A BÉZIER-BASED APPROACH TO UNSTRUCTURED MOVING MESHES*

David Cardoze† Alexandre Cunha‡ Gary L. Miller† Todd Phillips§
Noel Walkington§

We present a new framework for maintaining the quality of two dimensional triangular moving meshes. The use of curved elements is the key idea that allows us to avoid excessive refinement and still obtain good quality meshes consisting of a low number of well shaped elements. We use B-splines curves to model object boundaries, and objects are meshed with second order Bézier triangles. As the mesh evolves according to a non-uniform flow velocity field, we keep track of object boundaries and, if needed, carefully modify the mesh to keep it well shaped by applying a combination of vertex insertion and deletion, edge flipping, and edge smoothing operations at each time step. Our algorithms for these tasks are extensions of known algorithms for meshes built of straight–sided elements and are designed for any fixed-order Bézier elements and B-splines. Although in this work we have concentrated on quadratic elements, most of the operations are valid for elements of any order and they generalize well to higher dimensions. We present results of our scheme for a set of objects mimicking red blood cells subject to a precomputed flow velocity field.

Keywords

mesh generation, computational geometry, Bézier triangles, B-splines, finite element method, moving meshes

1. INTRODUCTION

The goal of this paper is to describe our development of geometric algorithms and software for the efficient simulation of deforming soft tissues. Of particular interest is the simulation of red blood cells at the scale of individual cells that are being transported in a fluid. This is the focus of the Sangria project at CMU [20], whose goal is to develop parallel geometric and numerical algorithms for the simulation of complex flows with dynamic interfaces. Our target application is the simulation of blood flow at the microscopic level where individual cell deformations and their interactions with the surrounding fluid have to be accounted. Due to the complexities involved in the simulation of thousands of individual cells, no simulations of blood flow at this scale exists. However they are crucial to the development of artificial organs, and better models of macroscopic blood flow. We must maintain the deforming interfaces with a geometric object such as a mesh and develop efficient geometric algorithms to update the underlying mesh as time evolves. In this paper we concentrate on such geometric algorithms. Our ultimate goal is to simulate blood flow in three dimensions, here we focus first on two-dimensional simulations, utilizing many procedures which generalize well to higher dimensions.

Our approach is to explicitly represent interfaces and boundaries by moving the mesh with their movement. More generally the mesh will move with the fluid and tissues as well. Thus an edge in our mesh represents a portion of the tissue so even if the edge is initially straight it will in general move to a curve with time.

Our moving mesh algorithms support the Lagrangian formulation of fluid mechanics. In the Lagrangian paradigm for simulating fluid flows, domain boundaries and object interfaces are continuous parts of the fluid they are embedded in and as such they inherit the same motion experienced by the fluid. Mesh vertices, representing both fluid particles and the embedded objects, move subject to the same velocity field, which makes it easy to deform and track dynamic object interfaces. A challenging aspect of this approach is to make sure that the moving meshes remain acceptable and of good quality at all times during the progression of the physical simulation.

Severe motion of the vertices from one time step to the next may produce tangled or poorly shaped elements which in turn may lead to unrecoverable errors in the numerical solver. It is well known that the presence of such bad elements in the mesh may prevent a simulation to advance and yield meaningful results. Therefore, maintaining the good quality of an evolving mesh is of crucial importance in the Lagrangian paradigm.

The work we present here is a step towards the broader acceptance and application of the Lagrangian formulation for simulating time dependent problems. Our goal is to improve the development of simulations by addressing the geometric aspects of moving meshes. Contrary to other methods, we do not resort to a complete remeshing from scratch at each

*Supported in part by NSF Grants CCR–9902091, CCR-9706572, and ACI-0086093.
†Computer Science Department
‡Laboratory for Mechanics, Algorithms, and Computing
§Department of Mathematics

SCG'04, June 8–11, 2004, Brooklyn, New York, USA.

time step to achieve good quality. When mesh elements deform beyond acceptable limits, we restore good quality by applying a set of local topological and geometrical operations which are extensions of known algorithms used in linear meshes.

We present four basic accomplishments in this work:

- A new framework for moving curved meshes based on existing algorithms for linear meshes.
- A new metric for analyzing the quality of curved elements.
- An implementation of the meshing methods discussed.
- Some experimental results verifying the quality of these simulation methods.

The rest of this paper is organized as follows. In Sections 2 and 3 we discuss or motivations and related work. In section 4 we describe Bézier curves, Bézier triangles, B-splines and explain the reasons why we opted to use them as geometric building blocks of our simulation process. In Section 5, we present the basic operations we use to modify evolving meshes and the algorithmic methods we use to ensure mesh quality. In Section 6 we describe how all the respective procedures fit together in our simulation process. In Section7 we describe our prototype. In Section 8, we present some experimental results. Finally, in Section 9, we discuss future work.

2. MOTIVATION

Previous curved meshing methods have been primarily concerned with respecting the curved nature of the domain boundary. Our approach is new, in that curves are allowed and indeed do exist everywhere in the mesh. There are several reasons motivating this design choice. The first is software design. From the perspective of the implementation, once one has implemented the ability to handle curvilinear elements on or near the boundary, it is relatively straightforward to extend this notion to the entire mesh.

The second and most important concern in moving meshes is the accuracy of the deformation. A linear moving mesh has only one of two options when deforming. A linear approximation to the deformation can be made, or the mesh can be pushed forward in a nonlinear way and then projected back onto a linear mesh. The first option is unwanted, as we desire a high degree of accuracy in our deformation procedure. The second option is also unwanted, since it requires a rather costly projection everywhere in the mesh. Performing such a projection not only requires more computation time, but also introduces re-interpolation error everywhere in the mesh.

As such, we find that using a curved moving mesh is exactly suited to our needs. We take a high-order approximation to the deformation (same degree as the mesh), and encounter no re-interpolation error. We must be wary, however, since allowing the curves of the mesh to continually deform arbitrarily can (and will) result in such severe curvature that our meshing methods will fail, since they are based on linear methods and expect the curves to be near linear. Still, our methods for reducing the severity of curves are localized, and therefore introduce less re-interpolation error than methods which force the entire mesh to be linear.

3. RELATED WORK

The main body of related work consists of work with curved meshing methods and with moving mesh methods. To the best of our knowledge, there exists no other work in dealing with curved moving meshes.

A good deal of work with curved meshes has been done by Luo, Shephard, et al. [15]. They describe methods to obtain a mesh with curved elements starting from a straight mesh. They also describe the use of Bézier polynomials to define the shapes of their elements. The work makes use of meshing operations that have proved well in practice, mainly edge splits and edge collapses. This work also discuss examining the Jacobian of Bézier elements for shape validity. Another important work in the field of curved meshing was done by Boiven and Ollivier-Gooch [3] who presented an algorithm extending Delaunay refinement to mesh a static domain with a curved boundary.

In the area of moving linear meshes, two major ideas have prevailed. One is to completely remesh the domain at every time step, simply preserving the mesh values at the vertices. The other idea is to make only local mesh modifications as necessary. We have chosen the later approach, since the former is ill-suited for higher order meshes. We are unwilling to remesh, since it would involve discarding (or at the very least downgrading) all of the mesh nodes not located at vertices.

Antaki et al. [1] motivated by the microstructural blood flow problem developed a two-dimensional parallel dynamic mesh Lagrangian method for flows with dynamic interfaces. They took the first approach described above, that is to remesh the domain at each time step. One drawback of their work was the inability to coarsen the mesh. Using the second approach described for moving linear meshes Kuprat et al. [11] describe a three-dimensional system for moving meshes, X3D. Their system modifies the mesh topology to maintain the Delaunay property as the mesh moves. It also provides mesh smoothing to optimize mesh quality, and mesh refinement by means of point insertions. Other work in moving linear meshes with local modifications is described in [23, 2].

In the case when their is no flow such as simulating a shock wave one may coarsen and refine a non-moving mesh. A nice example appears in [14].

4. MESH AND ELEMENT TYPES

For our mesh elements, we have chosen to use Bézier triangles, along with B-Splines to represent boundary geometry. In addition we have a mesh hierarchy consisting of three levels. In this section we discuss these concepts.

4.1 Bézier curves and B-Splines

Bézier curves and triangles were selected for defining mesh elements for two main reasons. Firstly, using curved instead of linear elements allows us to use meshes with far fewer elements both for representing geometry and for obtaining accurate numeric solutions. Secondly, Bézier curves and triangles have a number of mathematical properties leading to elegant algorithms.

Bézier curves are completely defined by their control points which form the control polygon. Similarly Bézier triangles are completely defined by their control points which form a control net. See Figure 1. For more information about Bézier curves and triangles see [8, 10] among others.

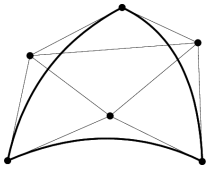

Figure 1: A quadratic Bézier triangle: The boundary of the quadratic triangle is shown in bold. The control net consists of six vertices and four straight triangles.

B-splines are a convenient way for us to represent C^1 continuous curves. First, they allow us to represent object boundaries when we want to enforce C^1 continuity. Quadratic B-splines are made of a sequence of quadratic Bézier curves connected in such a way that the overall curve is C^1 continuous everywhere. They are completely determined by a control polygon or de Boor polygon, and a knot sequence. See Figure 2. For more information regarding B-splines the reader can refer to [8, 10].

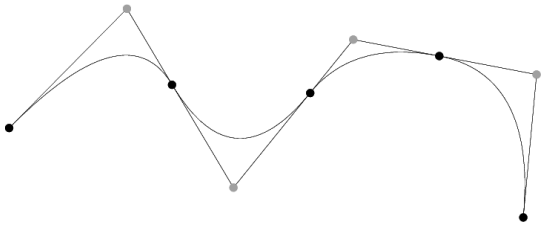

Figure 2: A quadratic B-spline and corresponding control polygon. The black points are the values the curve takes at the knots. The white points are the internal points of the control polygon.

4.2 Mesh Hierarchy

In our meshing methods we consider three different level of meshes: the Bézier mesh, the control mesh, and the logical mesh. The distinction between these three meshes is very useful in describing and defining mesh improvement methods for curved elements, although all three meshes need not be distinctly represented in the implementation. The curved mesh is of course the highest level mesh. This is a mesh of the domain, and it is on this mesh that functions are defined. The logical (or linear) mesh is the straight mesh formed by connecting the vertices of the curved mesh and maintaining the same topology. If the curved mesh is not very curved, we expect in will inherit most of the properties of the logical mesh. This is the crux of many of the meshing methods presented herein. The control mesh is obtained from the Bézier mesh by replacing every curved triangle by four straight triangles. The vertices of these triangles are the vertices and control points of the curved triangle. By controlling the validity and quality of this control mesh, we hope to control the validity and quality of our curved elements. Figure 3 shows a Bézier mesh and its logical and control meshes.

5. MESH AND ELEMENT OPERATIONS

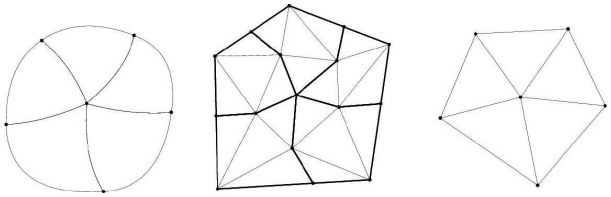

Figure 3: A Bézier mesh (left), its control mesh(center), and logical mesh(right)

When utilizing curved meshes, many basic topological and geometric functions of the mesh must be handled in a delicate fashion. In Section 5.2, we describe implementations of some basic geometric procedures for Bézier elements. In 5.3, we use these primitives to implement a standard set of operations for a dynamic mesh.

5.1 Definitions

We consider a mesh as a set of elements K, and we assume that for each $k \in K$, there is a geometric mapping $\chi_k(\xi)$ which maps $\hat{K}$ to K, where $\hat{K}$ is defined to be the unit right triangle, parameterized in barycentric coordinates ξ. The k subscripts may be omitted where obvious.
We also consider the existence of u_k for each $k \in K$, where u_k is a mapping from $\hat{K}$ to R^n, representing an n-dimensional function defined on k.

5.2 Mesh Modification Operations

In this section we discuss the primitive topological modifications that are allowed on the mesh. These routines are deliberately simple, so that their generalizations to higher order elements and higher dimensional meshes can be understood. We subsequently will use these operations as building blocks for mesh improvement algorithms.

5.2.1 Insert-Point

Insert-Point is a routine which takes an element k and an interior barycentric coordinate ξ and subdivides the triangle into three new triangles (see Figure 4). The geometric position of $\mathbf{p} := \chi(\xi)$ and the control points of the new edges are determined by evaluation using the de Casteljau algorithm. Control values for the function carried by the mesh are also determined by running the de Casteljau algorithm on $u(k)$.

Quite often one desires to insert a point $\mathbf{p}$ into a given element without foreknowledge of it's Bézier coordinates. The problem of finding the Bézier coordinates is simply that of finding a root to the equation $\chi(\xi) - \mathbf{p} = 0$. Using the multi-dimensional form of Newton's method for this root-finding works extremely well in practice. An alternative method known as "Bézier Clipping" (similar to interval bisection) is discussed in [18]. We note that when inserting points, inserting the exact geometric point specified is not crucial to any of our methods. In the case where Newton's root finding is converging to a point very near the boundary of our Bézier triangle (i.e. one of the barycentric coordinates is zero) then we may give up on the approximate solution and return a nearby point on the boundary of the triangle.

The Insert-Point routine as described above is valid for any dimension and for Bézier triangles of any degree.

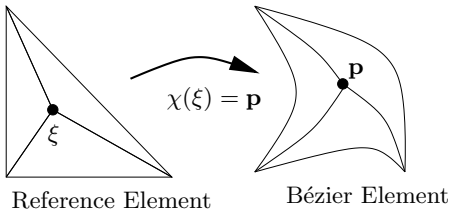

Figure 4: Point insertion through element subdivision.

5.2.2 Remove Point

Remove point is a simple routine that takes a vertex v of degree three and removes it from the mesh (see Figure 7). Since v is of degree three, the three triangles adjacent to v are discarded and a new triangle is created from the edges of those triangles not adjacent to v. Since we must maintain continuity of the mesh both geometrically and with respect to u, the function carried by the mesh, the edges of the new triangle contain exactly the same control information as before. Any control values that are internal to the new triangle must be determined by some projection from functions defined on the three old triangles to a function defined on the new triangle. The L^2 projection is most often desired, and can be approximated easily in a least-squares fashion. Note that in the case of quadratic elements, there are no internal control points, hence no such projection is needed. This method for point removal generalizes to higher dimensions, however the respective constraints on input vertices to this operation are more difficult to achieve. That is to say, it is more difficult to force vertices into degree four in three dimensions (see section 5.3.3). We also note that this operation is special cased for boundary vertices in the obvious way.

5.2.3 Edge Flipping

One procedure used heavily in linear mesh improvement algorithms is the bistellar edge flip. This operation has a valuable extension to Bézier triangles.

If we view the control nets of the two adjacent Bézier triangles in question, then the flipping operation can be viewed as the flipping of four edges in the control nets (Figure 5). This viewpoint has the advantage that it can be immediately generalized to higher order Bézier elements. In 3 dimensions the various categories of bistellar flips correspond to the same categories of bistellar flips in the control mesh.

After flipping the topology, we must choose new control values for the internal control points to attempt to preserve the function u carried by the mesh. A projection (normally L^2 approximation) is determined from functions defined on the old triangles to functions defined on the new triangles. This procedure is straightforward in any dimension with any degree Bézier simplices.

5.2.4 Edge Smooth

Since all of the preceding operations are merely extensions of known linear operations on meshes, it becomes necessary to introduce a mesh modification operation which works only on the curved properties of the mesh. The operation smooth edge does just that. It takes as input an edge

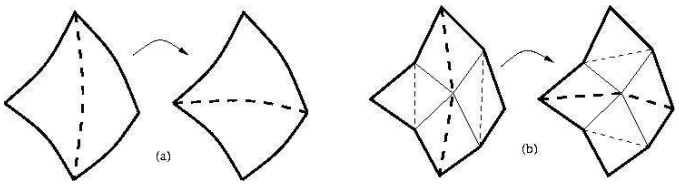

Figure 5: (a) The dashed edge is flipped in the curved mesh. (b) The corresponding dashed edges are flipped in the control mesh.

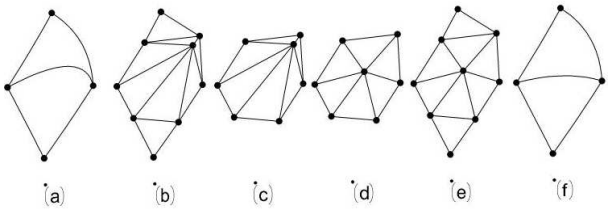

Figure 6: (a) An edge to be smoothed is identified based on it's quality. (b) The control mesh (c) Isolate the star of the control point to be move (d) Move control point based on linear mesh improvement techniques (e) Update the control mesh (f) better quality elements

of the mesh to be modified. New positions for the internal control points of the edge are determined which will alter the geometry of the curve without affecting the any topological properties of the mesh. This procedure is illustrated by Figure 6. Choosing a new geometric position for control points is discussed in Section 5.3.4.

Once a new geometric position for the control points has been determined, new control values must be chosen to maintain the function u being carried by the mesh. As before, a least-squares approximation is implemented.

5.3 Mesh Improvement Methods

As the mesh domain moves and deforms, it becomes necessary to modify the mesh in order to maintain certain quality guarantees. We describe several procedures for ensuring the quality of the curved mesh. These procedures are built from the operations of the previous section for reasons already described. The algorithms we have chosen to use for mesh improvement are all based on linear mesh improvement techniques. The reason for this decision is twofold.

Firstly, the analysis of these techniques for linear meshes in two dimensions is extremely well understood, and various quality guarantees exist. Our hope is that by implementing similar algorithms for curved meshes we will see good results in practice.

Secondly, there is no well-established definition of what it means to be a quality curved mesh. Intuitive notions suggest that curved elements that are close in shape to linear elements are of roughly the same quality as the linear element. This idea is based on standard quality guarantees for high order functions defined on linear elements. To take advantage of this intuition, we strive to make curved elements close to their linear counterparts whenever necessary. Some mathematics (see Section 5.3.4) also suggests that the lin-

ear quality of the control mesh of a Bézier triangle affects its quality as an element. It is certainly true that the validity of a Bézier element relies on the validity of its control mesh ([22, 15]).

5.3.1 Delaunay Logical Mesh

Recall that the logical mesh is the linear mesh underlying our curved mesh by connecting the vertices. After the mesh is distorted, the logical mesh may contain many triangles of poor aspect ratio. We therefore use traditional incremental Delaunay algorithms to enforce the Delaunay property on the logical mesh. This can be accomplished using solely the edge flip operation, thus localizing any mesh modifications to those areas where distortion occurs.

5.3.2 Mesh Refinement

As the mesh is distorted, elements may be stretched too large to capture the desired properties of the simulation, or elements may develop a poor aspect ratio that cannot be cured by edge flipping. In this case, we choose to refine the mesh. We select elements for removal from the mesh for two reasons. First, if a curved element has an area that is too large (as dictated by the numerics of the simulation concerned) then the element must be refined. Second, if the logical triangle underlying an element develops too poor of an aspect ratio, then the element must be removed.

For both cases, we use the method of inserting circumcenters as in [19]. Their are two issues that arrive for curved meshes when using this method. The first is point location of the circumcenter to be inserted into the mesh.

The standard topological walk through the linear mesh is generally viable, but may fail due to overly curved elements. Our method is this: once the point has been located in the linear mesh, we attempt to find it's Bézier coordinates in that element (as in section 5.2.1). If these Bézier coordinates are valid, i.e. all positive, then we have located the point. Otherwise we attempt to locate the point in a neighboring element. The direction of search can be determined by which Bézier coordinates are negative. For any set of elements whose curvature is bounded, the number of attempts is extremely small (rarely more than 2).

The other issue that arises when implementing Delaunay refinement on a curved mesh is that of boundary protection. Encroachment of boundaries is determined by protective lenses around the boundary curves. The size for these lenses is determined by an extension of the normal diametral circle. This extension for Bézier curves is defined in [17].

5.3.3 Mesh Coarsening

When the mesh is distorted, many elements may be condensed to sizes much smaller than are necessary for numerical accuracy. Mesh coarsening is then required to keep the mesh from becoming too large. We perform mesh coarsening by means of the Douglas-Peucker algorithm [7] and the function-based coarsening paradigm of Talmor et al. [16, 21].

The main idea of the function-based coarsening paradigm is as follows. First a sizing function for the mesh is obtained. The sizing function may come from the numerics of the simulation, and may also be relative to the geometric necessities of the domain. In either case, the sizing function is assumed to be approximately Lipschitz, and the resulting mesh after coarsening will approximately be at least as well-graded.

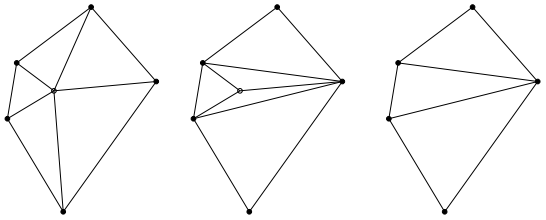

Figure 7: The star of a point to be deleted (left). The configuration after all new Delaunay ears have been obtained but before the vertex is removed (middle). The mesh with the vertex removed (right)

At every vertex of the mesh a scaled version of the sizing function is used to define the radius of a sphere centered at that point. A maximal subset of the mesh vertices is selected so that none of the corresponding spheres intersect. The selected vertices are kept and the others are slated for removal.

This basic approach must be modified when there are boundaries that must be maintained, as in our case. We run the Douglas-Peucker algorithm on the control polygon of our B-spline boundaries to determine a subset of the control points that must be kept to maintain the geometry of the boundary. These points will be present in the coarsened mesh. The rest of the points, interior or boundary points, are chosen to be in the coarsened mesh according to the sizing function as described above.

Once we have determined a subset of vertices to be removed from the mesh, we incrementally remove each vertex. This localizes the changes to the mesh into only the areas where coarsening is necessary.

The procedure to remove a vertex v from the mesh is based on the algorithm of Deviller [6] that deletes a vertex of the Delaunay triangulation and retriangulates the resulting cavity to obtain a new Delaunay triangulation. The algorithm iteratively identifies an 'ear' of the cavity that should be contained in the Delaunay triangulation of the cavity upon removal. This ear is then added to the mesh by way of an edge flip. At the end of this process the neighborhood of v has the form shown in figure 7 that is required for the point remove operation. Deviller's algorithm generalizes to three-dimensions, but the implementation is more involved [6]. We also note that our procedure is special cased for the deletion of boundary vertices in the obvious way.

5.3.4 Curve Smoothing

All of our previous mesh improvement methods have involved only the linear properties of the mesh, ensuring the quality of the underlying linear mesh. In order to attempt to transfer these quality guarantees to the curved mesh, we now implement the curve smoothing procedure. The first step is for us to identify those elements that are considered to be "too curved". To identify these elements, we propose the metric:

$$\int_k \frac{|J|}{A}$$

where k is the element being considered, A is the area of this curved element, and J is the Jacobian of the geometric mapping χ_k. This metric represents the distortion of the curved triangle from it's underlying linear triangle. This

metric for determining overly curved triangles is selected for several reasons.

Firstly, this metric is always equal to one for any linear triangle. This means that the deviation of this metric from one measures the 'curvature' of a Bézier triangle independent of the shape of its underlying linear triangle, thus isolating the curved properties from the linear properties of the triangle, since the linear properties have already been addressed by the preceding methods. Secondly, this metric is easy to bound for Bézier triangles. The area of a Bézier triangle can be quickly evaluated as a fixed linear combination of the geometric control points (see [8]). The Jacobian is the derivative of the Bézier function χ, and so is itself a Bézier function, and can be bounded by the convex hull of it's control points. And so any triangles for which this metric is too far from one (as determined by the problem), will have their edges smoothed. To select new geometric positions for control points, any linear mesh improvement methods for locally repositioning vertices may be applied to the control mesh. In the next two subsections, we describe the mesh improvement method we selected and examine the Jacobian of a Bézier triangle to justify the choice of this class of methods for repositioning the control points.

5.4 Jacobian of a Bézier Triangle

Algorithms and estimators for the Jacobian $J(\xi)$ of the function $\chi(\xi)$ are critical to our approach. In 2D the Jacobian is the 2 by 2 matrix of partial derivatives. In this section we show how to compute $J(\xi)$ using a generalization of de Casteljau's algorithm given the control mesh. Using this method we then show how to bound $|J(\xi)|$ from below for all ξ in the triangle. The positivity of the determinant will be used to determine the validity of a curved element as in [15]. A similar formulation for the Jacobian of Bézier simplices was independently presented by Vavasis [22].

When using Bézier elements, there is an important correspondence between the conditioning of J and the geometric shape of the control net for an element. This has motivated our geometric presentation of the Jacobian. We only discuss the quadratic Bézier triangle case, but the methods generalize for higher order Bézier elements and Bézier tetrahedra. When using the Bernstein polynomial basis for quadratic Bézier triangles, J may be computed directly and can be expressed as a linear Bézier triangle whose control points are the vector pairs $\mathbf{A}'$ and $\mathbf{B}'$, where $\mathbf{A} = 2\mathbf{A}'$ is the x-partial derivative of the Bézier triangle at the control point, and $\mathbf{B} = 2\mathbf{B}'$ is the y-partial derivative. See Figure 8(b). We have that:

$$J(\xi) = \begin{pmatrix} | & | \\ \frac{\partial \chi}{\partial \xi_1} & \frac{\partial \chi}{\partial \xi_2} \\ | & | \end{pmatrix} = \xi_1 \begin{pmatrix} | & | \\ \mathbf{A}_1 & \mathbf{B}_1 \\ | & | \end{pmatrix} + \xi_2 \begin{pmatrix} | & | \\ \mathbf{A}_2 & \mathbf{B}_2 \\ | & | \end{pmatrix} + \xi_3 \begin{pmatrix} | & | \\ \mathbf{A}_3 & \mathbf{B}_3 \\ | & | \end{pmatrix}$$

We are also concerned with the determinant $|J|$ of the Jacobian matrix, which in this case may be expressed as its own scalar-valued quadratic Bézier polynomial in two variables. If we define the scalar value $\mathbf{C}_{ij} = \mathbf{A}_i \times \mathbf{B}_j$, then convex combinations of these cross products are the control points for $|J|$. The layout of these control points for $|J|$ is shown in Figure 8(c).

Using the convex hull property of polynomials in Bézier form, $|J|$ for this element can be bounded by the maximum and minimum of the C_{ij}'s.

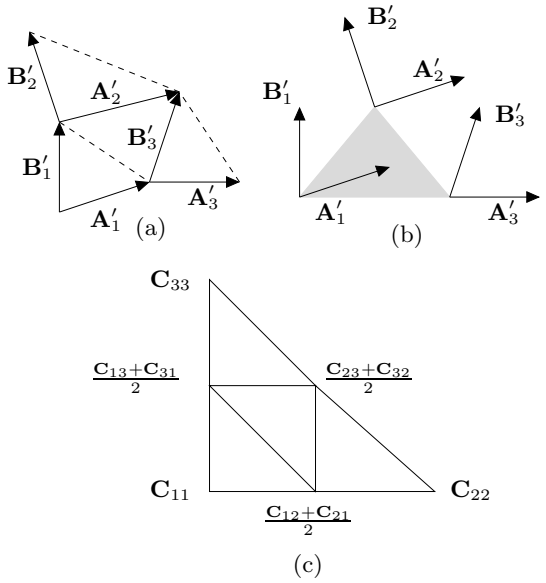

Figure 8: (a)Bézier control net for an element shown with the relevant vectors. (b) The Jacobian is a linear combination of the A-B pairs of vectors. (c) The determinant of the Jacobian as a Bézier polynomial

In this light, it becomes necessary to maintain the linear quality of the triangles in the control mesh, since their aspect ratio bounds the normalized cross-products which in turn bounds the normalized conditioning of the Jacobian. This motivates the use of linear mesh improvement techniques on the *control* mesh in order to improve the curved mesh. Figure 6 provides an illustration of improvement methods applied to the control mesh.

5.5 Smoothing control points

The positioning of an internal control point belonging to a curved edge may sometimes require extra caution. This happens in two situations. First, if the new position due to the motion of an internal control point results in a tangled control mesh, it needs to be modified to produce a valid mesh. This is rather a procedure to validate the mesh than to help improve its quality. In the second situation, when a quadratic edge has a high curvature resulting in very distorted but still valid elements, the internal point is relocated increasing the quality of the triangles incident to it and consequently lowering the edge curvature (see Figure 6). Note that in the former case we might decide improving the mesh quality after validating it. In both situations, only internal control points are repositioned.

To perform point relocation, we turn our attention to smoothing methods. Our goal here is not to improve current smoothing methods but rather use them to help us solve our immediate prototyping needs.

Smoothing is modeled as an unconstrained optimization problem, where a single internal control point is repositioned at a time, while all others are maintained fixed. This is a common approach adopted in many works, including [5, 9].

Each smoothing call solves

$$\max_{\mathbf{x}\in K} \min_{i\in M} \{q_i(\mathbf{x})\} \quad (1)$$

where M is the index set of the triangles incident to the control point located at $\mathbf{x}$, and q_i gives the quality value of triangle i in M. Provided that

$$\Phi(\mathbf{x}) = \min_{i\in M} \{q_i(\mathbf{x})\}$$

is a semi-convex function for all $\mathbf{x}$ in the feasible region K, we are safe on adopting an unconstrained optimization approach.

To fix tangled elements, we use the triangle area A_i as the quality measure in (1). In this case, Φ is a convex function everywhere, and the optimizer is able to find a valid position for the violating control point, no matter where it is initially located, producing untangled elements. Since the exclusive goal here is to form a valid mesh, convergence tolerance of the numerical solution is not relevant and we halt the optimizer as soon as a valid position is found.

When smoothing is not used to untangle elements but to actually increase the quality of valid triangles, the quality metric applied is

$$\tau = \frac{4\sqrt{3}A}{{l_1}^2 + {l_2}^2 + {l_3}^2},$$

where l_i is the length of the i–th edge of the triangle, $\tau \in (0, 1]$ for valid triangles, $\tau = 1$ for an equilateral triangle, and τ approaches zero for flat triangles with small or large angles. A negative τ ($A < 0$) indicates an invalid triangle that needs to be fixed. The control point is allowed to move only in the kernel of its surrounding star–shaped polygon.

Figure 9 shows the level sets of Φ when $q_i = A_i$ and $q_i = \tau_i$. Note that when smoothing with τ, points outside the polygon would have a chance to drift away from the desired solution because there Φ is not convex.

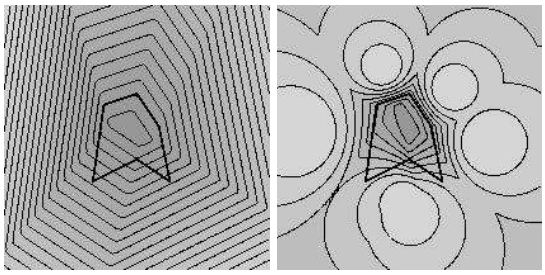

Figure 9: Level curves of Φ for area (left) and quality metric τ. Note in the right figure the sinks outside the polygon.

In our prototype, we adopted CFSQP [12] to solve numerically the optimization problem. It is very stable and it has demonstrated to be efficient enough for our current goals. CFSQP is a C implementation of the nonlinear programming algorithm FSQP (Feasible Sequential Quadratic Programming), a variation on the standard SQP scheme generating feasible iterates [13]. CFSQP is capable of solving *minimax* problems of a set of smooth objective functions with linear and nonlinear equality and inequality constraints. When solving problems with many sequentially related objective functions, such is our smoothing case, CFSQP gives the option to use an algorithm specially designed for this type of problems. This algorithm considerable reduces the computation time by selecting a small subset of the objective functions for inclusion in the quadratic programming subproblems.

6. THE SIMULATION PROCESS

All of our methods are designed around doing finite element computations with a curved moving mesh. In this setup, a field function u must be carried on the mesh, and the mesh must evolve over each discrete time step. We work in the class of so-called "isoparametric" finite element solutions, wherein the solution to the finite element problem is computed in the same basis as the geometric basis of the mesh, i.e. u_k for an element k is determined by control values at the control points of χ_k.

Our basic simulation is the same for any problem being solved. Input for a simulation is given as a topological cell complex representing the boundary, whose edges are B-splines, which may be closed loops. The initial points along each boundary edge are taken to be the knots of the B-spline, so that each corresponding edge in the mesh is a single quadratic segment.

At each time step the process proceeds in three stages. First, a finite element solution is computed for the given time step. Since this solution was computed in the basis of the mesh, we now have control values associated with every control point of the mesh.

Next, we push the mesh forward. To do this, we need a velocity field defined on the mesh itself. Obtaining such a field is discussed in section 6.1. Once we have obtained a velocity field in the basis of the mesh, we simply push each control point forward with a linear displacement ($\Delta x = v\Delta t$), unless we have a closed-form solution for the flow field, in which case we employ a Runge-Kutta scheme.

Lastly, we apply the mesh improvement methods of section 5. We apply them in the order presented in this work, i.e. we enforce the Delaunay property, refine the mesh as necessary, coarsen the mesh as necessary, and then apply edge smoothing. The staging of this type of mesh improvement for moving meshes is not necessarily specific. One might wonder if coarsening before refinement would yield better results. Different approaches to the staging for linear moving meshes have been proposed in [23] and [2]. A simultaneous refinement-coarsening scheme (non-moving mesh) is discussed in [14].

6.1 Simulations Implemented

We have implemented the ability to process two distinct types of simulations with curved moving meshes. The first type of simulation implemented is for Convection-Diffusion problems. In this problem setup, a velocity field is given *a priori* defined on the relevant area of R^2. A concentration (representing temperature or some other continuum) is defined on the mesh. At each time step, the mesh domain is to move and the concentration is to diffuse according to the finite element solution to the scalar diffusion equation. To project the velocity field from R^2 onto the mesh, we simply take the projection that interpolates the velocity field at each of the control points.

In the second setup, we simulate an incompressible Navier-Stokes fluid. A three-dimensional field function is defined on

the mesh, representing velocity in two dimensions and pressure. The mesh is to move forward according to this velocity, and then a new field function is calculated as the solution to the finite element computation. In this setup, no projection of the velocity field is necessary, as it is computed by the solver, and so is already defined in the mesh basis.

7. PROTOTYPE

The current implementation of these methods is written using the object oriented scripting language Ruby, and extensions for Petsc, OpenGL and GLUT. As such this is not a performance impelmentation by any mean, but is none the less robust and useful for testing the practicality of the methods we have described.

Our base topological classes are based around the **CellComplex** approach of Brisson [4]. An object of this class is a collection of objects of class **Cell** which can have arbitrary dimension. A **CellComplex** object stores adjacency information among these cells. We use this class in two ways, the representation of an input, and as a representation of the final mesh itself. By associating geometric information with each **Cell**, we derive two base classes, a **BoundaryComplex** and a **Bézier Mesh**.

The **BoundaryComplex** associates with each of its edges a **B-Spline**, which has a series of control points. The **Bézier Mesh** class associates a **Bézier Triangle** with each of its two dimensional **Cell**s and a **Bézier Edge** with each of its one-dimensional **Cell**s. Every triangle in this mesh has a pointer (generally just a small integer) referring to it's containing face in the **BoundaryComplex**. Similarly, every boundary edge in the mesh contains a reference to the **B-Spline** containing it, as well as it's parametric coordinates on that containing **B-Spline**. These references greatly ease the partioning of the mesh and enforcing of boundary conditions as needed by a finite-element solver.

Every **Bézier Triangle**, **Bézier Edge**, and vertex of the mesh has associated with it the appropriate number of control points, depending on degree of Bézier polynomials desired. For the case of quadratic Bézier functions, every vertex has one control point, edges have three, and triangles have six. Continuity between dimensions is created by using references to common control points. That is to say, a **Bézier Triangle** has pointers to the control points on its boundary, and the **Bézier Edges** that bound this triangle have pointers to the same control points.

If we are to carry an n-dimensional function on our mesh, then each control point has associated with it an (n+2)-dimensional control value (generally denoted as u). The first two dimensions of this control value represent the geometric position of the control point, the rest represent the control values of the function in the basis of the mesh. This is consistent with the isoparametric approach to the finite element method as described in section 6.

8. EXPERIMENTAL RESULTS

We have performed several simulations to test our curved moving mesh implementation. These simulations have been designed to exercise the various features of the mesher, as well as to develop interaction between the mesher and a solver. Several simulations were run with prescribed velocity fields in order to test severe distortions of the mesh domain. We show screenshots from these in Figures 10 and 12. The first figure mimics three blood cells distorting and wrapping around an obstacle, the second figure mimics a single cell body being squeezed through an orifice. These are pure convection simulations that track the positions and deformations of the cells as they move in a velocity field.

We have also conducted experiments while interfacing with a finite element solver. The incrompressible Navier-Stokes model for a fluid is solved at every time step as the mesh evolves over time. Our unique approach allows us to track the interface between various bodies in the fluid, and to prescribe discontinuous viscosities within the mesh domain. Snapshots of such a simulation are shown in Figure 11.

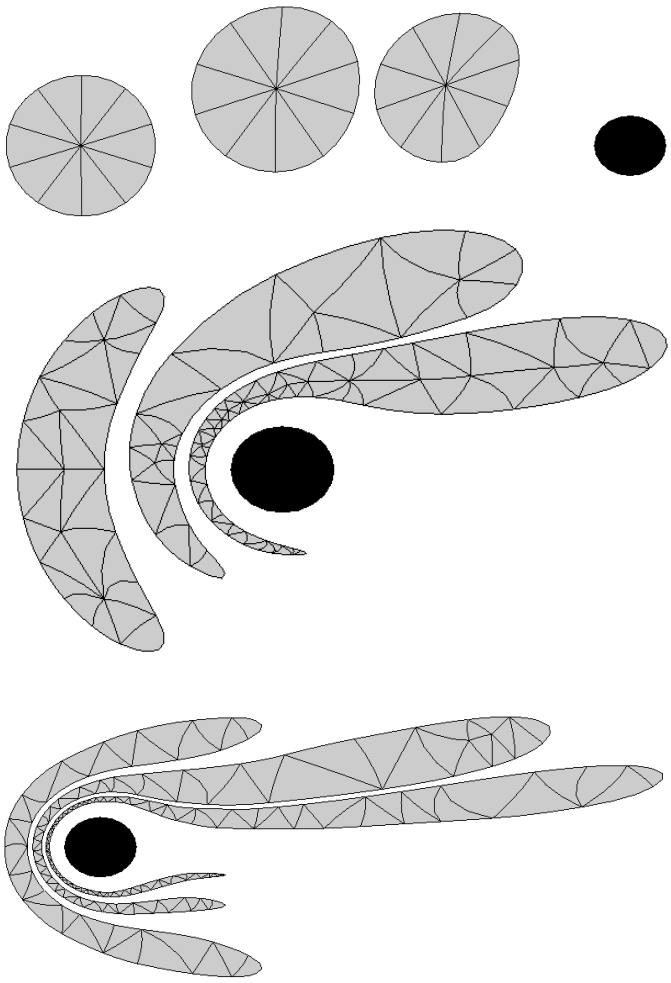

Figure 10: Three cells are considered as a mesh domain. A velocity field moves them from left to right, while an obstacle impedes the path of the cells. A quality mesh is dynamically maintained as the boundaries undergo severe distortion. Smooth boundaries are maintained.

9. FUTURE WORK

We have presented a framework and an implementation of a simulation system for curved moving meshes for Lagrangian methods. While the majority of the methods de-

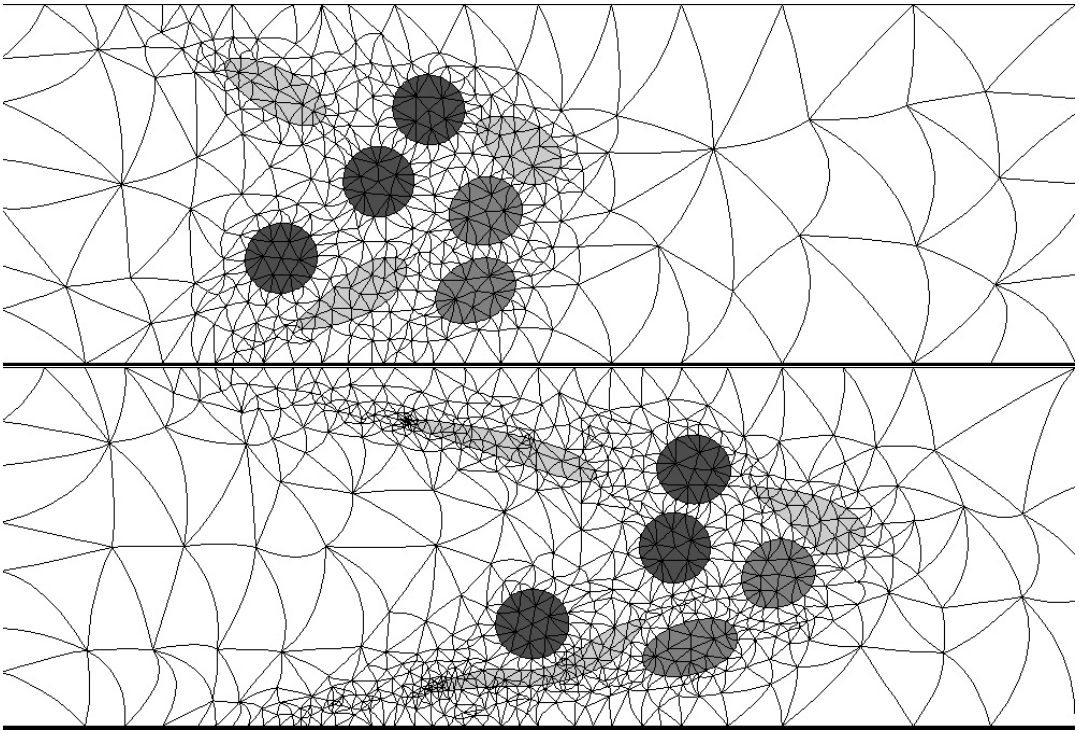

Figure 11: Several bodies flow through a tube with velocities and pressures governed by solutions to Navier-Stokes equations. Discontinuous viscosities are shown with shading. The darker bodies are more viscous, while the lighter bodies are less viscous.

scribed are trivially extended to higher-order Bézier triangles, the implementation as it stands is based on quadratic triangles. A more generic code for handling higher order Bézier triangles and the associated caveats is in the planning phase. Obviously, the principal extension of this work would be into three dimensions. We have chosen our basic mesh modification operations so that they have clear three dimensional analogues. The main difficulty lies in the algorithmic mesh improvement methods. We based our curved methods on very well-understood linear methods in two dimensions, however the analogous methods in three dimensions are not all well-understood, and in some cases do not exist. Overall, the geometric concerns of mesh improvement in three dimensions must be better understood before they can be extended to curved moving meshes.

10. REFERENCES

[1] James F. Antaki, Guy E. Blelloch, Omar Ghattas, Ivan Malcevic, Gary L. Miller, and Noel J. Walkington. A parallel dynamic-mesh lagrangian method for simulation of flows with dynamic interfaces. In *Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*, 2000.

[2] T.J. Baker. Mesh movement and metamorphosis. In *Proceedings, 10th International Meshing Roundtable*, pages 387–396. Sandia National Laboratories, October 7–10 2001.

[3] Charles Boivin and Carl F. Ollivier-Gooch. Guaranteed-quality triangular mesh generation for domains with curved boundaries. *International Journal for Numerical Methods in Engineering*, 55 (10):1185–1213, 2002.

[4] E. Brisson. Representing geometric structures in d dimensions: Topology and order. In *Symposium on Computational Geometry*, pages 218–227, 1989.

[5] Scott A. Canann, Joseph R. Tristano, and Matthew L. Staten. An approach to combined laplacian and optimization–based smoothing for triangular, quadrilateral and quad–dominant meshes. In *7th International Meshing Roundtable*, pages 479–494. Sandia National Laboratories, 1998.

[6] Olivier Devillers. On deletion in delaunay triangulation. *International Journal of Computational Geometry and Applications*, 12:193–205, 2002.

[7] D.H. Douglas and T.K. Peucker. Algorithms for the reduction of the number of points required to represent a line or its caricature. *The Canadian Cartographer*, 10(2):112–122, 1973.

[8] G. Farin. *Curves and Surfaces for CAGD: A Practical Guide.* Morgan Kaufman, 2002.

[9] Lori A. Freitag, Mark Jones, and Paul Plassmann. An efficient parallel algorithm for mesh smoothing. In *Fourth International Meshing Roundtable*, pages 47–58, Albuquerque, New Mexico, October 1995. Sandia National Laboratories.

[10] J. Gallier. *Curves and Surfaces in Geometric Modeling: Theory and Algorithms.* Morgan Kaufman, 1998.

[11] A. Kuprat, D. George, E. Linnebur, R. K. Smith, and H. E. Trease. Moving adaptive unstructured 3-d

meshes in semiconductor process modeling applications. *VLSI Journal*, 6(1-4):373–378, 1998.

[12] Craig Lawrence, Jial L. Zhou, and André L. Tits. User's guide for CFSQP version 2.5. Technical Report TR–94–16r1, University of Maryland, College Park, 1997.

[13] C.T. Lawrence and A.L. Tits. A computationally efficient feasible sequential quadratic programming algorithm. *SIAM Journal on Optimization*, 11(4):1092–1118, 2001.

[14] X.-Y. Li, S.-H. Teng, and A. Üngör. Simultaneous refinement and coarsening: adaptive meshing with moving boundaries. In *7th International Meshing Roundtable*, pages 201–210, Dearborn, Mich., 1998.

[15] Xian-Juan Luo, Mark S. Shephard, Jean-Francois Remacle, Robert M. O'Bara, Mark W. Beall, Barna Szabo, and Ricardo Actis. p-version mesh generation issues. In *Proceedings, 11th International Meshing Roundtable*, pages 343–354. Sandia National Laboratories, September 15-18 2002.

[16] Gary L. Miller, Dafna Talmor, and Shang-Hua Teng. Optimal coarsening of unstructured meshes. *Journal of Algorithms*, 31(1):29–65, Apr 1999.

[17] Todd Phillips. Delaunay refinement for curved boundaries. Poster Presentation, 11th International Meshing Roundtable, September 2002.

[18] S. H. M. Roth, P. Diezi, and M. H. Gross. Triangular Bézier clipping. Technical Report Rep 347, Institute of Scientific Computing, ETH Zurich, 2000.

[19] Jim Ruppert. A delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms*, 18(3):548–585, 1995.

[20] The Sangria Project. Supported by NSF ITR ACI-0086093.

[21] Dafna Talmor. *Well-Spaced Points for Numerical Methods*. PhD thesis, Carnegie Mellon University, Pittsburgh, August 1997. CMU CS Tech Report CMU-CS-97-164.

[22] S. Vavasis. A bernstein-bezier sufficient condition for invertibility of polynomial mapping functions. Unpublished, Nov 2001.

[23] Jie Wan, Suleyman Kocak, and Mark S. Shephard. Automated adaptive forming simulations. In *Proceedings, 12th International Meshing Roundtable*, pages 323–334. Sandia National Laboratories, September 14–17 2003.

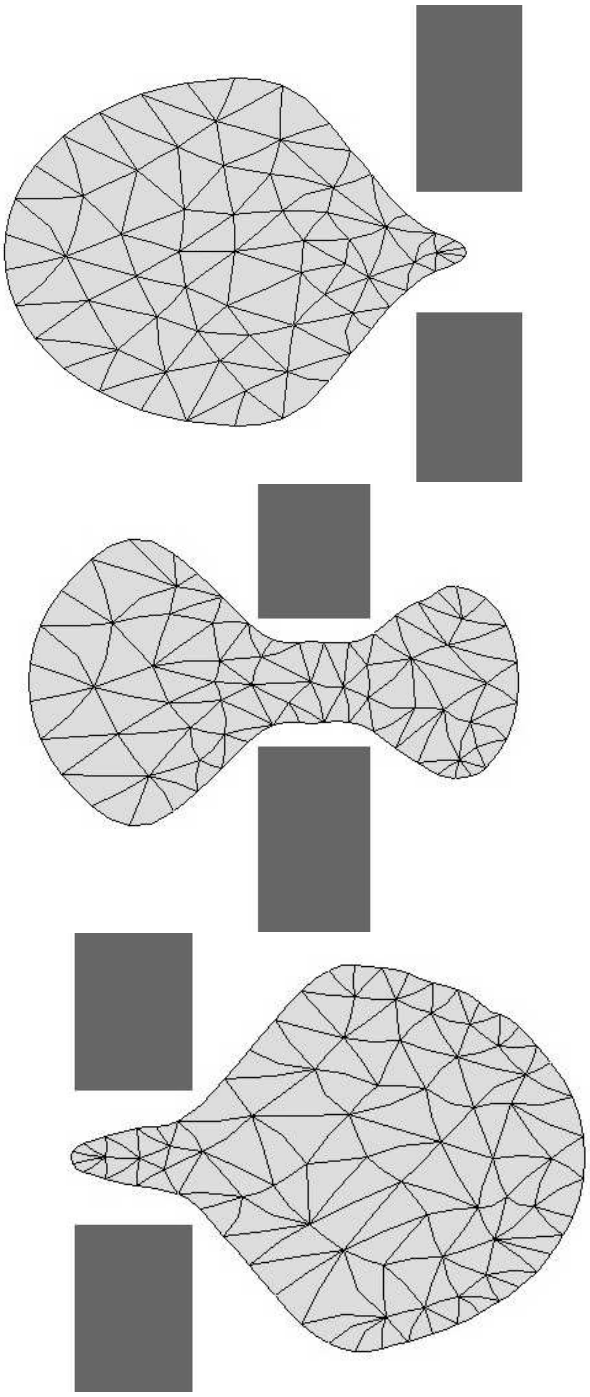

Figure 12: A cell body is pushed from left to right through a slit by the flow field. The body is compressed through the slit, and the mesh must be refined. As the body exits the slit, the mesh is coarsened.