Hierarchical, Wavelet-based Data Structures for Structured Grids

Joerg Meyer

University of California, Irvine
Department of Electrical Engineering and Computer Science
644E Engineering Tower
Irvine, CA 92697-2625
jmeyer@uci.edu

Abstract

Multi-dimensional, regular, structured grids can be represented in the form of hyper-cubes, i.e., vector-valued meshes of multiple dimensions. Volumetric data sets and time-varying volumetric data sets, as they occur in medical or geo-spatial applications, are special three- and four-dimensional cases. The storage space requirements of such higher-dimensional data sets can be quite significant. In order to store and access such data sets efficiently, the spatial or temporal coherence in each dimension must be analyzed and utilized.

Introduction

Structured grids are generated in many areas of simulation, for example, in finite element simulations or in field simulations. Structured grids also occur in medical applications, such as volumetric scans. Especially in medical scans, grids are usually regular and recently became multi-dimensional (multiple scan values or gradient- and tensor-based methods). Due to the complexity of such multi-dimensional, regular, structured grids, real-time image processing, storage, transmission and rendering become more challenging.

We present a novel technique for efficient compression, storage and access of multi-dimensional, regular, hyper-cube grids using wavelet-based decomposition and reconstruction methods. Due to their potential for an efficient implementation using integer arithmetic, Haar wavelets are employed by this algorithm. The method is asymmetric, which means that conversion of the data set into a wavelet representation and data compression have a higher complexity and therefore take more time than uncompressing and reconstructing the data. This is a desired property in many rendering applications.

It will be shown how the algorithm can be easily extended to higher-dimensional or vector-valued data sets. The algorithm becomes more efficient with increasing

dimensions, because the spatial or temporal coherence within each dimension contributes to the data reduction in the wavelet representation and the compression step.
The method is suitable for interactive rendering applications, because it enables fast extraction of low-frequency subsets, which can be rendered as low-resolution preview images. By preserving the spatial and temporal information of the data even during the compression step, the method also allows for gradual refinement of subvolumes or the entire data set, and instant multi-resolution region-of-interest selection, gradient extraction, slicing, cutting and down-projection.

Wavelet Decomposition for Hypercubic Meshes

Since we want to access the data set in a hierarchical fashion, we have to convert it into a multiresolution representation. This representation must be chosen in a way that the reconstruction can be performed most efficiently with minimal computational effort [1]. Haar wavelets fulfill these properties. They also have the advantage that they can be easily implemented as integer arithmetic. The Haar Wavelet scheme is described in [2] and shown in figure 1 (*n* indicates the compression level as a power of two).

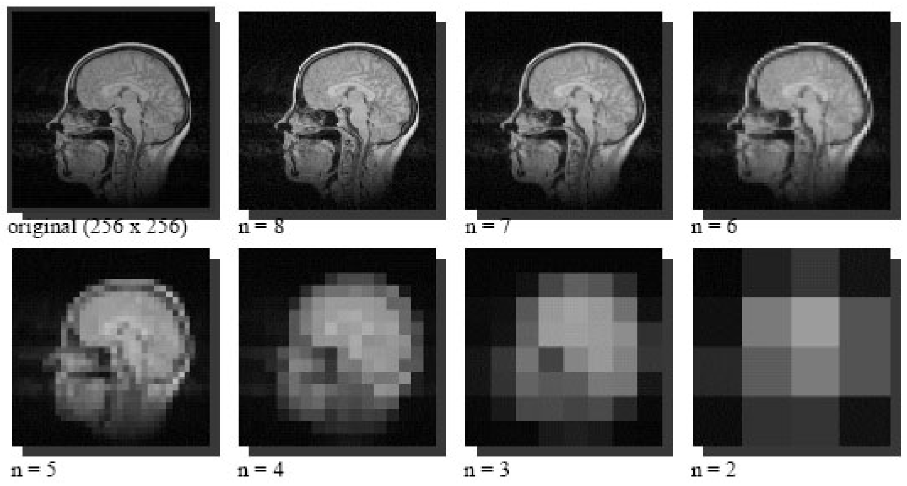

Figure 1. Haar wavelet compression scheme (2-D).

Hypercubes

The wavelet scheme can be easily extended into higher dimensions. One complete pass of the non-standard decomposition scheme comprises of iteration through all spatial dimensions. In the 2-D case shown in figure 1, the decomposition is first executed in the x direction, and then in the y direction. For higher-dimensional cases, the cycle must be completed for all dimensions before moving on to the next level of detail.

For vector-valued data, each vector component is simply treated as a separate dimension of the data set. Spatial coherency can only be obtained when applying the non-standard decomposition scheme sequentially for each dimension.

Similar wavelet schemes have been developed for tetrahedral meshes, and the scheme works in a similar way.

An additional complication arises when the temporal domain is taken into account. The following method shows how the temporal dimension can be treated in the same way as a spatial dimension.

Time-varying Tetrahedral Mesh Decimation

Extremely high decimation rates can be obtained by taking the temporal domain into account. Instead of tetrahedra, we consider a mesh of hypertetrahedra that consists of tetrahedra that are connected across the temporal domain (figure 2).

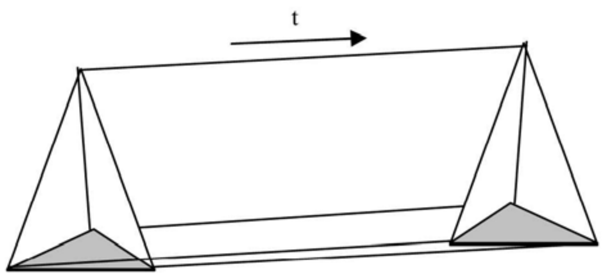

Figure 2. Hypertetrahedron.

We define a hypertetrahedron as a set of at least two (possibly more) tetrahedra where all four vertices are connected in the time domain. Intuitively, a hypertetrahedron represents a tetrahedron that changes shape over time. Every time step represents a snapshot of the current shape. Without loss of generality, we can assume that the time domain is a linear extension of a three-dimensional Cartesian coordinate system. As a consequence, we connect corresponding vertices with linear, i.e. straight, line segments that indicate the motion of a vertex between two or more discrete time steps. Since many tetrahedra do not change significantly over time, hypertetrahedra can be collapsed both in the temporal domain as well as in the spatial domain.

This results in hypertetrahedra that are either stretched in space or in time. Mesh decimation in time means that a hypertetrahedron (4-D) that does not change over time can be represented by a simple tetrahedron (3-D), just like a tetrahedron can be represented by a single point. The opposite direction (expansion of a tetrahedron to a hypertetrahedron over time) is not necessary, because a hypertetrahedron is collapsed only if it does not change significantly in a later time step.

The latter of the previously mentioned cases turns out to restrict the decimation ratio significantly. Since we do not allow hypertetrahedron expansion from a tetrahedron (split in the temporal domain), a large potential for decimation is wasted. Also, for practical purposes the given approach is not very suitable, because we need to be able to access the position of each vertex in the mesh at every time step if we want to navigate in both directions in the temporal domain. The reconstruction of this information and navigation in time with VCR-like controls requires a global view of the data. This means that the data cannot be processed sequentially for an infinite number of time steps. Consequently, the algorithm is not scalable.

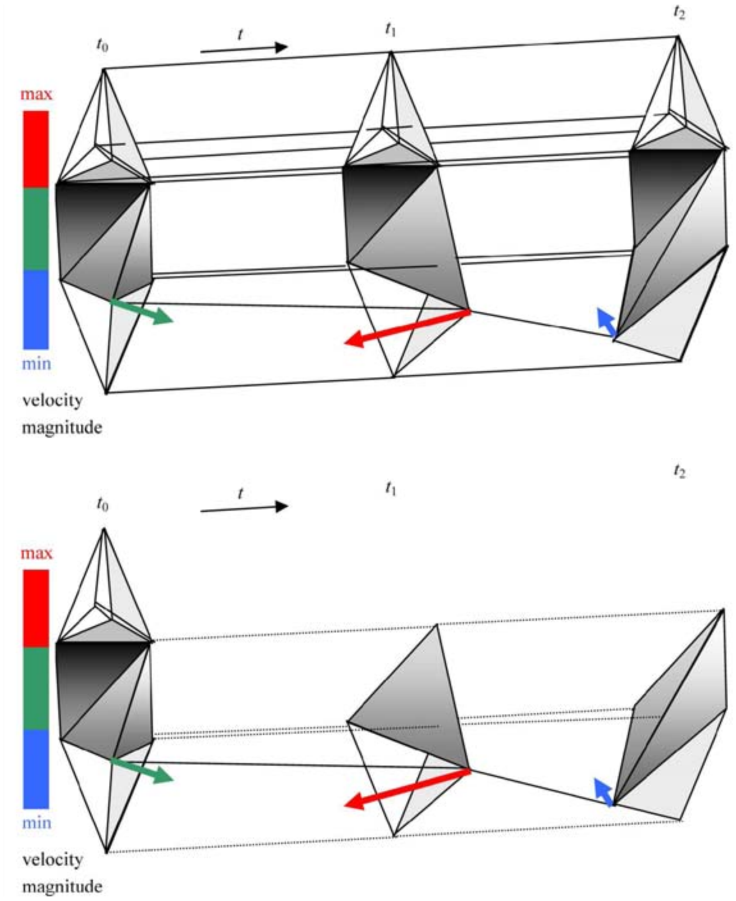

Figure 3. One affected node, two affected tetrahedra.

Figure 3 shows an example where one node is affected by a velocity vector. The velocity is proportional to the displacement, because all time steps have the same temporal distance. Therefore, the arrow indicates the position of the node in the next time step. Two tetrahedra are affected by this displacement and change over time. In this example, it would be sufficient to store the time history of the affected node (solid line) or the affected tetrahedra (dotted lines). In order to reconstruct the mesh, it would be necessary to search for the most recent change in the time history of each node, which would require keeping all time histories of all nodes in memory during playback. This becomes particularly obvious if forward and backward navigation in time is considered.

Even though this method offers a very compact representation of a time-varying tetrahedral mesh, we propose a different approach that enables easier playback (forward and backward) of all the time steps in a simulation. The standard method would be to decimate the mesh for each time step separately by applying QTetFusion [3] or some other mesh decimation technique. However, this approach would result in different meshes for every time step, leading to 'jumps' and flicker in the visualization. This would be very disruptive in an animation or on a virtual reality display.

Therefore, we use a different approach. The idea is to preserve every time step, which is necessary for playback as an animation and for navigation in time. The mesh that has the greatest distortion due to the earthquake (the velocity vector values associated with each grid node) is identified, and then decimated. All the decimation steps that were necessary to reduce the complexity of this mesh are recorded. For the record, it is sufficient to store the IDs of the affected tetrahedra in a list, because for the given application the IDs of the tetrahedra are identical in all time steps. Since tetrahedra are only removed but never added, the IDs are always unique. These recorded steps are then used to guide the decimation of the remaining meshes, i.e., the meshes of the other time steps are decimated in the exact same manner as the one whose features are supposed to be preserved.

Figure 4 shows that the decimation of t_0 and t_1 is guided by t_2, because t_2 is more distorted than any of the others. The decimated mesh should represent all displaced nodes in the most accurate way, because these are the ones that represent the significant features in the given application. Isotropic regions, i.e., areas that are not affected by the earthquake, such as the three tetrahedra at the top that are simplified into a single tetrahedron, expose only little variance in the data attributes, and consequently do not need to be represented as accurately, i.e., with the same error margin, as the significantly changing feature nodes in the other time steps. Comparing the top scenario (before decimation) and the bottom scenario (after decimation), the image shows that the tetrahedra on the top that were simplified in the selected t_2 time step are also simplified in the other two time steps (t_0 and t_1).

The question that remains is how to identify this 'most distorted' mesh. Instead of using complex criteria, such as curvature (topology preservation) and vector gradients (node value preservation), we simply divide the length of the displacement vector for each node by the average displacement of that node, calculate the sum of all these ratios, and find the mesh that has the maximum sum, i.e., the maximum activity. If there is more than one mesh with this property, we use the one with the smallest time index. The average activity of a node is the sum of all displacement vector lengths for all time steps divided by the number of time steps. This means that we consider those nodes in the mesh that expose a lot of activity, and try to represent the mesh that contains these active nodes in the best possible way. The mesh decimation algorithm is applied only to this one mesh. All other meshes are decimated according to this guiding mesh, using the same node indices for the collapse sequence.

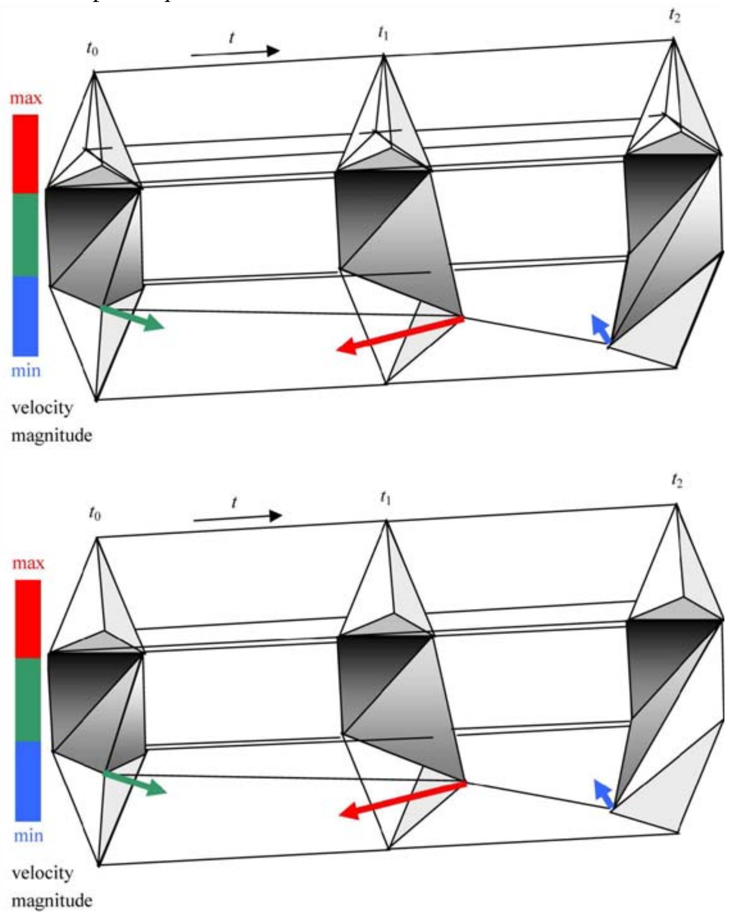

Figure 4. Preservation of temporal resolution, decimation guided by t_2.

Summary

We have shown that wavelet decomposition and tetrahedral mesh decimation can be used in higher-dimensional data sets. In general, the compression ratio obtained after transforming the data set into the given representation is higher than for the original data set.

References

[1] Schneider, Timna Esther, “Multiresolution-Darstellung von 2D-Schichtdaten in der medizinischen Bildverarbeitung,” thesis; Department of Computer Science, University of Kaiserslautern, Germany, December 1997.

[2] Meyer, J., Borg, R., Hamann, B., Joy, K.I. and Olson, A.J., Network-based rendering techniques for large-scale volume data sets. in: Farin, G., Hamann, B. and Hagen, H., eds., Hierarchical and Geometrical Methods in Scientific Visualization, Springer-Verlag, Heidelberg, Germany, pp. 283-296, 2002.

[3] Chopra, Prashant, and Joerg Meyer. Topology Sensitive Volume Mesh Simplification with Planar Quadric Error Metrics, IASTED International Conference on Visualization, Imaging, and Image Processing (VIIP 2003), Benalmádena, Spain, pp. 908-913, September 8-10, 2003.