Using Quadratic Simplicial Elements for Hierarchical Approximation and Visualization

David F. Wiley[a], Henry R. Childs[b], Bernd Hamann[a], Kenneth I. Joy[a], and Nelson L. Max[a,c]

[a] Center for Image Processing and Integrated Computing (CIPIC), Department of Computer Science, University of CA, Davis, CA 95616-8562, U.S.A.;
email: {wiley, hamann, joy}@cs.ucdavis.edu

[b] B Division, Lawrence Livermore National Laboratory, Mail Stop L-098, 7000 East Avenue, Livermore, CA 94550, U.S.A.;
email: childs3@llnl.gov

[c] Center for Applied Scientific Computing (CASC), Lawrence Livermore National Laboratory, 7000 East Avenue, L-551, Livermore, CA 94550, U.S.A.;
email: max2@llnl.gov

ABSTRACT

Best quadratic simplicial spline approximations can be computed, using quadratic Bernstein-Bézier basis functions, by identifying and bisecting simplicial elements with largest errors. Our method begins with an initial triangulation of the domain; a best quadratic spline approximation is computed; errors are computed for all simplices; and simplices of maximal error are subdivided. This process is repeated until a user-specified global error tolerance is met. The initial approximations for the unit square and cube are given by two quadratic triangles and five quadratic tetrahedra, respectively. Our more complex triangulation and approximation method that respects field discontinuities and geometrical features allows us to better approximate data. Data is visualized by using the hierarchy of increasingly better quadratic approximations generated by this process. Many visualization problems arise for quadratic elements. First tessellating quadratic elements with smaller linear ones and then rendering the smaller linear elements is one way to visualize quadratic elements. Our results show a significant reduction in the number of simplices required to approximate data sets when using quadratic elements as compared to using linear elements.

Keywords: quadratic elements, higher-order finite elements, approximation, hierarchical approximation, data-dependent approximation, visualization, spline, refinement, multiresolution

1. INTRODUCTION

Scalar and vector field data often contain discontinuities that should be preserved for data approximation and visualization purposes. It is important to approximate domain boundaries—including geometry such as a car body, an aircraft, or a ship hull—and the locations of field discontinuities, represented by curves and surfaces, well. To better approximate these curves and surfaces we investigate the use of curved quadratic simplicial elements. (We do not address the problem of extracting discontinuities from a given scalar or vector field data set; we assume that this information is known.) We consider the cases of data (i.e., dependent field variables) defined over two-dimensional (2D) and three-dimensional (3D) domains.

We only consider curved simplicial elements that are quadratic. In the 2D case, we use curved triangles whose edges may be either straight-line segments or parabolae; in the 3D case, we use curved tetrahedral elements whose edges may be either straight-line segments or curves, and faces are either planar or curved triangles. Generally, we will refer to both non-curved and curved simplicial elements as just simplicial elements. We use a quadratic polynomial transformation to map the so-called *standard simplex* to the corresponding curved simplicial region in space. Furthermore, we use a quadratic polynomial defined over each simplicial element to locally approximate the dependent variable(s). We use curved elements with curved edges or faces to better approximate domain boundaries and discontinuities.

Our overall goal is the construction of a hierarchical data representation over 2D or 3D domains, using a best-approximation approach based on curved quadratic finite elements and quadratic polynomials defined over these elements. We start with a coarse decomposition of the domain, using a relatively small number of simplicial elements

and placing curved simplices in areas where domain boundaries or field discontinuities occur. We then compute a (globally) best least squares approximation, a quadratic spline approximation for the dependent variable(s) that is C^0-continuous. The physical loci of discontinuities play the same roles as domain boundaries: Two simplicial elements may share the same edge (face)—geometrically—defining the locus of a discontinuity, but the field function defined over the two elements is discontinuous along the shared edge (face). We compute local errors for each simplicial element, and we bisect a certain percentage of the elements with largest errors, update the simplicial domain decomposition accordingly, and compute a new best quadratic spline approximation. We iterate this process until a specified error condition is met or the number of simplicial elements exceeds some threshold.

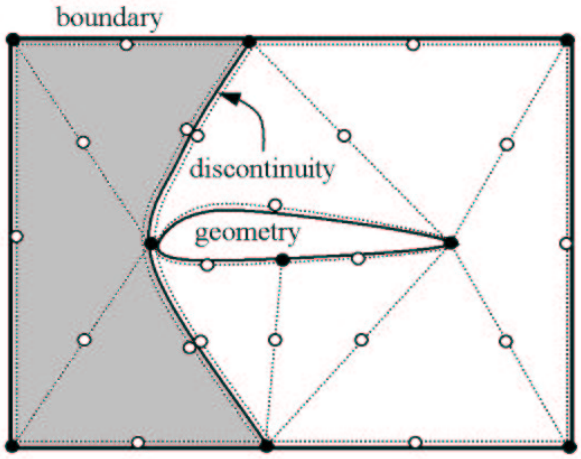

Figure 1. Decomposition of domain around a wing, using 2D curved quadratic simplices. There are two distinct regions in the field defined by the vertices and edges: the region left of the discontinuity (shaded) and the region right of the discontinuity. (The bullets and circles indicate the six polynomial coefficients associated with each element.)

Our approach belongs to the class of *refinement* methods. These methods are based on the principle of refining intermediate data approximations by inserting additional points or elements until a certain termination criterion is satisfied. We have developed our method with a focus on the needs of massive scientific data visualization, see [6]. To enable interactive visualization of massive data, it is possible to use either low-resolution best approximations everywhere or to adaptively "insert" high-resolution approximations locally into an otherwise relatively coarse approximation. Our approximation algorithm is based on these steps:

- **Initial simplicial domain decomposition.** Assuming that either a polygonal (polyhedral) or an analytical definition is known for all boundaries and field discontinuities, we construct a coarse simplicial decomposition of this domain. We use curved edges (faces) in areas where they are needed to better approximate curved boundaries or discontinuities. (The quadratic transformations, mapping the standard simplex defined in so-called *parameter space* to deformed simplices in so-called *physical space*, are defined by specifying corresponding point pairs in the two spaces such that one obtains a one-to-one, bijective mapping.) Figure 1 shows a possible initial simplicial decomposition of space around a wing cross section.
- **Best approximation.** In the 2D case, each simplicial element has six associated *knots* (having specific locations in the domain and associated coefficients), one knot per corner and one knot per edge. Six knots in parameter space are associated with six points in physical space, and this association defines the needed quadratic mapping for a simplex. (Accordingly, the number of knots is ten in the 3D case.) For simplicity, we consider only knots that are uniformly distributed along the edges of the standard simplex. We associate a quadratic polynomial with each simplicial element, which approximates the dependent variable(s) over the corresponding region in space. We represent each quadratic basis polynomial in so-called *Bernstein-Bézier form*, see [4]. Assuming that the function to be approximated is known in analytical form, it is possible to compute the unique best quadratic spline approximation defined as a linear combination of the quadratic basis functions. The best approximation, understood in a least squares sense, is the result of solving the *normal equations*, see [3].

- **Adaptive bisection.** We compute an error value for each simplicial element once a best approximation is computed. We use the L_2 norm to compute simplex-specific error values. The set of simplices is ordered according to the simplex-specific error values. To compute a next-level best quadratic approximation we determine a certain percentage of simplices with largest error values and bisect them by splitting them at the midpoint of their longest edge. If a simplex's longest edge is not unique, we choose the edge to be split randomly. In the case of curved edges, we use arc length to determine the longest edge to be bisected. Splitting a simplex into two simplices induces additional splits for all those simplices that share the split edge. We update a simplicial domain decomposition by considering all edge bisections and compute a new best approximation. We repeat the process of identifying simplices with largest errors, bisecting these simplices, and computing a new best approximation until we obtain an approximation for which the largest simplex-specific error is below a certain error threshold or until a maximal number of simplices is reached.
- **Hierarchical data representation.** To support level-of-detail visualization we can store multiple best approximations of different resolutions. For each best approximation, we need to store the polynomial coefficients of each simplicial element—for its shape and the field defined over it. Considering a non-curved simplicial element, we only need to store its three (four) corner points and the coefficients of the field defined over the element. Considering a curved element, we need to store all polynomial coefficients defining the shape of the element, in addition to the coefficients of the quadratic polynomial defining the field over the element. We store a fixed number of best approximations such that either the number of simplices increases in a specified fashion or the maximal simplex-specific error decreases in a certain way from one resolution to the next.

2. PREVIOUS WORK

Related work in the areas of hierarchical data representation and visualization is discussed in, for example, [11, 16]. Simplification methods are described in [9, 18]. Wavelet methods, in general, work well for rectilinear 2D and 3D grids and are described in [2, 14]. Refinement methods, similar to our method, are described in [7, 8]. Data-dependent triangulation schemes, i.e., schemes concerned with the construction of approximations using near-optimally shaped and placed simplicial elements, are described in [13]. Our method is also a *grid generation* method, and references for this area are [10, 17]. Finite element methods are discussed in [19].

3. MAPPING THE STANDARD SIMPLEX

In the 2D case, the standard simplex in parameters space is the triangle with corners (0, 0), (1, 0), and (0, 1). We associate a 2D quadratic Bernstein-Bézier polynomial $B^2_{i,j}(u,v)$, defined as

$$B^2_{i,j}(u,v) = \frac{2!}{(2-i-j)!\ i!\ j!}(1-u-v)^{2-i-j}u^i v^j,\ i,j \geq 0,\ i+j \leq 2, \qquad (1)$$

with each corner and midpoint of each edge. The six basis polynomials correspond to the six knots $\mathbf{u}_{i,j} = (u_{i,j}, v_{i,j}) = \left(\frac{i}{2}, \frac{j}{2}\right),\ i,j \geq 0,\ i+j \leq 2$ in parameter space, see Figure 2. We map the standard triangle to a

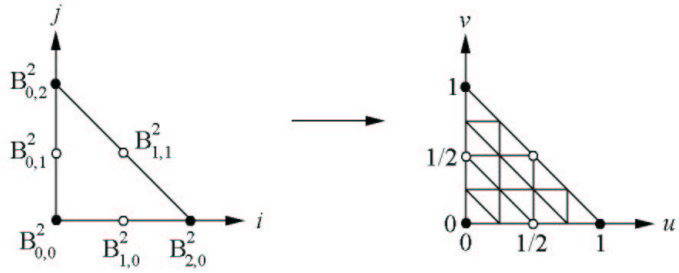

Figure 2. Correspondence between 2D basis functions $B^2_{i,j}$ in "index" space and knots (indicated by bullets and circles) in uv-parameter space.

curved triangular region in physical space by mapping the six knots $\mathbf{u}_{i,j}$ in parameter space to six corresponding points $\mathbf{x}_{i,j} = (x_{i,j}, y_{i,j})$ in physical space, using a quadratic mapping. The quadratic mapping is defined as

$$\mathbf{x}(\mathbf{u}) = \begin{pmatrix} x(u,v) \\ y(u,v) \end{pmatrix} = \sum_{i,j} \mathbf{b}_{i,j} B^2_{i,j}(\mathbf{u}) = \begin{pmatrix} \sum_{i,j} c_{i,j} B^2_{i,j}(u,v) \\ \sum_{i,j} d_{i,j} B^2_{i,j}(u,v) \end{pmatrix}, \; i, j \geq 0, \; i + j \leq 2 \,. \tag{2}$$

The mapping between parameter and physical space must be one-to-one. Figure 3 depicts a general mapping of the standard triangle in uv-parameter space to a curved triangle in xy-physical space.

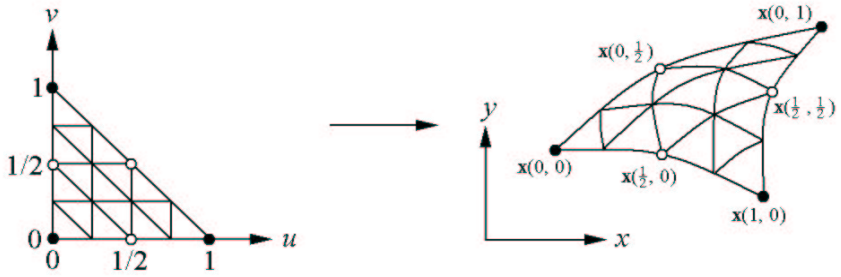

Figure 3. Correspondence between uv-parameter space and xy-physical space.

In the 3D case, the standard simplex is the tetrahedron with corners (0, 0, 0), (1, 0, 0), (0, 1, 0), and (0, 0, 1). We associate a 3D quadratic Bernstein-Bézier polynomial $B^2_{i,j,k}(u,v,w)$, defined as

$$B^2_{i,j,k}(u,v,w) = \frac{2!}{(2-i-j-k)!\; i!\; j!\; k!} (1-u-v-w)^{2-i-j-k} u^i v^j w^k, \; i, j, k \geq 0, \; i + j + k \leq 2 \,, \tag{3}$$

with each corner and midpoint of each edge. The ten basis polynomials correspond to the ten knots $\mathbf{u}_{i,j,k} = (u_{i,j,k}, v_{i,j,k}, w_{i,j,k}) = \left(\frac{i}{2}, \frac{j}{2}, \frac{k}{2}\right), \; i, j, k \geq 0, \; i + j + k \leq 2$ in parameter space. We map the standard tetrahedron to a curved tetrahedral region in physical space by mapping the ten knots $\mathbf{u}_{i,j,k}$ in parameter space to ten corresponding points $\mathbf{x}_{i,j,k} = (x_{i,j.k}, y_{i,j,k}, z_{i,j,k})$ in physical space, using a quadratic mapping. The quadratic mapping, in the 3D case, can be extended from 2. Figure 4 depicts the mapping of the standard tetrahedron in parameter space to a curved tetrahedron in physical space.

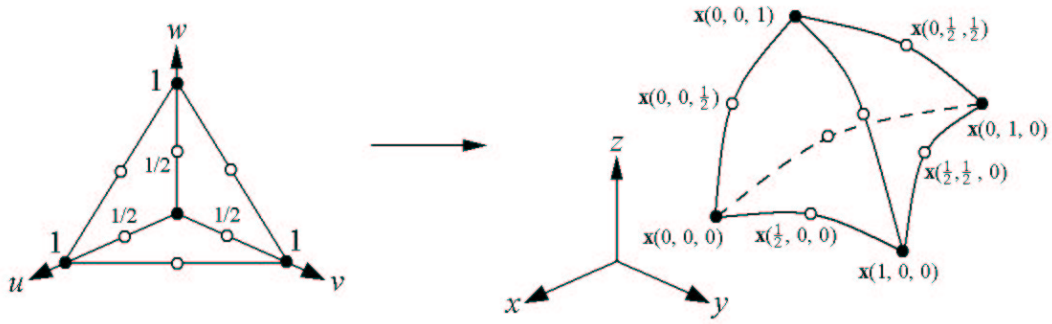

Figure 4. Mapping of standard tetrahedron in parameter space to curved tetrahedron in physical space.

4. INITIAL DOMAIN DECOMPOSITION

The main objective driving the development of our method is to construct a hierarchical representation of very large scientific data, where real-time and adaptive data visualization is crucial. Data sets resulting from computational simulations are typically defined on a grid, and the dependent variables are associated with either the vertices (also called *nodes*) or the elements defining the grid. Either of these types of data can be approximated by our method. The original grid, its boundaries, and possibly known locations of field discontinuities (in the dependent field variables) influence how we define an initial simplicial decomposition of the domain.

The objective is to initially represent the domain with a relatively small number of (curved) simplicial elements, using curved elements where they help to better approximate domain boundaries and (known) field discontinuities. We assume that we are provided with data representing field discontinuities and the domain boundary. This data is specified as a set of points and their connectivity. In the 2D case, the connectivity information defines linear splines. In the 3D case, the connectivity information defines linear spline (triangular) surfaces. Figure 5 shows a sample input and one desirable initial domain decomposition.

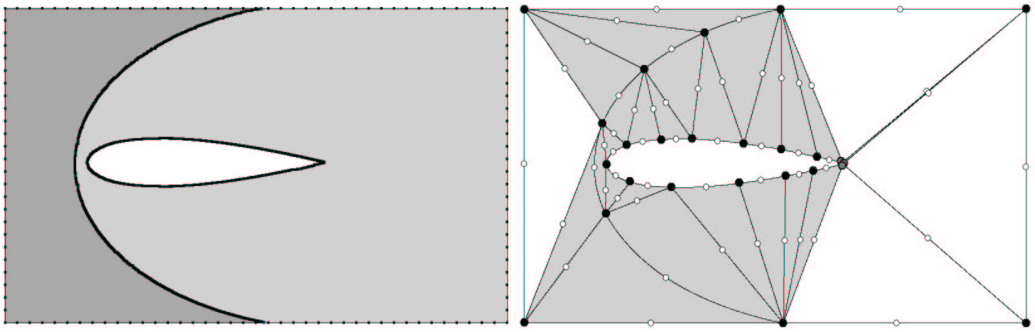

Figure 5. Sample input specification (left) and possible initial decomposition (right) utilizing curved (shaded) and non-curved simplices. Input points are shown as solid black squares; many points are used to represent the linear spline along the discontinuity between the dark-gray and light-gray regions (left image) and along the wing boundary. Bullets and circles represent corner and edge vertices of the elements in the initial decomposition.

We construct an initial decomposition by performing this sequence of steps: First, we classify the points; next, we determine connected paths; subsequently, we determine region boundaries; then, we determine a region hierarchy; next, we fit quadratic curves to the paths; and, finally, we triangulate (tetrahedralize) the regions. We describe these steps in more detail for the 2D case:

1) **Point classification.** We classify a point as *detached*, *hanging*, *normal*, or *junction* if it is connected to zero, one, two, or more than two edges, respectively. Detached points are considered extraneous and are ignored through the rest of the region identification process.
2) **Path determination.** In this step, the goal is to find the set of *paths* in the input data, where a path is defined as a sequence of two or more points beginning and ending with hanging or junction points and containing any number of normal points in-between. A *loop path* is a path beginning and ending with the same point—having at least one normal point in-between. To determine the paths, we first mark all the edges such that it is possible to determine if it has been traversed. Then, we select a seed edge at random and consider one of the endpoints; if the considered endpoint is normal, we traverse to the neighboring edge. We continue this traversal until the neighboring edge has already been traversed (forming a loop) or the edge's endpoint is classified as hanging or junction. If a loop path cannot be formed, we continue traversing in the opposite direction by considering the second endpoint of the seed edge. We then select a non-traversed edge at random and perform the same traversal. All paths are determined when all edges have been traversed. Figure 6 shows a sample classification and illustrates edge traversal.

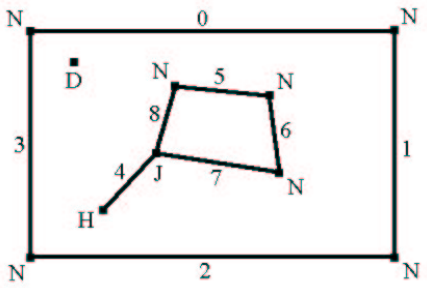

Figure 6. Point classification and path determination. Points are classified as detached (D), hanging (H), normal (N), and junction (J). There are three paths in this example, defined by the three edge sequences 0123, 4, and 8567.

3) **Determination of region boundaries.** In this step, we find the paths that bound the various regions. In general, a region is bounded by a sequence of paths, but, it is possible to have non-closed regions, which we call *crack regions*. We select a seed path at random and traverse the path in an arbitrary direction. If the endpoint in the direction we traverse is a junction, we make a right-hand turn—and consider all the paths that connect to this endpoint and select the path that contains the edge that most closely represents a right-hand turn by considering the incident angles. We repeatedly make right-hand turns until the seed path is reached. Then, we traverse the seed path in the same direction, making left-hand turns at junctions. These two traversals, in general, yield two distinct region boundaries. It is possible to trace the same region boundary twice; if this is the case, we discard one of the results. A crack region is formed when the resulting sequence of paths is not closed, i.e., when the seed path cannot be reached by only traversing the obtained sequence either forward or backward. If a hanging point is encountered during traversal, we perform a U turn and continue the traversal over the already traversed path(s). Once a not-yet-traversed path is reached, we "cut off" the sequence of paths that was traversed twice, forming a *crack region*. We then continue normal traversal, but, due to a possible U turn, we may encounter already traversed paths. When this is the case, we form additional crack regions by applying a cut-off step when a non-traversed path is reached. We must take special care when the traversal returns to a path that has already been traversed. This case leads to the formation of an *island region*. In this case, a closed region is formed, and we cut off the sequence that defines the island and continue traversal. Figure 7 shows an example.

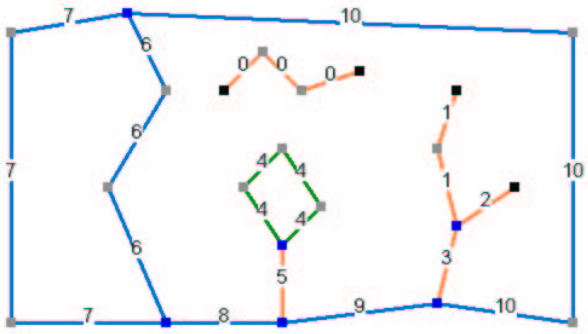

Figure 7. Region boundary construction. Points are classified as hanging (black), normal (gray), and junction (blue). Edges are labeled according to the path to which they belong. Blue paths (6, 7, 8, 9, and 10) denote closed region boundaries. Orange paths (0, 1, 2, 3, and 5) indicate crack regions. The green path (4) indicates an island region. The regions are defined by path sequences 0, 1-2-3, 4, 5, 6-10-9-8, and 6-7.

4) **Determination of region hierarchy.** To determine which regions are inside other regions, we sort the regions based on size of their bounding boxes. Starting with the minimum-area region, we iteratively test whether any non-shared point is contained in the next-larger-area region. If this is not the case, then we proceed to the next-larger-area region, etc. Once we have found a containing (parent) region, we add the smaller region as a child to the parent.
5) **Fitting curves to paths.** We fit quadratic curves to the paths by finding sequences of points that curve in the same direction. (We limit the angle by which a sequence of points can curve, to prevent "spiral" cases.) Points that are co-linear are fitted exactly by a quadratic curve. To preserve the polygonal domain boundary exactly, boundary paths are not approximated. Quadratic curves are fitted to points using least squares approximation. We perform the approximation relative to the axis formed by the endpoints of a sequence. For simplicity, we only consider quadratic

curves whose three control points are located at the endpoints and at the midpoint of the axis. Error is determined by the L_2 norm of the distance between the fitted curve and the points that define the piecewise linear spline being approximated.

6) **Triangulating regions.** We compute the initial domain decomposition in the following manner: We traverse the region hierarchy from top to bottom (largest region to smallest children) and triangulate the space between each region and its first-level child region(s). We form a constrained Delaunay triangulation using the piecewise linear spline defined by the sequence of control points for the quadratic curves that approximate the paths that define region boundaries. Once triangulated, we remove triangles that lie outside the region and inside child regions. Then, we construct curved triangles by considering the relationship between the curve control points and the triangles in the triangulation. Figure 8 shows an example of converting a linear triangulation of control points to a set of curved triangles.

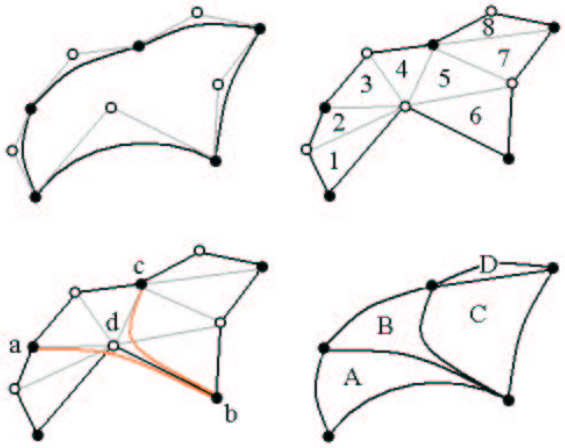

Figure 8. Triangulation of 2D region. We begin with a set of points defining the five control polygons of the quadratic Bézier curves associated with the paths that define the region to be triangulated (upper-left corner); bullets and circles denote end and middle control points, respectively. We construct a triangulation for the control points (upper-right corner). Then, we consider pairs of triangles, such as 1-2 and 3-4, whose shared edges have endpoints that are middle control points (circles). We form new curves (lower-left corner, orange)—using a-d-b and b-d-c as control polygons—to define appropriate edges for the curved triangles A and B (lower-right corner) that replace pairs 1-2 and 3-4, respectively. Triangle 8 is converted into D. We complete the triangulation by converting triangle triple 5-6-7 into curved triangle C.

In the 3D case, the input to our algorithm is a set of points and an associated triangulation(s). The various steps of our method can be extended and can be described on a high level as follows:

1) **Point and edge classification.** We classify a point as detached, hanging, or normal if it is connected to zero, one, or more than one edges, respectively. We classify an edge as detached, hanging, normal, or junction if it is connected to zero, one, two, or more than two triangles, respectively. Detached points and edges are considered extraneous and are ignored through the rest of the region identification process.
2) **Patch determination.** We grow a set of *patches* that are "planar" such that all the normal vectors of the member triangles are within a specified tolerance. (All triangles in a patch are connected by normal edges.)
3) **Determination of region boundaries.** We traverse patches across their boundaries to neighboring patches and make right- and left-hand turns to determine the region boundaries.
4) **Determination of region hierarchy.** We use the bounding box volume of region boundaries as our sorting criterion and perform an iterative merging step that constructs the hierarchy.
5) **Fitting curved surfaces to patches.** We fit curves to the boundary edges of the patches. Then, using the 2D approximation method in the following section, we fit curved surfaces to the patches.
6) **Tetrahedralization.** We tetrahedralize the control net for the regions and convert the resulting mesh into a curved tetrahedral mesh to yield an initial tetrahedralization.

5. BEST APPROXIMATION

We assume that the field to be approximated over a domain is known analytically. Should this not be the case, e.g., in the case of *scattered data*, it is possible to construct an analytical representation by performing a prior data interpolation or approximation step, see [5]. In the case that a data set is defined on a grid, the required analytical definition is given by a piecewise linear function for a simplicial (triangular or tetrahedral) grid and a piecewise bilinear (trilinear) function in the case of quadrilateral (hexahedral) grid cells. We denote the analytical function to be approximated by $F(\mathbf{x})$. Based

on an initial domain decomposition, we compute the corresponding best piecewise quadratic approximation of $F(\mathbf{x})$ by solving the normal equations, see [3]. The normal equations determine the set of coefficients for the desired quadratic spline representation.

Corner vertices of simplicial elements may be shared by any number of simplices, and we denote the basis function associated with a corner vertex $\mathbf{v}_i$ by $f_i(\mathbf{x})$. An edge of a simplicial element may be shared by no more than two simplices in the 2D case and by an arbitrary number of simplices in the 3D case. We denote a basis function associated with the midpoint of a simplex edge e_j by $g_j(\mathbf{x})$. We refer to the set of simplices sharing a common corner vertex as the *platelet* of this corner, and we call the set of simplices sharing a common edge *edge neighbors*. Thus, a platelet defines the region in space over which a basis function associated with the corresponding corner vertex is non-zero. Edge neighbors, associated with a particular edge, define the region in space over which a basis function associated with this edge is non-zero. Figure 9 shows the two types of basis functions that arise in the bivariate case.

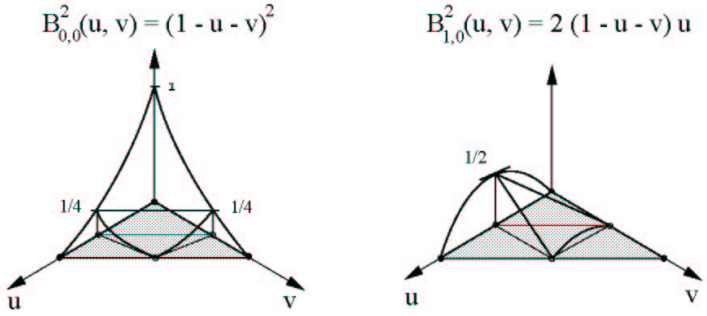

Figure 9. Types of bivariate basis functions associated with corner (left) and edge (right).

We denote a best approximation by $a(\mathbf{x})$, and we write it as a linear combination of the basis functions associated with all distinct simplex corners (f_i) and simplex edges (g_j). Assuming that there are m distinct corners and n distinct edges, we can write a best approximation as the sum

$$a(\mathbf{x}) = \sum_{i=1}^{m} c_i f_i(\mathbf{x}) + \sum_{j=1}^{n} d_j g_j(\mathbf{x}). \tag{4}$$

We must solve the normal equations to obtain the unknown coefficients c_i and d_j. In matrix form, the normal equations are

$$\begin{bmatrix} \langle f_1, f_1\rangle & \cdots & \langle f_1, f_m\rangle & \langle f_1, g_1\rangle & \cdots & \langle f_1, g_n\rangle \\ \vdots & & \vdots & \vdots & & \vdots \\ \langle f_m, f_1\rangle & \cdots & \langle f_m, f_m\rangle & \langle f_m, g_1\rangle & \cdots & \langle f_m, g_n\rangle \\ \langle g_1, f_1\rangle & \cdots & \langle g_1, f_m\rangle & \langle g_1, g_1\rangle & \cdots & \langle g_1, g_n\rangle \\ \vdots & & \vdots & \vdots & & \vdots \\ \langle g_n, f_1\rangle & \cdots & \langle g_n, f_m\rangle & \langle g_n, g_1\rangle & \cdots & \langle g_n, g_n\rangle \end{bmatrix} \begin{bmatrix} c_1 \\ \vdots \\ c_m \\ d_1 \\ \vdots \\ d_n \end{bmatrix} = \begin{bmatrix} \langle F, f_1\rangle \\ \vdots \\ \langle F, f_m\rangle \\ \langle F, g_1\rangle \\ \vdots \\ \langle F, g_n\rangle \end{bmatrix}, \tag{5}$$

where $\langle G, H\rangle$ denotes the inner product of the functions $G(\mathbf{x})$ and $H(\mathbf{x})$, i.e.,

$$\langle G, H\rangle = \int_{\text{Common Domain of } G \text{ and } H} G(\mathbf{x}) H(\mathbf{x}) d\mathbf{x}. \tag{6}$$

In our construction, we must compute inner products over simplices. Since all simplicial elements in physical space are defined by quadratic mappings of the standard simplex, we can simplify integration by making use of the *change-of-*

variables theorem, see [12], which relates integration in physical space to integration in parameter space. In the 2D case, integrals are computed according to the formula

$$\int_{\text{Physical Simplex}} \mathrm{G}(\mathrm{x},\mathrm{y})\,dxdy = \int_{\text{Standard Simplex}} \mathrm{G}(\mathrm{x}(\mathrm{u},\mathrm{v}),y(u,v))\,J(u,v)\,dudv, \tag{7}$$

where $J(u,v)$ denotes the *Jacobian* associated with the mapping of the standard simplex to the corresponding simplex in physical space. The Jacobian is the determinant

$$J(u,v) = \begin{vmatrix} \frac{\partial}{\partial u}x(u,v) & \frac{\partial}{\partial v}x(u,v) \\ \frac{\partial}{\partial u}y(u,v) & \frac{\partial}{\partial v}y(u,v) \end{vmatrix}. \tag{8}$$

The 3D case is a straightforward extension.

The matrices involved in the best-approximation step are sparse, since all basis functions have local support. Several methods exist for bandwidth reduction, efficient factorization, and inversion of such sparse matrices, for example, see [15]. We use an efficient sparse matrix representation and system solver to compute the coefficients in linear time.

The computation of the inner products appearing in the normal equations requires us to integrate over simplicial elements. While the change-of-variables theorem reduces this integration to integration over the standard simplex, we still need to perform relatively expensive numerical integration for the calculation of the inner products appearing on the right-hand side of the normal equations, i.e., the integrals of the types $\langle F, f_i \rangle$ and $\langle F, g_j \rangle$. Since $F(\mathbf{x})$ can, in general, be any analytically defined function, numerical integration can become expensive. We use *Romberg integration* for the computation of these right-hand-side inner products, see [1, 8].

Once we have computed a best approximation for a particular simplicial domain decomposition, we analyze the local approximation quality to identify simplices that should be refined (bisected) to further improve approximation quality. In the following section, we discuss adaptive bisection.

6. ADAPTIVE BISECTION

For each simplicial element S_i in a particular domain decomposition, we compute a local approximation error e_i. We define this error as

$$e_i = \left(\int_{S_i} \left(F(\mathbf{x}) - a(\mathbf{x})\right)^2 d\mathbf{x} \right)^{\frac{1}{2}}. \tag{9}$$

Selecting and bisecting simplices of maximal error are the two steps used to refine the mesh. In general, we choose a certain percentage of the simplices to be refined.

We refine a simplicial element by bisecting at the midpoint of its longest edge, using arc length. All simplices sharing the split edge are bisected to avoid "hanging nodes." The bisection step is shown in Figure 10. Bisection steps lead to new simplicial domain decompositions, and we must compute new best quadratic spline approximations for each one.

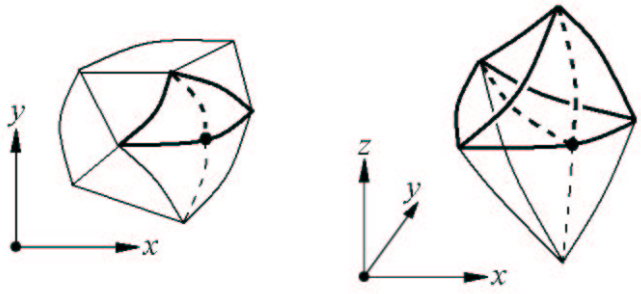

Figure 10. Bisection of simplices in 2D and 3D cases. The darker simplices are the ones selected for bisection.

We continue to bisect a certain percentage of simplices until either the number of simplices in a decomposition exceeds some user-specified maximal number or until an approximation is obtained whose global error is less than a user-specified tolerance. (We define the global error of an approximation as the sum of all local simplex errors.) The final result of our method is a set of independent "levels" of best quadratic spline approximations that can be used for interactive and/or adaptive, level-of-detail visualization.

7. RESULTS

We have tested our method for several test data. We visualize quadratic simplices by tessellating them using many linear elements. Figure 11 compares a curved quadratic approximation to a linear approximation. Figure 12 compares a curved quadratic approximation to a linear approximation. Figure 13 shows different stages of construction for a 3D initial decomposition. Results were computed on a 1.8GHz Pentium IV graphics workstation with 512MB of main memory.

8. CONCLUSIONS

Curved simplicial elements can be used to more compactly approximate data than non-curved simplicial elements. In general, the use of higher-order simplices should be considered as they can produce better-quality approximations with a smaller number of simplices.

Higher-order simplices are growing in importance in visualization as researchers are also using them more frequently for domain decomposition in numerical simulations. Thus, visualization of these simplices is also important because of their increasing popularity. Direct higher-order visualization techniques, including volume visualization techniques, must be developed to take advantage of higher-order elements.

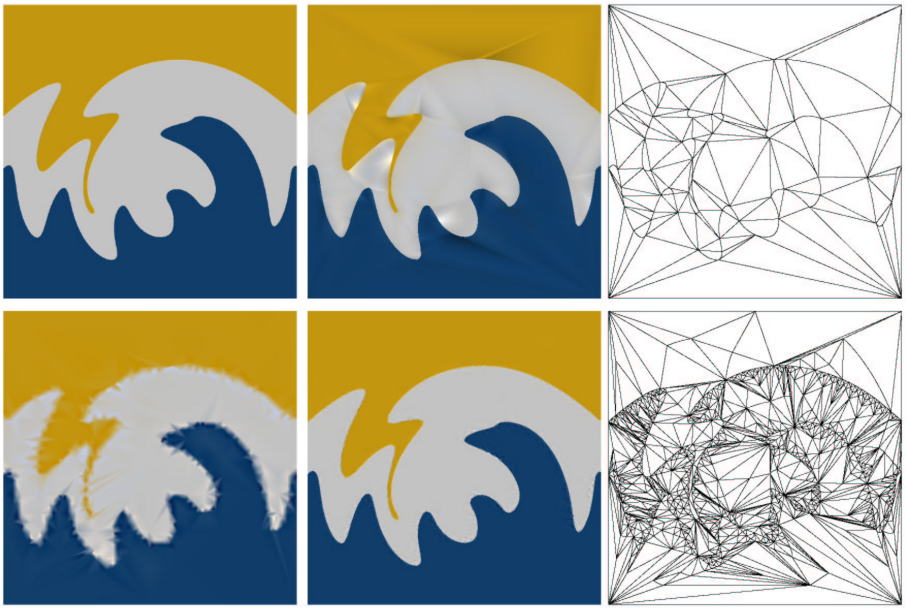

Figure 11. Comparison of a curved quadratic approximation to a linear approximation. Original image, consisting of 800x800 pixels, is shown in the upper-left corner. One linear approximation is shown in the lower-left corner. Two curved quadratic approximations are shown in the middle; their corresponding triangulations are shown on the right side. The linear approximation has 754 knots, 1483 linear simplices, an error of 1477.62, and a computation time of 42 seconds. The top quadratic approximation has 411 knots, 134 simplices, and an error of 24.99. The bottom quadratic approximation has 3392 knots, 1473 simplices, and an error of 0.07. Computation times were 24 and 82 seconds, respectively.

Figure 12. Comparison of a curved quadratic approximation to a linear approximation. Original image, consisting of 1536x1024 pixels, is shown in the upper-left corner. One linear approximation is shown in the upper-right corner. Two curved quadratic approximations are shown on the bottom. The linear approximation has 5816 knots, 11482 linear simplices, an error of 6120.42, and a computation time of 535 seconds. The left quadratic approximation has 12235 knots, 4066 simplices, and an error of 90.03. The right approximation has 63928 knots, 29355 simplices, and an error of 1.21. Computation times were 284 seconds and 1206 seconds, respectively.

Figure 13. Different stages of construction for a 3D initial decomposition. The given triangulation is shown on the left side. (The domain boundary has been removed for clarity.) Patches are shown in the middle. Boundaries of the patches are shown on the right side.

9. ACKNOWLEDGEMENTS

This work was performed under the auspices of the U.S. Department of Energy by University of CA Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48. This work was supported by the National Science Foundation under contract ACI 9624034 (CAREER Award), through the Large Scientific and Software Data Set Visualization (LSSDSV) program under contract ACI 9982251, and through the National Partnership for Advanced Computational Infrastructure (NPACI); the National Institute of Mental Health and the National Science Foundation

under contract NIMH 2 P20 MH60975-06A2; the Army Research Office under contract ARO 36598-MA-RIP; and the Lawrence Livermore National Laboratory under ASCI ASAP Level-2 Memorandum Agreement B347878 and under Memorandum Agreement B503159. We also acknowledge the support of ALSTOM Schilling Robotics and SGI. We thank the members of the Visualization and Graphics Research Group at the Center for Image Processing and Integrated Computing (CIPIC) at the University of CA, Davis. Furthermore, we acknowledge the support received by the members of B Division, Lawrence Livermore National Laboratory.

10. REFERENCES

1. Boehm, W. and Prautzsch, H. (1993), *Numerical Methods*, A K Peters, Ltd., Wellesley, MA.
2. Bonneau, G. P. (1999), Optimal triangular Haar bases for spherical data, in: Gross, M., Ebert, D. S. and Hamann, B., eds., *Visualization '99*, IEEE Computer Society Press, Los Alamitos, CA, pp. 279–284.
3. Davis, P. J. (1975), *Interpolation and Approximation*, Dover Publications, Inc., New York, NY.
4. Farin, G. (2002), *Curves and Surfaces for Computer Aided Geometric Design*, fifth edition, Academic Press, San Diego, CA.
5. Franke, R. (1982), Scattered data interpolation: Tests of some methods, Math. Comp. 38, pp. 181–200.
6. Hagen, H., Müller, H. and Nielson, G. M., eds. (1993), *Focus on Scientific Visualization*, Springer-Verlag, New York, NY.
7. Hamann, B. and Jordan, B. W. (1998), Triangulations from repeated bisection, in: Dæhlen, M., Lyche, T. and Schumaker, L. L., eds., *Mathematical Methods for Curves and Surfaces II*, Vanderbilt University Press, Nashville, TN, pp. 229–236.
8. Hamann, B., Jordan, B. W. and Wiley, D. F. (1999), On a construction of a hierarchy of best linear spline approximations using repeated bisection, IEEE Transactions on Visualization and Computer Graphics 5(1), pp. 30–46; 5(2), p. 190 (errata).
9. Hoppe, H. (1997), View-dependent refinement of progressive meshes, in: Whitted, T., ed., *Proceedings of SIGGRAPH 1997*, ACM Press, New York, NY, pp. 189–198.
10. Knupp, P. M. and Steinberg, S. (1993), *Fundamentals of Grid Generation*, CRC Press, Boca Raton, FL.
11. Kreylos, O. and Hamann, B. (1999), On simulated annealing and the construction of linear spline approximations for scattered data, in: Gröller, E., Löffelmann, H. and Ribarsky, W., eds., *Data Visualization '99* (Proc. EUROGRAPHICS-IEEE TCCG Symposium on Visualization), Springer-Verlag, Vienna, Austria, pp. 189–198.
12. Marsden, J. E. and Tromba, A. J. (1988), *Vector Calculus*, third edition, W. H. Freeman and Company, New York, NY.
13. Nadler, E. (1986), Piecewise linear best L_2 approximation on triangulations, in: Ward, J. D., ed., *Approximation Theory V*, Academic Press, Inc., San Diego, CA, pp. 499–502.
14. Nielson, G. M., Jung, I. H. and Sung, J. (1997a), Haar wavelets over triangular domains with applications to multiresolution models for flow over a sphere, in: Yagel, R. and Hagen, H., eds., *Visualization '97*, IEEE Computer Society Press, Los Alamitos, CA, pp. 143–149.
15. Press, W. H., Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P. (1992), *Numerical Recipes in C*, second edition, Cambridge University Press, New York, NY.
16. Staadt, O. G., Gross, M. H. and Weber, R. (1997), Multiresolution compression and reconstruction, in: Yagel, R. and Hagen, H., eds., *Visualization '97*, IEEE Computer Society Press, Los Alamitos, CA, pp. 337–346.
17. Thompson, J. F., Soni, B. K. and Weatherill, N. P., eds. (1999), *Handbook of Grid Generation*, CRC Press, Boca Raton, FL.
18. Xia, J. C. and Varshney, A. (1996), Dynamic view-dependent simplification for polygonal meshes, in: Yagel, R. and Nielson, G. M., eds., *Visualization '96*, IEEE Computer Society Press, Los Alamitos, CA, pp. 327–334.
19. Zienkiewicz, O. C. (1977), *The Finite-Element Method in Engineering Science*, McGraw-Hill, London, United Kingdom.