Mesh Displacement: An Improved Contouring Method for Trivariate Data

Doug Moore *
Rice University

Joe Warren †‡
Rice University

August 22, 1995

Abstract

Trivariate data arise in a wide range of industrial and medical applications. Some of the most powerful techniques for visualizing these data involve computing piecewise linear approximations to isocontours of the data [WMW86, LC87]. This paper describes a simple optimization that may be applied to the piecewise linear approximations produced by these techniques. The benefits of this optimization include:

- Reducing the number of triangles in the resulting approximation by 40% to 50% without a significant loss of accuracy in the approximation.
- Improving the shape of the remaining triangles while eliminating badly shaped triangles produced by current methods.
- Enabling simple computation of the topology of the resulting mesh.

Keywords - computer graphics, medical imaging, surface reconstruction, visualization

1 Introduction

Scalar valued data defined over a rectangular, three-dimensional grid appear in a wide range of applications. In medical imaging, both computed tomography (CT) and magnetic resonance (MR) data usually take the form of density values spaced over a cubic grid [LC87]. In graphics, trivariate functions are often used to define "fuzzy" or "soft" objects [Bli82, WMW86]. During rendering, these functions are generally sampled over a regular cubic grid [Blo88].

Among the most popular methods for visualizing scalar data over a cubic grid is by computing *isocontours* of the data. Typically, the data are used to define some continuous

*Supported in part by Center for Research in Parallel Computation

†Supported in part by NSF grant CCR 89-03431

‡Authors' address: Department of Computer Science, P.O. Box 1892, Houston, Tx 77251

function $F(x, y, z)$ that interpolates the data. An isocontour of F is simply the set of all points (x, y, z) such that

$$F(x, y, z) = c.$$

Displaying several isocontours of F for appropriate values of c is often an effective tool for understanding the behavior of the original trivariate data. In general, the problem of computing an isocontour $F = c$ can be transformed trivially into the simpler problem of computing the zero contour of $F - c = 0$. Therefore, we focus on approximating the zero contour of scalar data specified over a cubic grid.

Several authors have suggested similar methods for creating zero contours. Their methods process the data separately on each cube of the cubic grid, and use linear interpolation along the edges of a cube to compute a collection of points lying on the zero contour. In the original "marching cubes" algorithm, these intersections are connected to form edges and triangles using a table lookup based on the signs of the data at the vertices of the defining cube [LC87]. Figure 1 illustrates a two-dimensional version of this technique. Note that the rightmost case is ambiguous and could also be triangulated as shown by the dotted lines. Figure 2 presents a table for the three-dimensional case like the one in the original marching cubes paper. Unfortunately, this particular method does not guarantee a contour that is continuous, because adjacent cubes that share a face like the rightmost case in figure 1 may subdivide differently [Dür88]. Others have suggested an alternative method that disambiguates that case by sampling the function at the center of the ambiguous face [WMW86]. We call methods like these, that compute the vertices of the resulting contour using linear interpolation along edges of the cubic mesh, *edge-based interpolation* methods.

Figures 8(a)-11(a) show several surface contours created using edge-based interpolation methods. One drawback of these methods is that the surface meshes they produce can be highly irregular, even for simple trivariate data. These irregularities consist of tiny triangles, produced when the contour passes near a vertex of the cubic mesh, and narrow triangles, produced when the contour passes near an edge of the mesh. In our experience, such triangles can account for up to 50% of the triangles in some surface meshes. These badly shaped elements often degrade the performance of rendering algorithms and finite element analysis

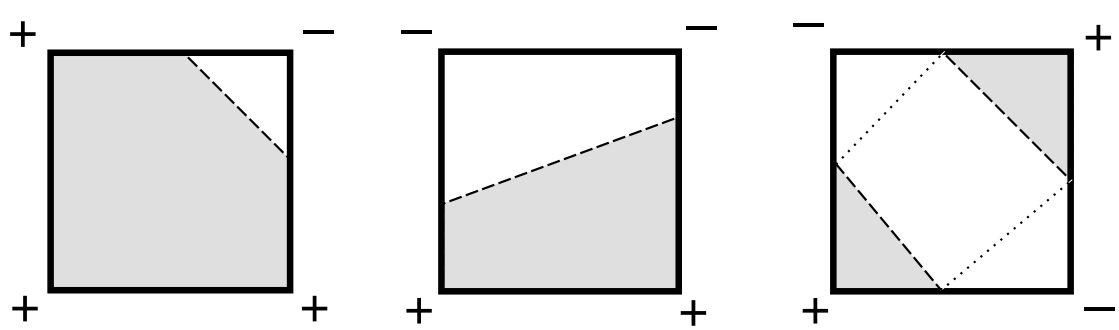

Figure 1 Edge-based interpolation in two dimensions

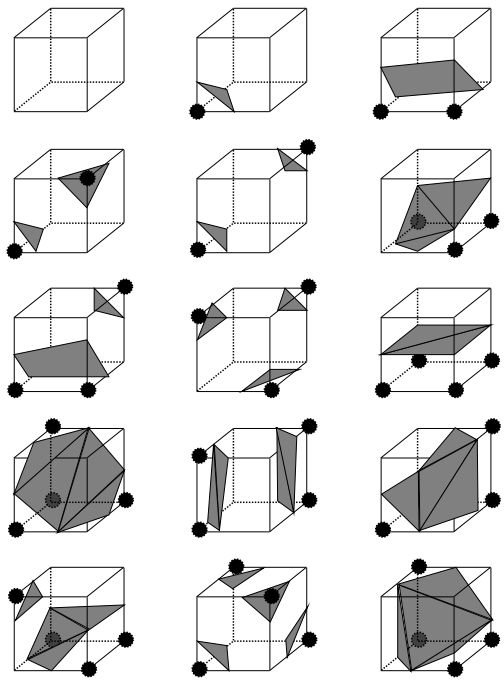

Figure 2 Edge-based interpolation in three dimensions

applied to the mesh while contributing little to the overall accuracy of the approximation.

This paper describes a method for removing these badly shaped triangles from the resulting mesh with little loss of accuracy in the resulting approximation. This method takes as input a surface contour produced by an edge-based interpolation method and produces a new surface contour free of badly shaped triangles. The number of triangles in the new contour is often reduced by as much as 50%.

2 Mesh Displacement

The algorithm described here is a generalization of the mesh displacement algorithm that we first described in a previous paper [MW90]. This algorithm is a contouring method for scalar data specified over a simplicial mesh. Instead of producing a piecewise linear contour using linear interpolation, vertices of the simplicial mesh are displaced to lie on or near the appropriate contour. The result is a new simplicial mesh whose set of faces contain as a subset an approximation to the appropriate contour. We propose to generalize this method to cubic meshes.

2.1 The method

We begin with some definitions. Let S be a surface mesh that is piecewise linear over a cubic mesh C with the property that the vertices of S lie on the edges of C. We say that S is *embedded* in C. By definition, edge-based interpolation methods produce embedded surface

meshes. Given a vertex v of the cubic mesh C, the convex hull of those points on the edges of C lying nearest to v is called the *vertex orbit* of v and denoted $N(v)$. For vertices in the interior of C, the vertex orbits form a cubic grid of octahedra that touch only at their vertices. Those points where S intersects the edges of C inside the vertex orbit of v are the *satellites* of v.

The following technique, called the mesh displacement algorithm, eliminates small and narrow triangles in S by displacing vertices of C to lie on or near S.

1. For each triangle in S,
    if S intersects the orbits of three distinct vertices
        then produce a triangle connecting these three vertices
        else the triangle collapses to a vertex or edge
2. For each vertex v of C,
    if S intersects the vertex orbit of v
        then displace v to the centroid of its satellites

The first step of the method defines the topology of a new mesh connecting vertices of C. All the vertices of S lying in the same vertex orbit are coalesced into a single vertex in the resulting mesh. Thus, small triangles lying in a single vertex orbit are collapsed to a vertex. Narrow triangles connecting two vertex orbits are collapsed to an edge. Figure 3 illustrates this for the two dimensional case. This perspective shows that if the original surface mesh S is continuous, then the topological mesh produced in the first step of the algorithm must also be continuous.

In the second step, the vertices of the topological mesh are displaced to lie on or near the original embedded surface mesh. Since each new vertex position is chosen to be at the centroid of a small cluster of points lying on S, the new approximation usually diverges only slightly from S. If S itself is an approximation to a smooth surface arising from some other representation, information from that representation can be used to reposition the vertex to lie precisely on the original smooth surface.

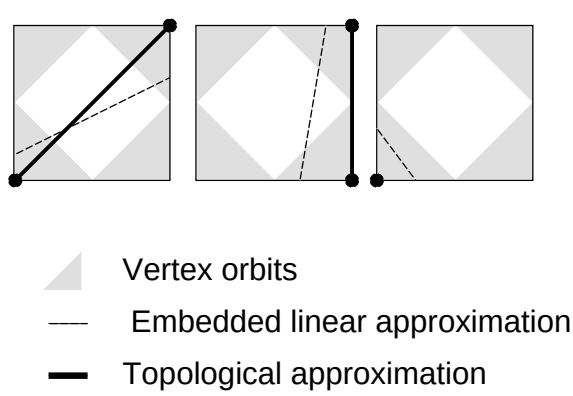

Figure 3 Two dimensional case table for mesh displacement

Figure 4 illustrates this method applied to a two-dimensional mesh. The upper portion illustrates the result of the first step. The lower portion illustrates the output of the second step. The short edges in the upper portion of the figure have been collapsed to form vertices in the lower portion.

2.2 Experimental results

The mesh displacement method of the previous section has been coded and run on a range of test data. Figure 8(a) shows the zero contour produced by the marching cubes method on a 13 by 13 by 13 grid of data from a function that defines a sphere. Figure 8(b) shows the result of applying mesh displacement to figure 8(a). Figures 9, 10, and 11 show the meshes resulting from applying the two methods to data derived from the function for a hyperboloid, a CT scan of the upper portion of a human femur, and a randomly generated cubic polynomial, respectively.

Mesh displacement improves the resulting surface meshes in two ways. First, by removing small and narrow triangles from the surface mesh, the resulting mesh is significantly compressed. This effect decreases the rendering time and storage space required for such meshes. Table 1 presents statistics for the four examples of the figures and several other data sets generated from randomly chosen degree three functions. In these examples, the compression averaged 40% to 50%. Of course, not every data set will experience such compression. For example, a planar contour perpendicular to a coordinate axis will experience no compression during mesh displacement. However, for most data sets arising in practice, mesh displacement achieves significant compression.

	edge-based		*aspect ratios*			*mesh disp*		*aspect ratios*		
Example	verts	faces	avg	dev	min	verts	faces	avg	dev	min
sphere	672	1328	.666	.259	.055	354	704	.840	.086	.685
hyperboloid	596	1088	.617	.306	.007	264	464	.871	.100	.597
femur	2095	4150	.679	.254	.000	1246	2462	.838	.084	.415
cubic0	809	1503	.648	.295	.002	398	723	.869	.104	.426
cubic1	842	1562	.655	.298	.004	439	796	.869	.109	.367
cubic2	648	1179	.660	.278	.003	385	690	.839	.092	.279
cubic3	854	1587	.650	.285	.000	457	834	.851	.097	.426
cubic4	739	1375	.663	.258	.001	438	795	.833	.085	.457
cubic5	922	1722	.635	.289	.000	472	860	.856	.099	.410
cubic6	746	1373	.651	.293	.002	397	714	.857	.095	.568
cubic7	817	1517	.634	.295	.000	408	739	.859	.101	.495
cubic8	868	1592	.664	.267	.001	510	921	.843	.099	.330
cubic9	834	1553	.644	.287	.001	423	769	.851	.106	.389

Table 1 Comparison of edge-based interpolation and mesh displacement

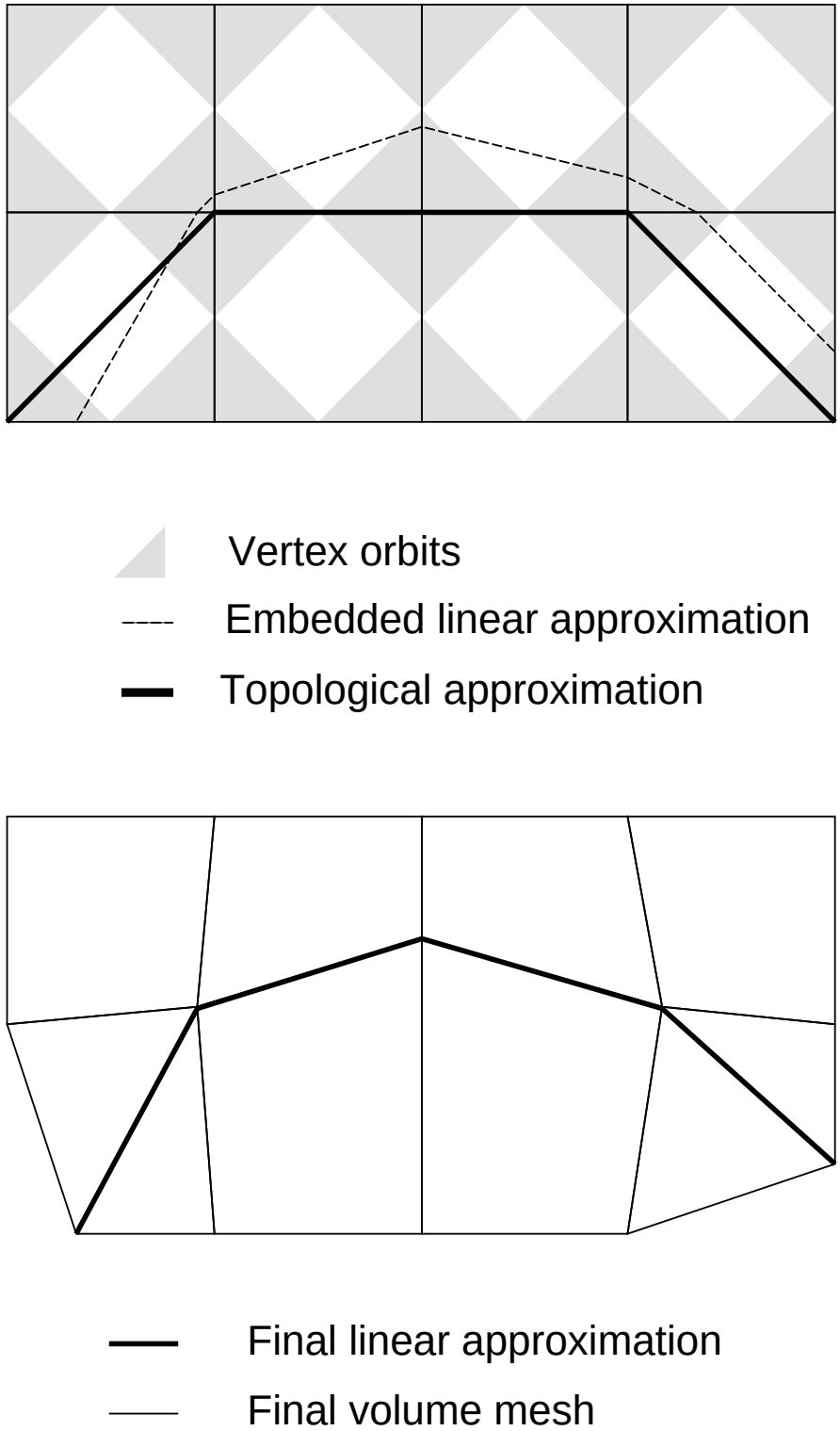

Figure 4 An example of mesh displacement

2.3 Shape bounds

The second improvement from mesh displacement involves the shape of the triangles in the new mesh. By eliminating small and narrow triangles, mesh displacement produces only triangles with good shape. A more precise statement requires a formal measure of the shape of a triangle. One standard measure is the *aspect ratio* of a triangle, the ratio of the radius of its inscribed circle to the radius of its circumscribing circle. This ratio is normalized by multiplying by a factor of two so that an equilateral triangle has aspect ratio one. The aspect ratio measures how close the three vertices of a triangle are to being collinear.

Table 1 also includes statistics on the aspect ratios of triangles produced by the two methods. Mesh displacement produces meshes whose average aspect ratio is significantly better than those produced by edge-based interpolation methods. Perhaps a more important measure is the worst case aspect ratio of triangles produced by these methods. Triangles with small aspect ratio can degrade the performance of polygon-based rendering methods such as Gouraud and Phong shading. In finite element analysis, elements with small aspect ratio can generate ill-conditioned systems of linear equations [Zie78].

Edge-based interpolation methods can and do produce triangles with arbitrarily small aspect ratios, as noted in table 1. For the mesh displacement method, we believe that the aspect ratios of the resulting triangles are bounded below by a positive constant. To support our claim, we offer the following argument.

A triangle with aspect ratio near zero has almost collinear vertices. In the second step of the algorithm, each vertex of the cubic mesh C is displaced inside its orbit. Only those cases in which the orbits of three vertices of a cube contain collinear points can lead to triangles with arbitrarily small aspect ratio. By inspection, the only case in which this is possible is pictured in figure 5, where four cubes share a common edge. Three vertices on the lower left cube have vertex orbits that are collinear along the bold line L. Edge-based interpolation has produced the triangle abc in the lower left cube and the triangle bcd in the lower right cube. Vertex c lies on the interior edge shared by all four cubes, but near the vertex v.

During mesh displacement, vertices c and d lie in the orbit of vertex v and are collapsed to their centroid e. If the cubes have unit size, the new triangle abe has sides of length $\frac{5}{4}$, $\frac{\sqrt{2}}{2}$, and $\frac{\sqrt{5}}{4}$. The resulting triangle has an aspect ratio of about 0.0503.

To achieve this lower bound would require highly unusual input data. In practice, the mesh displacement algorithm rarely produces triangles whose aspect ratio is worse than 0.4. For all thirteen examples of table 1, only six triangles possess aspect ratios of less than 0.4. The smallest aspect ratio that we have observed in practice is about 0.25.

3 Boundary containment

Edge-based interpolation methods produce surface meshes whose boundaries are contained in the boundaries of the original cubic grid. The boundary curves of surfaces produced by mesh displacement do not necessarily conform to the boundaries of the cubic mesh. If such boundary containment is desired, we suggest two possible approaches.

One solution is to apply mesh displacement only to those vertices of S that lie in the orbits of interior vertices of C. Those vertices of S that lie in the orbit of a particular

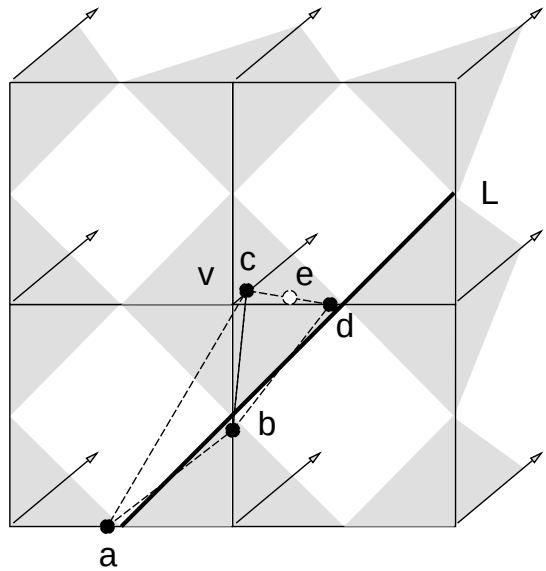

Figure 5 Worst case aspect ratios in mesh displacement

interior vertex of C are coalesced into a single vertex. Those vertices of S that lie in the orbits of vertices on the boundary of C are left unchanged. This method has the advantage of simplicity, but can still leave several badly shaped triangles along the boundaries of the final surface mesh.

Another solution allows mesh displacement to be applied in all but a rare set of circumstances. This technique, called *boundary classification*, classifies a satellite p of a vertex v into one of two categories. If p and v lie in exactly the same set of faces on the boundary of C, then p is a *primary* satellite of v. Otherwise, p is a *secondary* satellite of v. If v lies on a face of the boundary of C, its primary satellites lie on that face. If v lies on an edge of the boundary of C, its primary satellites lie on that edge of C.

Boundary classification is incorporated into the mesh displacement algorithm by replacing the second step with the following:

2. for each vertex v of C
    if v has a primary satellite
        displace v to the centroid of its primary satellites
    else
        treat each secondary satellite of v as lying in a distinct vertex orbit

This approach is correct because displacing v to the centroid of its primary satellites keeps the vertex on the same facet of the boundary of C. If v has a single secondary satellite, displacing v to this satellite does not affect the boundary behavior of S. If v has two or more secondary satellites, it is not always possible to coalesce these satellites so that the boundary of the resulting surface mesh still lies on the boundary of C. For example in figure 6, S passes near a corner vertex v of C. Vertex v has two secondary satellites, p and q. Coalescing p and q into a new vertex and forcing the new vertex to simultaneously lie on

the edges containing p and q results in the new vertex being positioned at v. However, v is an unsatisfactory location for the new vertex since it may lie up to $\frac{\sqrt{2}}{4}$ units from S. In this rare case, we suggest that each secondary satellite be treated as lying in a distinct vertex orbit and assigned a different vertex index.

Figure 7 illustrates three examples of mesh displacement with boundary classification in two dimensions. In the middle and right examples, the vertex v has a single primary satellite. Vertex v is displaced to this primary satellite, so that the resulting curve still terminates at the boundary of the mesh. In the leftmost example, v has a single secondary satellite. Since v has only a single secondary satellite, the original embedded curve does not exit the boundary of the mesh in the orbit of v. Thus, displacing v to its secondary satellite does not affect the termination of the curve. The examples given in this paper were computed using boundary classification so that, for instance, the boundary curves in figure 11 remain planar after mesh displacement.

Boundary classification also has the advantage that the behavior of the final mesh along a boundary is determined solely by the scalar data lying on the containing boundary of the cubic grid. Thus, if two adjacent cubic grids share the same scalar data along a common boundary, the two zero contours produced by boundary classification are guaranteed to be continuous along the common boundary.

Requiring boundary containment affects the aspect ratios of the resulting triangles in several ways. First, mesh displacement with boundary classification can produce triangles with arbitrarily small aspect ratios. Degenerate triangles can arise, as depicted in figure 5. If the vertex v lies on a face of the boundary of C, then d is a primary satellite of v

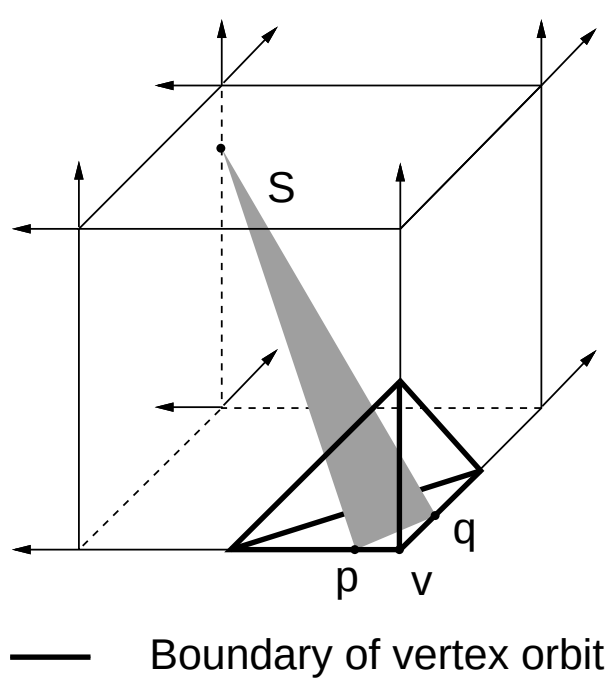

Figure 6 A vertex with two secondary satellites

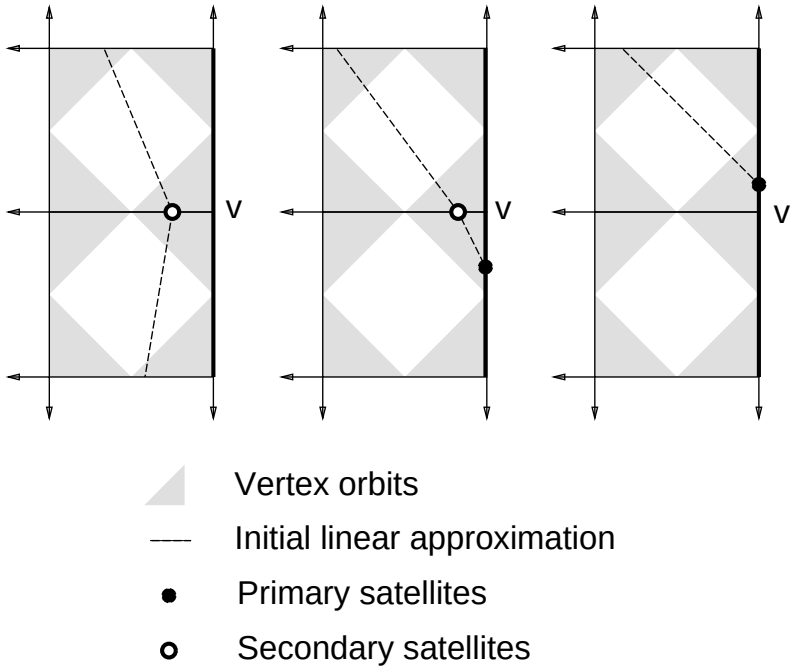

Figure 7 Several examples of boundary classification

and c is a secondary satellite of v. Displacing v to d yields the triangle abd whose aspect ratio can be arbitrarily small. If the aspect ratio of the final mesh is important, we suggest explicitly testing for this circumstance and deleting the triangle abd. Since this triangle lies completely on the boundary of the cubic grid, the resulting surface is continuous and its boundary curves still lie on the boundaries of C.

For some embedded surface meshes, the three goals of boundary containment, bounded aspect ratio, and accurate approximation of the original mesh are difficult to achieve simultaneously. In figure 6, the darkened triangle has an edge from p to q whose length can be made arbitrarily close to zero. Any method that accurately approximates S and maintains boundary containment must produce a surface mesh with vertices at p and q. Because of the vanishingly small distance between p and q, arbitrarily many vertices may need to be inserted into the resulting surface mesh to achieve a lower bound on the aspect ratio of the resulting triangles. In these case, we simply note that mesh displacement does not degrade the aspect ratio of the final mesh.

4 Implementation and performance advantages

Piecewise linear meshes are typically represented using two sets of data: vertex locations and triangle topology. A table of Cartesian coordinates stores vertex locations, with each vertex identified by its position in the table. Triples of vertex indices define triangles. The table of all triples defines the topology of the surface mesh. This representation allows several triangles to logically share a common vertex, typically decreasing the size of the vertex table

by nearly a factor of six. This technique also makes later calculations, such as Gouraud shading for graphical display, simpler.

For edge-based interpolation methods, identifying triangles sharing a common vertex usually involves some type of hashing or a table lookup based on the edge representation. For mesh displacement, such identification is simple. Since each vertex of the final mesh is topologically a vertex of the original cubic mesh, vertex indices can be assigned during the first step and stored with the three-dimensional array of vertex data. Other information for a vertex such as the number of primary and secondary satellites it currently possesses and the centroids of those satellites can be stored in an auxiliary table with as many entries as there are vertices in the surface mesh. A pass through this auxiliary table allows creation of the actual vertex locations during the second step of the method. More sophisticated implementations of marching cubes that only store a few slices of the data cube at a time can use mesh displacement as well, by interleaving the two steps of the algorithm with the processing of the slices. The algorithm requires only the vertex indices for the slices being processed at any given time and the auxiliary data for the part of the surface mesh contained in those slices.

We have modified an existing implementation of the original marching cubes algorithm to use mesh displacement with boundary classification. Complete revision and debugging of this code required about two hours and involved insertion of around 100 extra lines of code. Our installation of mesh displacement on top of the marching cubes algorithm improved the performance of the resulting code because the surface mesh produced by mesh displacement was significantly smaller than the mesh produced by marching cubes.

References

[Bli82] J. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):235–256, 1982.

[Blo88] Jules Bloomenthal. Polygonalization of implicit surfaces. *Computer Aided Geometric Design*, 5:341–355, 1988.

[Dür88] M. J. Dürst. Additional reference to marching cubes. *Computer Graphics*, 22(2):72–73, 1988.

[LC87] W. Lorenson and H. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *Computer Graphics*, 21(4):163–169, 1987.

[MW90] Doug Moore and Joe Warren. Adaptive mesh generation II: Packing solids. Technical Report TR 90-139, Rice University, Department of Computer Science, 1990.

[WMW86] G. Wyvil, C. McPheeters, and B. Wyvil. Data structure for *soft* objects. *The Visual Computer*, 2:227–234, 1986.

[Zie78] O. C. Zienkiewicz. *The Finite Element Method.* McGraw Hill, 1978.

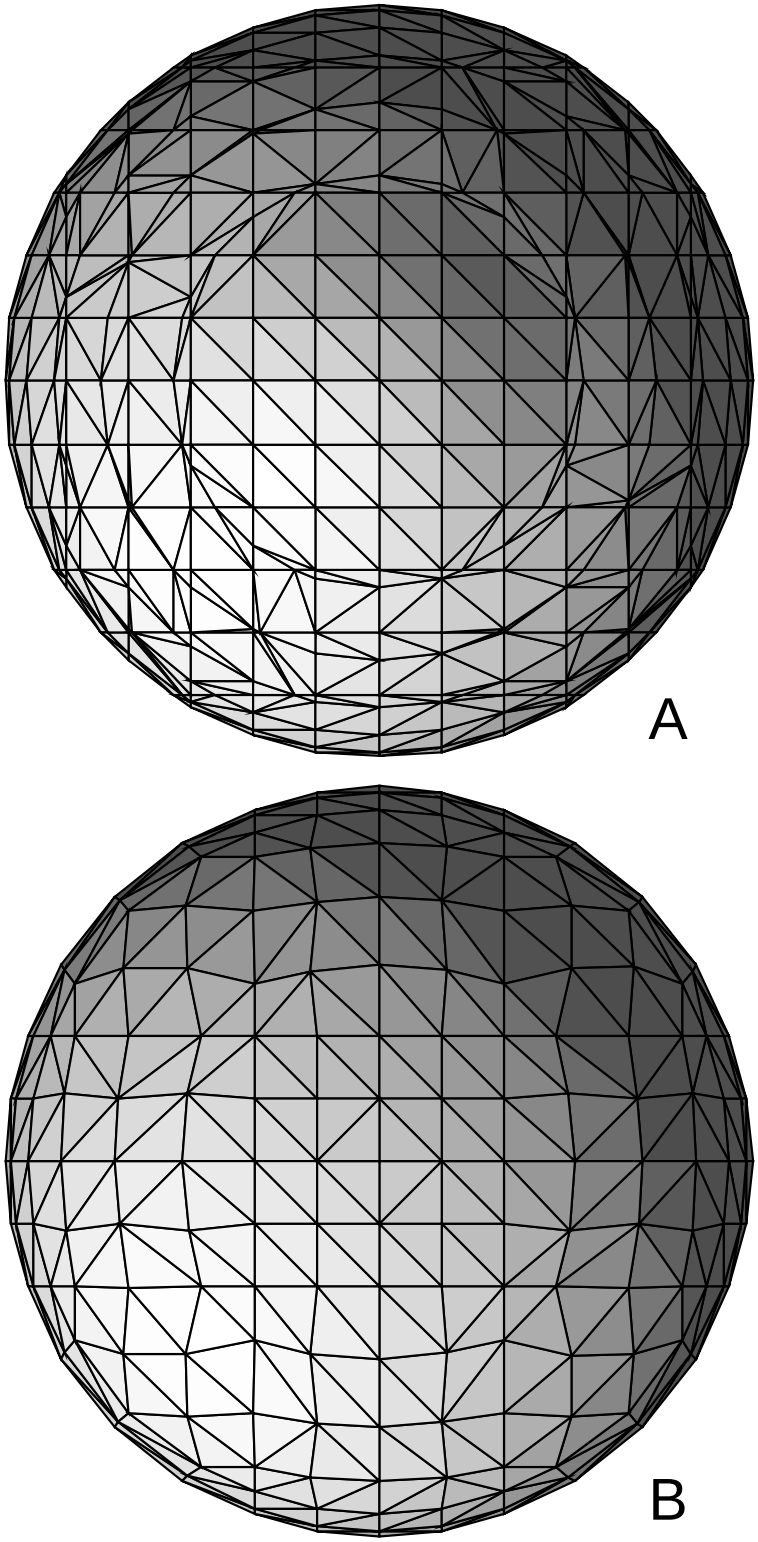

Figure 8 Two approximations to a sphere

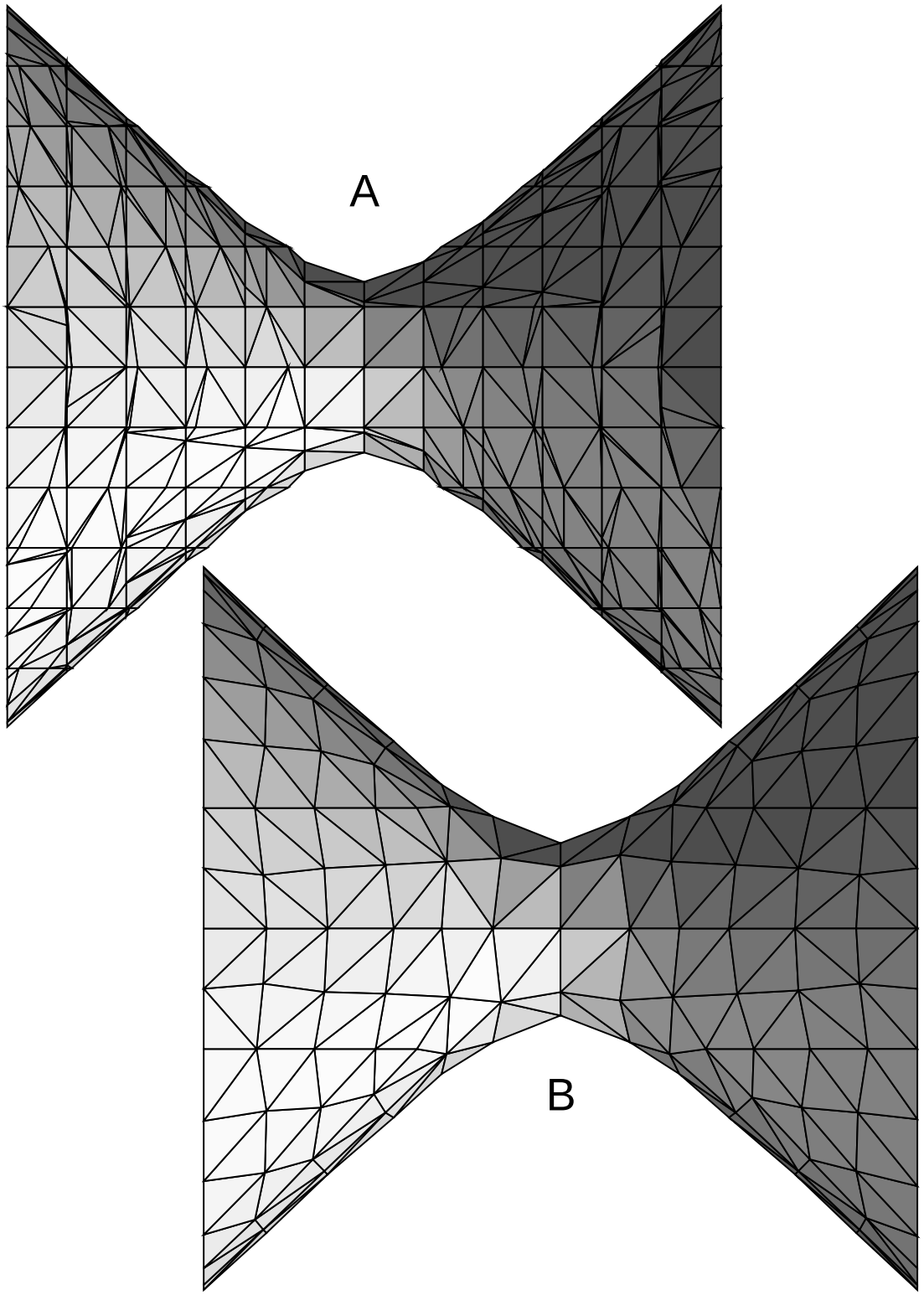

Figure 9 Two approximations to a hyperboloid

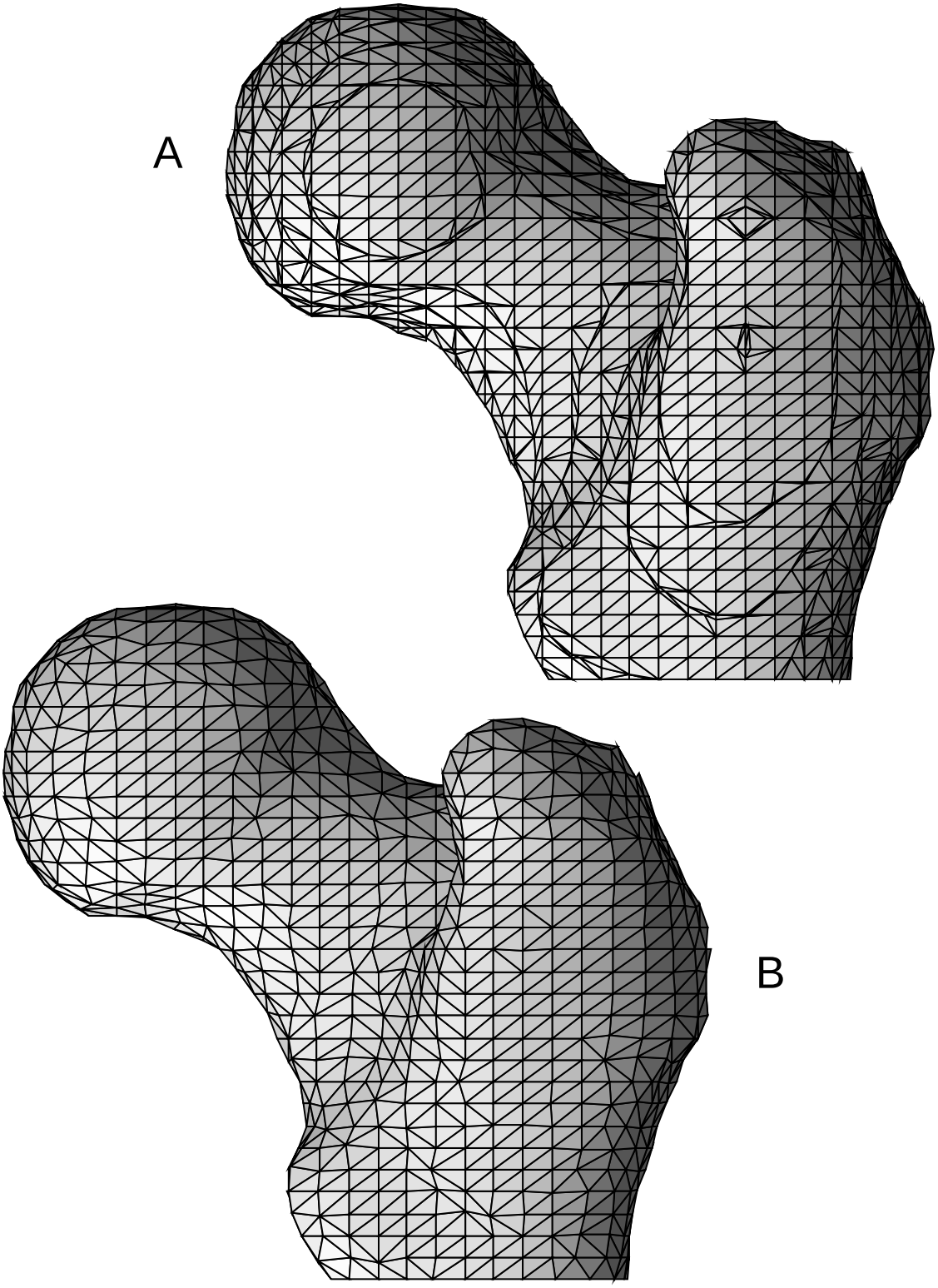

Figure 10 Two approximations to the head of a femur

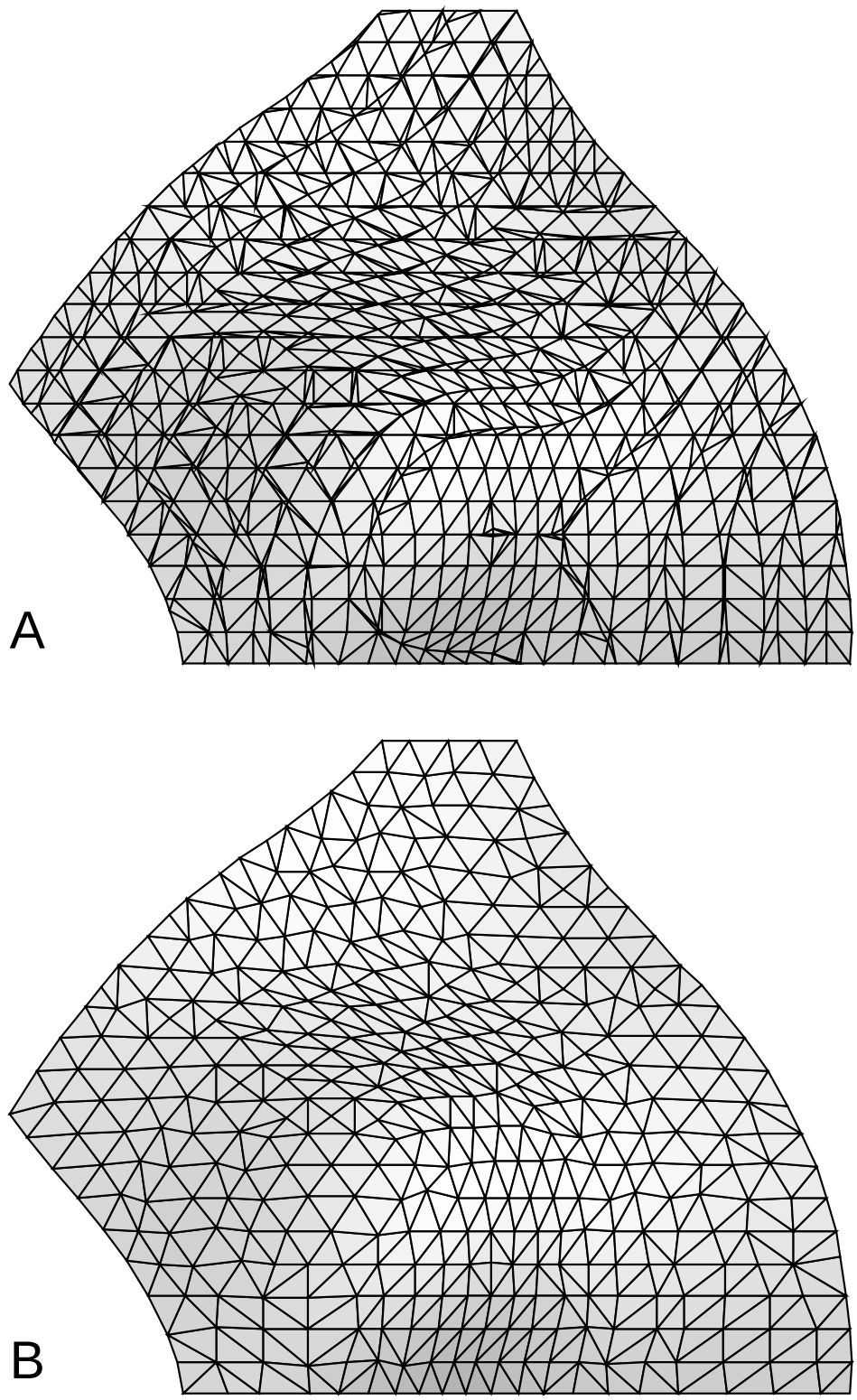

Figure 11 Two approximations to an implicit cubic patch