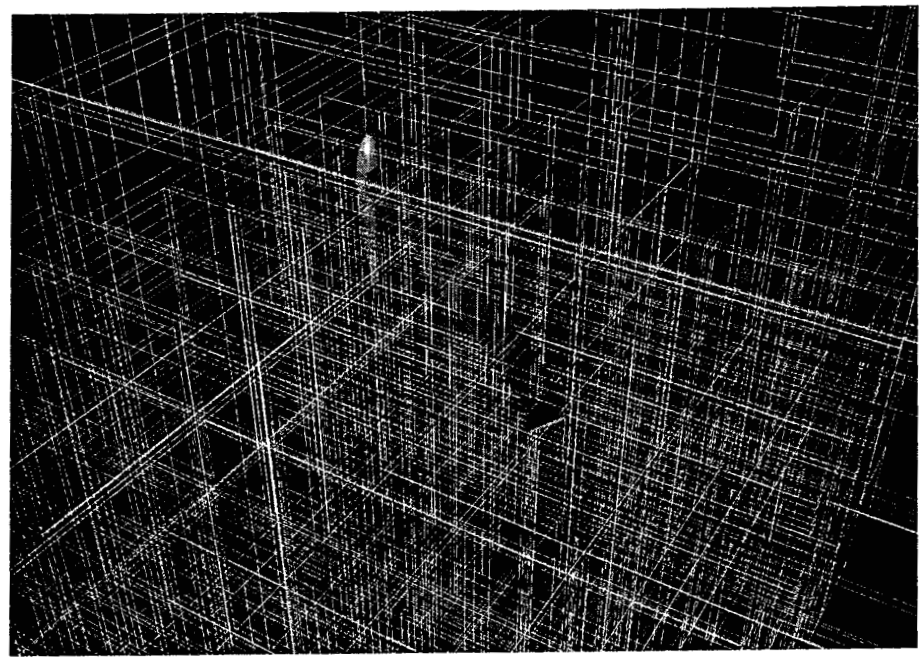

Figure 2: A close-up of the Rotor data set: The mesh consists of 210 zones, 206 of which are regular meshes. The remaining 4 zones are curvilinear and surround the blade, blade end, and central hub.

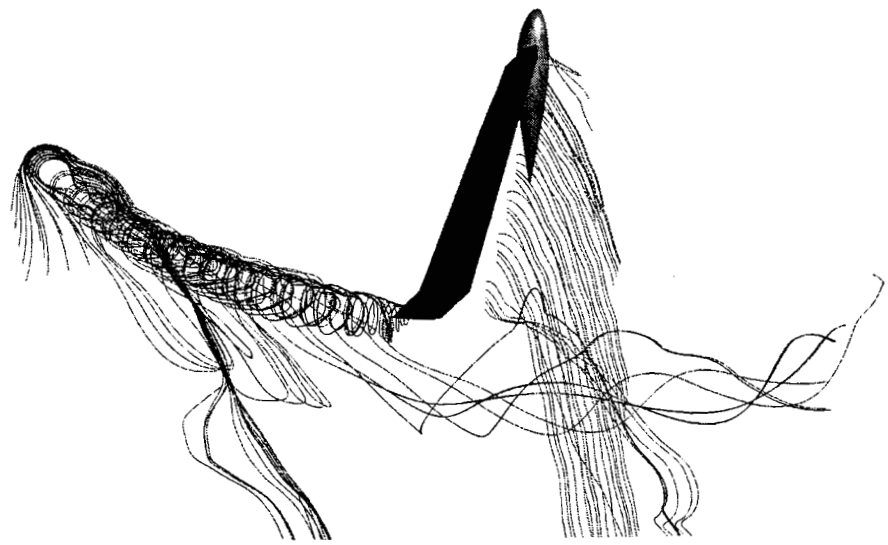

Figure 3: An illustration of how streamlines can diverge when the original submeshes are replaced by regular meshes.

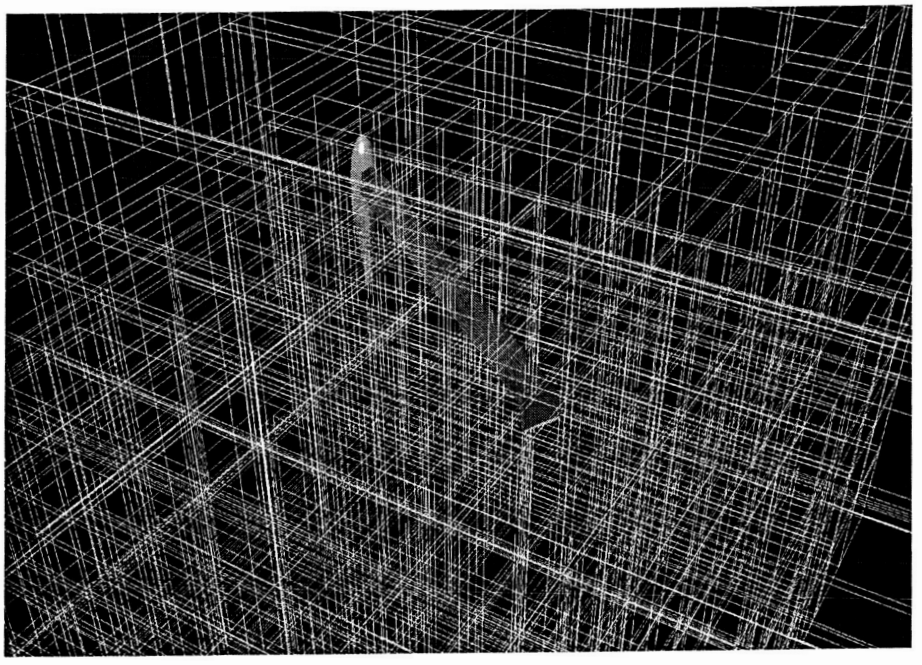

Figure 2: A close-up of the Rotor data set: The mesh consists of 210 zones, 206 of which are regular meshes. The remaining 4 zones are curvilinear and surround the blade, blade end, and central hub.

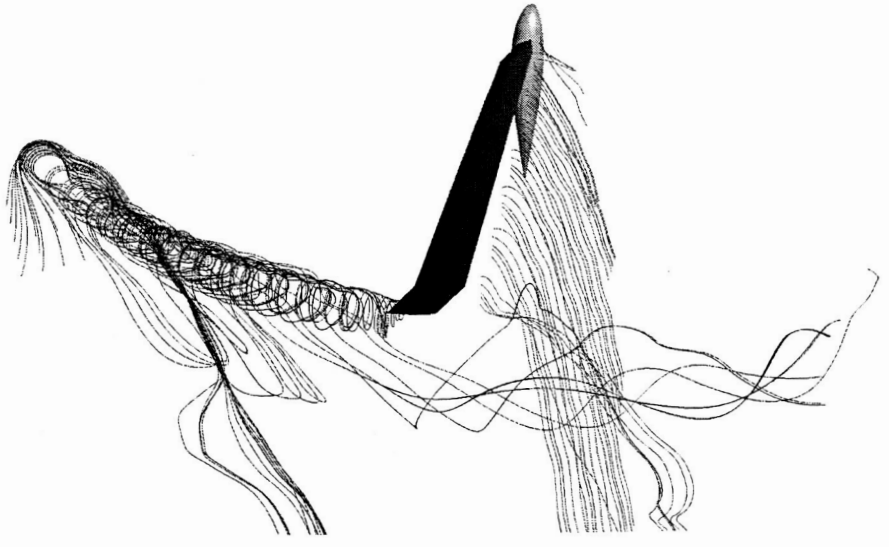

Figure 3: An illustration of how streamlines can diverge when the original submeshes are replaced by regular meshes.

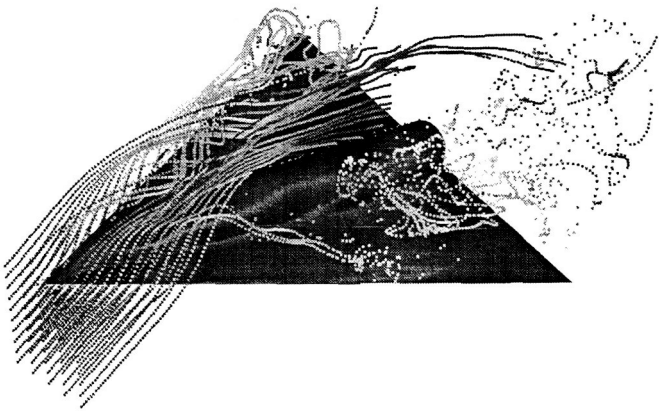

Figure 4: A snapshot from the rolling delta wing simulation: The wing surfaces and particles are color mapped with pressure.

the loaded mesh data. In addition, we believe that the time reduction was small due to file layout. Because vertex and IBLANK data blocks are adjacent in the disk file, the time to load the vertex and IBLANK data together is not much more than the time to load only the IBLANK data since both sets of data can be loaded using a single disk seek.

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 reduce the amount of geometry data that would have to be accessed.

7 Conclusion

We have presented a discovery algorithm for identifying regularities in large, curvilinear data sets. We then show how we can exploit 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 53%, 42% and 18% for the rotor, rolling delta wing, and turbo-pump data sets, respectively. We expect that the same techniques could be applied to many other data, in particular other time-series data sets. Our data model makes 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 IBLANK's could be further improved. Allocating 32 bits per vertex for the amount of information IBLANK's contain is known to be not particularly efficient. Using more sophisticated compression techniques, such as those proposed by Hultquist [12], we anticipate that the performance of multi-zone data visualizations could be significantly improved. With time-series data in particular, there are opportunities to replace IBLANK's in regions that do not vary with time with a more efficient representation. Another possibility would be to replace solution data at IBLANK 0 (invalid) vertices with a special value, such as a NaN. With this replacement invalid solution data could be detected without having to consult IBLANK's at all. With the turbo-pump, for example, this technique would speed up the scalar mapping on the turbo-pump blades.

Acknowledgements

We could like to thank Roger Strawn, Neal Chaderjian, and Cetin Kiris for providing the data sets utilized for 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 [28] software movement, and SourceForge in particular.

References

[1] G. Abram and L. Treinish. An extended data-flow architecture for a data analysis and visualization. In *Proceedings of Visualization '95*, pages 263–270. IEEE Computer Society Press, 1995.

[2] D. M. Butler and M. H. Pendley. A visualization model based on the mathematics of fiber bundles. *Computers in Physics*, 3(5):45–51, Sep/Oct 1989.

[3] N. M. Chaderjian and L. B. Schiff. Numerical simulation of forced and free-to-roll delta-wing oscillations. *Journal of Aircraft*, 33(1):93–99, Jan/Feb 1996.

[4] Y.-J. Chiang and C. T. Silva. I/O optimal isosurface extraction. In Roni Yagel and Hans Hagen, editors, *IEEE Visualization '97*, pages 293–300. IEEE, November 1997.

[5] Y.-J. Chiang, C. T. Silva, and W. J. Schroeder. Interactive out-of-core isosurface extraction. *IEEE Visualization '98*, pages 167–174, October 1998. ISBN 0-8186-9176-X.

[6] M. Cox and D. Ellsworth. Application-controlled demand paging for out-of-core visualization. In *Proceedings of Visualization '97*, pages 235–244. IEEE Computer Society Press, October 1997.

[7] D. Ellsworth. Accelerating demand paging for local and remote out-of-core visualization. Technical Report NAS-01-004, NAS Division, NASA Ames Research Center, June 2001.

[8] *Field Model.* `http://field-model.sourceforge.net`.

[9] R. Haber, B. Lucas, and N. Collins. A data model for scientific visualization with provisions for regular and irregular grids. In *Proceedings of Visualization '91*, pages 298–305, October 1991.

[10] W. Hibbard. VisAD: Connecting people to computations and people to people. *Computer Graphics*, 32(3), 1998.

[11] W. Hibbard, C.R. Dyer, and B. Paul. A lattice data model for data display. In *Proceedings of Visualization '94*, pages 310–317, October 1994.

[12] J. P. M. Hultquist. Improving the performance of particle tracing of curvilinear grids. Technical report, National Aeronautics and Space Administration, 1994. RNR-94-009.

[13] T. Itoh and K. Koyamada. Automatic isosurface propagation using an extrema graph and sorted boundart cell lists. *IEEE Transactions on Visualization and Computer Graphics*, 1(4):319–327, December 1995.

[14] D. Kenwright and R. Haimes. Vortex identification – applications in aerodynamics: A case study. In *Proceedings of Visualization 1997*, pages 413–416. IEEE Computer Society Press, October 1997.

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

[16] C. Law, K. Martin, W. Schroeder, and J. Temkin. A multithreaded streaming pipeline architecture for large structured data sets. In *Proceedings of Visualization '99*, pages 225–232, October 1999.

[17] B. Lucas et al. An architecture for a scientific visualization system. In *Proceedings of Visualization '92*, pages 107–114. IEEE Computer Society Press, 1992.

[18] P. Moran. Field model: An object-oriented data model for fields. Technical report, National Aeronautics and Space Administration, 2001. NAS-01-005.

[19] P. Moran and C. Henze. Large data visualization with demand-driven calculation. In *Proceedings of Visualization '99*, pages 27–33. IEEE Computer Society Press, October 1999.

[20] P. Moran, C. Henze, and D. Ellsworth. The FEL 2.2 user guide. Technical report, National Aeronautics and Space Administration, 2000. NAS-00-002.

[21] Stephen M. Pizer and Victor L. Wallace. *To Compute Numerically.* Little, Brown, and Company, Boston, MA, 1983.

[22] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C.* Cambridge University Press, Cambridge, United Kingdom, 1998.

[23] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics.* Prentice-Hall Inc., New Jersey, second edition, 1997.

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

[25] H.-W. Shen, L.-J. Chiang, and K.-L. Ma. A fast volume rendering algorithm for time-varying fields using a time-space partitioning (TSP) tree. In *Proceedings of Visualization '99*, pages 371–377. IEEE Computer Society Press, October 1999.

[26] H.-W. Shen and D. Kao. UFLIC: A line integral convolution algorithm for visualizing unsteady flows. In *Proceedings of Visualization '97*, pages 317–322. IEEE Computer Society Press, October 1997.

[27] H.-W. Shen, T. Sandstrom, D. Kenwright, and L.-J. Chiang. *VisTech Library User and Programmer Guide.* National Aeronautics and Space Administration, 1999.

[28] Open Source. `http://www.opensource.org`.

[29] P. M. Sutton and C. D. Hansen. Isosurface extraction in time-varying fields using a temporal branch-on-need tree (T-BON). *IEEE Visualization '99*, pages 147–154, October 1999.

[30] S.-K. Ueng, C. Sikorski, and K.-L. Ma. Out-of-core streamline visualization on large unstructured meshes. *IEEE Transactions on Visualization and Computer Graphics*, 3(4):370–380, December 1997.

[31] P. Walatka, P. Buning, L. Pierce, and P. Elson. *PLOT3D User's Manual.* National Aeronautics and Space Administration, July 1992. NASA Technical Memorandum 101067.