An Edge-preserving, Data-dependent Triangulation Scheme for Hierarchical Rendering

James C. “Fritz” Barnes[1], Bernd Hamann[1,2], and Kenneth I. Joy[1]

Center for Image Processing and Integrated Computing (CIPIC)
Department of Computer Science
University of California, Davis

Abstract

In many applications one is concerned with the approximation of functions from a finite set of given data sites with associated function values. We describe a construction of a hierarchy of triangulations which approximate the given data at varying levels of detail. Intermediate triangulations can be associated with a particular level of the hierarchy by considering their approximation errors. This paper presents a new data-dependent triangulation scheme for multi-valued scattered data in the plane. We perform piecewise linear approximation based on data-dependent triangulations. Our scheme preserves edges (discontinuities) that might exist in a given data set by placing vertices close to edges. We start with a coarse, data-dependent triangulation of the convex hull of the given data sites and subdivide triangles until the error of the piecewise linear approximation implied by a triangulation is smaller than some tolerance.

1 Introduction

We present an algorithm for the approximation of bivariate scattered data based on a data-dependent triangulation scheme. Considering approximation error, a triangulation can be associated with a particular level in an approximation hierarchy. Our iterative algorithm refines an initial, coarse triangulation of a subset of the given data sites[3] by subdividing triangles. The subdivision process produces a sequence of piecewise linear functions which improve the approximation to the given scattered data with each subdivision. The method can be applied to general multi-valued

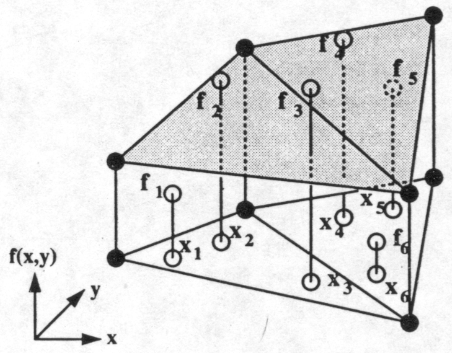


Figure 1. Triangle mesh approximating given scattered data

scattered data, *i.e.*, one can use it for the approximation of color images or vector fields. We assume that a vector of m function values $\mathbf{f}_i = (\mathrm{f}_{i,1}, ..., \mathrm{f}_{i,m})$ are given at random locations $\mathbf{x}_i = (\mathrm{x}_i, \mathrm{y}_i), i = 1, ..., n$. This is illustrated in Figure 1.

Our algorithm is based on subdividing triangles based on approximation error. We start with an initial, coarse triangulation whose convex hull coincides with the convex hull of all original data sites. Triangles are split into two, three, or four subtriangles, depending on which split leads to the (locally) best approximation. Since the process of subdividing triangles is based on inserting new vertices along triangle edges, vertices of an intermediate triangulation do not generally coincide with original data sites.

We use the differences between function values at the original data sites and the function values implied by a particular triangulation, *i.e.*, its associated piecewise linear function, as an error measure. Subdivision terminates when the error associated with the entire triangulation is smaller than a specified tolerance. Usually, one uses the term data-dependent in the context of triangulations when it is the goal to approximate some function well by a piecewise linear function. Thus, to achieve a good approximation, tri-

[1]*Center for Image Processing and Integrated Computing (CIPIC), Department of Computer Science, University of California, Davis, CA 95616-8562, USA*; E-mail: `barnes,hamann,joy@cs.ucdavis.edu`

[2]Co-Director of CIPIC

[3]We use the term *site* for the location of an original datum.

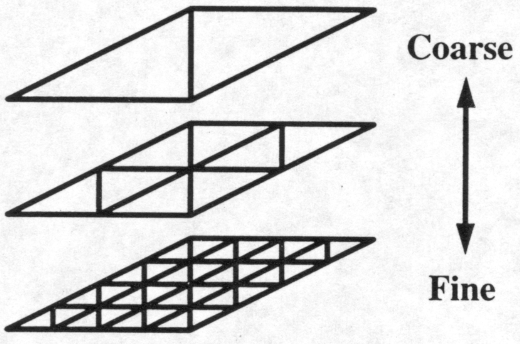


Figure 2. A multiresolution pyramid

angles do not necessarily have good aspect ratios, see *e.g.*, [5, 6, 24].

In the context of approximation over triangulated domains, a data-dependent triangulation scheme adaptively generates a domain triangulation leading to a "small" approximation error. The techniques described in [12, 13, 14, 15, 25], deal with the problem of decimating triangular surface meshes and adaptive refinement of tetrahedral volume meshes. These approaches are aimed at the concentration of points in regions of high curvature. This paradigm can be used to either eliminate points in nearly linearly varying regions (*decimation*) or to insert points in highly curved regions (*refinement*). The data-dependent triangulation scheme that we describe in this paper is based on the principle of refinement: our algorithm inserts points in areas with large approximation errors.

In principle, our technique is related to the idea of constructing a *multiresolution pyramid*, *i.e.*, a data representation hierarchy of triangulations with increasing precision, see [9]. Figure 2 shows a multiresolution hierarchy of triangles where the top level contains a coarse triangulation and as we *descend* the hierarchy finer triangulations are visible. The pyramid concept has also been extended to the adaptive construction of tetrahedral meshes for scattered scalar-valued data, see [2, 3]. *Multiresolution methods* have been applied to polygonal approximations of surfaces. Such approaches are described in [4, 7, 18]. Our data-dependent technique can be viewed as a hierarchical method for representing scattered data by multiple levels of triangulations, yet our approach is not based on the construction and application of *wavelet* bases.

Edges, or discontinuities, are common in many of the data sets that we have considered. These discontinuities may be ridge lines in terrain data, bone boundaries in medical imaging data, or object edges in digital images. Previous techniques such as [23] discuss decimation of a triangle mesh utilizing the dihedral angle between triangles as a measure of whether an edge exists. Another approach is to recognize the linear "coherence" of discontinuities [22]; in this refinement scheme, when a triangle is subdivided, one attempts to place a triangle edge along the discontinuity in the data set. While the approach discussed in [22] extracts the set of points lying on discontinuities from the initially given data and uses these points as vertices in each triangulation level, we compute points on the boundaries as part of each refinement step.

In [17], an elegant triangle decimation scheme is described for general surface triangulations. Our scheme is, in contrast, a refinement scheme: we start with a coarse representation and iteratively refine triangulations. Furthermore, our approach is tailored to the domain triangulation of bivariate functions.

A survey paper of scattered data approximation for bivariate and trivariate data is [19]. In [20], various scattered data interpolation techniques (scalar-valued, trivariate case) are discussed and compared. The method that we describe in the following relies to a high degree on *geometric modeling* and *computational geometry* concepts; they can be found in [8, 21].

2 Adaptive triangle refinement

The adaptive refinement process is started by creating an initial triangulation. Knowing this initial triangulation, we iteratively refine intermediate triangulations until a triangulation is obtained whose global approximation error (E_{RMS} or E_{MAX} – root-mean-square error or maximal absolute error) is smaller than some tolerance.

Refining an intermediate triangulation consists of four basic steps: (i) additional vertices are generated along the edges of an existing triangle; (ii) function values are approximated for each of the new vertices (and certain existing vertices in the neighborhood); (iii) a new triangulation is constructed for the set of "old" and inserted vertices; and (iv) an error estimate is computed for the new triangulation. These steps are iterated until a certain approximation error condition is satisfied. The adaptive placement of new vertices along the edges of existing triangles is crucial. The algorithm is adaptive in a twofold sense: (a) an intermediate triangulation is refined locally in regions with large errors; and (b) the location of new vertices is chosen in order to minimize error.

We consider error estimates measuring the deviation of a triangular approximation and the original data. To do this we compute the absolute differences between the function values given at the original data sites and the piecewise linear approximation implied by a triangulation. Denoting the set of k original data sites lying inside a particular triangle τ or on its boundary by $\mathbf{x}_i$, $i = 1, ..., k$, with associated function values $\mathbf{f}_i$, the *local root-mean-square (LRMS) error)* is defined as

$$E_{LRMS} = \sqrt{\frac{1}{k}\sum_{i=1}^{k} ||\mathbf{L}(\mathbf{x}_i) - \mathbf{f}_i||^2}, \qquad (1)$$

where $|| \; ||$ denotes the Euclidean norm and $\mathbf{L}$ the linear polynomial implied by the triangle τ. We have also considered using the maximum error but have discovered that the results clearly favor the root-mean-square error measure. (We ensure that the ranges of the components of a vector-valued function are always normalized.)

In each refinement step, we identify the triangle with maximal local approximation error and subdivide it in a locally optimal way. More precisely, we store the triangles as an ordered list by considering their associated errors. From this list we select a certain percentage of triangles that we consider for the next refinement step. Provided that at least two original sites are inside a selected triangle, we can subdivide it. In the case that there is only one original site inside a selected triangle we connect this site with the triangle's vertices. A single subdivision step increases the number of triangles by no more than six: the triangle selected for subdivision will be split into no more than four triangles, and the subdivision process generates new vertices on triangle edges which causes implied splits with neighboring triangles. The split point along the triangle edges are connected to the opposite vertex of the (up to) three neighbor triangles. We must compute function value estimates for all new vertices and must update the function value estimates for all old vertices whose *tiles* have changed as a result of the re-triangulation process.

In the following subsections, we discuss how to obtain the initial triangulation, how to refine a triangle, how to estimate function values for new vertices, and how to preserve edges that might exist in an original data set.

2.1 Initial triangulation

We consider the convex hull of the set of given data sites as the *natural* boundary of a data set. We use the *Jarvis' march* algorithm to compute the convex hull. This may result in multiple collinear points on the interior of edges defining the convex hull. We remove these collinear points, see [21]. After applying the above technique we have a minimal K points which define the convex hull.

Using the convex hull, we compute a data-dependent triangulation for the minimal point set defining the convex hull. In general, one has to consider all possible triangulations of this point set and select the one that minimizes one's error measure. A possible initial triangulation is shown in Figure 3. Computing all possible initial triangulations cannot be done efficiently when the convex hull is defined by a relatively large number of points. In this case we propose

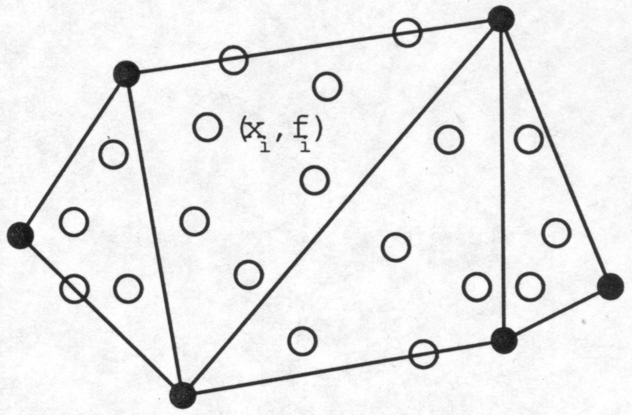


Figure 3. Coarse initial triangulation of domain

to construct any triangulation of the K points and then apply *simulated annealing* in order to obtain a better, possibly optimal, data-dependent triangulation [16, 24].

Remark. For many practical applications, it might be sufficient to simply use the four vertices defining the corners of the bounding box containing all original sites. After all, many real-world data sets are defined on a uniform, rectilinear grid whose convex hull coincides with its bounding box.

2.2 Subdividing a triangle

Whenever we subdivide a triangle with vertices $\mathbf{V}_1, \mathbf{V}_2$, and $\mathbf{V}_3$ we split it, topologically, into four subtriangles, using variable split points $\mathbf{S}_{1,i}, \mathbf{S}_{2,j}$, and $\mathbf{S}_{3,k}$ on the edges $e_1 = \overline{\mathbf{V}_2\mathbf{V}_3}$, $e_2 = \overline{\mathbf{V}_3\mathbf{V}_1}$, and $e_3 = \overline{\mathbf{V}_1\mathbf{V}_2}$. We consider combinations of (N+1) candidate split points per edge. The candidate split points are spaced uniformly on the triangle edges, unless one has to consider discontinuities in the original data set. Thus, the sets of candidate split points are

$$\begin{aligned} &\left\{\mathbf{S}_{1,i} = \tfrac{N-i}{N}\mathbf{V}_2 + \tfrac{i}{N}\mathbf{V}_3 \mid i = 0, ..., N\right\}, \\ &\left\{\mathbf{S}_{2,j} = \tfrac{N-j}{N}\mathbf{V}_3 + \tfrac{j}{N}\mathbf{V}_1 \mid j = 0, ..., N\right\}, \\ &\left\{\mathbf{S}_{3,k} = \tfrac{N-k}{N}\mathbf{V}_1 + \tfrac{k}{N}\mathbf{V}_2 \mid k = 0, ..., N\right\}. \end{aligned} \qquad (2)$$

Figure 4 shows the possible split points for $N = 3$.

As shown in Figure 5, we consider four possibilities when subdividing a triangle. The vertex triples defining the four subtriangles for each of the four possibilities are

- $(\mathbf{V}_1, \mathbf{S}_{3,k}, \mathbf{S}_{2,j})$, $(\mathbf{V}_2, \mathbf{S}_{1,i}, \mathbf{S}_{3,k})$, $(\mathbf{V}_3, \mathbf{S}_{2,j}, \mathbf{S}_{1,i})$, $(\mathbf{S}_{1,i}, \mathbf{S}_{2,j}, \mathbf{S}_{3,k})$,
- $(\mathbf{V}_1, \mathbf{S}_{1,i}, \mathbf{S}_{2,j})$, $(\mathbf{V}_1, \mathbf{S}_{3,k}, \mathbf{S}_{1,i})$, $(\mathbf{V}_2, \mathbf{S}_{1,i}, \mathbf{S}_{3,k})$, $(\mathbf{V}_3, \mathbf{S}_{2,j}, \mathbf{S}_{1,i})$,
- $(\mathbf{V}_1, \mathbf{S}_{3,k}, \mathbf{S}_{2,j})$, $(\mathbf{V}_2, \mathbf{S}_{2,j}, \mathbf{S}_{3,k})$, $(\mathbf{V}_2, \mathbf{S}_{1,i}, \mathbf{S}_{2,j})$, $(\mathbf{V}_3, \mathbf{S}_{2,j}, \mathbf{S}_{1,i})$, and

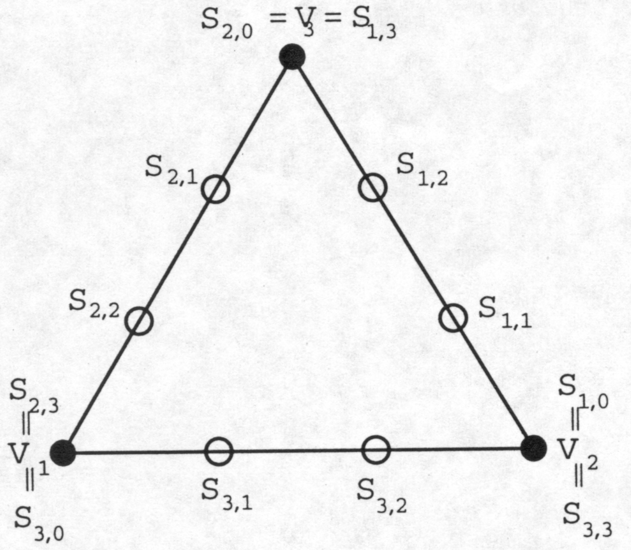

$S_{2,0} = V_3 = S_{1,3}$
$S_{2,1}$
$S_{1,2}$
$S_{2,2}$
$S_{1,1}$
$S_{2,3} = V_1 = S_{3,0}$
$S_{1,0} = V_2 = S_{3,3}$
$S_{3,1}$
$S_{3,2}$

Figure 4. Uniformly spaced split points for $N = 3$

- $(\mathbf{V}_1, \mathbf{S}_{3,k}, \mathbf{S}_{2,j})$, $(\mathbf{V}_2, \mathbf{S}_{1,i}, \mathbf{S}_{3,k})$, $(\mathbf{V}_3, \mathbf{S}_{3,k}, \mathbf{S}_{1,i})$, $(\mathbf{V}_3, \mathbf{S}_{2,j}, \mathbf{S}_{3,k})$.

Certain vertex triples imply degenerate triangles, *i.e.*, triangles with zero area. This is the case when a point $\mathbf{S}_{1,i}$, $\mathbf{S}_{2,j}$, or $\mathbf{S}_{3,k}$ coincides with a corner of the triangle being refined. We eliminate such degenerate subtriangles from the mesh, leading to the possible triangulations shown in Figure 6.

We have to approximate function values for each of the candidate split points. These estimates depend on the future local re-triangulation itself. Therefore, one has to compute the resulting errors – we use E_{RMS} (the maximum of all local E_{LRMS} values) – for each possible re-triangulation. There will be certain re-triangulations which minimize the maximum of the local approximation errors (*min-max error criterion*). We select one of these error-minimizing re-triangulations: If there is only one re-triangulation minimizing the maximum of the local error estimates, we select that one; if there are multiple ones to choose from (each one minimizing the maximum of the local errors), we select the one that maximizes the minimum angle in the local re-triangulation (*max-min angle criterion*).

When inserting a new vertex along an edge of a particular triangle, we must split the neighbor triangle sharing this edge into two subtriangles as well (*knot-to-knot condition*). When applying the subdivision step to a particular triangle, we consider all possible re-triangulations of this triangle and its neighbor triangles. We determine the effect on the resulting errors associated with all triangles resulting from refinement, the ones replacing existing ones, and choose that re-triangulation that locally minimizes E_{LRMS}.

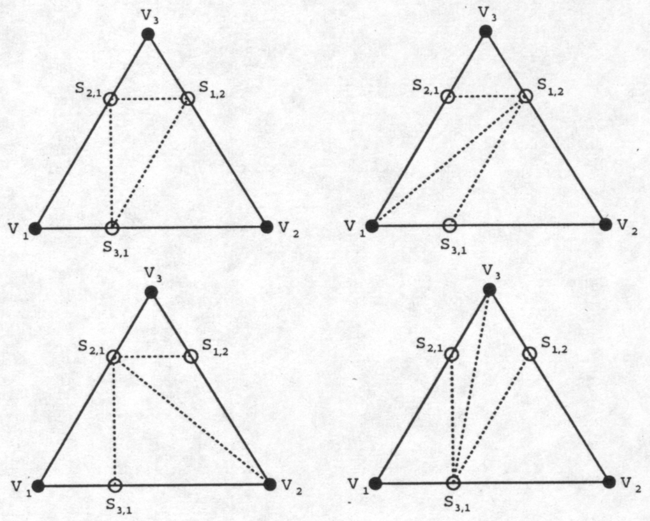


Figure 5. Four possible re-triangulations

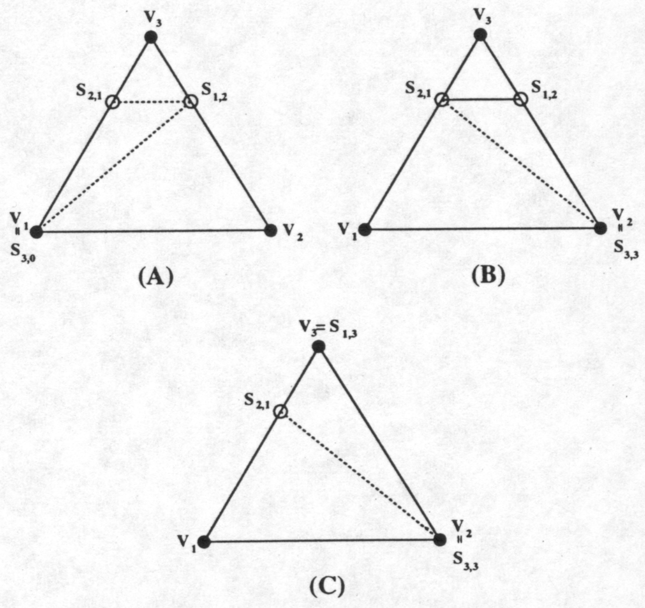


Figure 6. Degenerate subdivision

It is our experience that one should also consider the geometrical quality of triangles being generated and avoid triangles whose geometrical quality is below an acceptable tolerance. To evaluate the geometric quality of a trinagle we consider the triangle's minimum angle. Although we wish to use long, skinny triangles that model datasets in a data-dependent fashion, we wish to avoid long, skinny triangles that do not contain a certain minimum number of original data sites. It is our experience that skinny triangles containing few original data sites result in visual "spikes" that we would like to avoid. When considering the different splits of a triangle we group the possible splits into two categories: those that exceed the minimum angle threshold and those that do not. When refining an intermediate triangulation, we consider the set of all possible re-triangulations that do not lead to skinny triangles. From this set we choose the one that locally minimizes the approximatino error. Should all possible re-triangulations introduce skinny triangles, we choose the one that maximizes the resulting minimum angle.

2.3 Estimating function values

The function value for a vertex in the triangulation is estimated using a neighborhood of nearby data sites. Whenever triangles are subdivided as a result of inserting additional vertices we must estimate new function values for all vertices in the triangulation whose associated *tile*, see Figure 7, change as a result of the subdivision process. This set of vertices is given by the set of inserted split points and additional "neighboring vertices" whose tiles have changed as a result of refinement. The tiles are needed to compute function value estimates based on a local approximation procedure. The local approximation procedure only considers original data lying inside a tile. The tile associated with a vertex is constructed in two steps:

- **Platelet construction.** The *platelet* associated with a vertex is the set of triangles sharing this vertex. The platelet is determined in a first step.
- **Tile construction.** The *tile* associated with a vertex **V** is computed in a second step. The tile is the region bounded by the polygon obtained by connecting the centroids of the triangles defining **V**'s platelet with the midpoints of those edges in **V**'s platelet that have **V** as an end point. When an edge of a platelet triangle lies on the boundary of the domain, *i.e.*, on the convex hull, we connect this edge's midpoint and **V** to obtain a closed polygon as tile boundary. Figure 7 illustrates the tile construction.

We use an inside/outside test for simple, closed polygons to determine the set of original data inside a tile, see [10].

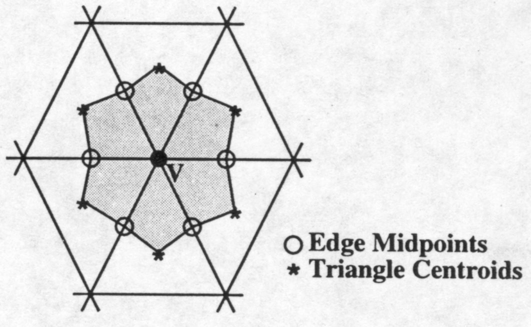


Figure 7. Tile construction

We consider this set of original data for a localized, generalized version of *Shepard's method*, considering gradient estimates (g_i^x, g_i^y) at the original data sites $\mathbf{x}_i$, to estimate a function value f_{app} for a vertex $\mathbf{v} = (x, y)$, see [11]. The function value $f_{\text{app}}(x, y)$ is defined as

$$f_{\text{app}}(x,y) = \begin{cases} f_{\text{orig}}, & \text{if } \mathbf{v} \text{ coincides with an original data site with value } f_{\text{orig}} \\ f_{\text{avg}}, & \text{otherwise.} \end{cases} \tag{3}$$

where

$$f_{\text{avg}} = \frac{\sum_{i=1}^{L} (\mathbf{f}_i + g_i^x(x - x_i) + g_i^y(y - y_i)) \,/\, ({d_i}^2)}{\sum_{i=1}^{L} 1/{d_i}^2} \tag{4}$$

Here, L is the number of original sites inside the tile, $\mathbf{f}_i$ is the function value associated with an original data site $\mathbf{x}_i$, and d_i is the Euclidean distance between the new vertex $\mathbf{V}$ and $\mathbf{x}_i$. The same approach can be used when dealing with multi-valued data by applying it to each of the multiple function values. The gradient estimates (g_i^x, g_i^y) are computed in a pre-processing step as described in the next section.

2.4 Preserving edges and gradient approximation

In many practical applications, data sets contain edges or discontinuities, manifested by very large gradient magnitudes along these edges. In order to capture edges it is crucial to place vertices very close to the edges themselves. Otherwise, a piecewise linear approximation can not reflect sharp changes in function values. In order to preserve edges, we compute gradient estimates for each original data point and consider these estimates in the process of placing certain vertices close to discontinuities. In particular, images often contain sharp edges that are to be preserved.

When subdividing a particular triangle, we determine whether there are large gradients along an edge that is to

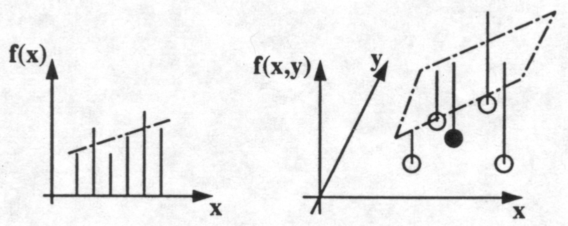


Figure 8. Computing gradient estimates

be split. If this is the case, we place the vertex to be inserted along this edge close to the discontinuity. We estimate the gradient values and use these for adaptive splitting in the presence of edges as follows:

- **Gradient estimation.** We estimate gradients at original data sites in a pre-processing step, using a linear polynomial computed as a least squares approximation. For each original data site $\mathbf{x}_i$, we consider it and its six closest neighbor sites to obtain a local linear approximation $f_i(x, y)$ of the form $g_i^x x + g_i^y y + c_i$.

- **Splitting triangles containing edges.** Having gradient estimates available at the original data sites, we can determine where edges occur. We assume that a particular site is close to an edge if its associated gradient magnitude is larger than some threshold. "Critical sites" are the data sites whose gradient magnitude exceeds the threshold. We use an *adhoc* principle that we have found to work well: We interpret an original data site as being close to an edge if its associated gradient magnitude is in the top five percent of all gradient magnitudes.

  When splitting a triangle that contains critical sites, we identify the critical sites closest to the triangle's edges and project the critical sites onto the edges. These projections are used as the final split points. This principle is illustrated in Figure 9. In more detail, the splitting algorithm follows these steps:

  - Identify the critical sites $\mathbf{b}_1$, $\mathbf{b}_2$, and $\mathbf{b}_3$ – points in the original data set – that are closest to the triangle edges e_1, e_2, and e_3, respectively.
  - Project $\mathbf{b}_1$, $\mathbf{b}_2$, and $\mathbf{b}_3$ onto e_1, e_2, and e_3, respectively, and use the projections as split points.

3 Results and examples

We have applied our method to data sets with and without discontinuities. We have compared our data-dependent scheme with a simple bisection algorithm that bisects the triangle with largest approximation error, but produces only

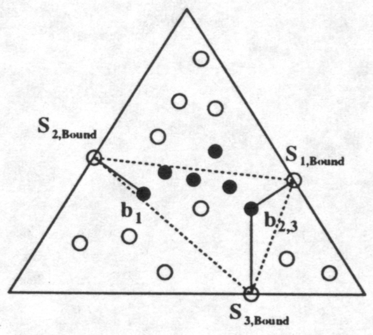


Figure 9. Preserving edges; solid disks represent edge points

triangles that are geometrically similar to the original ones, see [1].

We provide four examples demonstrating the strengths of our method:

- a continuous "cliff function," defined as

  $$f(x, y) = \tanh(9 * y - 9 * x) + 1)/90,$$

  where $x, y \in [-1, 1]$, is shown in Figures 10 and 11, sampled on a 80 × 80 uniform rectilinear grid,

- a discontinuous "disk function," defined as

  $$f(x, y) = \begin{cases} 1 & x^2 + y^2 = r^2, r \in [0, 1] \\ 0 & \text{Otherwise} \end{cases},$$

  where $x, y \in [-1, 1]$, is shown in Figure 12, sampled on a 80 × 80 uniform rectilinear grid,

- a continuous trigonometric vector-valued function, defined as

  $$\begin{aligned} \text{red}(x, y) &= (\cos 4\pi x) * (\cos 4\pi y), \\ \text{green}(x, y) &= (\cos 4\pi(x + .25)) * (\cos 4\pi(y - .25)), \\ \text{blue}(x, y) &= (\cos 4\pi(x + .33)) * (\cos 4\pi(y - .33)), \end{aligned}$$

  where $x, y \in [-1, 1]$, is shown in Figure 13, sampled on a 80 × 80 uniform rectilinear grid,

- a discrete San Francisco bay digital-elevation Model (DEM) data set, given as a 60 × 60 uniform rectilinear grid, shown in Figure 14. This data set consists of contoured color levels, each representing height above sea level. Blue represents sea level; red, green, and purple indicate increasing height.

The "cliff function" demonstrates the usefulness of data-dependent triangulations for approximating sampled data with long narrow "cliff regions." For this data set, the simple bisection scheme introduces artifacts in the Gouraud-shaded images. These artifacts consist of shading from high to low that does not reflect the orientation of the "cliff." The "disk function" is discontinuous. Our algorithm manages to represent this data set with fewer triangles than the bisection scheme due to its ability to preserve discontinuities.

When considering the trigonometric function we note that the bisection scheme performs slightly better than our scheme. In images from the bisection scheme (B) and (D) of Figure 13 we see artifacts due to the underlying triangulation.

4 Conclusions and future work

We have discussed a new technique for the construction of data-dependent triangulations for multi-valued, bivariate scattered data. Our scheme identifies and preserves discontinuities that might exist in a given data set. We have tested our method for various data sets and can conclude that one can achieve data approximations within some specified tolerance with much fewer triangles, in comparison to a simple bisection scheme.

Our algorithm has potential applications in scattered data approximation, visualization, and image compression. We plan to extend our method to scattered data in three dimensions and apply it to time-varying multi-valued data sets.

5 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 under ASCI ASAP Level-2 Memorandum Agreement B347878 and under Memorandum Agreement B503159; and the North Atlantic Treaty Organization (NATO) under contract CRG.971628 awarded to the University of California, Davis. We also acknowledge the support of Alstom/Shilling Robotics, Davis, California; Chevron; and Silicon Graphics, Inc. We 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] R. E. Barnhill and F. F. Little. Adaptive triangular cubature. *Rocky Mountain Journal of Mathematics*, 14(1):53–75, 1984.

[2] M. Bertolotto, L. D. Floriani, and P. Marzano. Pyramidal simplicial complexes. In C. Hoffmann and J. Rossignac, editors, *Third Symposium on Solid Modeling and Applications*, pages 153–162, New York, NY, 1995. Association for Computing Machinery, ACM Press.

[3] P. Cignoni, L. D. Floriani, C. Montani, E. Puppo, and R. Scopigno. Multiresolution modeling and visualization of volume data based on simplicial complexes. In *1994 Symposium on Volume Visualization*, pages 19–26, Los Alamitos, CA, 1994. IEEE Computer Society Press.

[4] A. D. DeRose, M. Lounsbery, and J. Warren. Multiresolution analysis for surfaces of arbitrary topological shape. Technical Report 93-10-05, Department of Computer Science and Engineering, University of Washington, Seattle, WA, 1993.

[5] N. Dyn, D. Levin, and S. Rippa. Data dependent triangulations for piecewise linear interpolation. *IMA Journal of Numerical Analysis*, 10:137–154, 1988.

[6] N. Dyn, D. Levin, and S. Rippa. Algorithms for the construction of data dependent triangulations. In J. C. Mason and M. G. Cox, editors, *Algorithms for Approximation II*, pages 185–192. Chapman and Hall, New York, NY, 1990.

[7] M. Eck, A. D. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In R. Cook, editor, *Proceedings of SIGGRAPH 1995*, pages 173–182, New York, NY, 1995. ACM Press.

[8] G. Farin. *Curves and Surfaces for CAGD*. Academic Press, San Diego, CA, fourth edition, 1997.

[9] L. D. Floriani. A pyramidal data structure for triangle-based surface description. *IEEE Computer Graphics & Applications*, 9(2):67–78, 1989.

[10] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics*. Addison Wesley Publishing Company Inc., Reading, MA, second edition, 1990.

[11] R. Franke. Scattered data interpolation: Tests of some methods. *Math. Comp.*, 38:181–200, 1982.

[12] T. S. Gieng, B. Hamann, K. I. Joy, G. L. Schussman, and I. J. Trotts. Constructing hierarchies for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 4(2):145–161, 1998.

[13] B. Hamann. A data reduction scheme for triangulated surfaces. *Computer Aided Geometric Design*, 11(2):197–214, 1994.

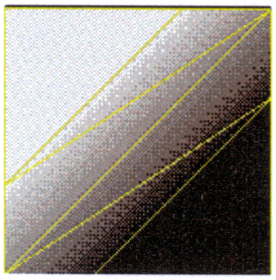
[14] B. Hamann and J. L. Chen. Data point selection for piecewise trilinear approximation. *Computer Aided Geometric Design*, 11(5):477–489, 1994.

[15] B. Hamann and J. L. Chen. Data point selectionfor piecewise linear curve approximation. *Computer Aided Geometric Design*, 11(3):289–301, 1994.

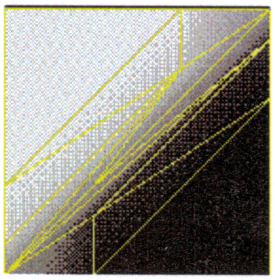
[16] B. Hamann, H. J. Thornburg, and G. Hong. Automatic unstructured grid generation based on iterative point insertion. *Computing*, 55(2):135–161, 1995.

[17] H. Hoppe. Progressive meshes. In H. Rushmeier, editor, *Proceedings of SIGGRAPH 1996*, pages 99–108, New York, NY, 1996. ACM Press.

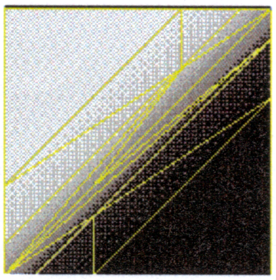
[18] M. Lounsbery. *Multiresolution Analysis for Surfaces of Arbitrary Topological Shape*. dissertation, Department of Computer Science and Engineering, University of Washington, Seattle, WA, 1994.



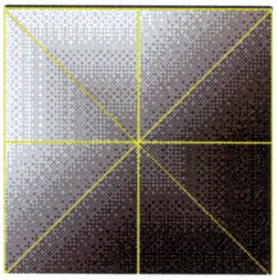
(a) 25% error (6 triangles)



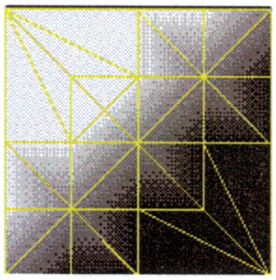
(b) 15% error (14 triangles)



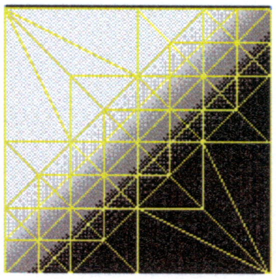
(c) 5% error (16 triangles)



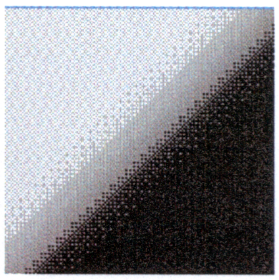
(d) 25% error (8 triangles)



(e) 15% error (30 triangles)

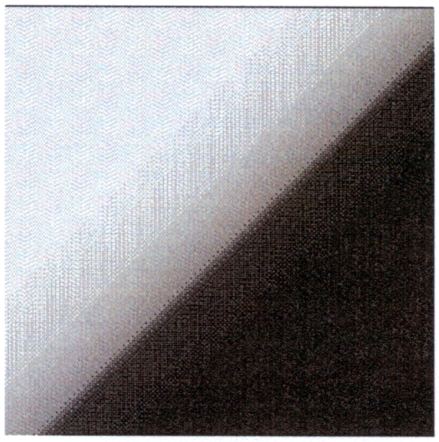


(f) 5% error (104 triangles)

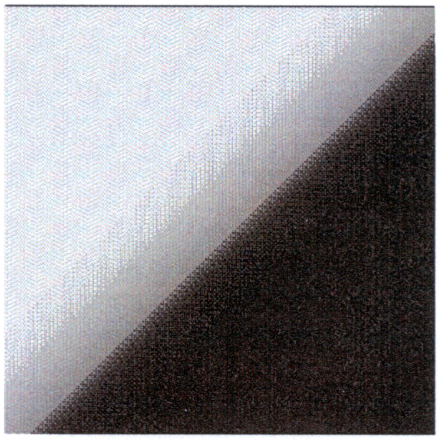


(g) Original dataset (12,800 triangles)

Figure 10. Cliff function examples, (a-c) data dependent method, (d-f) bisection method

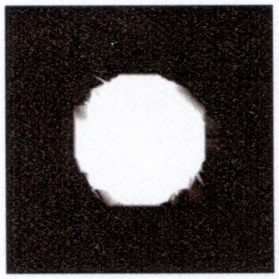


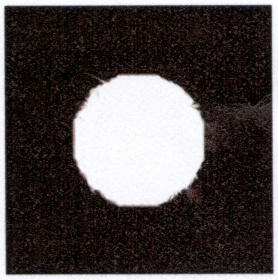
(a) Data Dependent Triangulation (16 Triangles)

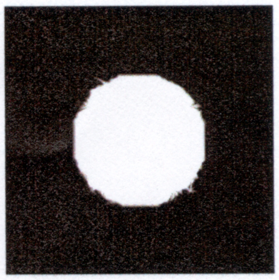


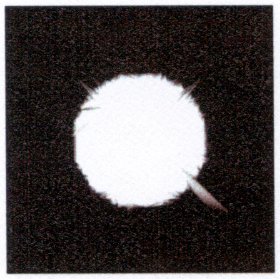
(b) Actual Data (12,800 Triangles)

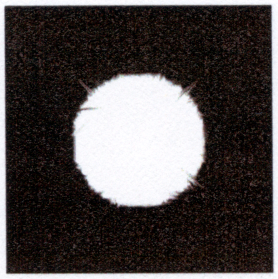
Figure 11. Comparison of actual and approximated meshes

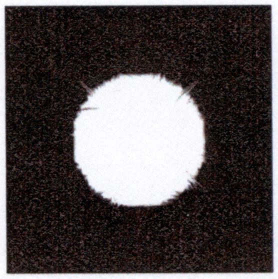

(a) 60% error (764 triangles)


(b) 30% error (956 triangles)


(c) 10% error (1178 triangles)


(d) 60% error (803 triangles)


(e) 30% error (1057 triangles)


(f) 10% error (2281 triangles)

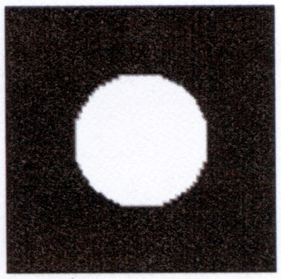
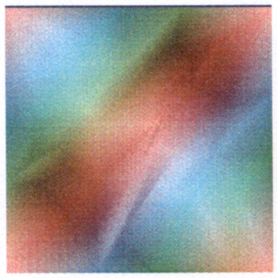
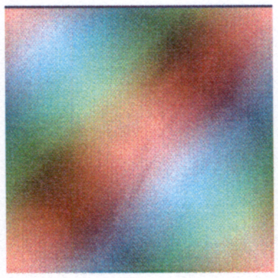
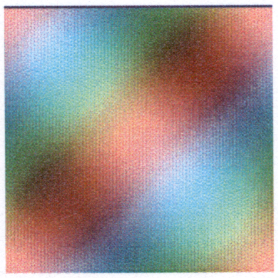
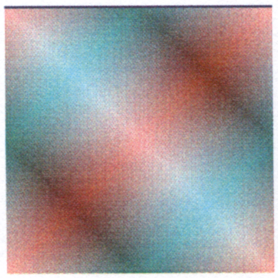

(g) Original dataset (12,800 triangles)

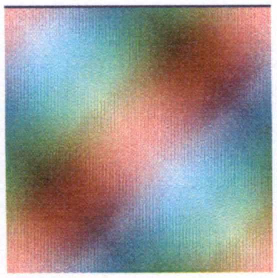
Figure 12. Disk function examples, (a-c) data dependent method, (d-f) bisection method

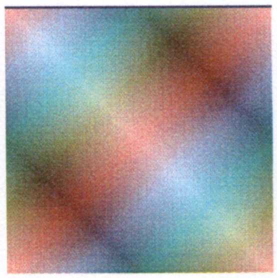

(a) 20% error (109 triangles)


(b) 10% error (233 triangles)


(c) 5% error (421 triangles)


(d) 20% error (16 triangles)


(e) 10% error (48 triangles)


(f) 5% error (202 triangles)

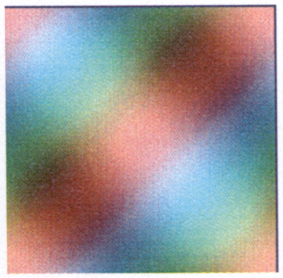
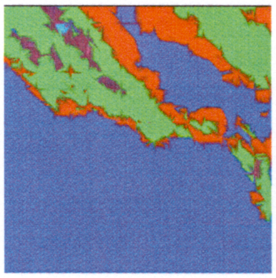
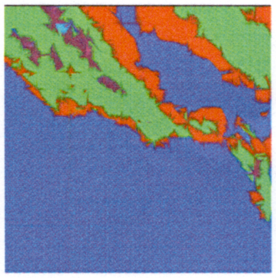

(g) Original dataset (12,800 triangles)

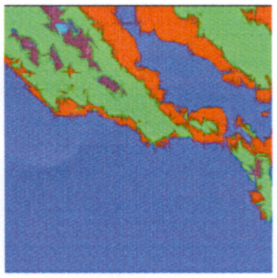
Figure 13. Continuous smooth function, (a-c) data dependent method, (d-f) bisection method



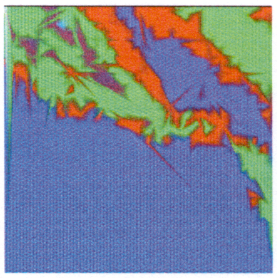
(a) 10% error (3355 triangles)



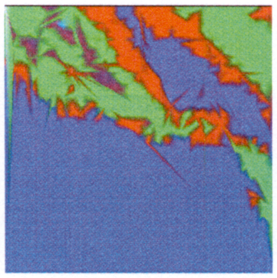
(b) 5% error (4019 triangles)



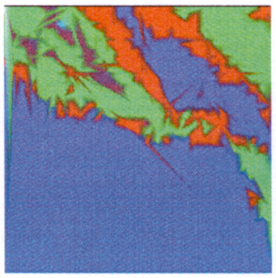
(c) 2% error (5252 triangles)



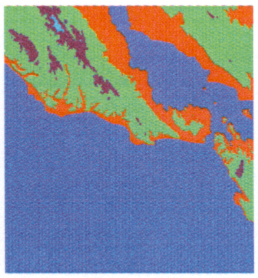
(d) 10% error (2402 triangles)



(e) 5% error (3633 triangles)



(f) 2% error (6213 triangles)



(g) Original dataset (12,800 triangles)

Figure 14. San francisco bay example, (a-c) data dependent method, (d-f) bisection method

[19] G. M. Nielson. Scattered data modeling. *IEEE Computer Graphics and Applications*, 13(1):60–70, 1993.

[20] G. M. Nielson and J. Tvedt. Comparing methods of interpolation for scattered volumetric data. In D. F. Rogers and R. A. Earnshaw, editors, *State of the Art in Computer Graphics*, pages 67–86. Springer-Verlag, 1993.

[21] F. P. Preparata and M. I. Shamos. *Computational Geometry*. Springer-Verlag, New York, NY, third printing edition, 1990.

[22] L. L. Scarlatos and T. Pavlidis. Hierarchical triangulation using terrain features. In *Proceedings IEEE Conference on Visualization '90 Proceedings*, pages 168–175, 1990.

[23] W. J. Schroeder, J. A. Zarge, and W. Lorensen. Decimation of triangle mesh. In *Proceedings of SIGGRAPH 1992*, pages 65–70, New York, NY, 1992. ACM Press.

[24] L. L. Schumaker. Computing optimal triangulations using simulated annealing. *Computer Aided Geometric Design*, 10(3-4):329–345, 1993.

[25] I. J. Trotts, B. Hamann, K. I. Joy, and D. F. Wiley. Efficient and robust simplification of tetrahedral meshes. In D. S. Ebert, H. Hagen, and H. E. Rushmeier, editors, *Visualization '98*, pages 287–295, Los Alamitos, California, 1998. IEEE Computer Society Press.