A BÉZIER-BASED APPROACH TO UNSTRUCTURED MOVING MESHES

David Cardoze Alexandre Cunha[1] Gary L. Miller
Todd Phillips[2] Noel Walkington[2]

September 9, 2003

CS-CMU-03-166

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[1]Laboratory for Mechanics, Algorithms and Computing
[2]Department of Mathematics

This research was sponsored in part by National Science Foundation (NSF) grants no. CCR–9902091, CCR-9706572, and ACI-0086093. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the NSF or the US government.

Keywords: mesh generation, computational geometry, Bézier curves, Bézier triangles, B-splines, finite element method, quadratic elements

Abstract

We present in this report a new framework for maintaining good 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 curve smoothing operations at each time step. Our algorithms for these tasks are extensions of known algorithms for meshes build of straight–sided elements and are designed for any fixed-order Bézier elements and B-splines. We discuss a calculus of geometric primitives for Bézier curves and triangles that we employ to implement such operations. 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 *a priori* computed flow velocity field. As a pure geometric exploration, our method does not account for neither refinement nor coarsening dictated by the simulation results.

1 Introduction

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 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 on a complete remesh from scratch at each 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.

In our prototype system, we adopt a multilevel representation for meshes, from input specification to Delaunay refined meshes. In each level we make use of the cell-tuple data structure of Brisson [3, 4] which allows representing regular cell complexes of arbitrary dimension as well as permits access to ordering information and imposition of orientation on the complex. To our great benefit, we have augmented this data structure with self-loops and multiedges.

The rest of this paper is organized as follows: In section 3, we comment works related to ours. 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 to keep them well-shaped. Section 6 outlines local smoothing operations that are utilized for improving the quality of curved elements. Section 7 discusses contingencies related to small angles in the mesh that arise due to the mesh movement. In Section 8 we describe how all the respective procedures fit together in our simulation process and in Section 9 we describe our prototype implementation. In Section 10, we present some experimental results. Finally, in Section 11, we discuss future work.

2 The Sangria Project

The goal of the Sangria project 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.

From a computational perspective the main challenges in microstructural blood flow simulations are the development of numerical algorithms that stably and accurately deal with the interaction of moving and deforming domains and the moving fluids around them. 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. The target application is to simulate blood flow in three dimensions, here we focus first on two-dimensional simulations, utilizing many procedures which generalize well to higher dimension applications.

3 Related Work

Luo et al. [16] describe methods to obtain a mesh with curved elements starting from a straight mesh. They use Bernstein polynomials to describe the shape of their elements. They make use of operations such as edge split, edge swap and edge collapse to correct for invalid shapes. They look at the Jacobian of the map to determine shape validity. They however do not consider moving meshes.

Kuprat et al. [13] 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. They however deal with straight elements.

Also 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 generated a new mesh from scratch at every time step to deal with distortion as the mesh moves. Their mesh used linear elements and did not provide for mesh coarsening. Our approach is different in that we use quadratic elements for our meshes, and we do not recompute the mesh from scratch at every step. As an alternate approach, we make local modifications as the mesh moves to keep it well shaped.

4 Mesh and Element Types

4.1 Bézier Curves and Triangles

We decided to use Bézier curves and triangles to model our mesh elements for two main reasons. First, 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. Second Bézier curves and triangles have very nice mathematical properties that lead to elegant algorithms.

Bézier curves are completely defined by their control points which form the control polygon. See Figure 1. Similarly Bézier triangles are completely defined by their control points which form a control net. See Figure 2. For more information about Bézier curves and triangles see [9, 11] among others.

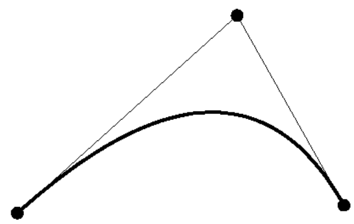

Figure 1: A quadratic Bézier curve: The quadratic curve is shown in bold. The control polygon consists of three vertices and two straight segments.

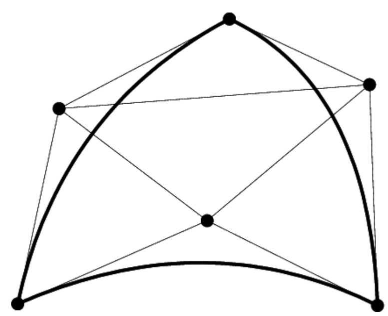

Figure 2: A quadratic Bézier triangle: The boundary of the quadratic triangle is shown in bold. The control net consists of six vertices and four straight triangles.

4.2 B-Splines

B-splines are a convenient way for us to represent C^1 continuous curves. We use B-splines in two important ways in our code. First, they allow us to represent object boundaries when we want to enforce C^1 continuity. In our target applications of simulating red cells moving down arteries we use quadratic B-splines to represent cell boundaries. 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. Second, B-splines make it much easier for us to move or slide a mesh point on the curve while only affecting the shape of the curve represented by the B-spline. Section 7 uses this property to slide mesh points on the curve to handle small boundary angles that may arise as the mesh moves.

See Figure 3. For more information regarding B-splines the reader can refer to [9, 11].

We move B-splines by sampling points on them and applying the flow field to them to obtain new positions. Then we perform a least-squares fit to obtain a new control polygon that best fits the new positions.

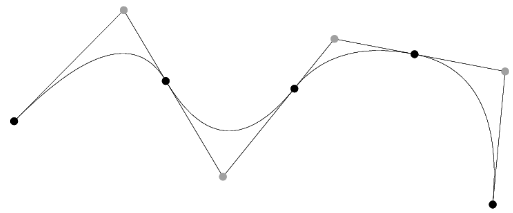

Figure 3: A quadratic B-spline and corresponding control polygon. The black points are the values the curve takes at the knots. The white points are the internal points of the control polygon.

4.3 Mesh Hierarchies

In our simulations we consider three different level of meshes: the Bézier mesh, the control mesh, and the logical mesh. The logical and control meshes are straight meshes. 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. The logical mesh is obtained from the Bézier mesh by replacing every curved triangle by the straight triangle that results from straightening every edge. See Figure 4. Figure 5 shows a Bézier mesh and its logical and control meshes.

As we will see below, different operations are performed on different levels of the mesh hierarchy. Also as the mesh moves, one of our invariants is that the logical mesh is Delaunay.

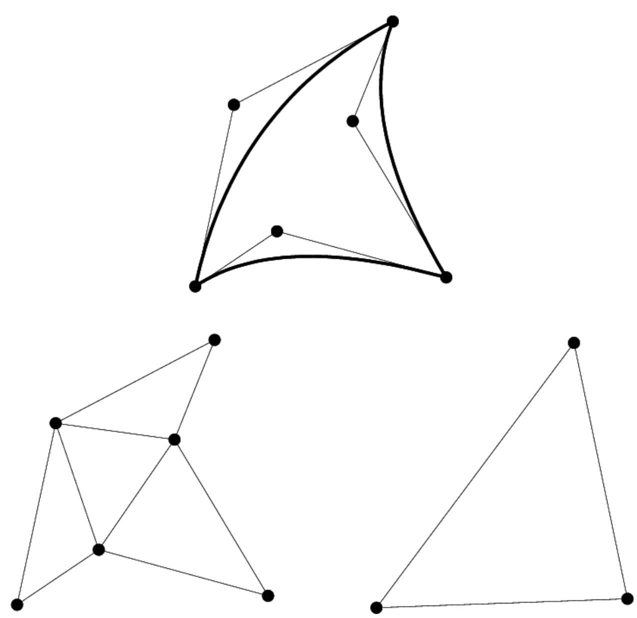

Figure 4: Bézier triangle and corresponding control (left) and logical triangle (right)

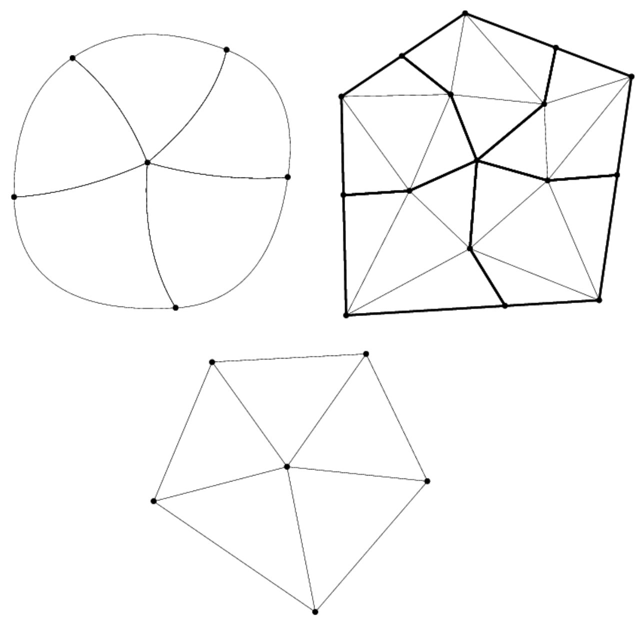

Figure 5: A Bézier mesh (left), its control mesh(right), and logical mesh(bottom)

5 Mesh and Element Operations

When utilizing curved meshes, many basic topological and geometric functions of the mesh must be handled in a delicate fashion. In 5.1, we describe implementations of some basic geometric procedures for Bézier elements. In 5.2, we use these primitives to implement a standard set of operations for a dynamic mesh.

At a high level our strategy to perform exact floating test for linear case such as line-side tests. But for more complicated test such as curve side test we will only provide function that correctly return one of three answers. Suppose that $C(t)$ is a parameterized curve, p is point, and $\epsilon > 0$ is a small constant then our test must return `below`, `above`, or t_0. If it returns `below` or `above` then p must be below or above the curve C. On the other hand, if the functions returns t_0 then $|p - C(t_0)| < \epsilon$. We can show that these approximate curve side test are fast and sufficient for our applications. Jnsson and Vavasis give exact algorithms for curve side test our hope was to find quicker and simpler methods, [12].

5.1 Geometric Primitives

5.1.1 Point in a Quadratic Bézier Triangle

In order to determine whether a point **p** is inside a quadratic Bézier triangle T, we proceed as follows. First we shoot a random ray r from **p** and count the number of carrier segments corresponding to the edges of T that r intersects. Second we count the number of carrier regions corresponding to the edges of T that **p** belongs to. The carrier segment of a Bézier curve is the straight segment between its endpoints. The carrier region is that region bounded by the curve and its carrier segment. See Figure 6. If the parity of the sum of two quantities is even, then $\mathbf{p}$ is outside of T, otherwise $\mathbf{p}$ is inside of T. As with all meshing applications, the effect of finite-precision calculations must be considered. For straight-line predicates, namely the determinant line-side test, the adaptive precision methods of [21] are used. For the determination of whether $\mathbf{p}$ belongs to a carrier region, we must use a more subtle technique. A general curve side test is implemented which recursively subdivides the Bézier curve using the DeCasteljou algorithm [9]. After a certain level of precision is reached, the algorithm termilnates and yields a point on the curve which is within ε of $\mathbf{p}$.

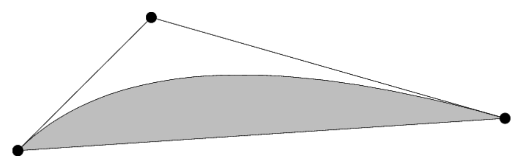

Figure 6: The carrier of a Bézier curve

5.1.2 Jacobian of a Bézier Triangle

When using curved elements, an important concern for the accuracy of the solution is the Jacobian matrix $J(\xi)$ for the function $B(\xi)$ that maps the reference element to the curved element, where ξ parametrizes the reference triangle in barycentric coordinates. This matrix may be examined to determine the validity of a curved element for quality metrics of curved elements as in [16].

When using Bézier elements, there is an important correspondence between the conditioning of the J and the geometric shape of the control net for an element. The quadratic Bézier triangle is presented herein, but the methods utilized are the same for higher order Bézier elements and are similar for Bézier tetrahedra. When using the Bernstein basis polynomials 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 $\mathbf{B}$ shown in Figure 7(b), so we have that:

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

For many applications 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 $\mathbf{C}_{i,j} = \mathbf{A}_i \times \mathbf{B}_j$, then convex combinations of these cross products are control points for $|J|$. The layout of these control points for $|J|$ is shown in Figure 7(c).

Using the convex hull property of polynomials in Bézier form, the $|J|$ for this element can be bounded by the maximum and minimum of $C_{i,j}$. A similar formulation for the Jacobian of Bézier simplices is discussed in [23]. In this light, it becomes necessary to maintain the quality of the triangles in the control mesh, which we further discuss in section 5.2.4.

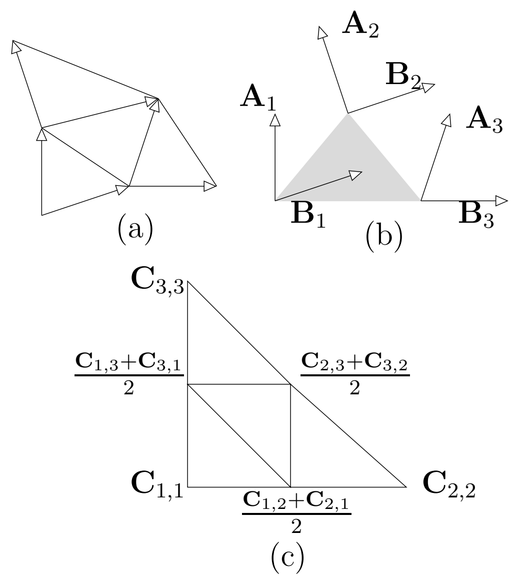

Figure 7: (a)Bézier control net for an element shown with the relevant vectors. (b) The Jacobian is a convex combination of the **A**-**B** pairs of vectors. (c) The determinant of the Jacobian as a Bézier polynomial

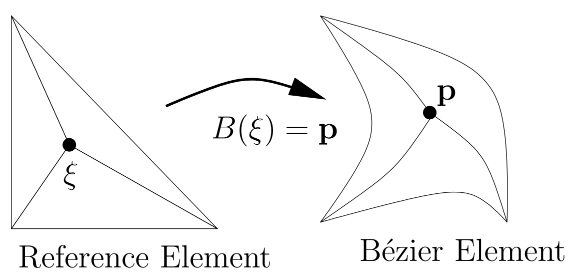

Figure 8: Finding reference coordinates for $\mathbf{p}$ is locating a root of the polynomial $B(\xi) - \mathbf{p}$.

5.2 Mesh Maintenance Operations

As the mesh domain moves with respect to the solution field, 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. The main operation is that of an edge-flip. This makes function interpolation as the mesh is modified easier.

5.2.1 Mesh Refinement

When the mesh domain changes, mesh elements may shift in a way that makes them unsuitable due to size or shape requirements. In either case, we use the traditional method of inserting circumcenters. The circumcenter of a curved element is taken to be the point $\mathbf{p}$ that is the circumcenter of the overlying logical triangle.

Upon establishing the geometric coordinates of $\mathbf{p}$, the point must then be topologically located in the curved mesh. This is important so that $\mathbf{p}$ may be inserted in a topologically valid way. The standard topological walk through the graph is generally viable, but may fail due to overly curved elements. Our approach is to supplement this method with the point in triangle primitive of section 5.1.1. The combination of these two methods yields a point location procedure that is reasonably fast in practice.

If $\mathbf{p}$ encroaches boundary edge, then instead of inserting $\mathbf{p}$ the boundary edge will be split. Encroachment of boundaries is determined by protective lenses around the boundary curves. The size for these lenses is an extension of Ruppert's Delaunay refinement algorithm [20]. Protective lenses for Bézier curves are defined in [19]. When a boundary edge is refined a knot is also inserted into the overlying boundary B-spline, introducing another degree of freedom in the piecewise boundary.

Once the element containing $\mathbf{p}$ has been located, it becomes necessary to find the local coordinates of the point in the element, for geometric reasons and for proper interpolation. If the geometric basis function for the curved element is B, then the problem is to find ξ such that $B(\xi) = \mathbf{p}$ (Figure 8). In our case, the function B is a quadratic in two variables, and a root may be found by Newton's method or any other standard procedure.

Having found the reference coordinates ξ, the point $\mathbf{p}$ may be inserted into the mesh using DeCasteljou's algorithm for the subdivision of the Bézier triangle into three new triangles. The subdivision process yields geometric basis functions for the new edges in the curved mesh.

5.2.2 Edge Flipping

Throughout the lifetime of the mesh, it is desirable to keep mesh elements from becoming too poorly shaped. One approach that yields quality meshes is to ensure that the overlying logical mesh has the Delaunay property. This can be accomplished by topologically flipping edges. The problem that arises is to define a new curved edge between the two elements.

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 9). This viewpoint has the advantage that it is very easily generalized to higher order Bézier elements. Upon flipping these four edges, the smoothing techniques described in Section 6 may then need to be applied to the control mesh in order to maintain valid and quality elements.

5.2.3 Vertex Deletion

The procedure to remove a vertex $\mathbf{v}$ from the mesh is based on the algorithm of Deviller [7] that deletes a vertex of a 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 $\mathbf{v}$ has the form shown in figure 10. At this point $\mathbf{v}$ is deleted from the topology of the mesh. The three edges and the three triangles adjacent to $\mathbf{v}$ are deleted, and the triangle formed by the three vertices that were adjacent to $\mathbf{v}$ is added.

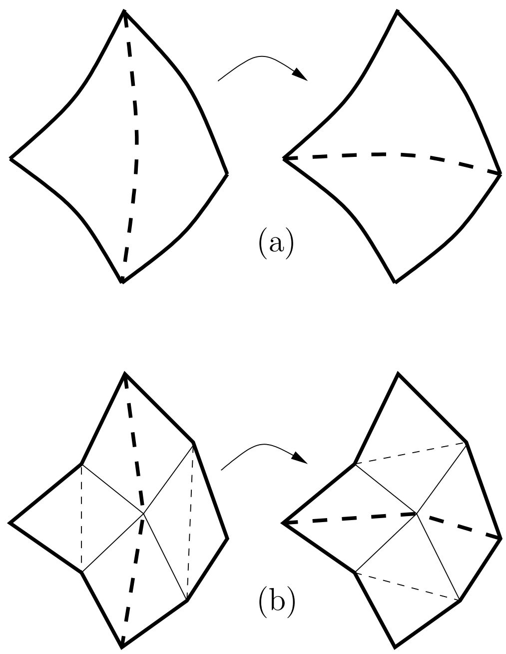

Figure 9: (a) The dashed edge is flipped in the curved mesh. (b) The corresponding dashed edges are flipped in the control mesh.

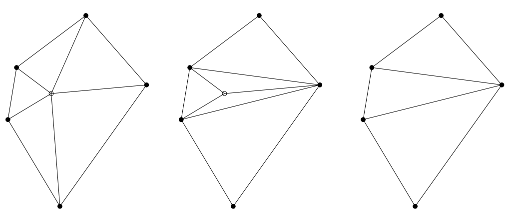

Figure 10: The star of a point to be deleted (left). The configuration after all new Delaunay ears have been obtained but before the vertex is removed (middle). The mesh with the vertex removed (right)

5.2.4 Curve Smoothing

In addition to maintaining the quality of the logical mesh as in 5.2.1, it is also necessary to manage the curvature of individual elements. As a quality metric for the curvature of a Bézier triangle, we propose the use of the ratio of the determinant of the Jacobian ($|J|$) over the area (A) of the Bézier triangle, both of which can be easily computed. For linear triangles, this ratio is always one, which motivates a natural quality metric for a Bézier triangle through measuring the deviation of this ratio from one. It is also important that neither the area nor the Jacobian determinant be negative anywhere in the Bézier triangle, as this results in an invalid element. Using this criteria to identify poor quality elements, we make improvements by smoothing the control points of the Bézier triangles in question. Smoothing techniques are applied to points in the control mesh as in Figure 11 and in Section 6. The improvement in the geometric quality of the control net for the the Bézier triangle

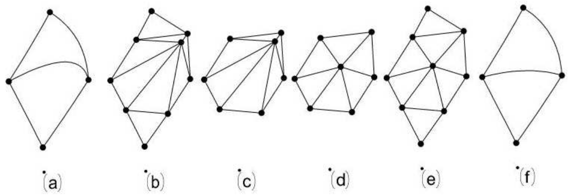

Figure 11: (a) An edge to be smoothed is identified based on it's quality. (b) The control mesh (c) Isolate the star of the control point to be move (d) Move control point based on linear mesh improvement techniques (e) Update the control mesh (f) better quality elements

improves the determinant of its Jacobian as shown in section 5.1.2.

5.2.5 Mesh coarsening

Mesh coarsening is required to keep the mesh from becoming too large. We perform mesh coarsening by means of the Douglas-Peucker algorithm [8] and the function-based coarsening paradigm of Talmor et al. [17], [22].

The main idea of the function-based coarsening paradigm is as follows. First a coarsening function for the mesh is obtained. This is an (approximate) Lipschitz function which describes the desired spacing of the mesh nodes. The amount by which a Lispchitz function varies from one point to another is bounded, and hence it is well suited to express the spacing of well-graded meshes. At every point of the mesh a scaled value of the coarsening function is used to define the radius of a sphere centered at that point. A maximal subset of the mesh nodes is seleteced so that none of the corresponding spheres intersect. The selected points are kept and the other ones are removed.

This basic approach has to be modified when there are boundaries that have to be kept, as in our case. Our overall procedure to smooth the mesh is as follows. We run the Douglas-Peucker algorihm on the control polygon of our B-spline boundaries to determine a subset of the control points to be kept. 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 procedure described above.

We should also mention that contrary to other approaches to mesh coarcening, once we have selected a set of mesh nodes to be kept, we don't recompute a new mesh from scracth. Starting from the initial mesh, we remove one by one the vertices that were not selected using the vertex deletion algorithm we described above.

6 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 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 11). 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, 10]. Each smoothing call solves

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

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 12 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.

In our prototype, we adopted CFSQP [14] 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 [15]. 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

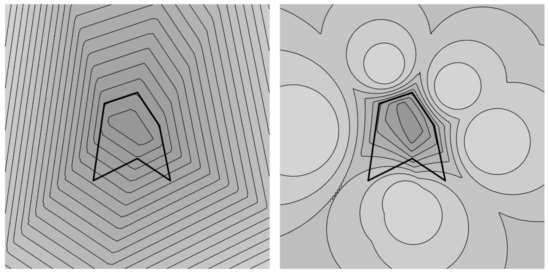

Figure 12: Level curves of Φ for area (left) and quality metric τ. Note in the right figure the sinks outside the polygon.

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.

7 Handling Small Angles

As the mesh domain changes and the quality of the mesh is maintained, it is very important to prevent small angles in the mesh from causing unbounded refinement due to the ping-pong effect that is caused due to mutual encroachment. Heuristics for handling small boundary angles are well understood in the static case [18], but issues arise when the mesh is dynamic.

The first problem is that small boundary angles may arise during the simulation which were not small angles at the outset. This is solved by bookkeeping near all angles in the input boundaries as in Figure 13(a). The task seems tedious, but in practice there are relatively few boundary angles since the boundaries are represented as differentiable B-splines.

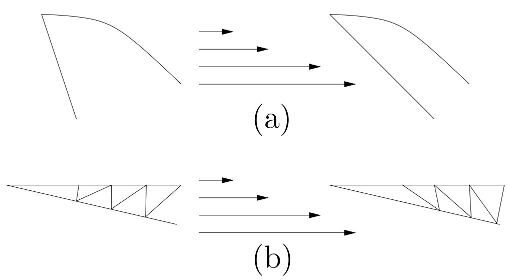

Figure 13: (a) Small angles can arise due to shearing forces. (b) Shearing forces can also cause stable configurations to become poorly shaped.

The method of corner-lopping, originally proposed by Ruppert[20], is employed to create isosceles triangles at input angles. Because the boundaries have curvature, they are refined so that the secant lengths of the neighboring mesh edges are equal, as in [6, 2]. The corner-lopping method is useful for preventing mutual encroachment. But as Figure 13(b) illustrates, when the domain is next moved the carefully constructed isosceles configuration goes awry. It is undesirable to refine near this angle at every time step, but it is also undesirable to allow the triangle spanning the small angle to be sheared into a poorly conditioned obtuse shape. The method of secant length snapping is now introduced as a solution.

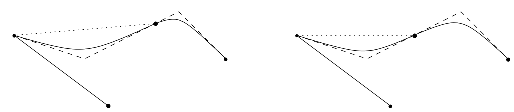

Figure 14: Secant length (dotted line) near small angles can be adjusted using the knot sequence of boundary B-splines (deBoor controls shown dashed).

The knot sequence of the underlying B-spline of one of the boundary edges may be adjusted *without* changing its DeBoor control points in order to vary the position of the knot in the B-spline. This allows flexibility in the secant length of the quadratic piece in question, with only a small local change to the boundary shape. The knot may be allowed to slide along the line connecting the two relevant DeBoor control points. If a suitable secant length cannot be achieved by adjusting the knot sequence (within a reasonable tolerance), then the secant length snapping procedure will yield to refinement of the boundary.

8 The Simulation Process

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 several stages. First the given solution for velocity is approximated as a piecewise quadratic on the control mesh. If the solver in question utilizes the geometric basis functions as a finite-element basis then this projection is trivial, otherwise a least-squares projection is used.

Next the differentiability condition on the boundary B-splines must be enforced. The piecewise continuous solution in the control mesh is projected to a new B-spline using least squares. The geometric basis of the control mesh is then updated to reflect this change and re-interpolation is done.

Now that an acceptable velocity solution has been found for the control mesh, the entire mesh is moved forward using ordinary differential equation methods for the velocity.

After the mesh has been moved, it remains to clean the mesh using the methods of section 5.2. First the Delaunay property is enforced on the logical mesh using edge flips as described in 5.2.2. Triangles with poor curvature are then smoothed using control point smoothing. Lastly skinny triangles are eliminated through circumcenter and midpoint insertions.

9 The Prototype

We currently have a prototype that implements the ideas that we have described. It is written using the object oriented scripting language Ruby, and extensions for OpenGL and

GLUT.

The base class to represent geometric objects in our prototype is the **CellComplex** class. It is based on the cell-tuple approach of Brisson [3]. An object of this class is a collection of objects of class **Cell** which can have arbitrary dimension. A **CellComplex** object stores the incidence relation among these cells. From this class we derive the **BézierMesh** class which represents two dimensional meshes where the 1-cells and 2-cells are Bézier curves and triangles respectively of arbitrary degree. These edges and triangles are represented by objects of class **BézierEdge** and **BézierTriangle** both of which are derived from the class **Cell**.

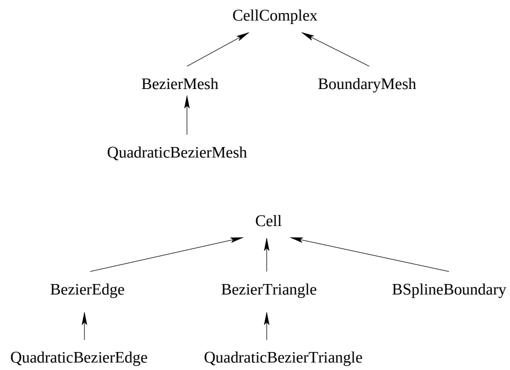

Figure 15: The Class Hierarchy

From the **BézierMesh** class we derive the **QuadraticBézierMesh**. This class represents Bézier meshes where the edges and triangles are of degree two, which are represented by objects of class **QuadraticBézierEdge** and **QuadraticBézierTriangle** which are derived from **BézierEdge** and **BézierTriangle** respectively. In the current implementation, all of our meshes are instances of the **QuadraticBézierMesh** class.

We model object boundaries with quadratic B-splines which are implemented by the class **QuadraticBSplines**. The set of boundaries is modeled with an object of the class **BoundaryMesh** which is derived from the **CellComplex** class. Objects of the class **BoundaryMesh** are two-dimensional complexes where every edge is a quadratic B-spline boundary which are represented by the **BSplineBoundary** class. The edges can be self-loops, and in fact in most cases they are.

The class **Flow** implements flow fields. It has methods to track the motion of particles in the flow using a Runge-Kutta method of order 4.

The simulation itself is carried out by an object of the **Simulator** class. An object of these class tracks a quadratic Bézier mesh and the associated quadratic B-spline boundaries as they move in a given flow. Finally there is a driver routine which class the simulator at given time steps and calls the operations described above to keep the mesh and the boundaries well formed as time evolves.

10 Experimental Results

Figure 16 shows a cross-sectional photograph of red blood cells moving past an obstacle in blood flow and becoming distorted. We have run several simulations which mimic the flow of red blood cells past a cylindrical obstacle in order to examine the capabilities of the geometric mesh maintenance operations in section 5.2. For these simulations, we have assumed the a priori existence of a flow field around the obstacle. Only the cells themselves are meshed and updated, so no interaction between the cells or the containing fluid is modeled. Flipping operations are utilized to maintain the Delaunay property for the logical mesh, and vertex insertion is used to maintain a minimum angle in the logical mesh of .45 radians ($\approx 25.7°$). The entire mesh is coarsened every 20 timesteps using a geometric spacing function to increase the distance to the nearest neighbor. Figures 17-20 show several snapshots of the simulation running.

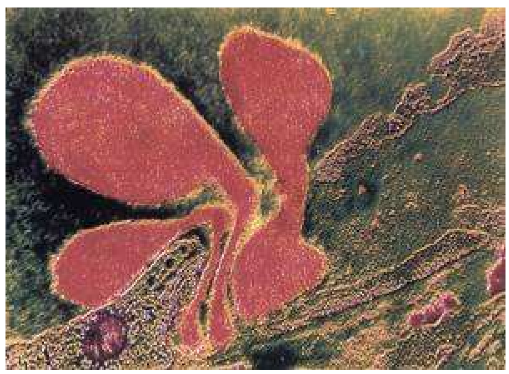

Figure 16: A cross section of red blood cells moving past an obstacle.

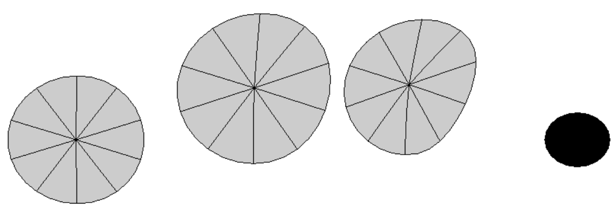

Figure 17: Initial configuration for the simulation. A flow field moves the gray cells around the black hole.

11 Future Work

We have presented work in progress towards the development of a simulation system for unstructured moving meshes for Lagrangian methods. Aside from the goals of extension to three dimensions, there are several other goals for design of curved moving mesh simulations.

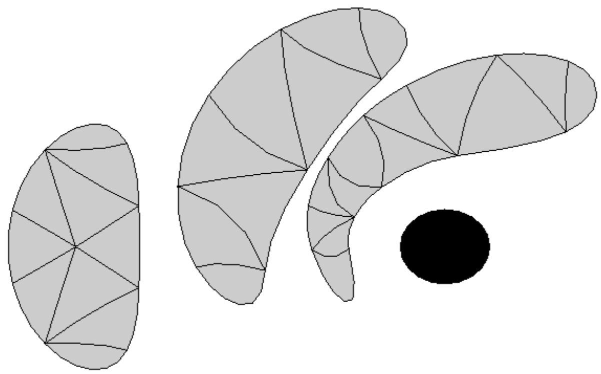

Figure 18: Differentiable boundaries are maintained with B-splines.

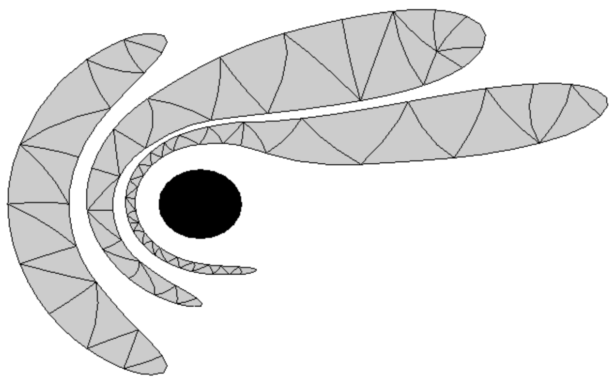

Figure 19: Boundary refinement occurs in areas of increased curvature to maintain a differentiable boundary.

There are several numerical constants involved in the mesh coarsening procedures. We would like to do more experiments to determine optimal values for them. Further work is also necessary in examining the interplay of dynamic interfaces within the mesh. Applying elastic constraints on boundaries affects the methods for which continuity is maintained. A model that can accomodate the tearing of boundaries is also desired in some moving mesh simulations and must be examined. We are also considering NURBS to determine if we would gain from their use. In addition we are in the process of adding code to peform function interpolation on the mesh as it is modified by operations such as edge flips.

12 Acknowledgements

We would like to thank Clemens Kadow and Guy Blelloch for helpful discussions.

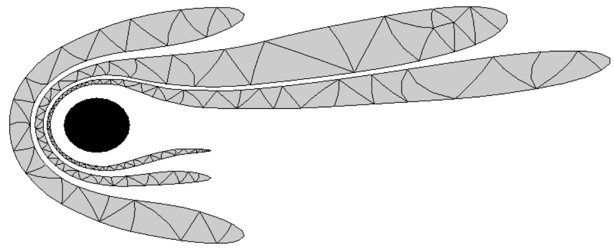

Figure 20: Curve-smoothing maintains good curvature of Bézier triangles.

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] 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.

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

[4] E. Brisson. Representing geometric structures in d dimension: Topology and order. *Discrete and Computational Geometry*, 9:387–426, 1993.

[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] Alexandre Cunha, Scott Canann, and Sunil Saigal. Automatic boundary sizing for 2D and 3D meshes. In *Trends in unstructured mesh generation*, volume AMD-220, pages 65–72. The American Society of Mechanical Engineers, 1997.

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

[8] 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.

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

[10] 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.

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

[12] G. Jónsson and S. Vavasis. Accurate solution of polynomial equations using macaulay resultant matrices. unpublished, Feb 2001.

[13] 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.

[14] 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.

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

[16] Xian-Juan Luo, Mark S. Shepard, 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.

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

[18] Steven Elliot Pav. *Delaunay Refinement Algorithms.* PhD thesis, Department of Mathematics, Carnegie Mellon University, Pittsburgh, Pennsylvania, May 2003.

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

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

[21] Jonathan Richard Shewchuk. Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates. *Discrete & Computational Geometry*, 18(3):305–363, October 1997.

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

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