Procedural Generation of Triangulation-Based Visualizations

Gunther H. Weber[1,2], Bjoern Heckel[1], Bernd Hamann[1], and Kenneth I. Joy[1]

[1] Center for Image Processing and Integrated Computing (CIPIC), Department of Computer Science, University of California, Davis, CA 95616-8562, USA
[2] AG Graphische Datenverarbeitung und Computergeometrie, Fachbereich Informatik, Universität Kaiserslautern, D-67653 Kaiserslautern, Germany

Abstract

We present a new visualization approach based on procedural grid generation for scattered data sets. Instead of pre-computing triangulations for a given set of levels in a data hierarchy, a triangulation within a specified region of interest (ROI) is computed by considering scattered data of a pre-computed scattered data hierarchy. By choosing an appropriate level in the hierarchy the number of samples representing the data set in the ROI can be kept sufficiently small to interactively generate a triangulation and a visualization of the data within the ROI. This allows one to generate the triangulation on-the-fly during the visualization process.

1 Introduction

Visualization of large-scale scientific data sets is a field of increasing importance. Often, data are given as scattered data, *i.e.*, data *samples* consisting of a position and a data value without connectivity. A commonly used solution to deal with the visualization of large-scale data sets is to use a *hierarchical representation*. A coarse data representation allows one to get an overview of the data set. However, it is also necessary to enable a user to view a given region of interest (ROI) in the data domain at fine detail. This necessitates the availability of different resolution levels. A hierarchical representation that stores a data set using different levels of detail is therefore necessary.

Many visualization techniques depend on connectivity information, see [6, 7]. If connectivity information is required, it is possible to generate a set of hierarchical grids representing a data set at different resolutions. However, if the original data set consists only of unconnected sample points only triangulations for a set of representations at given resolutions are computed[1].

In order to achieve sufficient accuracy in high-resolution representations these have to consist of a relatively large amount of samples. Thus, the computational cost necessary to generate a triangulation for these levels becomes prohibitively high. This makes the approach of computing a grid for an entire hierarchy level impractical.

At a higher levels of detail it is not necessary to generate a triangulation for the complete set of data associated with this level in the hierarchy. We note that these levels are only accessed when a user concentrates on a specific ROI in a data set. Thus, it is only required to generate a triangulation for the region currently of interest to a viewer. We present an innovative approach to on-the-fly grid generation enabling efficient localized visualization. Instead of pre-computing grids for a fixed set of levels of detail, we apply a procedural triangulation scheme. Depending on the size of an ROI, an appropriate level of detail of the scattered data hierarchy is accessed. This is done in a way such that the number of samples used to approximate a data set remains constant. This number can be kept sufficiently small, thereby making it possible to generate a triangulation procedurally as part of the visualization process.

[1]If the original data set is defined on a grid, one can use methods that allow access to all intermediate grid levels, see [5].

2 Related work

Related work is described in the grid generation and data set simplification literature. An overview of grid types and grid generation techniques is given in Chapter 3 of [8].

Related work in data simplification is grid-based simplification, including, *e.g.*, the work of Hoppe *et al.* [5]. The authors simplify a triangular mesh by a series of edge collapses. By storing the simplified mesh along with the performed collapse operations, it is possible to access all intermediate levels of detail. In this regard the method is similar to the use of the scattered data hierarchy we discuss in this paper. Staadt and Gross [11] have developed a similar algorithm for tetrahedral meshes.

3 Generating the hierarchy

Our procedural triangulation scheme relies on the availability of a hierarchical data representation. This hierarchy can be derived from a *clustering* scheme. Clustering, see [1], is a classical data analysis technique commonly used in statistics. It has been applied to *surface reconstruction* by Heckel *et al.* [3], and this approach has been further generalized by Heckel *et al.* [4] for vector field simplification. In the context of vector field visualization, the basic idea is to partition a vector field into clusters corresponding to coherent regions characterized by vectors of similar direction and length. Each cluster has a *representant*. It is computed as average of all positions and vectors, *i.e.*, each component of the representant is the arithmetic mean of this component of all samples in that cluster. Thus, a representant is placed in the center of the cluster; its vector is the average of all vectors within the cluster.

This hierarchical cluster representation is computed using a top-down strategy. First, all vectors are assigned to a single cluster representing an entire field. Subsequently, the following steps are performed:

1. The cluster with the largest error — according to a given error metric — is determined.
2. This cluster is split into two clusters.

This process is iterated until the error of all clusters is below a given threshold and a simplified representation of a field has been obtained. A detailed description of the clustering process can be found in [4]. Instead of using the original data set, it is possible to use the representants of the clusters resulting from performing the split steps. By storing all intermediate clusters one has access to a hierarchical representation.

4 The clustering tree

The clustering process yields a hierarchical representation of a field without sample connectivity. This representation is a binary tree which we call the *clustering tree*.

DEFINITION 1 (CLUSTERING TREE)
The *clustering tree* is a binary tree corresponding to the evolution of the clustering process. Each node in this tree has the following properties:

- It corresponds to a cluster created during the clustering process.
- It describes the region of the field containing the samples associated to its corresponding cluster.
- If a finer representation of this region is available it has two descendants; otherwise, it has no descendants.

Each node contains the following information:

1. the sample $s_{\mathrm{node}} = (\mathbf{x_{node}}, \mathbf{v_{node}})$, which is the representant of the cluster corresponding to this node;
2. the error describing the deviation of the vector field from the representant in the region described by this node; and
3. a "split number" that indicates when the cluster corresponding to this node was split.

Using the split number, the exact progression of the clustering process can be reproduced, and any desired level of detail within the limits of the given cluster hierarchy can be accessed. Figure 1

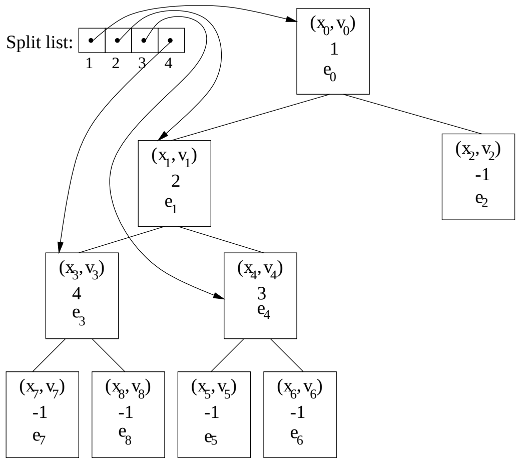


Figure 1: Clustering tree and array mapping split numbers to nodes.

shows a simple example of a clustering tree. The nodes contain the representant (first row), the split number (second row), and the error for this node (last row). In this representation, any level of detail constructed from the hierarchy is represented by a thread connecting the nodes belonging to that level of detail. Starting with a thread containing only the root node, each thread can be constructed from the previous one by replacing the node with the lowest split number along the thread with its two descendants. The *split list*, depicted in Figure 1, provides a simple mapping between the split number and nodes in the clustering tree and allows constant-time access to the node describing the next split process. This is shown in Figure 2. The thread corresponding to the current hierarchy level is drawn using solid arrows starting at "currentThread" and ending at "NULL." The thread representing the next level can be constructed by replacing the grey-colored node in the clustering tree by its two descendants. The dashed arrows indicate the difference between the original and the resulting thread. It is also possible to reverse the refinement of vector field. This can be done by successively replacing the two children of the last split node by their parent.

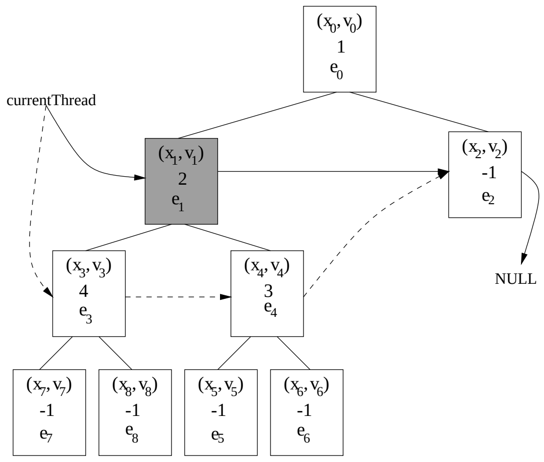


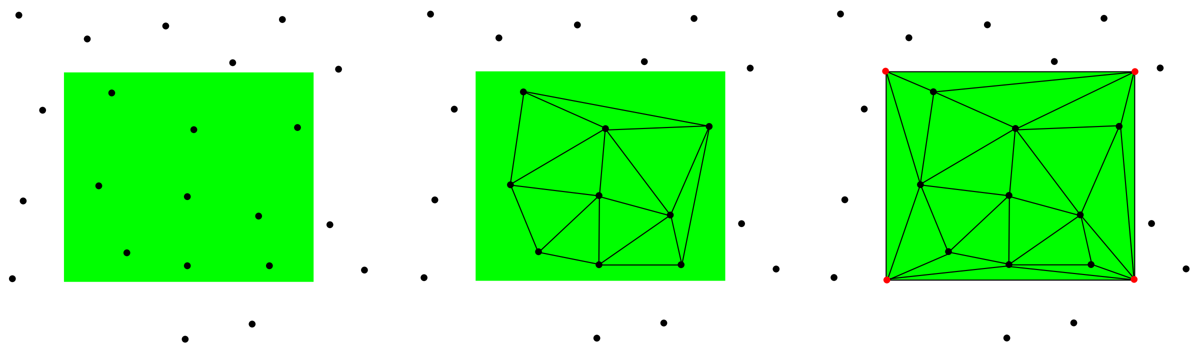
Figure 2: Refining a thread in the clustering tree.

5 Procedural Triangulation

In order to generate triangulations procedurally during the visualization process, one has to explicitly choose an ROI (shaded region in Figure 3(a)) in a data set (a vector field). One does not have to be aware of the hierarchy to choose this region. Our approach is completely transparent in regard to the underlying hierarchy. Once a user has chosen her/his ROI, the samples within that region are triangulated using a Delaunay triangulation, see [10, 9]. This is shown in Figure 3(b). We note that the triangulation does not completely "fill" the ROI. This can be remedied by taking additional samples outside the ROI into account, but this is not a trivial task. If the ROI is not completely contained within boundaries of the vector field, it is not possible to fill the ROI completely, even when samples outside this region are used. Additionally, it is computationally expensive to check whether the ROI is completely inside the vector field. It is possible to avoid these costly computations and special cases: By interpolating the values at the vertices of the ROI with a scattered data interpolant, *e.g.*, Hardy interpolation, see [2], and adding these to the triangulation, it is possible to generate a triangulation filling the complete ROI, see Figure 3(c). Adding these vertices to the triangulation causes the boundaries of the triangulation (now containing the added vertices) and the ROI to coincide.

Once an ROI has been triangulated it is possible to use this triangulation in conjunction with a mesh-based interpolation scheme to estimate values inside that region. Furthermore, it is possible to use this triangulation for other algorithms that require a tetrahedral mesh, *e.g.*, stream line generation, see [7].

So far, we have not addressed the role of the hierarchy for the procedural generation of a simplical mesh. The basic idea is to represent the vector field within an ROI using a fixed number of samples. Thus, the effort necessary for the visualization is independent of the size of the ROI. Furthermore, choosing a smaller ROI automatically increases the accuracy of the representation, since the same number of samples is used to approximate the smaller region.



(a) Choosing an ROI.

(b) Triangulation of points within ROI.

(c) Adding vertices of ROI to triangulation.

Figure 3: Steps for computing a procedural triangulation.

Keeping the number of samples in an ROI constant is achieved by using the clustering tree and the thread representing a particular hierarchy level. Initially, the complete vector field is chosen as ROI and approximated by a user-specified number of samples. Whenever the ROI is changed, the number of samples in the ROI are counted. Depending on whether there are too many or too few samples in the region, the split steps are reversed or further split steps are reproduced. The result is an ROI that contains, if possible, a constant number of samples regardless of its size and position. In order to keep track of the number of samples currently within the ROI, it is necessary to determine how each performed split operation affects this number.

We note that this approach refines the complete vector field until the desired number of samples is generated for the ROI, despite the fact that only the samples inside the ROI are used afterwards.

6 Results

Figures 4 and 5 show triangulations of ROIs in the "Blunt Fin" data set. Figure 4 shows the triangulation for a large ROI. We note that the triangulation is fine in regions of high variation. Figure 5 illustrates that a smaller ROI is approximated with a finer grid and thus yields a more accurate representation of the data set.

7 Future work

Our procedural triangulation scheme can be improved in many different ways. It may be advantageous to replace the Delaunay triangulation with a computationally less expensive triangulation procedure. This would allow one to use a larger number of samples to approximate the data set in an ROI and obtain a more accurate representation at the same speed.

One drawback of our current scheme is the fact that the hierarchy is used to refine an entire vector field and not just the part of it that is within the ROI. This could be avoided by only performing split operations that yield new samples within an ROI. Furthermore, this would eliminate the need to add interpolated values for the vertices to the triangulation. By starting with a coarse approximation of a data set and only refining the ROI it may be possible to fill the complete ROI without additional samples.

8 Acknowledgments

This work was supported by the National Science Foundation under contract ACI 9624034 (CAREER Award), the Office of Naval Research under contract N00014-97-1-0222, the Army Research Office under contract ARO 36598-MA-RIP, the NASA Ames Research Center through an NRA award under contract NAG2-1216, the Lawrence Livermore National Laboratory through an ASCI ASAP Level-2 contract under W-7405-ENG-48 (and B335358, B347878), and the North Atlantic Treaty Organization (NATO) under contract CRG.971628 awarded to the University of California, Davis. We also acknowledge the support of Silicon Graphics, Inc., and thank the members of the Visualization Thrust at the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis.

References

[1] A. D. Gordon. Hierarchical classification. In R. Arabie, L.J. Hubert, and G. DeSoete, editors, *Clustering and Classification*, pages 65–105. World Scientific Publishers, River Edge, NJ, 1996.

[2] R. L. Hardy. Theory and applications of the multiquadric-biharmonic method: 20 years of discovery 1968–1988. *Computers and Mathematics with Applications*, 19:163–208, 1990.

[3] B. Heckel, Antonio E. Uva, and B. Hamann. Clustering-based generation of hierarchical surface models. In C.M. Wittenbrink and A. Varshney, editors, *Proceedings of Visualization 1998 (Hot Topics)*, pages 50–55. IEEE Computer Society Press, Los Alamitos, CA, October 1998.

[4] B. Heckel, G. H. Weber, B. Hamann, and Kenneth I. Joy. Construction of vector field hierarchies. To appear in Proceedings IEEE Visualization '99, IEEE Computer Society Press, Los Alamitos, October 1999.

[5] H. Hoppe. Progressive meshes. In Holly Rushmeier, editor, *SIGGRAPH '96 Conference Proceedings*, Annual Conference Series, pages 99–108. ACM SIGGRAPH, Addison Wesley, August 1996.

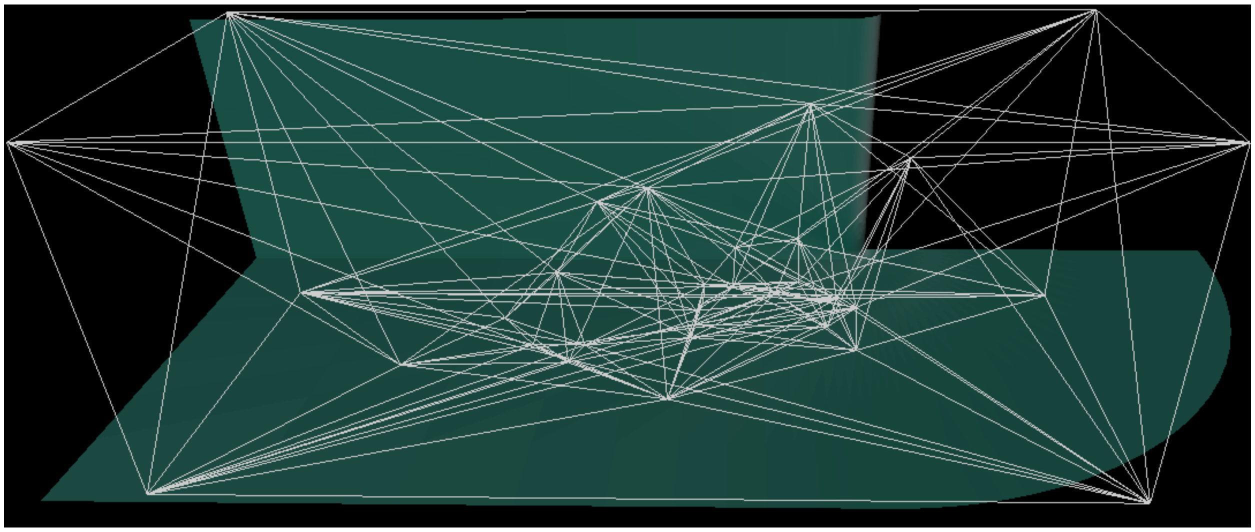


Figure 4: Triangulation of a large ROI in “Blunt Fin” data set (courtesy of NASA Ames Research Center).

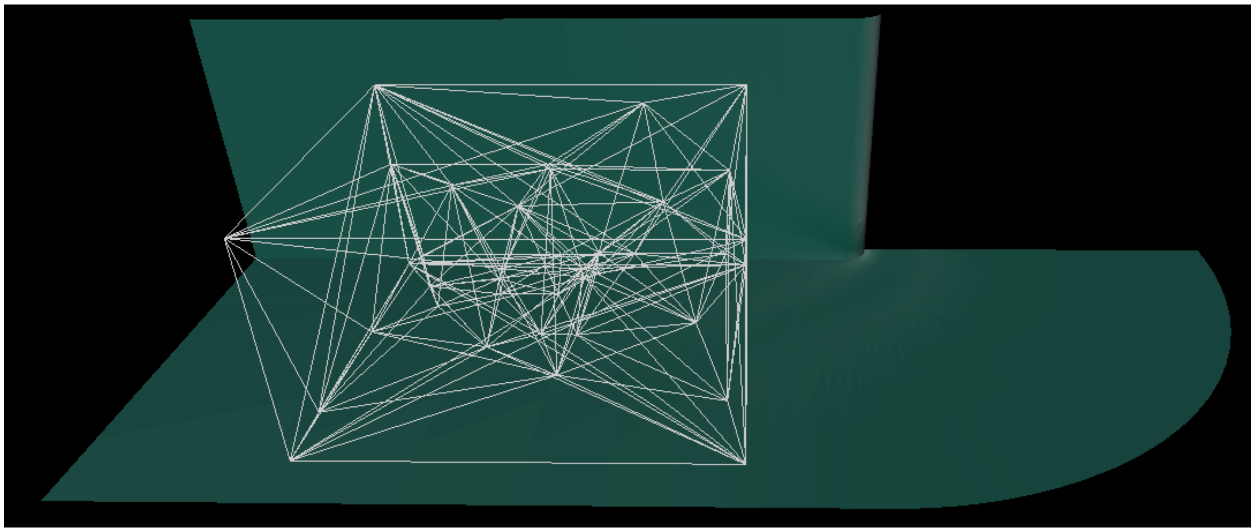


Figure 5: Triangulation of a small ROI in “Blunt Fin” data set (courtesy of NASA Ames Research Center).

[6] D. N. Kenwright. *Dual stream function methods for generating three-dimensional stream lines*. PhD thesis, Department of Mechanical Engineering, University of Auckland, August 1993.

[7] D. Knight and G. Mallinson. Visualizing unstructured flow data using dual stream functions. *IEEE Transactions on Visualization and Computer Graphics*, 2(4):355–363, December 1996.

[8] G. M. Nielson, H. Hagen, and H. Müller, editors. *Scientific Visualization: Overviews, Methodologies, Techniques*. IEEE Computer Society, Los Alamitos, California, 1997.

[9] Atsuyuki Okabe, Barry Boots, and Kokichi Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Probability and Mathematical Statistics. John Wiley & Sons, Chichester, England, September 1992. Foreword by D. G. Kendall.

[10] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, 1985.

[11] Oliver G. Staadt and Markus H. Gross. Progressive tetrahedralizations. In David S. Ebert, Hans Hagen, and Holly Rushmeier, editors, *Proceedings of Visualization 98*, pages 397–402. IEEE Computer Society Press, Los Alamitos, California, October 1998.