# THREE-DIMENSIONAL REPRESENTATION OF EVOLVING INTERFACES IN COMPUTATIONAL FLUID DYNAMICS

By
ABHISHEK KHANNA

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2000

ACKNOWLEDGEMENTS

I would sincerely like to thank Dr. Wei Shyy for his useful suggestions, belief in my capabilities and for extending my financial support to continue this work. I express my gratitude to Dr. H.S. Udaykumar for his constant support and guidance throughout this work. I would also like to thank him for spending a considerable amount of his valuable time with me during the course of this research work. I wish to thank Dr. Jorg for his help and for the useful discussions we have had together. My thanks are due to Vivek Jayaraman for helping me in getting familiar with the tracker code developed by him. I would also like to thank Ramji Kamakoti and Rajkumar Vaidyanathan for helping me in making my report. Equally, I wish to express my heartfelt gratitude to my parents for their endless love and encouragement.

TABLE OF CONTENTS

page

ACKNOWLEDGEMENTS ..... ii

NOMENCLATURE ..... v

ABSTRACT ..... vii

CHAPTERS

1 INTRODUCTION ..... 1

Moving-boundary Problems ..... 1
Three-dimensional Interface Tracking ..... 3
Overview of the Report ..... 5

2 REVIEW OF CURVES AND SURFACES ..... 6

Implicit and Parametric Forms ..... 6
Bézier Curves ..... 8
Polynomial Bézier Curves ..... 8
Rational Bézier Curves ..... 12
Bézier Surfaces ..... 13
Polynomial Tensor Product Bézier Surfaces ..... 14
Rational Bézier Surfaces ..... 15
B-Spline Curves ..... 16
Polynomial B-Spline Curves ..... 17
Nonuniform Rational B-Spline Curves ..... 21
B-Spline Surfaces ..... 22
Polynomial B-Spline Surfaces ..... 22
Nonuniform Rational B-Spline Surfaces ..... 23
Bézier Triangles ..... 24
Polynomial Bézier Triangles ..... 24
Rational Bézier Triangles ..... 28

3 INTERFACE TRACKING METHOD ..... 30

$C^1$ Surface Spline Algorithm ..... 34
Mesh Refinement ..... 40
Point Insertion ..... 42
Diagonal Swapping ..... 46
Mesh Update ..... 48
Mesh Update on $C^1$ Spline Surface ..... 48
Mesh Update on Surface Triangulation ..... 50

4 RESULTS ..... 57

Complex Feature Representation ..... 57
Grid Refinement ..... 62
No Grid Refinement ..... 68
Grid Adaptation ..... 70

5 CONCLUSIONS AND FUTURE WORK ..... 81

APPENDICES

A POINTS AND VECTORS ..... 83

B BARYCENTRIC COORDINATES ..... 85

C DATA STRUCTURES FOR INTERFACE TRACKING ..... 86

Node Data Structure ..... 86
Edge Data Structure ..... 86
Triangle Data Structure ..... 87
Surface Mesh Data Structure ..... 87

REFERENCES ..... 89

BIOGRAPHICAL SKETCH ..... 91

## NOMENCLATURE

The following list gives the meanings of symbols that appear in this report.

| | |
|---|---|
| $A, B, C$ | Coefficients of Dupin indicatrix |
| $A_i$ | Edge coefficient |
| $a, b$ | Real numbers |
| $a_{i1}, a_{i2}$ | Blend ratios of cell $i$ |
| $B_{\mathbf{i},n}$ | Bivariate Bernstein polynomial of degree $n$ |
| $B_{i,n}$ | Univariate Bernstein polynomial of degree $n$ |
| $\mathbf{C}$ | Curve |
| $C'$ | Preliminary cell center |
| $C$ | Cell center |
| $c$ | Cell |
| $\mathbf{d}$ | Direction vector |
| $d, e, f$ | Elements of direction vector $\mathbf{d}$ |
| $d$ | Distance of projection of new point |
| $\mathcal{E}$ | Euclidean or point space |
| $E, F, G$ | Coefficients in the mean curvature equation |
| $\mathbf{e1}, \mathbf{e2}, \mathbf{e3}$ | (1, 0, 0), (0, 1, 0) and (0, 0, 1) respectively |
| $F$ | Function to be minimized to get Dupin indicatrix |
| $f$ | Real-valued function |
| $H$ | Mean curvature |
| $k1, k2, k3$ | Mean curvature values at the nodes of a triangle |
| $L$, $M$, $N$ | Coefficients in the mean curvature equation |
| $m, n$ | Integers |
| $m$ | Scaling factor of Dupin indicatrix |
| $N_{i,p}$ | B-spline basis function of degree $p$ |
| $\mathbf{n}$ | Normal vector |
| $\mathbf{P}$ | Point on the control polygon |
| $P$ | Bezier coefficients of a cubic triangular patch |
| $p, q$ | Integers |
| $\mathbf{Q}$ | Intermediate point |
| $Q$ | Bezier coefficients of a quadratic triangular patch |
| $\mathcal{R}$ | Linear or vector space |
| $R$ | Radius of a sphere |
| $R_{i,n}$ | Rational basis function of degree $n$ |
| $r$, $s$ | Integers |
| $r$ | Circumradius of a triangle |
| $\mathbf{S}$ | Surface |
| $\mathbf{t}$ | Unit tangent vector along an arbitrary direction |
| $\mathbf{t}_i$ | Direction represented by unit tangent vector |
| $\mathbf{t}_1, \mathbf{t}_2$ | Tangent directions corresponding to principal curvatures |

| | |
|---|---|
| $U, V$ | Knot vectors |
| $\mathbf{u}$ | Point in the parametric plane |
| $\mathbf{u}, \mathbf{v}$ | Vectors |
| $u, v$ | Coordinates of a point in the parametric plane |
| $\overline{V}$ | Vertex |
| $V$ | Vertex |
| $\mathbf{v}_n$ | Velocity along normal direction |
| $\mathbf{v}_1, \mathbf{v}_2$ | Vectors |
| $v_x, v_y, v_z$ | Velocity along $x$, $y$ and $z$ directions |
| $W, X, Y$ | Polynomial functions |
| $w$ | Weight |
| $w_n$ | Constant coefficient |
| $\mathbf{x}$ | Point in Euclidean space |
| $x, y, z$ | Coordinates of a point $\mathbf{x}$ |
| $\mathbf{x_c}$ | Circumcenter of a triangle |
| $\kappa_{n,i}$ | Normal curvature in direction $\mathbf{t}_i$ |
| $\kappa_{\mathbf{t}}$ | Normal curvature along direction $\mathbf{t}$ |
| $\kappa_1, \kappa_2$ | Principal curvatures |
| $\Psi$ | Angle between arbitrary normal section and $\mathbf{t}_1$ |
| $\Theta$ | Angle between an arbitrary plane and tangent plane |
| $\alpha_i$ | Constant coefficient |
| $\epsilon$ | Distance between tangent plane and plane of intersection |

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

THREE-DIMENSIONAL REPRESENTATION OF EVOLVING INTERFACES IN COMPUTATIONAL FLUID DYNAMICS

By

Abhishek Khanna

December 2000

Chair: Wei Shyy
Major Department: Aerospace, Engineering Mechanics and Engineering Science

Moving boundary problems arise in numerous important physical phenomena and they often form complex shapes during their evolution. Numerical methods for the solution of moving boundary problems primarily focus on interface tracking and on the solution of the field equations. The ability to track interfaces in two dimensions is well established. However, modification of the grid, representing the interface, as it evolves in a three-dimensional space introduces additional issues.

In the current work, moving interfaces are represented by triangulated surface grids. Continuous spline surfaces are also used as an alternative means of surface representation. The unstructured triangulated grids are restructured and refined based on the shape and size of the triangular elements in the grid. As the interface deforms, points are automatically added and mesh connectivity modified to ensure that the accuracy of the interface representation remains consistent.

Normals at the nodes of the surface mesh are required for the movement of the boundary and curvature plays an important role in problems involving capillarity. In the current work, methods for computing geometric features such as normals and curvatures on a triangulated surface are discussed. Curvature calculations are based on the mean of normal curvatures and on approximations to the Dupin indicatrix. Normals and curvatures are also computed on the parametric spline surface. This is done by the calculation of the partial derivatives of the coordinates of the surface nodes with respect to the parameter values.

Results are presented to show the complex feature representation capability of the tracker code. A comparison between the different methods, used for computing curvatures, is made here. Finally, the effect of mesh quality and accuracy of the point location on the curvature error is also investigated.

# CHAPTER 1
# INTRODUCTION

## Moving-boundary Problems

Moving-boundary problems arise in a variety of important physical phenomena. Some examples are materials processing, biofluid dynamics, flame propagation and fluid-structure interactions [1]. Such systems are often characterized by internal boundaries or by interfaces demarcating regions with different transport properties and mechanisms. Across these interfaces compositions, phases, material properties and flow features can change rapidly. The movement of interfaces is influenced by the flow field and the interfaces, in turn, affect the behavior of the flow. During their movement, the interfaces may form complex shapes and may change topologically while undergoing severe deformations such as dilatation, compression, fragmentation and collision.

The main challenge to numerical methods for the solution of moving boundary problems is that the boundary location and shape must be determined as part of the solution. The evolution of the boundary is closely coupled with transport of mass, continuity, momentum and energy. Within each domain, field equations must be solved with the location of the internal boundary being determined simultaneously. The interface is often a discontinuity and there is need to track it accurately in both time and space within the limits of finite grid resolution.

The challenges posed by fluid flow problems involving moving boundaries have resulted in a variety of computational techniques directed toward their solution [1]. These numerical methods can typically be classified as either Eulerian or Lagrangian [2]. Eulerian methods such as the level set method [3, 4] and the volume of fluid (VOF) method [5] do not keep track of the position and shape of the interface explicitly whereas Lagrangian methods such as the marker and cell (MAC) method [6] follow the interface explicitly, keeping track of both position and shape. While this makes the Lagrangian methods more complex, there is a significant advantage in knowing the interface position explicitly.

In tracking distorted interfaces with accuracy, a combination of the strengths of both Eulerian and Lagrangian methods has been shown to be useful for problems involving fluid-fluid (Unverdi and Tryggvason [7]) as well as solid-liquid (Udaykumar et al. [8]) interactions. These methods employ fixed structured grids and afford the simplicity and availability of well-tested flow solvers. The only moving component is the interface which is a lower-dimensional surface overlying the fixed grid and is explicitly tracked.

Solution of a moving boundary problem involves interaction of the flow solver with the interface tracking components. The effects of internal stationary as well as moving boundaries can be communicated to the flowfield by either incorporating changes in the control volume definitions that take place due to intersection of the interface with the background mesh (cut-cell approach [9]) or appropriate source term contributions (immersed boundary technique [10]). Information from the flowfield is transferred to the boundaries by interpolation.

The ELAFINT (Eulerian-Lagrangian Algorithm For INterface Tracking) method [8] is currently operational for the solution of two-dimensional problems. The current work is aimed at extending the ELAFINT algorithm by giving it three-dimensional interface tracking capabilities.

## Three-dimensional Interface Tracking

Representation of interface in two dimensions is done by connecting a series of points describing the curves. Points on a curve always have two neighbors and thus storing the connectivity between the points does not have to be separate from storing the points themselves if the ordering of points is maintained appropriately. Addition of points to the curve is also straightforward as the new points can be inserted at appropriate locations into an already-existing list of points. In three dimensions, however, the task is much more difficult. To represent arbitrary deformations of a boundary, the surface is tesselated using triangles. In terms of data structures alone, there is no set ordering of points that suggests itself. Thus, connectivity information cannot be stored implicitly if any flexibility in changing the mesh is desired.

The need to store connectivity information explicitly calls for the interface to be represented as a surface mesh. If the collection of points is stored as a structured grid, there is no flexibility in changing the mesh. Arbitrary addition or deletion of points is difficult when a complete regridding procedure has to be carried out at every time instant. The natural choice in such circumstances is to use an unstructured triangulated surface mesh to represent the surface.

Using triangular-element unstructured mesh offers several advantages. The grid can be restructured. Individual triangles and sides can be bisected or rearranged

to give new grid structures that better represent the evolving interface. Since the number of triangles meeting at a vertex is variable, increased accuracy in one region does not force unnecessary resolution in other areas. Triangles can cover a sphere more satisfactorily than rectangles without cusps or other local representation irregularities. Thus, free surfaces, complicated interfaces and boundaries of immersed objects can be represented accurately and economically.

A general-connectivity triangulated mesh involves nontrivial bookkeeping tasks associated with the grid and its connections as all the data structures have to be updated during triangle, edge and vertex additions, reconnections and subtractions. The present approach uses unstructured grids to represent the interface while the background (stationary) grid is structured. The interface is initialized by means of a triangulation provided by standard CAD software. Since the initial connectivity information is available to us, the issues of unstructured grid generation, from discrete point data, involving complex geometries can be bypassed.

After an initial interface position and shape is defined, the interface must move based on the velocity field. For the movement of the interface it is often required to know the normal at every node of the surface mesh. For the calculation of the surface forces which are distributed to the background structured mesh employed by the field equation solver, information about the curvature at all surface points is needed. In this work, normal and curvature calculations are performed on the parametric spline surface as well as on the triangulated mesh and accuracy of these methods is discussed.

While the generation of both two-dimensional planar as well as three-dimensional volume unstructured grids [11] is now not uncommon and largely straightforward, it is

important to note that unstructured surface grid generation is not. Further, methods to adapt irregular surface grids by adding surface points to an already-existing set of points are still being developed. In order to track arbitrarily moving interfaces successfully, it becomes important to deal with triangulation of sets of points that are not Delaunay triangulated at every time instant. This capability is demonstrated by the current work. The interface tracking code has been tested on various cases documented here which reveal its versatality and accuracy.

**Overview of the Report**

The first chapter of the report presents an introduction to the topic of moving boundary problems and gives a review of the numerical methods used for solving these problems. Interface tracking issues which form a part of the solution of moving boundary problems have also been discussed here. Chapter 2 deals with the basics of Bézier and B-spline curves and surfaces that form the theoretical basis of the $C^1$ surface spline algorithm discussed later in the report. Chapter 3 presents the interface tracking algorithm. Algorithms for mesh refinement and parametric surface approximation have been discussed in detail. Methods used for computing normals and curvatures at the nodes of the interface have also been highlighted. Chapter 4 shows several examples which demonstrate the capabilities of the interface-tracking code developed so far. Conclusions drawn from the current work are presented in Chapter 5.

# CHAPTER 2
# REVIEW OF CURVES AND SURFACES

This chapter reviews some of the modern methods used for representing surfaces in three-dimensions. Our particular interest is in the description and computation of surface features such as normals and curvatures for a general triangulated surface. The most popular methods for this purpose are currently Bézier and B-spline curves and surfaces [12, 13]. These parametric elements of curve and surface description have become the standard in geometric modeling. In this chapter we will detail the mathematical basis of these representations and point out their usefulness or otherwise vis-a-vis the accuracy of surface representation.

## Implicit and Parametric Forms

The two most common methods of representing curves and surfaces in geometric modeling are implicit equations and parametric functions. An implicit equation of a curve in the $xy$ plane has the form $f(x, y) = 0$. This equation describes an implicit relationship between the $x$ and $y$ coordinates of points lying on the curve. In the parametric form a curve can be represented as $\mathbf{C}(u) = (x(u), y(u))$ where $a \leq u \leq b$. In this form, each of the coordinates of a point on a curve is represented separately as an explicit function of an independent parameter $u$. The parametric representation of a curve is not unique. It is instructive to think of $\mathbf{C}(u) = (x(u), y(u))$ as the path traced out by a particle as a function of time where $u$ is the time variable and

$(a, b)$ is the time interval. The first and second derivatives of $\mathbf{C}(u)$ are velocity and acceleration of the particle respectively. A comparison of implicit and parametric methods is given as follows:

- By adding a $z$ coordinate, parametric method can be extended to represent arbitrary curves in three dimensional space, $\mathbf{C}(u) = (x(u), y(u), z(u))$. The implicit form only specifies curves in $xy$ (or $xz$ or $yz$) plane.

- It is cumbersome to represent bounded curve segments (or surface patches) with implicit form. Boundedness is, however, built into the parametric form through the bounds on the parametric interval.

- The parametric form is more natural for designing and representing a shape in a computer. The coefficients of many parametric functions possess considerable geometric significance and explain many properties of these curves.

- The complexity of many geometric operations depends greatly on the method of representation. For example, it is difficult to compute a point on a curve or surface in the implicit form whereas in the parametric form, it is not directly possible to determine whether a given point lies on the curve or surface.

- In the parametric form, sometimes one has to deal with parametric anomalies which are unrelated to the true geometry. For example, poles in a sphere, where all the longitudnal lines meet, are critical points in a parametric form although geometrically they are no different from any other point on the sphere.

- Parametric curves possess a natural direction of traversal (from $\mathbf{C}(a)$ to $\mathbf{C}(b)$ if $a \leq u \leq b$) whereas implicit curves do not.

From the above comparison, parametric approach appears to be an attractive option for representing geometry in a computer. The primary disadvantage, that is parametric anomalies, can be avoided by using parametric patches.

## Bézier Curves

### Polynomial Bézier Curves

These are parametric polynomial curves described over a set of given points. An $n^{th}$ degree Bézier curve, with parameter $u$, is defined as

$$\mathbf{C}(u) = \sum_{i=0}^{n} B_{i,n}(u)\mathbf{P}_i \qquad 0 \leq u \leq 1 \tag{2.1}$$

Here the basis (blending) functions $\{B_{i,n}(u)\}$ are classical $n^{th}$ degree Bernstein polynomials given by

$$B_{i,n}(u) = \frac{n!}{i!(n-i)!}u^i(1-u)^{n-i} \tag{2.2}$$

The geometric coefficients of this form $\{\mathbf{P}_i\}$ are called *control points*. The polygon formed by the points $\{\mathbf{P}_i\}$ is called the *control polygon*. Figure 2.1 shows a second-degree ($n = 2$) Bézier curve along with its control points.

In any curve (or surface) representation scheme, the choice of basis functions determines the geometric characteristics of the scheme. Some important properties of basis functions can be listed as follows:

- **Symmetry**: For any $n$, the set of polynomials $\{B_{i,n}(u)\}$ is symmetric with respect to the parameter value $u = 1/2$.

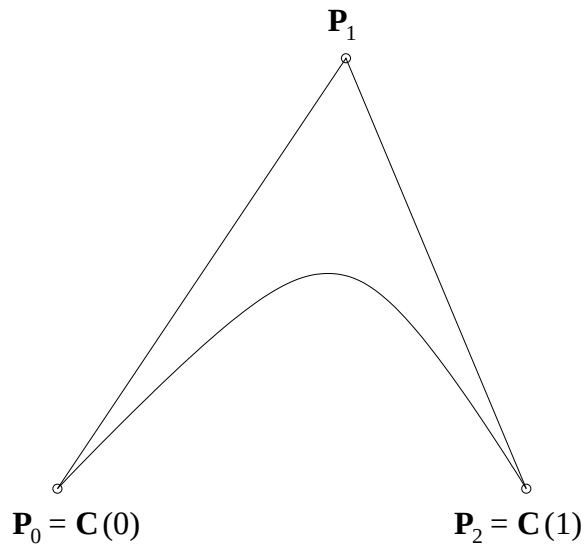



Figure 2.1: A second-degree Bézier curve along with its control points.

- $B_{0,n}(0) = B_{n,n}(1) = 1$ hence, a Bézier curve passes through its first and last control points. For an arbitrary set of points the curve does not pass through interior control points.

- **Nonnegativity**: $B_{i,n}(u) \geq 0$ for all $i, n$ and $0 \leq u \leq 1$.

- **Recursive definition**: $B_{i,n}(u) = (1-u)B_{i,n-1}(u)+uB_{i-1,n-1}(u)$ with condition $B_{i,n}(u) \equiv 0$ if $i < 0$ or $i > n$.

- $B_{i,n}(u)$ attains exactly one maximum on the interval $[0, 1]$.

- **Partition of unity**: $\sum_{i=0}^{n} B_{i,n}(u) = 1$ for all $0 \leq u \leq 1$. Partition of unity and nonnegativity property ensure a set of positive basis functions whose sum is equal to 1. From Equation 2.1, it is evident that points on a Bézier curve are convex barycentric combinations (Appendix A) of the specified control points. Therefore, these points lie within the convex hull of the control points of the corresponding Bézier curve.

- **Derivative**: $B'_{i,n}(u) = \frac{dB_{i,n}(u)}{du} = n(B_{i-1,n-1}(u) - B_{i,n-1}(u))$ with conditions $B_{-1,n-1}(u) \equiv 0$ and $B_{n,n-1}(u) \equiv 0$. This property is used to compute the derivative of a Bézier curve.

From the properties mentioned above, it is observed that the coefficients of a Bézier curve possess geometric flavor and give insight about the shape of the curve. The properties of Bézier curves are as follows:

- **Transformation invariance**: Bézier curves are invariant under the usual affine transformations such as rotations, translations and scalings, that is, one applies the transformations to the curve by applying it to the control polygon.

- **Convex hull property**: A Bézier curve is contained in the convex hull of its defining control points. This property ensures that the Bézier curve obtained from its control points is always bounded. A simple consequence of the convex hull property is that a planar control polygon always generates a planar curve.

- **Variation diminishing property**: No straight line intersects a given curve more times than it intersects the control polygon of the same curve (in three dimensions, replace 'straight line' with 'plane'). This property expresses the fact that a Bézier curve follows its control polygon rather closely and does not wiggle more than its control net.

Using the property of derivative of a basis function and definition of a Bézier curve, the general expression of derivative of a Bézier curve is given by

$$\mathbf{C}'(u) = \frac{d}{du}\left(\sum_{i=0}^{n} B_{i,n}(u)\mathbf{P}_i\right) = n\sum_{i=0}^{n-1} B_{i,n-1}(u)(\mathbf{P}_{i+1} - \mathbf{P}_i) \tag{2.3}$$

The above equation is used extensively to compute geometric features such as normals and curvatures at points on a Bézier curve. These points on the curve can be calculated by making use of the recursive definition of basis functions. Denoting a general $n^{th}$ degree Bézier curve by $\mathbf{C}_n(\mathbf{P}_0, \ldots, \mathbf{P}_n)$, the following relation holds true:

$$\mathbf{C}_n(\mathbf{P}_0, \ldots, \mathbf{P}_n) = (1-u)\mathbf{C}_{n-1}(\mathbf{P}_0, \ldots, \mathbf{P}_{n-1}) + u\mathbf{C}_{n-1}(\mathbf{P}_1, \ldots, \mathbf{P}_n) \tag{2.4}$$

Fixing $u = u_0$ and denoting $\mathbf{P}_i$ by $\mathbf{P}_{0,i}$, a recursive algorithm for computing the point $\mathbf{C}(u_0) = \mathbf{P}_{n,0}(u_0)$ on an $n^{th}$ degree Bézier curve is obtained as follows:

$$\begin{aligned} \mathbf{P}_{k,i}(u_0) &= (1-u_0)\mathbf{P}_{k-1,i}(u_0) + u_0\mathbf{P}_{k-1,i+1}(u_0) \\ & \qquad k = 1, \ldots, n \quad \text{and} \quad i = 0, \ldots, n-k \end{aligned} \tag{2.5}$$

This is the *de Casteljau Algorithm* for Bézier curves (Figure 2.2). It is a corner cutting process which yields the triangular table of points shown in Table 2.1.

The main attraction of the algorithm is the beautiful interplay between geometry and algebra. According to the de Casteljau algorithm, points on a Bézier curve can be obtained by performing a sequence of linear interpolations. The geometry underlying the algorithm enables us to infer some important properties of Bézier curves discussed before. Transformation invariance is a direct consequence of the algorithm because linear interpolation is affinely invariant and so is a finite sequence of them. Convex hull property also follows because every intermediate point, $\mathbf{P}_{k,i}$, is obtained as a convex barycentric combination of the previous set of points, $\{\mathbf{P}_{k-1,j}\}$, and at no step of the de Casteljau algorithm do we produce points outside the convex hull of the specified control points.

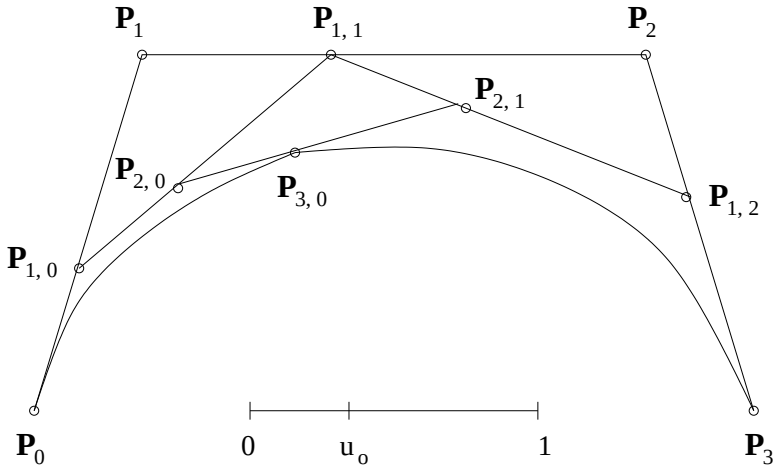



Figure 2.2: de Casteljau algorithm for a cubic Bézier curve showing intermediate points generated by repeated interpolation at parameter $u_o$.

**Rational Bézier Curves**

Although polynomials offer many advantages, there exist a number of important curve and surface types which cannot be represented precisely using polynomials such as circles, ellipses, hyperbolas, cylinders, cones, spheres etc. It is known from classical mathematics that all conic curves, including the circle, can be represented using *rational functions* which are defined as the ratio of two polynomials. In fact they are represented by rational functions of the form:

$$x(u) = \frac{X(u)}{W(u)} \qquad y(u) = \frac{Y(u)}{W(u)} \tag{2.6}$$

$X(u)$, $Y(u)$ and $W(u)$ are polynomials. For example, a circle of unit radius centered at the origin can be expressed as $x(u) = (1-u^2)/(1+u^2)$, $y(u) = (2u)/(1+u^2)$. On similar lines, we define an $n^{th}$ degree *rational Bézier curve* by

$$\mathbf{C}(u) = \frac{\sum_{i=0}^{n} B_{i,n}(u) w_i \mathbf{P}_i}{\sum_{i=0}^{n} B_{i,n}(u) w_i} \qquad 0 \le u \le 1 \tag{2.7}$$

$\mathbf{P}_i$ and $B_{i,n}(u)$ are the control points and Bernstein polynomials as before while $\{w_i\}$ is a set of scalars called *weights*. Thus, $W(u) = \sum_{i=0}^{n} B_{i,n}(u) w_i$ is the common

Table 2.1: Points generated by the de Casteljau algorithm.

| | | | | | | |
|---|---|---|---|---|---|---|
| $\mathbf{P}_0$ | | | | | | |
| | $\mathbf{P}_{1,0}$ | | | | | |
| $\mathbf{P}_1$ | | $\mathbf{P}_{2,0}$ | | | | |
| | $\mathbf{P}_{1,1}$ | | | | | |
| $\mathbf{P}_2$ | | $\vdots$ | | | | |
| | $\vdots$ | $\vdots$ | | $\mathbf{P}_{n-1,0}$ | | |
| $\vdots$ | $\vdots$ | $\vdots$ | $\cdots$ | | $\mathbf{P}_{n,0}$ | $= \mathbf{C}(u_0)$ |
| | $\vdots$ | $\vdots$ | | $\mathbf{P}_{n-1,1}$ | | |
| $\mathbf{P}_{n-2}$ | | $\vdots$ | | | | |
| | $\mathbf{P}_{1,n-2}$ | | | | | |
| $\mathbf{P}_{n-1}$ | | $\mathbf{P}_{2,n-2}$ | | | | |
| | $\mathbf{P}_{1,n-1}$ | | | | | |
| $\mathbf{P}_n$ | | | | | | |

denominator function. Normally $w_i > 0$ is assumed for all $i$ which ensures that $W(u) > 0$ for all $u \in [0, 1]$. A rational Bézier curve can be written as

$$\mathbf{C}(u) = \sum_{i=0}^{n} R_{i,n}(u)\mathbf{P}_i \tag{2.8}$$

In the above equation $\{R_{i,n}(u)\}$ are the rational basis functions for this curve form and can be expressed as

$$R_{i,n}(u) = \frac{B_{i,n}(u)w_i}{\sum_{j=0}^{n} B_{j,n}(u)w_j} \tag{2.9}$$

The $R_{i,n}(u)$ have properties very similar to those of the polynomial Bézier basis functions, $B_{i,n}(u)$. In fact, if $w_i = 1$ for all $i$ then $R_{i,n}(u) = B_{i,n}(u)$ for all $i$. Hence, $B_{i,n}(u)$ are a special case of $R_{i,n}(u)$ and polynomial Bézier curves are a special case of rational Bézier curves.

### Bézier Surfaces

A curve $\mathbf{C}(u)$ is a vector valued function of one parameter. It is a mapping of a straight line segment into Euclidean three-dimensional space. A surface is a vector

valued function of two parameters, $u$ and $v$, and represents a mapping of a region of the $uv$ plane into Euclidean three dimensional space. Thus a surface has a form $\mathbf{S}(u,v) = (x(u,v), y(u,v), z(u,v))$. In this section we shall discuss nonrational as well as rational Bézier surfaces.

**Polynomial Tensor Product Bézier Surfaces**

A tensor product scheme is a bidirectional scheme that uses basis functions and geometric coefficients. The basis functions are bivariate functions of $u$ and $v$ which are constructed as products of univariate basis functions. Polynomial Bézier surfaces are obtained by taking a bidirectional net of control points and products of univariate Bernstein polynomials. They can be expressed as

$$\mathbf{S}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_{i,n}(u)B_{j,m}(v)\mathbf{P}_{i,j} \qquad 0 \le u,v \le 1 \tag{2.10}$$

Properties of tensor product basis functions follow from corresponding properties of univariate basis functions. Properties of Bézier surfaces follow along the lines of the Bézier curves listed in Subsection 2. It is to be noted that unlike Bézier curves, there is no variation diminishing property for Bézier surfaces.

The de Casteljau algorithm can be easily extended to compute points on a Bézier surface (Figure 2.3 [13]). Let $(u_o, v_o)$ be fixed. Fixing $j_o$, $\mathbf{Q}_{j_o}(u_o) = \sum_{i=0}^{n} B_{i,n}(u_o)\mathbf{P}_{i,j_o}$ is the point obtained by applying the de Casteljau algorithm to the $j_o$ row of control points, that is, to $\{\mathbf{P}_{i,j_o}\}$, $i = 0,\ldots,n$. Therefore applying the de Casteljau algorithm $m+1$ times ($j_o = 0,\ldots,m$) yields $\mathbf{C}_{u_o}(v)$ and applying it once more to $\mathbf{C}_{u_o}(v)$ at $v = v_o$ yields $\mathbf{C}_{u_o}(v_o) = \mathbf{S}(u_o, v_o)$. This process requires $n(n+1)(m+1)/2+m(m+1)/2$ linear interpolations. By symmetry, $\mathbf{C}_{v_o}(u)$ can be computed first ($n+1$ applications

of de Casteljau) and then $\mathbf{C}_{v_o}(u_o) = \mathbf{S}(u_o, v_o)$. This, on the other hand, requires $m(m+1)(n+1)/2 + n(n+1)/2$ linear interpolations. Thus if $n > m$ compute $\mathbf{C}_{v_o}(u)$ first, then $\mathbf{C}_{v_o}(u_o)$. Otherwise, compute $\mathbf{C}_{u_o}(v)$ first, then $\mathbf{C}_{u_o}(v_o)$.

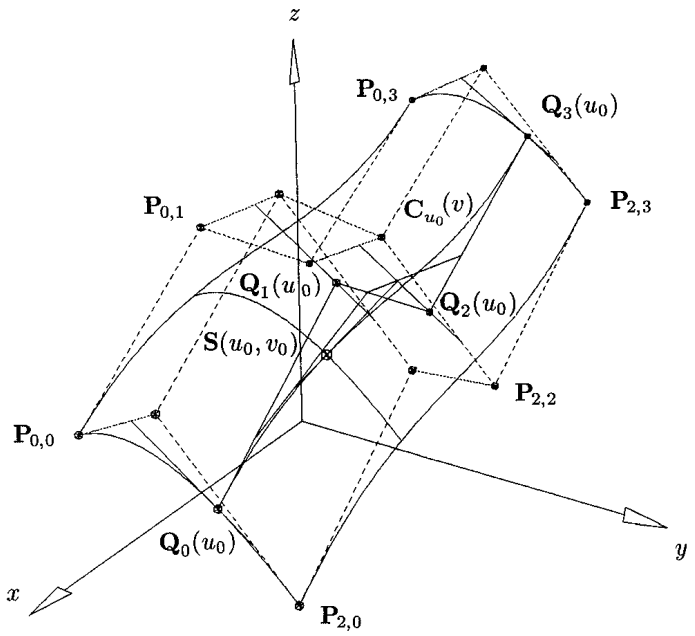



Figure 2.3: de Casteljau algorithm for a Bézier surface showing intermediate points generated by repeated linear interpolations at parameters $(u_o, v_o)$.

### Rational Bézier Surfaces

Rational surfaces, similar to rational curves, are useful in representing specific shapes such as spheres, ellipsoids etc. which cannot be represented as polynomial surfaces. A rational Bézier surface with degree $n$ in $u$ direction and $m$ in $v$ direction can be defined as

$$\mathbf{S}(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} B_{i,n}(u)B_{j,m}(v)w_{i,j}\mathbf{P}_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m} B_{i,n}(u)B_{j,m}(v)w_{i,j}} \qquad 0 \le u, v \le 1 \tag{2.11}$$

$\{w_{i,j}\}$ are the weights as before. The above given equation can be rewritten as shown below:

$$\mathbf{S}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} R_{i,j}(u,v)\mathbf{P}_{i,j} \tag{2.12}$$

$\{R_{i,j}(u,v)\}$ are the rational functions but they are not products of other basis functions. They can be expressed in terms of Bernstein coefficients as follows:

$$R_{i,j}(u,v) = \frac{B_{i,n}(u)B_{j,m}(v)w_{i,j}}{\sum_{r=0}^{n}\sum_{s=0}^{m} B_{r,n}(u)B_{s,m}(v)w_{r,s}} \tag{2.13}$$

Assuming $w_{i,j} > 0$ for all $i$ and $j$, the properties listed previously for nonrational Bézier surfaces (and product functions $B_{i,n}(u)B_{j,m}(v)$) extend naturally to rational Bézier surfaces. Furthermore, if $w_{i,j} = 1$ for all $i$ and $j$ then $R_{i,j}(u,v) = B_{i,n}(u)B_{j,m}(v)$ and the corresponding surface is nonrational.

## B-Spline Curves

Curves consisting of just one polynomial or rational segment are often inadequate. Some of their shortcomings are listed below:

- A high degree is required to satisfy a large number of constraints. For example, an $n-1$ degree polynomial is required that passes through $n$ given points. This task involves inversion of an $n \times n$ matrix and can be computationally expensive if $n$ is large. Hence, high degree curves are inefficient to process. They are also numerically unstable.

- A high degree is required to accurately fit some complex shapes.

- Single segment curves and surfaces can be shaped by means of their control points and weights but the control is not local. For example, movement of one control point of a Bézier curve alters the location of all the points on the curve changing the shape of the entire object being represented by the Bézier curve.

A possible solution is to use piecewise polynomial or rational curves and surfaces which join with some level of continuity. B-splines are attractive in this regard. Their piecewise polynomial or rational basis functions have nice analytic properties which follow along the lines of Bézier basis functions. This ensures that B-Splines have good geometric properties similar to their Bézier counterparts.

**Polynomial B-Spline Curves**

A Bézier curve is a one segment curve and therefore the degree of a Bézier curve is one less than the number of control points. However, since the B-spline curve is a piecewise curve, the degree of the B-spline segments has to be specified. A $p^{th}$ degree B-spline curve is defined by

$$\mathbf{C}(u) = \sum_{i=0}^{n} N_{i,p}\mathbf{P}_i \qquad a \leq u \leq b \tag{2.14}$$

Here $\{\mathbf{P}_i\}$ are the control points and $\{N_{i,p}(u)\}$ are the $p^{th}$ degree B-spline basis functions defined on a knot vector $U = \{\underbrace{a, \ldots, a}_{p+1}, u_{p+1}, \ldots, u_{m-p-1}, \underbrace{b, \ldots, b}_{p+1}\}$ as

$$N_{i,0} = \begin{cases} 1 & \text{if } u_i \leq u < u_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$N_{i,p} = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \tag{2.15}$$

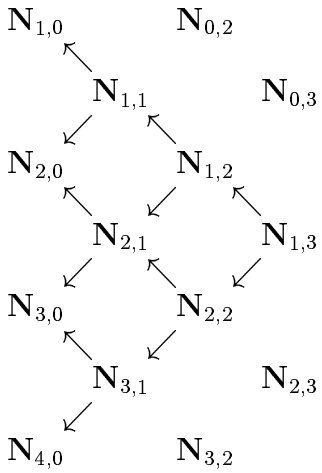
Hence, computation of a set of basis functions requires specification of a knot vector, $U$, and degree, $p$. The basis function equation can yield a quotient 0/0 which is defined to be zero. The half open interval, $[u_i, u_{i+1})$, is called the $i^{th}$ *knot span* and it can have zero length since knots need not be distinct.

Three steps are required to compute a point on a B-spline curve at a fixed given parameter value. They are enumerated as follows:

1. Find the knot span in which parameter $u$ lies.
2. Compute the nonzero basis functions.
3. Multiply the values of the nonzero basis functions with the corresponding control points.

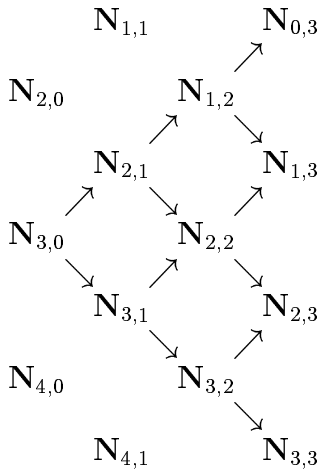
$\{N_{i,p}(u)\}$ are piecewise polynomials defined on the entire real line. Some properties of these basis functions are listed as follows:

- **Local support property**: $\{N_{i,p}(u) = 0\}$ if $u$ is outside the interval $[u_i, u_{i+p+1})$. This is illustrated by the following triangular scheme:



Notice that $N_{1,3}$ is a combination of $N_{1,0}$, $N_{2,0}$, $N_{3,0}$, and $N_{4,0}$. Thus $N_{1,3}$ is nonzero only for $u \in [u_1, u_5)$.

- In any given knot span, $[u_j, u_{j+1})$, atmost $p+1$ of the $N_{i,p}$ are nonzero, namely functions $N_{j-p,p}, \ldots, N_{j,p}$. This is shown as follows:




On $[u_3, u_4)$ the only nonzero zeroth degree function is $N_{3,0}$. Hence, the only cubic functions not zero on $[u_3, u_4)$ are $N_{0,3}, \ldots, N_{3,3}$.

- **Partition of unity**: For an arbitrary span, $[u_i, u_{i+1})$, $\sum_{j=i-p}^{i} N_{j,p}(u) = 1$ for all $u \in [u_i, u_{i+1})$.

- Except for the case $p = 0$, $N_{i,p}(u)$ attains exactly one maximum value.

- **Nonnegativity**: $N_{i,p} \geq 0$ for all $i$, $p$ and $u$.

- **Derivative**: The derivative of a basis function has a recursive definition where $N'_{i,p} = \frac{p}{u_{i+p} - u_i} N_{i,p-1}(u) - \frac{p}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u)$.

- A knot vector of the form $U = \{\underbrace{0, \ldots, 0}_{p+1}, \underbrace{1, \ldots, 1}_{p+1}\}$ yields Bernstein polynomials of degree $p$.

- If there are $m+1$ knots, there are $n+1$ basis functions where $m = n+p+1$. Also, $N_{0,p}(a) = 1$ and $N_{n,p}(b) = 1$.

Many properties of B-spline curves follow from those of their basis functions. Some of them are listed as follows:

- **Transformation invariance**: B-spline curves, like Bezier curves, are invariant under affine transformations such as rotations, translations and scalings.

- If $n = p$ and $U = \{\underbrace{0, \ldots, 0}_{p+1}, \underbrace{1, \ldots, 1}_{p+1}\}$ then $\mathbf{C}(u)$ is a Bézier curve.

- **Endpoint interpolation**: $\mathbf{C}(0) = \mathbf{P}_0$ and $\mathbf{C}(1) = \mathbf{P}_n$.

- **Convex hull property**: A B-spline curve is contained in the convex hull of its control polygon.

- **Local modification scheme**: Moving $\mathbf{P}_i$ changes $\mathbf{C}(u)$ only in the interval $[u_i, u_{i+p+1})$. This follows from the fact that $N_{i,p}(u) = 0$ for $u \notin [u_i, u_{i+p+1})$.

- Moving along the curve from $u = 0$ to $u = 1$, $\{N_{i,p}(u)\}$ functions act like switches. As $u$ moves past a knot, one $N_{i,p}(u)$ (and hence the corresponding $\mathbf{P}_i$) switches off and the next one switches on.

- **Variation diminishing property**: No plane has more intersections with the curve than with the control polygon (for two dimensional curves, replace the word plane with line).

Derivative of a B-spline curve $\mathbf{C}(u) = \sum_{i=0}^{n-1} N_{i,p}(u)\mathbf{P}_i$ defined on knot vector $U = \{\underbrace{0,\ldots,0}_{p+1}, u_{p+1},\ldots,u_{m-p-1}, \underbrace{1,\ldots,1}_{p+1}\}$ is given by

$$\mathbf{C}'(u) = \sum_{i=0}^{n-1} pN_{i,p-1}(u)\frac{\mathbf{P}_{i+1} - \mathbf{P}_i}{u_{i+p+1} - u_{i+1}} \tag{2.16}$$

The basis functions in the above equation have been computed on knot vector $U'$ obtained by dropping the first and last knots from the original knot vector $U$, that is, $U' = \{\underbrace{0,\ldots,0}_{p}, u_p,\ldots,u_{m-p-2}, \underbrace{1,\ldots,1}_{p}\}$.

**Nonuniform Rational B-Spline Curves**

NonUniform Rational B-Splines (NURBS) have become the industry standard for the representation, design and data exchange of geometric information processed by computers. The enormous success of NURBS can be attributed to the fact that they provide a unified mathematical basis for representing both analytic shapes such as conic sections as well as free-form entities such as car bodies and ship hulls and this feature makes NURBS an attractive tool for representing geometry in moving boundary problems. These elements are the general form of polynomial B-Splines and therefore polynomial Bézier elements.

NURBS curves are defined on a nonuniform knot vector. A $p^{th}$ degree NURBS curve is defined by

$$\mathbf{C}(u) = \frac{\sum_{i=0}^{n} N_{i,p}(u)w_i\mathbf{P}_i}{\sum_{i=0}^{n} N_{i,p}(u)w_i} \qquad a \le u \le b \tag{2.17}$$

$\{\mathbf{P}_i\}$ are the control points (forming a control polygon), $\{w_i\}$ are the weights and $\{N_{i,p}(u)\}$ the $p^{th}$ degree B-spline basis functions defined on a nonperiodic knot vector

$U = \{\underbrace{a, \ldots, a}_{p+1}, u_{p+1}, \ldots, u_{m-p-1}, \underbrace{b, \ldots, b}_{p+1}\}$. Unless otherwise stated, $a = 0$, $b = 1$ and $w_i > 0$ for all $i$. A NURBS curve can also be rewritten as

$$\mathbf{C}(u) = \sum_{i=0}^{n} R_{i,p}(u)\mathbf{P}_i \tag{2.18}$$

$\{R_{i,p}(u)\}$ are rational basis functions. They are piecewise rational functions on $u \in [0, 1]$ and can be expressed in terms of polynomial B-spline basis functions as

$$R_{i,p}(u) = \frac{N_{i,p}(u)w_i}{\sum_{j=0}^{n} N_{j,p}(u)w_j} \tag{2.19}$$

$\{R_{i,p}(u)\}$ have properties similar to those of $\{N_{i,p}(u)\}$ hence properties of NURBS are no different from those of polynomial B-spline curves.

## B-Spline Surfaces

Like polynomial Bézier surfaces, polynomial B-spline surfaces are tensor product patches of their basis functions. Rational B-Spline surfaces, on the other hand, are obtained by using piecewise rational functions.

### Polynomial B-Spline Surfaces

A polynomial B-spline surface, just like a polynomial Bézier surface, is a tensor product structured patch. It is obtained by taking a bidirectional net of control points, two knot vectors and products of the univariate B-spline basis functions as shown in the following equation:

$$\mathbf{S}(u, v) = \sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)\mathbf{P}_{i,j} \tag{2.20}$$

This surface is defined on knot vectors $U = \{\underbrace{0, \ldots, 0}_{p+1}, u_{p+1}, \ldots, u_{r-p-1}, \underbrace{1, \ldots, 1}_{p+1}\}$ and $V = \{\underbrace{0, \ldots, 0}_{q+1}, v_{q+1}, \ldots, v_{s-q-1}, \underbrace{1, \ldots, 1}_{q+1}\}$. $U$ has $r + 1$ knots and $V$ has $s + 1$. Hence $r = n + p + 1$ and $s = m + q + 1$.

Five steps are required to compute a point on a B-spline surface at fixed $(u, v)$ parameter values:

1. Find the knot span in which $u$ lies, say $u \in [u_i, u_{i+1})$.
2. Compute the nonzero basis functions $N_{i-p,p}(u), \ldots, N_{i,p}(u)$.
3. Find the knot span in which $v$ lies, say $v \in [v_j, v_{j+1})$.
4. Compute the nonzero basis functions $N_{j-q,q}(v), \ldots, N_{j,q}(v)$.
5. Multiply the values of the nonzero basis functions with the corresponding control points.

Properties of tensor product basis functions follow from corresponding properties of univariate basis funtions and properties of B-spline surfaces are similar to those of B-spline curves. The only exception to this is the variation diminishing property which is not known to exist for a B-spline surface.

**Nonuniform Rational B-Spline Surfaces**

A NURBS surface of degree $p$ in the $u$ direction and degree $q$ in the $v$ direction is a bivariate vector valued piecewise rational function of the form:

$$\mathbf{S}(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)w_{i,j}\mathbf{P}_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)w_{i,j}} \qquad 0 \le u, v \le 1 \tag{2.21}$$

$\{\mathbf{P}_{i,j}\}$ form a bidirectional control net, $\{w_{i,j}\}$ are weights, $\{N_{i,p}(u)\}$, $\{N_{j,q}(v)\}$ are nonrational basis functions on knot vectors $U = \{\underbrace{0, \ldots, 0}_{p+1}, u_{p+1}, \ldots, u_{r-p-1}, \underbrace{1, \ldots, 1}_{p+1}\}$

and $V = \{\underbrace{0,\ldots,0}_{q+1}, v_{q+1},\ldots,v_{s-q-1},\underbrace{1,\ldots,1}_{q+1}\}$ where $r = n+p+1$ and $s = m+q+1$. The surface can be rewritten as

$$\mathbf{S}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} R_{i,j}(u,v)\mathbf{P}_{i,j} \tag{2.22}$$

Here $\{R_{i,j}(u,v)\}$ are the rational basis functions. They are not a tensor product of the polynomial B-spline basis functions and are related to these functions as shown below:

$$R_{i,j}(u,v) = \frac{N_{i,p}(u)N_{j,q}(v)w_{i,j}}{\sum_{k=0}^{n}\sum_{l=0}^{m} N_{k,p}(u)N_{l,q}(v)w_{k,l}} \tag{2.23}$$

Assuming $w_{i,j} > 0$ for all $i$ and $j$, the properties listed previously for nonrational B-spline surfaces (and product functions $N_{i,p}(u)N_{j,q}(v)$) extend to rational B-spline surfaces. Furthermore, if $w_{i,j} = 1$ for all $i$ and $j$ then $R_{i,j}(u,v) = N_{i,p}(u)N_{j,q}(v)$ and the corresponding surface is nonrational.

## Bézier Triangles

Bézier triangles are a direct generalization of Bézier curves and the de Casteljau algorithm for triangular patches is an extension of the corresponding algorithm for curves. The control net now is of a triangular structure. The unstructured nature of the control net opens the possibility of using Bézier triangular patches along with an existing surface triangulation for representing an interface in a moving boundary problem. Tensor product surface patches cannot be used in this case because the surface information is stored in the form of an unstructured mesh.

### Polynomial Bézier Triangles

A polynomial Bézier triangle, as mentioned previously, is defined on a control net that has triangular structure. Figure 2.4 gives an example of a cubic patch (shaded

patch) with its control net. The numbering scheme used for control points is also shown.

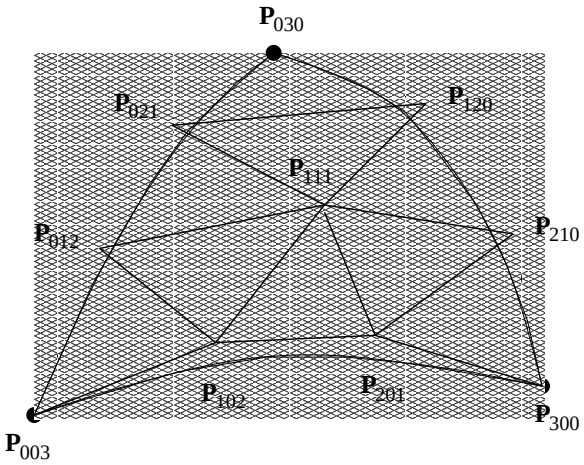



Figure 2.4: A cubic Bézier triangle with its control net.

With $n$ as the degree of the patch, the control net consists of $(n+1)(n+2)/2$ vertices. The numbers $(n+1)(n+2)/2$ are called *triangle numbers*. An $n^{th}$ degree Bézier triangle is defined as

$$\mathbf{S}(\mathbf{u}) = \sum_{|\mathbf{i}|=n} B_{\mathbf{i},n}(\mathbf{u})\mathbf{P}_{\mathbf{i}} \qquad 0 \leq u, v, 1-u-v \leq 1 \tag{2.24}$$

Here, the point $\mathbf{P}_{ijk}$ has been denoted by $\mathbf{P}_{\mathbf{i}}$. $|\mathbf{i}| = i+j+k$, therefore, $|\mathbf{i}| = n$ means $i+j+k = n$, always assuming $i, j, k \geq 0$. $\mathbf{u} = (u, v, 1-u-v)$ is a point in the parametric plane having barycentric coordinates (Appendix B). $\{B_{\mathbf{i},n}(\mathbf{u})\}$ are the bivariate Bernstein polynomials defined by

$$B_{\mathbf{i},n}(\mathbf{u}) = \frac{n!}{i!j!k!}u^i v^j (1-u-v)^k \qquad |\mathbf{i}| = n \tag{2.25}$$

$B_{\mathbf{i},n}(\mathbf{u}) = 0$ if $i, j, k < 0$ or $i, j, k > n$ for some subscripts. This follows standard convention for *trinomial coefficients*. Some properties of bivariate polynomials are stated as follows:

- **Endpoint interpolation**: A Bézier triangular patch passes through endpoints of the control net as $B_{00n,n} = B_{0n0,n} = B_{n00,n} = 1$ for all $0 \leq u, v, 1-u-v \leq 1$.

- **Nonnegativity**: $B_{\mathbf{i},n}(\mathbf{u}) \geq 0$ for all $\mathbf{i}$, $n$ and $0 \leq u, v, 1-u-v \leq 1$.

- **Partition of unity**: $\sum_{|\mathbf{i}|=n} B_{\mathbf{i},n}(\mathbf{u}) = 1$ for all $0 \leq u, v, 1-u-v \leq 1$.

- **Recursive definition**: Taking $\mathbf{e1} = (1,0,0)$, $\mathbf{e2} = (0,1,0)$ and $\mathbf{e3} = (0,0,1)$, $B_{\mathbf{i},n}(\mathbf{u}) = uB_{\mathbf{i}-\mathbf{e1},n-1}(\mathbf{u}) + vB_{\mathbf{i}-\mathbf{e2},n-1}(\mathbf{u}) + (1-u-v)B_{\mathbf{i}-\mathbf{e3},n-1}(\mathbf{u})$ relation holds for all Bernstein polynomials when $|\mathbf{i}| = n$.

Many properties of triangular Bézier triangles follow from those of the bivariate Bernstein polynomials. Some of them are listed below:

- **Transformation invariance**: Bézier triangles are invariant under usual affine transformations such as rotations, translations and scalings.

- **Boundary curves**: For a triangular patch, boundary curves are determined by control vertices at the boundaries (having at least one zero as a subscript). For example, a point on the boundary curve $\mathbf{P}_n(u, 0, 1-u)$ can be obtained by $\mathbf{P}_{r,\mathbf{i}}(u, 0, 1-u) = u\mathbf{P}_{r-1,\mathbf{i}+\mathbf{e1}} + (1-u)\mathbf{P}_{r-1,\mathbf{i}+\mathbf{e3}}$, which is the de Casteljau algorithm for Bézier curves.

- **Convex hull property**: This property is guaranteed since for $0 \leq u, v, 1-u-v \leq 1$, each $\mathbf{P}_{r,\mathbf{i}}$ is a convex combination of the previous $\mathbf{P}_{r-1,\mathbf{i}}$.

The de Casteljau algorithm for triangular patches is a direct generalization of the corresponding algorithm for curves (Figure 2.5).

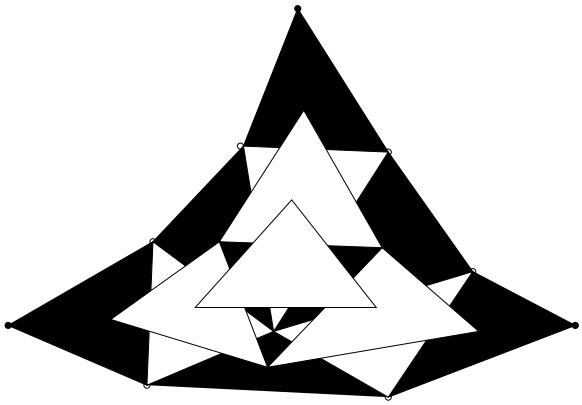


Figure 2.5: de Casteljau algorithm for a Bézier triangle showing the intermediate points generated.

Like the curve algorithm, the triangle algorithm uses repeated linear interpolation. Given a triangular array of points, $\{\mathbf{P_i}\}$, and $|\mathbf{i}| = n$, the recursive algorithm follows:

$$\mathbf{P}_{r,\mathbf{i}}(\mathbf{u}) = u\mathbf{P}_{r-1,\mathbf{i+e1}}(\mathbf{u}) + v\mathbf{P}_{r-1,\mathbf{i+e2}}(\mathbf{u}) + (1-u-v)\mathbf{P}_{r-1,\mathbf{i+e3}}(\mathbf{u})$$

$$r = 1, \ldots, n \quad \text{and} \quad |\mathbf{i}| = n - r \tag{2.26}$$

Here $\mathbf{P}_{0,\mathbf{i}} = \mathbf{P_i}$ and $\mathbf{P}_{n,\mathbf{0}}$ is the point with parameter value $\mathbf{u}$ on the triangular Bézier patch $\mathbf{P}_n(\mathbf{u}) = \mathbf{S}(\mathbf{u})$. As stated before, the triangular de Casteljau algorithm is completely analogous to the univariate one.

While discussing derivatives for tensor product patches, partials were considered because they are easily computed for those surfaces. The situation is different for triangular patches as the appropriate derivative here is the directional derivative. Let $\mathbf{u}_1$ and $\mathbf{u}_2$ be two points in the domain. Their difference $\mathbf{d} = \mathbf{u}_2 - \mathbf{u}_1$ defines a vector. The directional derivative of a surface at $\mathbf{x}(\mathbf{u})$ with respect to the direction vector, $\mathbf{d} = (d, e, f)$, is given by

$$D_{\mathbf{d}}\mathbf{x}(\mathbf{u}) = d\mathbf{x}_u(\mathbf{u}) + e\mathbf{x}_v(\mathbf{u}) + f\mathbf{x}_{1-u-v}(\mathbf{u}) \tag{2.27}$$

A geometric interpretation: in the domain, draw the straight line through $\mathbf{u}$ that is parallel to $\mathbf{d}$. This straight line is mapped to a curve on the patch. Its tangent vector at $\mathbf{x}(\mathbf{u})$ is the desired directional derivative. The partials of a Bézier patch are easy to compute and the directional derivative is given by

$$D_{\mathbf{d}}\mathbf{P}(\mathbf{u}) = n \sum_{|\mathbf{i}|=n-1} [d\mathbf{P}_{\mathbf{i}+\mathbf{e1}} + e\mathbf{P}_{\mathbf{i}+\mathbf{e2}} + f\mathbf{P}_{\mathbf{i}+\mathbf{e3}}] B_{\mathbf{i},n-1}(\mathbf{u}) \tag{2.28}$$

Thus a directional derivative is obtained by performing one step of the de Casteljau algorithm with respect to the direction vector $\mathbf{d}$ and $n-1$ more with respect to the position $\mathbf{u}$. In general, for an $r^{th}$ directional derivative, one has to perform $r$ steps of the de Casteljau algorithm with respect to $\mathbf{d}$ and the rest $n-r$ with respect to $\mathbf{u}$. Mixed derivatives can also be calculated this way by applying the de Casteljau algorithm along different directions. The order in which the operations are performed is irrelevant.

**Rational Bézier Triangles**

Following the familiar theme of generating rational curve and surface schemes, a rational Bézier triangle is defined as

$$\mathbf{S}(\mathbf{u}) = \frac{\sum_{|\mathbf{i}|=n} B_{\mathbf{i},n}(\mathbf{u}) w_{\mathbf{i}} \mathbf{P}_{\mathbf{i}}}{\sum_{|\mathbf{i}|=n} B_{\mathbf{i},n}(\mathbf{u}) w_{\mathbf{i}}} \tag{2.29}$$

Here, as usual, $\{w_{\mathbf{i}}\}$ are the weights associated with the control vertices $\mathbf{P}_{\mathbf{i}}$. For positive weights, the properties of polynomial Bézier triangles extend to rational Bézier triangles and if all weights are equal to one, the two forms are exactly the same.

This chapter presented methods for representing surfaces, in a parametric form, in three dimensions. Bézier and B-Spline tensor product surface patches result in

smooth representations of geometric shapes. However, these are structured patches and cannot be used for representing an unstructured triangulated surface available to us. Bézier triangular patches require a control net that has a triangular structure. This requirement poses a restriction as it is not be possible to break up an an arbitrary surface triangulation, where arbitrary number of triangles meet at every node, into triangular control nets. Therefore, the methods discussed so far are not directly applicable to us. We need an algorithm that uses the available surface triangulation to generate control nets for the Bézier triangles, such that these triangular patches join with some level of continuity.

# CHAPTER 3
# INTERFACE TRACKING METHOD

The challenges posed by moving boundary problems have resulted in a number of computational techniques directed towards their solution. Methods based on the combined Eulerian and Lagrangian approach require the interaction of the flow solver with the interface tracking components. The flow solver solves the governing flow equations over a stationary background mesh. The velocities computed over the cells of the background mesh are used to determine the velocity field of the moving surface. The tracker keeps track of the shape and position of the moving interface. It also computes geometric information such as normals and curvatures for the flow solver.

This chapter discusses a three-dimensional surface tracker, originally developed by Jayaraman et al. [14], for tracking interfaces in moving boundary problems. The moving interface is represented by a triangular-element unstructured mesh which offers flexibility in changing the mesh. Individual triangles, which do not satisfy the mesh quality criteria, can be bisected or rearranged to give new grid structures which better represent the interface. The purpose of the tracker is to refine and modify an existing mesh as it evolves in time in a moving boundary problem, hence, it does not include the unstructured grid generation methods used for triangulating a discrete set of points in space. The interface tracking algorithm is presented in Figure 3.1 and a description of its steps is given in the rest of the rest of the chapter.

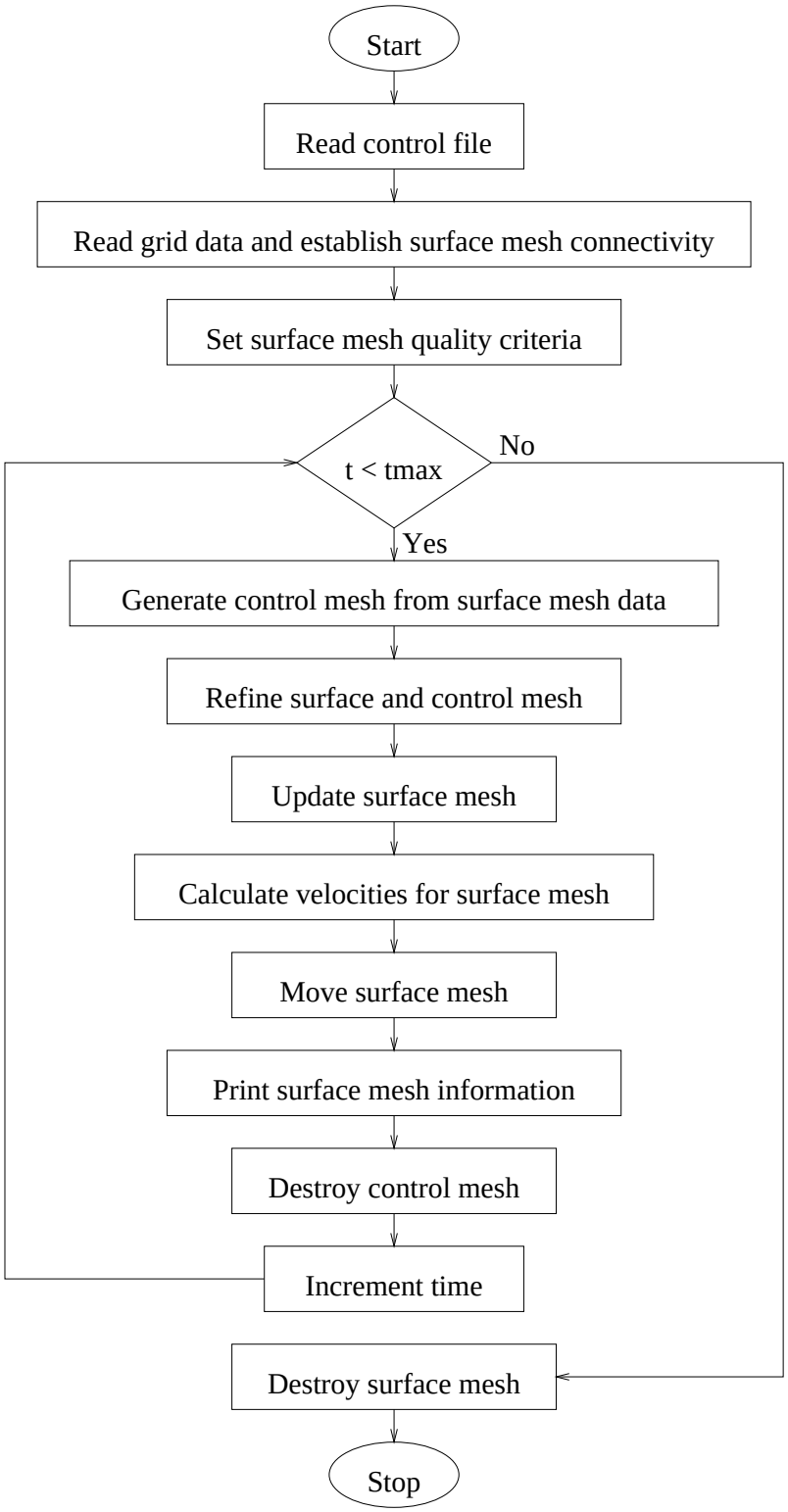



Figure 3.1: Interface tracking algorithm.

The interface tracking algorithm starts by reading the number of time steps and the time step size value from the control file.

In the next step, the initial unstructured surface triangulation data is applied to the tracker. This unstructured mesh, initializing the moving interface, is obtained from a standard CAD software. The number of nodes and their coordinates, number of triangles and the node numbers of every triangle are required to establish the surface mesh connectivity. With the information given above, we develop the node, edge, triangle and surface mesh data structures described in Appendix C.

Having provided an initial grid information, a mesh quality criterion is set. The tracker refines "big triangles" by breaking them into smaller triangles. A triangle is considered large enough to be bisected if its area exceeds the "average triangle area" times an "area factor" (which is set in this step). In order to get accurate results from the governing fluid flow equations, it is desirable to have triangles having a uniform size and a good aspect ratio. The length of edges of the surface mesh should typically be of the order of the grid spacing of the background field equation solver mesh. Thus the value of the average triangle area is computed on the basis of the size of the background Cartesian cell. For now, in order to test the interface tracker alone, the average triangle area is the mean of areas of all the triangles in the mesh.

The first step in the time loop is generation of a control mesh from surface mesh data. This is done using the $C^1$ Surface Spline algorithm [15] presented later in this chapter. The motive behind using a surface modeling scheme is to use the control mesh to generate a smooth surface which gives accurate normal and curvature values at every node.

Mesh refinement according to the size and shape of the triangles of the surface mesh follows next. A restructuring of the mesh does not generally involve movement of any of the vertices but does include many triangle, vertex and edge additions, reconnections and subtractions. As mentioned previously, the surface tracker bisects big triangles by breaking them up into smaller triangles. A triangle has a distorted shape if its circumcircle contains an opposite node of an adjacent triangle. This is called the circumcircle criterion and the edge common to the two triangles is a "bad edge" that has to be swapped. The details of these mesh refinement operations are included in this chapter.

Once mesh refinement is complete, the surface mesh is updated. This involves computations of normals and curvatures. The interface points move in the direction of their normals with a velocity which is obtained from the field equation solver. For the calculation of surface forces to be later distributed to the background structured mesh employed by the solver, information about the curvature at all the points is required. Details of normal computations and various methods used for computing curvature on the triangulated surface mesh as well as on the parametric surface obtained by the application of the $C^1$ Surface Spline algorithm are given in this chapter.

In order to evolve the interface in time, velocity is required at all the nodes of the mesh. For testing purposes, the tracker is currently working with user-input velocity functions.

With the velocity field available and normals known at all points, the interface is advected by a distance equal to velocity times the time increment value, obtained from the control file, in the normal direction.

The new interface position can be printed by storing the grid data in the output file. If required, normal and curvature values can also be plotted. This completes the time cycle. The control mesh, generated by the $C^1$ Surface Spline algorithm, is destroyed to be created again during the next cycle according to the new postions of the surface mesh points. The iteration counter is incremented before the control goes to the next cycle.

### $C^1$ Surface Spline Algorithm

The generation of control mesh is done using the $C^1$ Surface Spline algorithm proposed by Peters [15]. The method is applicable to irregular meshes with non-quadrilateral cells and with arbitrary number of cells meeting at each mesh node. Arbitrary freer-form surfaces are allowed to be modeled in the same conceptual frame work as tensor product B-Splines. In this report, however, the scheme is discussed for a triangulated surface.

The algorithm requires mesh points, mesh connectivity and blend ratios (knot spacings) as input. The output is a set of three-sided, cubic patches expressed in the Bernstein-Bézier form [16]. The mesh points serve as the control points of a smooth piecewise polynomial surface representation that is local, evaluates by averaging and obeys the convex hull property. The algorithm defines a spline space over irregular meshes by local averaging as do the other algorithms by Peters [17, 18]. The main improvement of the present approach over the earlier algorithms is in the number of triangular patches generated per mesh triangle. While earlier schemes generate 48 three-sided Bernstein-Bézier patches for each mesh triangle, the current algorithm reduces that number to 12. The $C^1$ smooth surface can be generated by applying

the triangular de Casteljau algorithm on each of these patches. The representation obtained has the following properties:

- **Free form modeling capability**: There are no restrictions on the number of triangles meeting at a mesh node.
- **Smoothness**: Surface splines form smooth surface parameterizations and in order to manipulate surface splines, it suffices to add, subtract or move mesh points locally.
- **Evaluation by averaging**: The coefficients in Bernstein-Bézier form can be obtained by performing averaging operations on the input mesh. The algorithm is local and can be interpreted as a rule for cutting an input mesh such that the control mesh obtained gives a spline surface.
- **Convex hull property**: The surface lies locally and globally in the convex hull of the input mesh.
- **Intuitive shape parameters**: Averaging process is geometrically intuitive. Blend ratios, analogous to knot distances, govern the depth of cuts into the input mesh to generate the control mesh. Smaller cuts result in a surface that follows the input mesh more closely and changes the normal direction more rapidly across the boundary.
- **Low degree parametrization**: The surface is parametrized by low degree polynomial patches and the representation can be extended to rational patches.

A six step algorithm for generating a surface spline control mesh is given below. The first three steps generate a refined mesh consisting of quadrilateral cells such that each original node of the unstructured surface mesh is surrounded by vertices of degree four. The last three steps generate the surface parametrization. In the following steps, all the coefficients $A$, $C$, $P$, $Q$, $V$ are points in space. All indices are interpreted modulo $n$ where $n$ is the number of quadrilateral cells, edges or Bernstein-Bézier patches meeting at a node.

**Step 1: Mesh Refinement.** Referring to Figure 3.2, for each edge with *node vertices* $V_i$, $V_j$, add an *edge vertex* $V_{ij} = (V_i + V_j)/2$ and for each triangle having node vertices $V_i$, $V_j$, $V_k$, add a *triangle vertex* $V_{ijk} = (V_i + V_j + V_k)/3$. Connect triangle vertex to all edge vertices of a triangle. This leads to the creation of a refined mesh of quadrilateral cells. Each triangle now contains three cells. All the edge vertices have a degree four whereas triangle vertices have degree three.

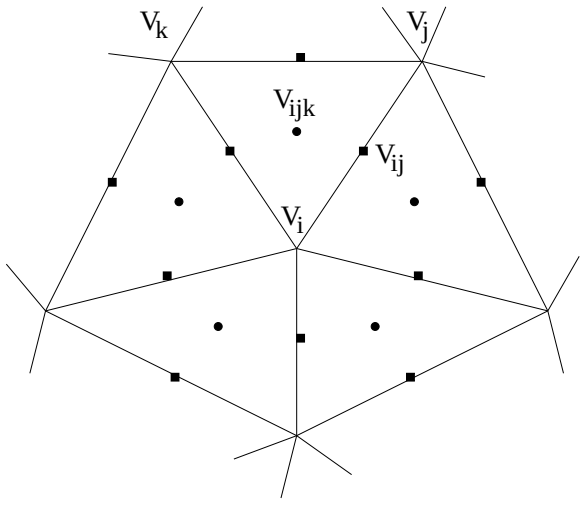



Figure 3.2: Mesh refinement step where the addition of edge vertices and triangle vertices leads to creation of refined mesh of quadrilateral cells.

**Step 2: Edge Cutting.** For every cell $c_i$, starting with the node vertex that corresponds to a point on the surface mesh and taking vertices $V_1$, $V_2$, $V_3$, $V_4$ (Figure 3.3), define a *preliminary cell center* $C_i'$ subject to blend ratios $0 < a_{i1}, a_{i2} < 1$ as follows:

$$C_i' = (1 - a_{i1})(1 - a_{i2})V_1 + (1 - a_{i1})a_{i2}V_2 + a_{i1}a_{i2}V_3 + a_{i1}(1 - a_{i2})V_4 \tag{3.1}$$

Next derive *cell centers* $C_i$ from $C_i'$ such that all $(C_i + C_{i+1})/2$ surrounding a vertex lie in the same plane. For each vertex $V$ surrounded by less than five cells, $C_i = C_i'$. When the number of cells, in anticlockwise order, is greater than four $(c_1, \ldots, c_n)$, the cell center corresponding to $c_i$ is given by

$$C_i = \bar{V} + 2w_n/n \sum_{j=1}^{n} \cos(2\pi j/n) C_{i+j}' \tag{3.2}$$

Here $\bar{V} = 1/n \sum_{i=1}^{n} C_i'$ and $0 < w_n < 1$. Also, $w_n = 1/(1 + \cos(2\pi/n))$ if $n$ is even and $w_n = 1/(2\cos(\pi/n))$ if $n$ is odd.

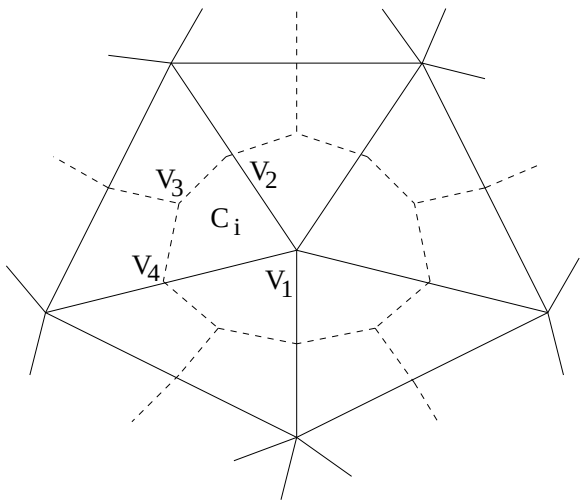



Figure 3.3: Edge cutting step where edges of the surface mesh are cut according to the blend ratios.

**Step 3: Quadratic Meshing.** For each vertex $V$ surrounded in order by cell centers $C_1, \ldots, C_n$, compute the final vertex location $\bar{V} = 1/n \sum_{i=1}^{n} C_i$. Note that $\sum_{i=1}^{n} C_i = \sum_{i=1}^{n} C_i'$. For each edge separating two cells with centers $C_{i-1}$, $C_i$, create an *edge coefficient* $A_i = (C_{i-1} + C_i)/2$. This step leads to a refined mesh such that three points are associated with each cell edge (Figure 3.4). These can be interpreted as the Bernstein-Bézier coefficients of a quadratic curve segment. Note that each vertex $\bar{V}$ lies in the same plane as the surrounding $A_i$'s.

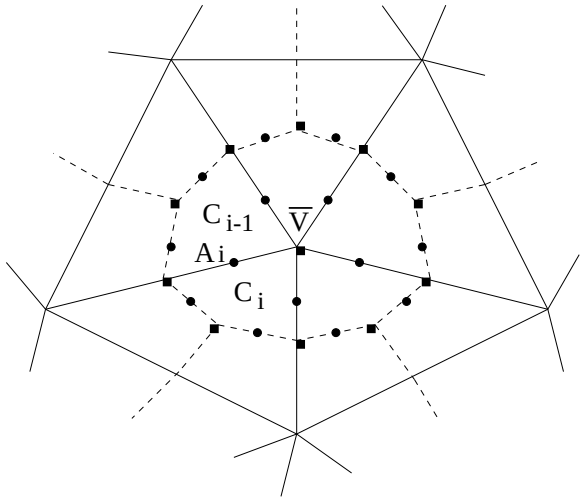



Figure 3.4: Quadratic meshing step which yields Bernstein-Bézier coefficients of a quadratic curve segment.

**Step 4: Quadratic Patching.** Each quadrilateral cell is made of four second-degree Bernstein-Bézier triangular patches as shown in Figure 3.5. For each vertex $\bar{V}$ set $Q_{020,i} = \bar{V}$, $Q_{110,i} = A_i$ where $i = 1, \ldots, n$ and compute the interior coefficients by averaging these values. Set coefficients $Q_{011,i} = Q_{101,i+1} = (Q_{110,i} + Q_{110,i+1})/2$ and $Q_{002} = (Q_{101,i} + Q_{011,i+1})/2$.

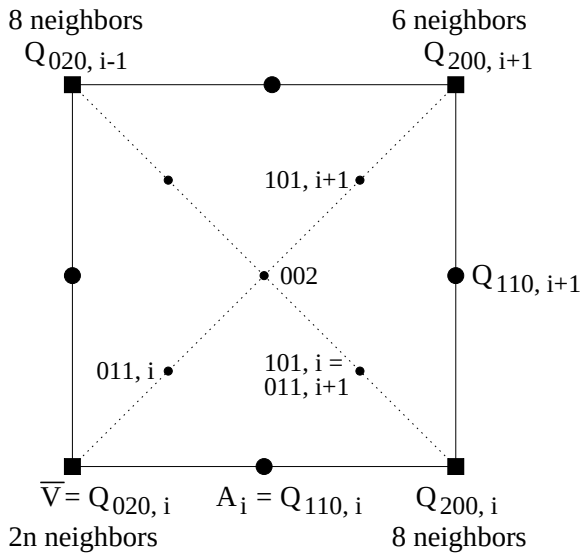



Figure 3.5: Quadratic patching step where internal coefficients are computed to give quadratic Bézier triangle patches.

**Step 5: Degree Raising.** The quadratic patches are reinterpreted as cubic patches using the following formulas:

$$
\begin{array}{ccccc}
P_{030} & & & & Q_{020} \\
& P_{021} & & \frac{Q_{020}+2Q_{011}}{3} & \\
P_{120} & & P_{012} & & \frac{Q_{002}+2Q_{011}}{3} \\
& P_{111} & \quad P_{003} = & \frac{Q_{110}+Q_{101}+Q_{011}}{3} & \quad Q_{002} \\
P_{210} & & P_{102} & & \frac{Q_{002}+2Q_{101}}{3} \\
& P_{201} & & \frac{Q_{200}+2Q_{101}}{3} & \\
P_{300} & & & Q_{200} &
\end{array}
\qquad \frac{Q_{020}+2Q_{110}}{3}
$$

The coefficients $P_{111}$, $P_{102}$, $P_{012}$ and $P_{003}$ are recomputed in the Step 6.

**Step 6: Twist Adjustment.** At vertex $\bar{V} = P_{030} = Q_{020}$ surrounded by $n$ quadrilateral cells, set $c = \cos(2\pi/n)$. Using $C_i$, the center of $i^{th}$ cell, compute coefficient $P_{111,i} = [(2-c)Q_{110,i} + (1+c)Q_{200,i} + 2C_i + Q_{020,i}]/6$. Finally, counting with $j$ around the four-neighbor vertex $Q_{002}$, $P_{102,j} = P_{012,j+1} = (P_{111,j} + P_{111,j+1})/2$, $P_{003,j} = (P_{102,0} + P_{102,2})/2$. Steps 5 and 6 are not necessary if $n = 4$. This step is required for getting $C^1$ continuous Bézier triangular patches.

The surface obtained by application of the surface spline algorithm is parametrized by Bernstein-Bézier patches that can be, depending on users choice, either rational or polynomial. There are 12 Bernstein-Bézier triangular patches generated for evry triangle of the unstructured surface mesh. Three sided patches ensure tangent plane continuity of the surface.

### Mesh Refinement

As the interface evolves in time, the positions of the mesh nodes change which might lead to large or skewed triangles having a large aspect ratio. For accuracy of the numerical computations, the spacing of the interface points should be of the order of the background Cartesian mesh spacing. Hence, it becomes important to refine the mesh periodically and retain an accurate representation of the interface. This is done according to the mesh refinement algorithm shown in Figure 3.6.

The algorithm starts by first creating a list of "big triangles", that is, a list of triangles whose area is greater than a specified value. As long as the list is not empty, the algorithm breaks up the big triangles into smaller parts by adding new points and it works on the biggest triangle first. By experience, this results in a better quality mesh that gives accurate curvature results as compared to results obtained by the refinement of any arbitrary "big triangle". Once all the "big triangles" are broken into smaller parts resulting in a surface made up of triangles having a uniform size, the tracker works on the skewed triangles. Diagonal swap method, which is essentially a mesh reconnection method, is employed for this purpose. The details of the point insertion and diagonal swapping procedures are discussed in detail in this section.

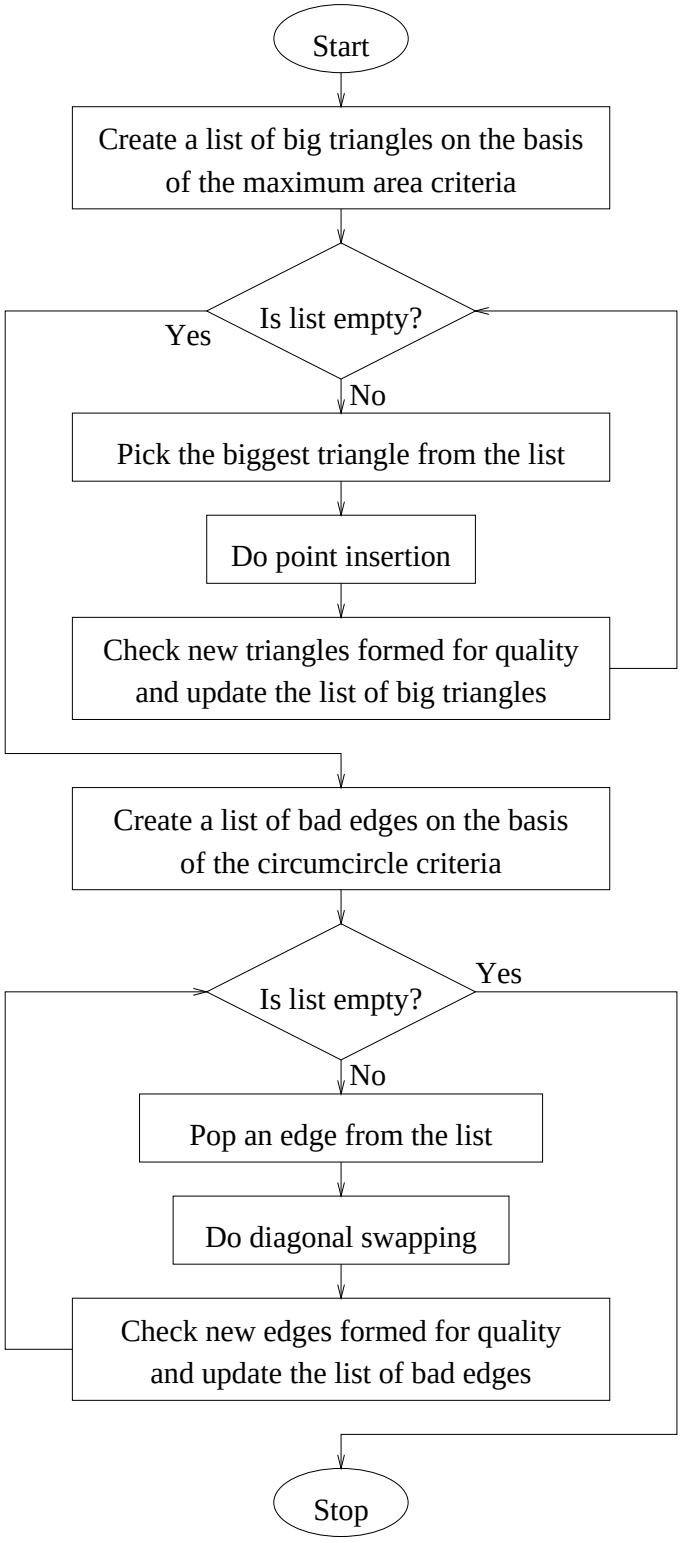



Figure 3.6: Mesh refinement algorithm.

New points are inserted into "big triangles", whose area exceeds a limit specified previously (based on the average triangle size and an area factor), according to the Bowyer-Watson point replenishment algorithm [19, 20] which was concieved for two-dimensional Delaunay triangulation [21]. Once all the big triangles are bisected into smaller ones, the refinement algorithm works on the distorted triangular elements that do not satisfy the circumcircle criterion. "Bad edges" are identified and diagonal swap operation is performed on the quadrilateral formed by two triangles sharing a bad edge. The details of point insertion and diagonal swap technique are presented below.

**Point Insertion**

Insertion of new point is done to break a big triangle into smaller triangles. The algorithm followed is shown in Figure 3.7.

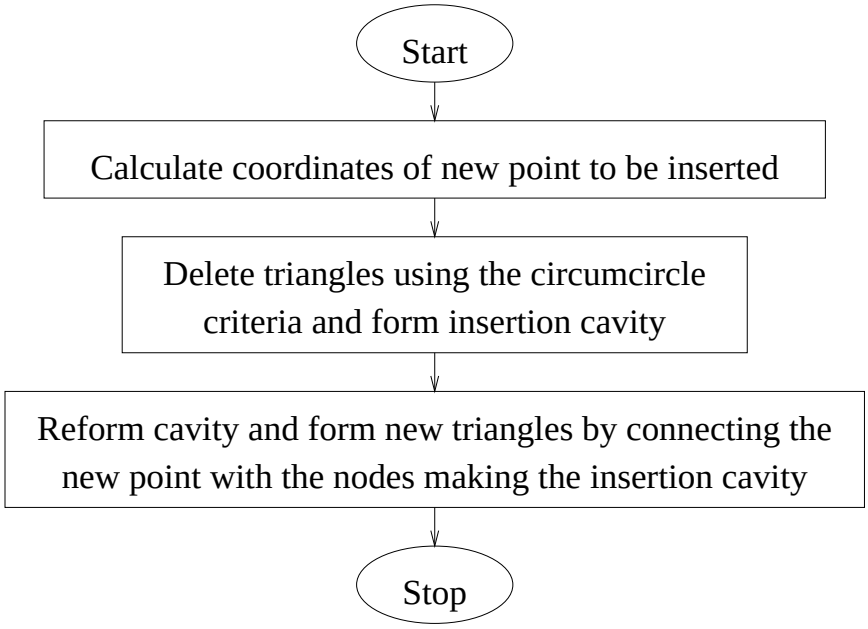



Figure 3.7: Point insertion algorithm.

To begin with, the original grid data of the interface includes coordinates of the nodes and node numbers of every triangle in the mesh. The connectivity information provided is of topological nature and the exact geometry of the mesh (how nodes in the mesh are connected to their respective neighbors) is not available to the tracker. Connection of nodes by straight lines is actually an assumption of the "real" surface. In such circumstances, it is not possible to find a unique location of the new point to be inserted. The tracker computes normals and curvatures at all nodes of the mesh for the field equation solver and the results of these calculations, especially curvature calculations, are very sensitive to the coordinates of the new point. Placement of the new point in the plane of the big triangle leads to zero curvature at that location whereas slight displacement of the point from the plane causes rapid variation of curvature. Keeping this in mind, the new point is inserted based on the local curvature information obtained from its neighborhood. The new point is made to lie on a sphere which passes through the nodes of the big triangle and whose curvature is equal to the average of the curvature value at the same three nodes.

Let $k_1$, $k_2$ and $k_3$ be the curvatures at the node locations of the triangle under consideration. Then $k = (k_1 + k_2 + k_3)/3$ is the curvature at the new point location. If $R$ is the radius of the sphere on which the new point is located then $R = 1/\|k\|$. Let $r$ be the circumradius of the triangle being refined. Since the sphere passes through the nodes of this triangle, the normal to the plane containing the triangle is in the same direction as the line that passes through center of the sphere and circumcenter of the triangle. The new point has to be placed so that it lies on the sphere, hence, it should be projected by a distance $d = R - \sqrt{R^2 - r^2}$ along the normal from the circumcenter.

The point can be projected above or below the triangle because theoretically there are two spheres of radius $R$ passing through three given points. The appropriate choice is made on the basis of sign of local surface curvature. The advantage of using a circumcenter is that, after projection, the new point is equidistant from all the nodes of the triangle. With this method of calculating the coordinates of the new point, three cases arise which are enumerated as follows:

1. $k = 0$ and as a result the radius, $R$, tends to infinity. This implies that the local surface around the new point is planar. Hence the new point is inserted at the circumcenter of the triangle.

2. $R > r$ and in this case the point placement is done by projecting a point along the normal from the circumcenter by a distance $d$.

3. $R < r$ which means that the surface triangle lies in a region of very high curvature (since $R = 1/\|k\|$). In such a situation, the new point is placed at the circumcenter of the triangle.

Once the coordinates of the new point are available, the subsequent step in the point insertion algorithm is the deletion of triangles according to the Bowyer-Watson criterion and formation of an "insertion cavity". The details of these operations are given below.

The Bowyer-Watson algorithm is based on the circumcircle property which ensures that no point of a Delaunay triangulation can lie within the circle circumscribed to any triangle. It is essentially a reconnection method since it computes how an

existing triangulation is to be modified because of insertion of a new point. The Bowyer-Watson algorithm is a right choice for the current application as it is required to maintain an accurate representation of a deforming interface by adding points where necessary. Even though the algorithm is for two dimensional triangulations, its modifcation is being used for three-dimensional surfaces.

The circumcircle criterion is illustrated in Figure 3.8. $\triangle ABC$ is a big triangle that needs to be refined because of its large size. Initially, its edges $AB$, $BC$ and $CA$ are "alive" because each of them is associated with two triangles of the mesh. Since $\triangle ABC$ is to be bisected, it is marked for deletion and all its edges are declared "dormant" because now each of them is a part of only one triangle of the surface mesh. Point $F$ is the new point inserted to break this triangle into smaller triangles. Distance between the circumcenter of $\triangle ACE$ (point $H$) and $F$ is less than the circumradius of $\triangle ACE$ hence that triangle is marked for deletion. Edges $CE$ and $EA$ change state from alive to dormant whereas edge $AC$, which was dormant previously, now becomes "dead". $\triangle ADB$ satisfies the circumcircle criterion discussed above and is not marked for deletion.

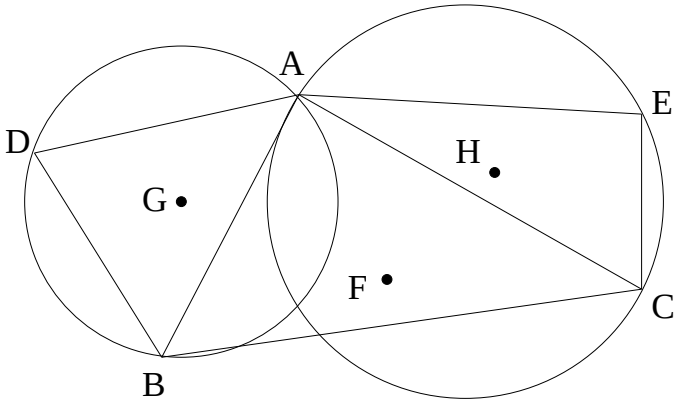



Figure 3.8: Bowyer-Watson criterion for marking triangles to be deleted in order to form an insertion cavity.

Each time a new point is inserted into the mesh, a search is carried out starting from the "insertion triangle" which is marked for deletion. The search then travels across the three edges of this triangle to its neighbors. If the neighboring triangles fail to satisfy the circumcircle criterion discussed above, they are marked for deletion. Any edges common to two deleted triangles are also identified (declared dead) to be removed later. The recursive process continues up to a specified number of levels which, by experience, is chosen such that it covers all the triangles whose circumradius is less than the distance between its circumcenter and the new point.

Once triangles, edges and nodes to be deleted are identified, an "insertion cavity" is constructed. The process is depicted in Figure 3.9 where $\triangle ABC$, $\triangle ADB$, $\triangle CBE$ and edges $AB$, $BC$ are deleted according to the Bowyer-Watson criterion. Edges $AD$, $DB$, $BE$, $EC$ and $CA$ are in a dormant state because each of them is a part of only one triangle of the surface mesh. These edges form the insertion cavity. $F$ is the new point to be inserted and new triangles are formed by connecting this new point with points $A$, $D$, $B$, $E$, $C$ that form the insertion cavity. This completes the point insertion algorithm which results in triangles having relatively uniform sizes.

**Diagonal Swapping**

This mesh restructuring technique is used after point insertion is complete and none of the triangles have area larger than the set criterion. Diagonal swapping is basically attempted to mend distorted triangles which do not satisfy the circumcircle criterion. The procedure can be clearly understood with the help of Figure 3.10. The distance between point $C$ and the circumcenter of $\triangle ABD$ is less that the circumradius of $\triangle ABD$. Hence, edge $BD$, common to the two triangles under consideration,

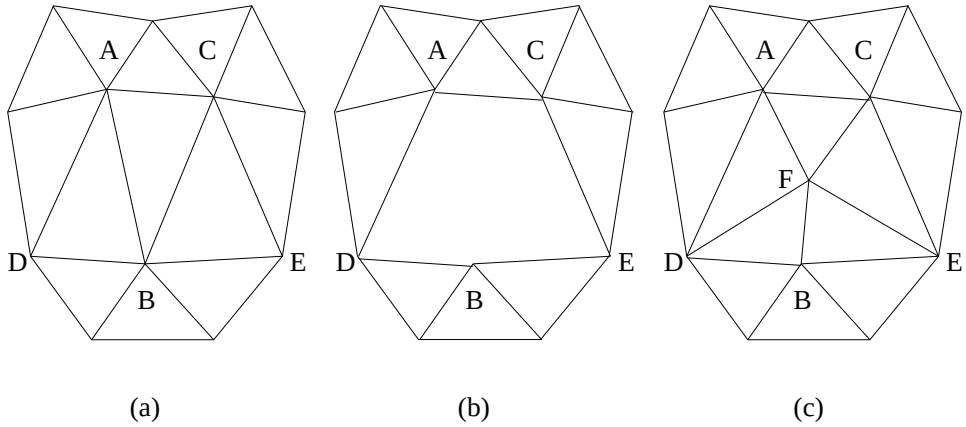



Figure 3.9: Bowyer-Watson algorithm: (a) Existing triangulation; (b) Formation of insertion cavity; (c) Reconnection of an existing grid around a newly inserted point.

becomes a "bad edge" that is to be swapped. A diagonal swap connects $A$ to $C$ and the newly formed triangles, $\triangle ABC$ and $\triangle ACD$, have a better aspect ratio than $\triangle ABD$ and $\triangle BCD$ in the original triangulation.

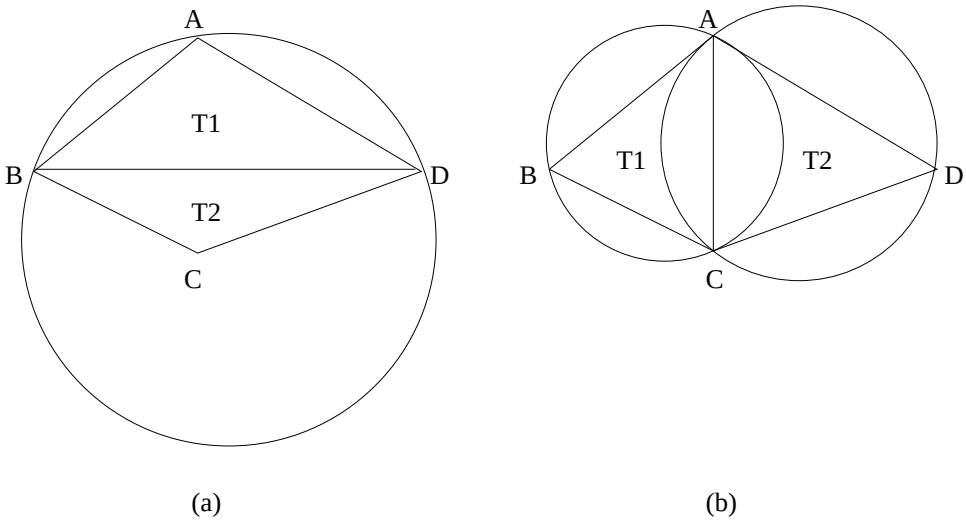



Figure 3.10: Diagonal swapping operation: (a) Triangles before diagonal swapping; (b) Triangles after diagonal swapping.

Every time a point insertion or diagonal swap operation is performed on the surface triangulation, corresponding changes are made to the control mesh. Deletion of a triangle causes deletion of three cells in the control mesh and creation of new triangles

leads to formation of new cells by local application of the surface spline algorithm discussed in the previous section. The control mesh is thus coupled dynamically with the surface triangulation.

## Mesh Update

After an initial interface position and shape is defined, the interface moves on the basis of the velocity field. The normal at all the nodes is required for the movement of the interface. Curvature plays an important role in the estimation of interface characteristics as well as interfacial fluid dynamics. At the end of the mesh refinement step, we have a surface approximated by an unstructured mesh. We also have a $C^1$ smooth surface composed of parametric Bernstein-Bezier patches obtained by the application of the triangular de Casteljau algorithm on the control mesh. The control mesh here is the outcome of the $C^1$ Surface Spline algorithm applied on the unstructured surface mesh. Methods of computing normals and curvatures on these surface representations are quite different because one representation is implicit while the other is parametric. The details of the methods used are given below.

### Mesh Update on $C^1$ Spline Surface

The application of $C^1$ Spline Algorithm results in a surface expressed in the Bernstein-Bézier form. Each parametric patch of the surface is of the form:

$$\mathbf{x} = \mathbf{x}(u, v) = (x(u, v), y(u, v), z(u, v)) \tag{3.3}$$

Here cartesian coordinates $x$, $y$, $z$ of a surface are differentiable functions of the parameter $u$ and $v$. For a Bézier triangle the parametric domain is a triangle and $(u, v, 1 - u - v)$ are the barycentric coordinates of a point in the reference triangle.

**Numerical calculation of normal**. The partials $\mathbf{x}_u$ and $\mathbf{x}_v$ at a point $\mathbf{x}$ span the tangent plane to the surface at $\mathbf{x}$. The normal of the tangent plane, $\mathbf{n}$, coincides with the normal to the surface at $\mathbf{x}$ and is given by

$$\mathbf{n} = \frac{\mathbf{x}_u \times \mathbf{x}_v}{\|\mathbf{x}_u \times \mathbf{x}_v\|} \tag{3.4}$$

The normal is of unit length and is perpendicular to $\mathbf{x}_u$ and $\mathbf{x}_v$. The normal together with unnormalized vectors $\mathbf{x}_u$ and $\mathbf{x}_v$ form a local coordinate system. If $n$ triangles abut at a particular node in the surface triangulation, $2n$ cubic Bézier patches meet at the corresponding node in the control mesh. A normal at a point $P$ is computed by taking the normalized mean of normals obtained from all the patches meeting at $P$.

**Numerical calculation of curvature**. Let $\mathbf{n}$ denote a unit vector normal to the surface $S$ at a point $P$ and consider the intersections, with $S$, of planes containing $\mathbf{n}$. Each of these planes meets the surface in a curve of intersection that is referred to as a normal section at $P$. A direction can be associated to each normal section, the direction of the unit tangent vector, $\mathbf{t}$, to the curve at $P$. Every normal section also has a normal surface curvature, the curvature, $\kappa_{\mathbf{t}}$, of the curve at $P$. These normal surface curvatures vary as the normal cutting plane rotates around $\mathbf{n}$. The maximum and minimum values of $\kappa_{\mathbf{t}}$, which occur in orthogonal directions, are referred to as the *principal curvatures* at $P$. If $\kappa_1$ and $\kappa_2$ denote the principal curvatures, the *mean curvature*, $H$, is given by

$$H = \frac{\kappa_1 + \kappa_2}{2} = \frac{NE - 2MF + LG}{2(EG - F^2)} \tag{3.5}$$

Here $E = \mathbf{x}_u\mathbf{x}_u$, $F = \mathbf{x}_u\mathbf{x}_v$, $G = \mathbf{x}_v\mathbf{x}_v$, $L = \mathbf{n}\mathbf{x}_{uu}$, $M = \mathbf{n}\mathbf{x}_{uv}$, $N = \mathbf{n}\mathbf{x}_{vv}$. Derivatives $\mathbf{x}_u$, $\mathbf{x}_v$, $\mathbf{x}_{uu}$, $\mathbf{x}_{uv}$, $\mathbf{x}_{vv}$ can be obtained by application of the de Casteljau algorithm for Bézier triangles with respect to the directions of derivative and the position $\mathbf{u} = (u, v, 1-u-v)$. Both $\kappa_1$ and $\kappa_2$ change sign if the normal, $\mathbf{n}$, is reversed and $H$ is affected by such a reversal.

**Mesh Update on Surface Triangulation**

Triangulated mesh yields data which is of implicit nature. The coordinates of the nodes of the mesh along with the connectivity information are available to us. The methods for computing normals and curvatures on such a surface are different from those applicable for a smooth parametric patch. This section is devoted to the ways of computing normals and curvatures on triangulated surfaces.

**Numerical calculation of normal**. Normal at a point $P$ is basically a vector which is perpendicular to the tangent plane at $P$. On a triangulated surface, planes containing triangles meeting at $P$ can be considered as tangent planes at $P$. Due to the absence of a unique tangent plane, there is no unique way of computing a surface normal in this case. The method used is based on geometrical averaging of normals obtained from triangles meeting at every node of the mesh. Consider a triangle having nodes $\mathbf{x}_{-1}$, $\mathbf{x}_0$ and $\mathbf{x}_{+1}$. Define vectors $\mathbf{v}_1 = \mathbf{x}_{-1} - \mathbf{x}_0$ and $\mathbf{v}_2 = \mathbf{x}_{+1} - \mathbf{x}_0$. The unit normal to the triangle is now simply

$$\mathbf{n} = \frac{\mathbf{v}_1 \times \mathbf{v}_2}{\|\mathbf{v}_1 \times \mathbf{v}_2\|} \tag{3.6}$$

A normal at a point $P$ is computed by normalizing the mean of normals obtained from all the triangles meeting at $P$.

**Surface curvature using the Dupin indicatrix**. For an arbitrary interface, a method for calculating curvatures for a given distribution of discrete grid points on the interface has been proposed by Todd and McLeod [22]. This method uses the Dupin indicatrix from given surface point data and is significantly more accurate than those using local quadric interpolants for the structured grid cases considered by Todd and McLeod. The approach along with some modifications made for computing surface curvatures is discussed here.

Let $\kappa_1$ and $\kappa_2$ denote the principal curvatures and $\mathbf{t}_1$, $\mathbf{t}_2$ denote the corresponding directions. Further, let $\Psi$ denote the angle between the direction $\mathbf{t}$ of an arbitrary normal section and the direction $\mathbf{t}_1$. Euler showed in 1760 [23, 24] that the normal curvature $\kappa_{\mathbf{t}}$ in an arbitrary direction $\mathbf{t}$ is related to the principal curvatures $\kappa_1$ and $\kappa_2$ by

$$\kappa_{\mathbf{t}} = \kappa_1 \cos^2 \Psi + \kappa_2 \sin^2 \Psi \tag{3.7}$$

If $x = \pm \cos \Psi / (|\kappa_{\mathbf{t}}|)^{1/2}$ and $y = \pm \sin \Psi / (|\kappa_{\mathbf{t}}|)^{1/2}$ then the Dupin indicatrix of the surface $S$ at the point $P$ is given by

$$\kappa_1 x^2 + \kappa_2 y^2 = \pm 1 \tag{3.8}$$

If the principal curvatures are of same sign, the normal curvature in any direction is also of that sign and the surface, locally, lies entirely on one side of the tangent plane at $P$. In this case, Equation (3.8) provides a nontrivial curve which is an ellipse. When the principal curvatures are of differing sign, that is, when $P$ is hyperbolic or

saddle point, Dupin indicatrix is a pair of hyperbolas. If axes other than those in the direction of the principal curvatures are used, Dupin indicatrix takes the form:

$$Ax^2 + 2Cxy + By^2 = \pm 1 \tag{3.9}$$

Hence if Dupin indicatrix at a point is known, so are principal curvatures and their directions. The mean curvature can be computed by taking the average of the principal curvature values. In turn, knowing estimates of normal curvatures in particular directions, some approximate fit to that data by a central conic of the form of Equation (3.9) can be made and from there, via a rotation, the Dupin indicatrix can be estimated. From the coordinates of points on the surface, however, the normal curvatures cannot be estimated directly. To circumvent this difficulty, a result due to Meusnier may be used. Let $\kappa$ be the curvature, at $P$, of a curve obtained by the intersection of an arbitrary plane with the surface $S$ and let $\kappa_{\mathbf{t}}$ be the normal curvature in the same direction. If $\Theta$ is the angle between the arbitrary plane and the normal plane then, according to Meusnier's theorem [23, 24], we get the following result:

$$\kappa_{\mathbf{t}} = \kappa \cos \Theta \tag{3.10}$$

The surface curvature at any point $P$ can be computed by estimating the planar curvatures in plane sections through $P$. Three distinct points in a three-dimensional space determine a plane. Within this plane these three points determine a circle and hence curvature and tangent vector along the circle. Let the three distinct points be $\mathbf{x}_{-1}$, $\mathbf{x}_0$ and $\mathbf{x}_{+1}$. Performing a translation so that $\mathbf{x}_0$ is now at the origin, it is

straightforward to obtain the circumcenter, $\mathbf{x}_c = (a, b, c)$, of the triangle formed by the three points. Knowing the circumcenter, the curvature vector is given by

$$\kappa = \frac{(a, b, c)}{(a^2 + b^2 + c^2)} \tag{3.11}$$

Defining us define vectors $\mathbf{u}$ and $\mathbf{v}$ as $\mathbf{u} = \mathbf{x}_{-1} - \mathbf{x}_0$ and $\mathbf{v} = \mathbf{x}_{+1} - \mathbf{x}_0$. The unit tangent vector to the circle passing through $\mathbf{x}_{-1}$, $\mathbf{x}_0$ and $\mathbf{x}_{+1}$, at the location $\mathbf{x}_0$, is given by

$$\mathbf{t} = \frac{(\mathbf{u} \times \mathbf{v}) \times \kappa}{\|(\mathbf{u} \times \mathbf{v}) \times \kappa\|} \tag{3.12}$$

Given sets of three points, we can find the tangent and curvature vectors of the circles passing through each of these sets. Hence for any point on the surface, two neighboring points can be selected and from these three points an estimation of the curvature of the planar section through the initial point can be obtained. As far as implementation is concerned, $\mathbf{x}_0$ and pairs of points lying on opposite sides of $\mathbf{x}_0$ are chosen. For each of these sets of three points, let $\mathbf{t}_i$ and $\kappa_i$ be the corresponding tangent and planar curvature vectors as indicated before. Each tangent vector $\mathbf{t}_i$ is tangent at the point $\mathbf{x}_0$ to the circle defined by the point set under consideration. Hence each pair of tangent vectors, $\mathbf{t}_i$ and $\mathbf{t}_j$, defines a plane that is tangent to two circles, each of which passes through the point $\mathbf{x}_0$. This plane is an approximation of the tangent plane to the surface at $\mathbf{x}_0$. Since the unit normal of the surface $S$ at point $P$ is known, Meusnier's theorem can be applied to give

$$\kappa_{n,i} = \kappa_i \cdot \mathbf{n} \tag{3.13}$$

Here $\kappa_{n,i}$ is an estimate of the normal curvature in the direction $\mathbf{t}_i$. Choosing a coordinate frame such that the $x-y$ plane is perpendicular to the vector $\mathbf{n}$, define

$$\mathbf{x}_i = (x_i, y_i) = \frac{\mathbf{t}_i}{(|\kappa_{n,i}|)^{1/2}} \tag{3.14}$$

In the above equation, $i = 1, \ldots, n$ where $n$ is the number of triangles meeting at point $P$. If $Ax^2 + 2Cxy + By^2 = \pm 1$ is the Dupin indicatrix then in the absence of numerical error we get

$$Ax_i^2 + 2Cx_iy_i + By_i^2 = \text{sgn}(\kappa_{n,i}) \tag{3.15}$$

Here again $i = 1, \ldots, n$. The above given equation has three unknowns ($A$, $B$ and $C$) and if $n > 3$, normal curvatures with their associated errors cannot all be expected to lie on a conic of this form. In this case, a least-squares approximation to the Dupin indicatrix can be made by minimizing the following function:

$$F = \sum_{i=1}^{n} (Ax_i^2 + 2Cx_iy_i + By_i^2 - \text{sgn}(\kappa_{n,i}))^2 \tag{3.16}$$

The minimum is given by values of $A$, $B$ and $C$ which satisfy the linear system of equations:

$$\begin{bmatrix} \sum x_i^4 & \sum x_i^3 y_i & \sum x_i^2 y_i^2 \\ \sum x_i^3 y_i & \sum x_i^2 y_i^2 & \sum x_i y_i^3 \\ \sum x_i^2 y_i^2 & \sum x_i y_i^3 & \sum y_i^4 \end{bmatrix} \begin{bmatrix} A \\ 2C \\ B \end{bmatrix} = \begin{bmatrix} \sum x_i^2 \ \text{sgn}(\kappa_{n,i}) \\ \sum x_i y_i \ \text{sgn}(\kappa_{n,i}) \\ \sum y_i^2 \ \text{sgn}(\kappa_{n,i}) \end{bmatrix} \tag{3.17}$$

All summations are over $i$ where $i$ varies from 1 to $n$. Now, the principal curvatures $\kappa_1$, $\kappa_2$ and the principal directions of curvature are eigenvalues and eigenvectors of the following matrix:

$$\begin{bmatrix} A & C \\ C & B \end{bmatrix}$$

$\kappa_1$ and $\kappa_2$ can be obtained by performing a rotation in the plane by an angle $\Theta$ where $\tan\Theta = 2C/(A-B)$. The Dupin indicatrix has a geometric interpretation as illustrated in Figure 3.11. A surface at $\mathbf{x}$ can be approximated by a paraboloid, that is, a Taylor expansion with terms up to second order. This leads to a simple interpretation of the Dupin indicatrix: the indicatrix, scaled down by a factor of $m$, can be viewed as the intersection of the surface with planes parallel to the tangent plane at $\mathbf{x}$ in the distance $\epsilon = \pm 1/(2m^2)$. The Dupin indicatrix can be assigned a sign depending on whether it is "above" or "below" the tangent plane.

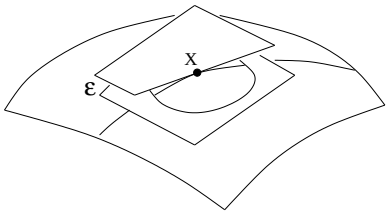



Figure 3.11: Dupin indicatrix (scale $1:m$) obtained by the intersection of the surface with the lower plane.

With all this information, different methods tried for numerically computing the mean curvature value are presented. All these methods, which are largely based on the Todd and McLeod approach, are discussed below:

1. **Mean of Normal Curvatures**: According to this method, surface curvature is estimated by computing the mean of the normal curvatures, $\{\kappa_{n,i}\}$, obtained by choosing sets of three points. This is a quick way of computing curvatures and the accuracy of values obtained will depend on the number of normal curvature values used and directions along which normal curvatures are calculated.

2. **Todd and McLeod Method**: In this method, the mean curvature value is obtained by taking the average of the maximum and minimum normal curvature of a conic whose curvature, $\kappa$, vs angle, $\Psi$, curve "closely" fits the set of normal curvature values, $\{\kappa_{n,i}\}$, obtained from the local geometry around point $P$. The details of obtaining a conic by method of least squares and principal curvature values by matrix rotation have already been discussed.

3. **Tangent Plane Intersection Method**: This method is based on fitting a conic at $P$ through points obtained by the intersection of two planes parallel to the tangent plane at $P$ and the local surface around that point. The two intersecting planes are at equal distance on both sides of tangent plane. This distance is chosen such that one of the planes passes through the midpoint of the edge having shortest $z$ component in the coordinate system where the $x-y$ plane is perpendicular to the normal vector, $\mathbf{n}$, at $P$. Once the intersection points are available, an estimate of Dupin indicatrix is made by following the least squares method discussed before. The sign of curvature now depends on the intersection plane used to get the intersection point. Rest of the procedure for obtaining mean curvature from the Dupin indicatrix is followed as outlined earlier.

# CHAPTER 4
# RESULTS

Several examples showing the potentialities of the code developed are given below. Complex interface representation capability is demonstrated first. Grid refinement ability, where points are inserted on a stationary sphere based on the area criteria, is shown next. Various methods of computing curvatures are compared later by expanding a sphere, uniformly in its normal direction, without any refinement. The effects of grid adaption on normal and curvature calculations of a sphere and an ellipsoid are also shown. These tests demonstrate the capabilities of the method as an important tool in interface representation.

## Complex Feature Representation

The ability of the tracker to represent complex shapes is demonstrated in this section. In Figure 4.1, velocity functions are prescribed to a sphere that result in the interface forming complex features. Figure 4.1a shows the initial sphere used to generate rest of the geometries. Figure 4.1b is a uniformly expanded sphere obtained by the velocity function: $v_n = 1$ where $v_n$ is the velocity in the direction of the surface normal. An ellipsoid (Figure 4.1c) is obtained by the following velocity function: $v_x = \pm x$ (velocity is positive when the $x$ coordinate is positive and negative when the $x$ coordinate is negative), $v_y = 0$, $v_z = 0$. Finally, Figure 4.1(d) shows the final shape obtained from the initial sphere by the velocity function: $v_n = \sin(|x|)\sin(|y|)\sin(|z|)$.

In all these cases, the features of interface are clearly captured by unstructured surface grids used. It is also clear that the features could not be represented as well if structured grids were used for the initial surface. It would be difficult to refine and insert new points to a structured surface as compared to an unstructured one.

Mesh refinement is based on the triangle quality and is done according to the mesh refinement algorithm discussed earlier in the report. We are associating quality values with every node of the surface mesh in order to investigate the effect of mesh quality on the geometric features such as normals and curvatures. The mesh quality value is set in two ways: first, by computing the ratio of the areas of the largest and the smallest triangles meeting at a node (area ratio) and second, by computing the mean of the aspect ratios of all the triangles meeting at a node (average aspect ratio). Here, aspect ratio of a triangle is the ratio of its circumradius and inradius which is equal to 2 for an equilateral triangle. Figure 4.2 shows the area ratio contours of the shapes discussed before. The original sphere has triangles of unequal areas which leads to nonhomogenous point insertion. Considering this fact, the results are fairly good. The average aspect ratio contours are depicted in Figure 4.3. It is observed that, even after considerable deformations, the average aspect ratio values do not change significantly. This clearly demonstrates the mesh refinement capability of the interface tracker. The use of Bowyer-Watson algorithm for point insertion and circumcircle criterion for diagonal swapping results in a good quality mesh. The mesh quality values, especially the average aspect ratio values, have an effect on the numerical curvature calculations performed. This correlation is highlighted in the rest of this chapter.

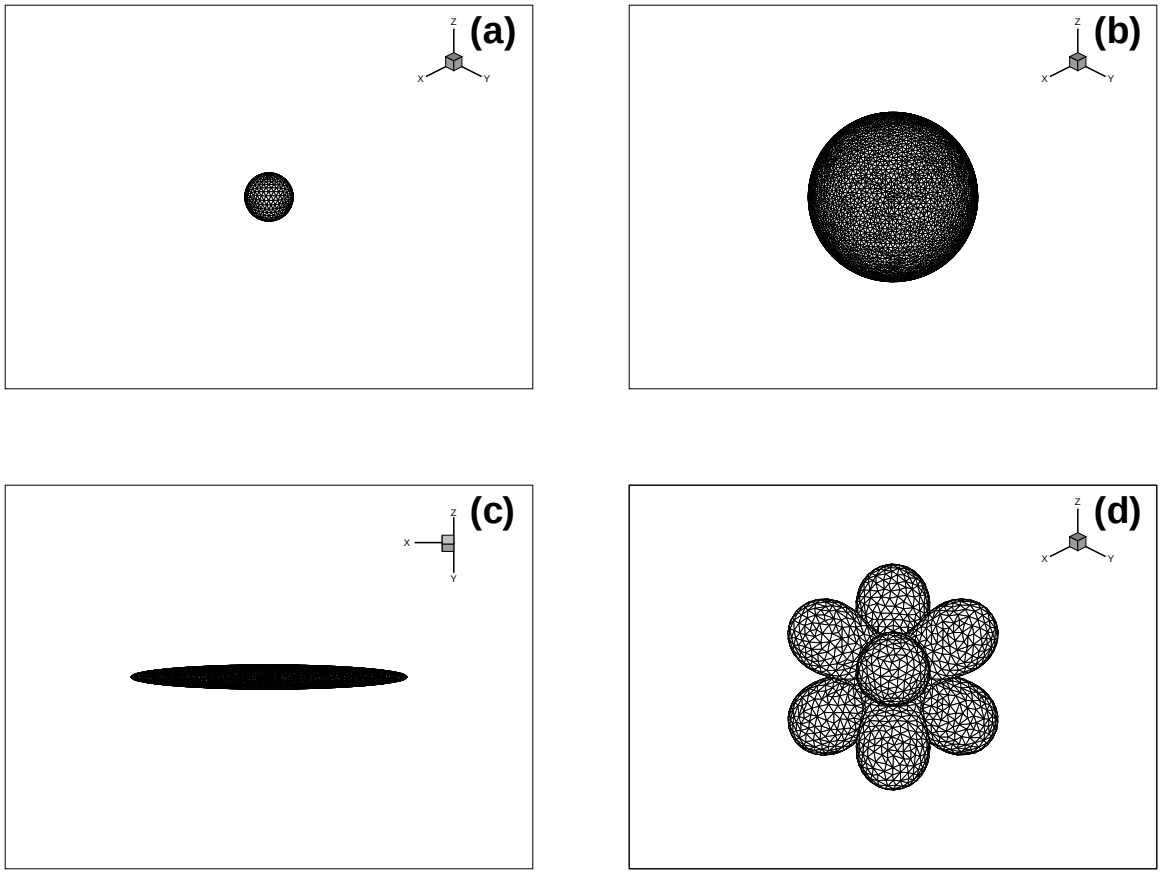



Figure 4.1: Grid refinement as a sphere expands into different shapes: (a) Initial sphere (402 points); (b) Final sphere (4,926 points); (c) Ellipsoid (3,794 points); (d) Final shape (2,454 points).

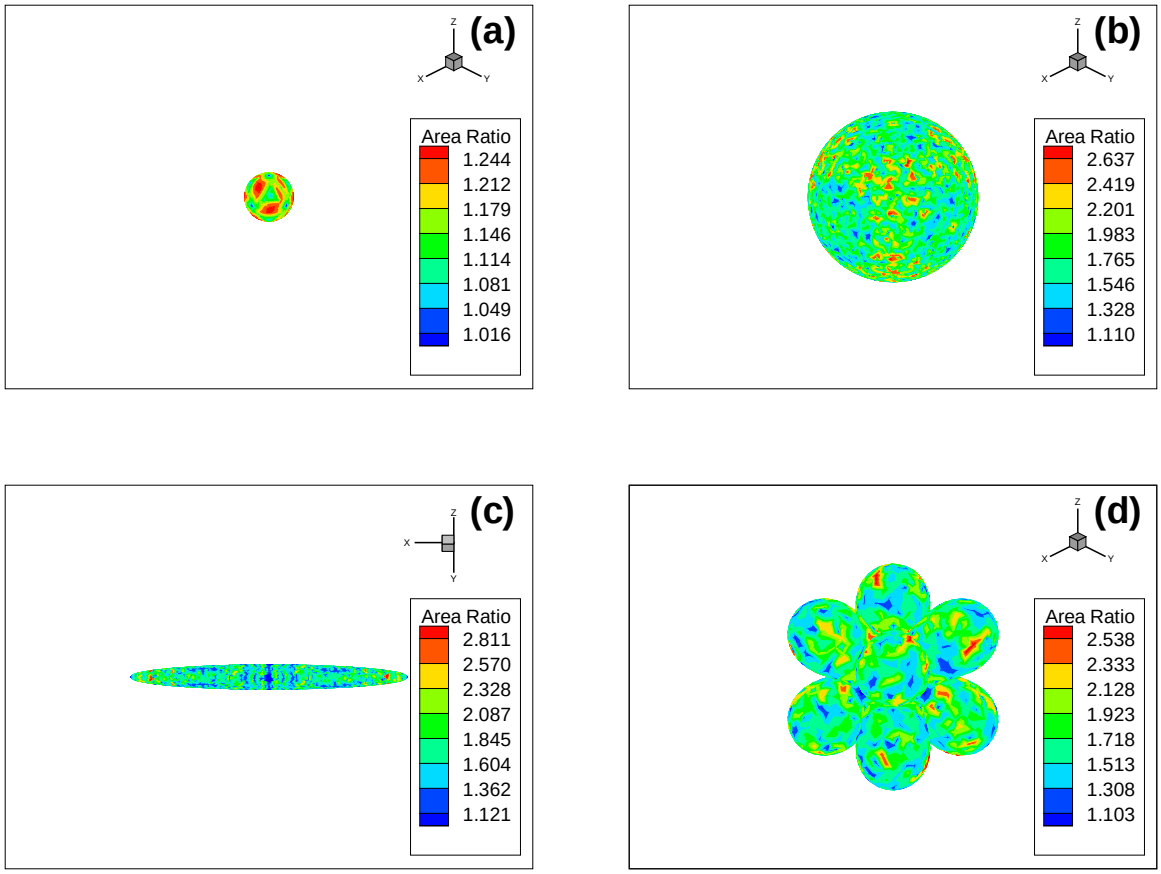



Figure 4.2: Maximum to minimum area ratio contours as a sphere expands into different shapes: (a) Initial sphere; (b) Final sphere; (c) Ellipsoid; (d) Final shape.

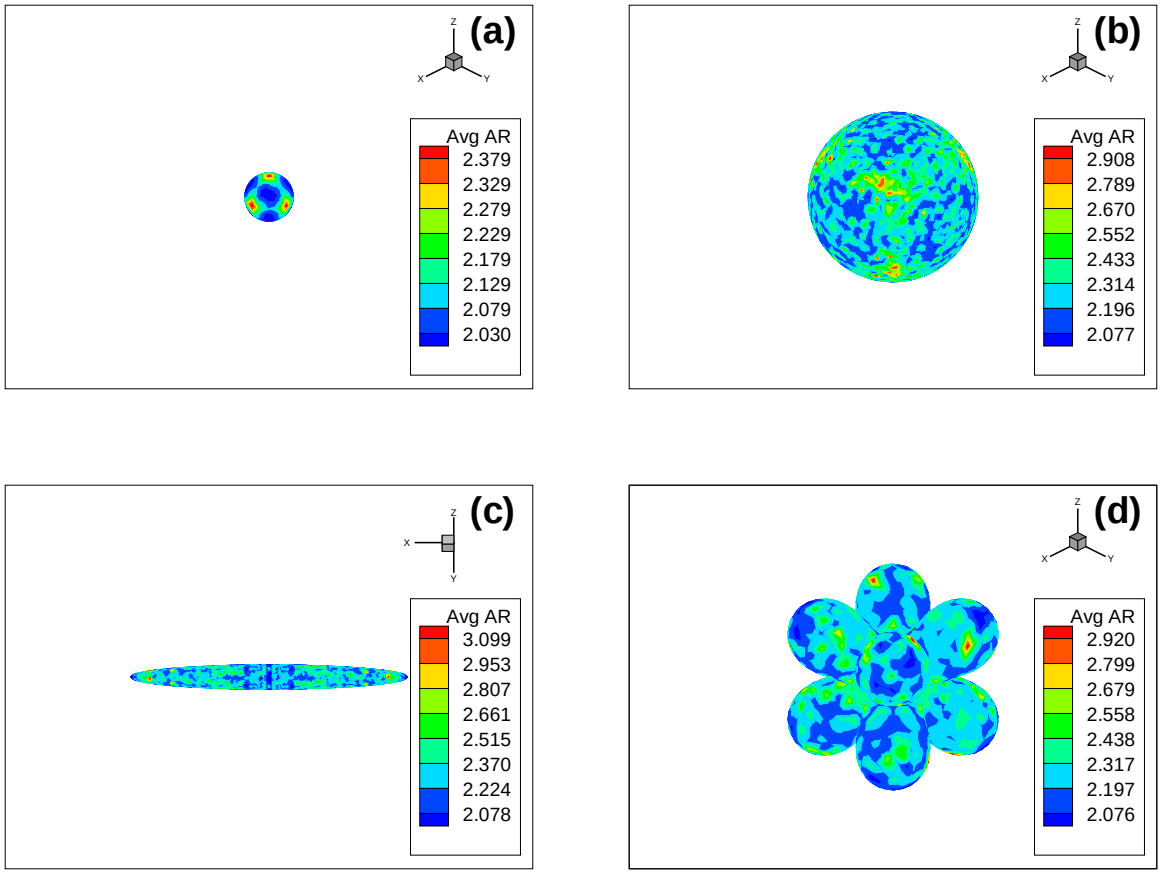



Figure 4.3: Average aspect ratio contours as a sphere expands into different shapes: (a) Initial sphere; (b) Final sphere; (c) Ellipsoid; (d) Final shape.

Table 4.1: Effect of area factor on the RMS radius and curvature errors as mesh refinement takes place on a stationary sphere.

| Area factor | Number of nodes | Curvature (% RMS) error | Radius (% RMS) error |
|---|---|---|---|
| 0.2 | 3,262 | 1.285 | $6.940 \times 10^{-3}$ |
| 0.4 | 1,502 | 0.996 | $7.067 \times 10^{-3}$ |
| 0.6 | 1,030 | 1.002 | $7.306 \times 10^{-3}$ |
| 0.8 | 770 | 2.561 | $5.685 \times 10^{-3}$ |
| 1.0 | 658 | 2.259 | $2.259 \times 10^{-3}$ |

**Grid Refinement**

In order to test the accuracy of the point-placement strategy and the curvature values obtained, consider a stationary sphere that is refined on the basis of the area criteria. Therefore, points are inserted on a sphere that remains fixed in space. Table 4.1 shows the effect of the area factor (a triangle is refined if its area is greater than the average triangle area times an area factor) on the root mean square (RMS) radius and curvature errors of a sphere. The initial radius and the theoretical value of curvature are used to normalize the errors (expressed in percentage). Initially, that is, before any mesh refinement takes place, we have the following: Curvature (% RMS) error = 0.488 and Radius (% RMS) error = $3.184 \times 10^{-5}$.

Curvature calculations are done using the Todd and McLeod method and the initial error in curvature, even though points accurately lie on a sphere, is because of the numerical computation of curvature from available discrete node data. Area

factor does not have appreciable effect on the radius (% RMS) error as this error increases marginally with the increase in the number of nodes. This information is an indication of the fact that point placement is highly accurate. As far as curvature (% RMS) error is concerned, it shows erratic behavior. Since the curvature error not only depends on the location of points but also on the way curvature is computed numerically, we have attempted to correlate the curvature errors with mesh quality measures discussed before. We have also compared the configuration of the sphere that gives accurate curvatures (area factor 0.4) with the one that results in relatively large curvature errors (area factor 0.8).

Figure 4.4 shows the initial configuration of a sphere about to be refined. Figure 4.4a shows the surface mesh composed of eight spherical patches. Here, curvature is computed using the Todd and McLeod approach. Normalized curvature (RMS) error contour is shown in Figure 4.4b. It is seen that there is some error in the zones where the surface patches meet. Figure 4.4c shows the normalized radius (RMS) error contour. Radius error is very low in this case. The area ratio contour is shown in Figure 4.4d and the average aspect ratio contour is illustrated in Figure 4.4e. It is strikingly clear that there is a direct correlation between the curvature error and the aspect ratio. Even though points of the surface mesh lie on a sphere, as is evident from the negligible radius error, we still get appreciable curvature error. Hence, besides the new surface-mesh nodes being placed accurately, surface-mesh triangles should have good aspect ratios as well. This is the conclusion we can draw for getting accurate curvature results from the discrete node and their connectivity information available to us.

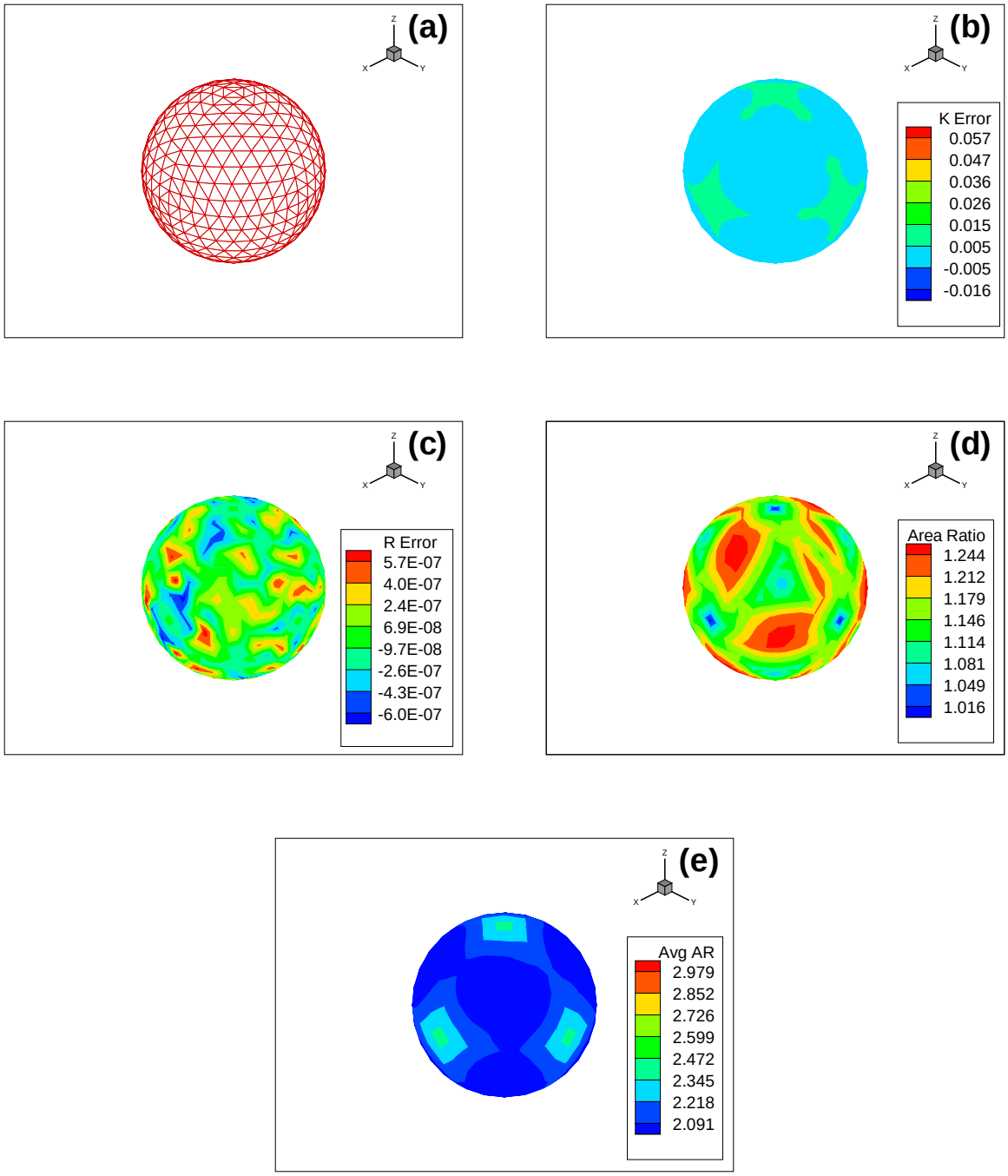



Figure 4.4: Initial configuration before mesh refinement takes place on a stationary sphere: (a) Initial surface mesh; (b) Contour of normalized curvature (RMS) error; (c) Contour of normalized radius (RMS) error; (d) Contour of area ratio; (e) Contour of average aspect ratio.

Figure 4.5 shows the final configuration of a stationary sphere that is refined with an area factor = 0.8. The initial uniform unstructured mesh loses its symmetry due to the insertion of a few points. The number of new points added is less than the number of points in the mesh originally. This results in a nonuniform triangulation as shown in Figure 4.5a. This fact is also evident from the area ratio (Figure 4.5d) and aspect ratio (Figure 4.5e) contours drawn on a stationary sphere that is refined by point insertion and diagonal swap operations. Here again, the normalized curvature error (Figure 4.5b) is high in a zone of triangles having high aspect ratios.

The case of a stationary sphere that is refined with an area factor = 0.4 (Figure 4.6) is quite different from the one discussed above. Since the maximum allowable area of a triangle is less in this case, there are more number of points inserted into the original surface mesh. The final configuration has more than double the number of points in the original triangulation, hence, every triangle in the initial unstructured surface mesh gets bisected resulting in a more uniform-looking triangulation (Figure 4.6a). This is also deducible from the area ratio (Figure 4.6d) and aspect ratio (Figure 4.6e) contours. Since the resulting triangulation is uniform, curvature errors (Figure 4.6b) are low.

The conclusion drawn from the cases of stationary grid refinement is that the mean curvature of the surface not only depends on the accuracy of the placement of the points on the surface but also on the quality of the surface mesh. The quality of the unstructured mesh affects the mean curvature value because we do not have a local analytical representation of the surface and curvature is being computed numerically from discrete point and their connectivity information available to us.

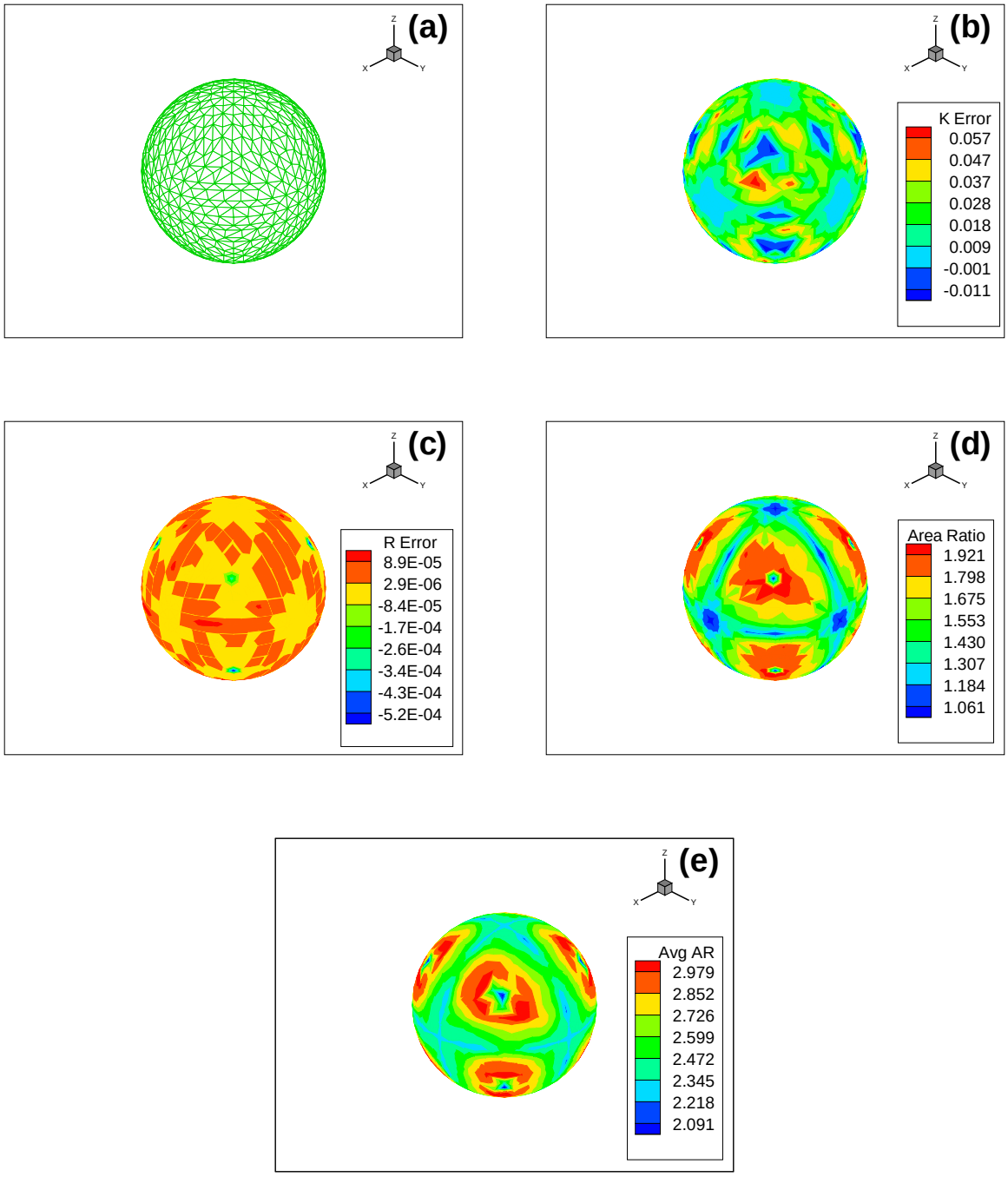



Figure 4.5: Final configuration of a stationary sphere that is refined with an area factor = 0.8: (a) Final surface mesh; (b) Contour of normalized curvature (RMS) error; (c) Contour of normalized radius (RMS) error; (d) Contour of area ratio; (e) Contour of average aspect ratio.

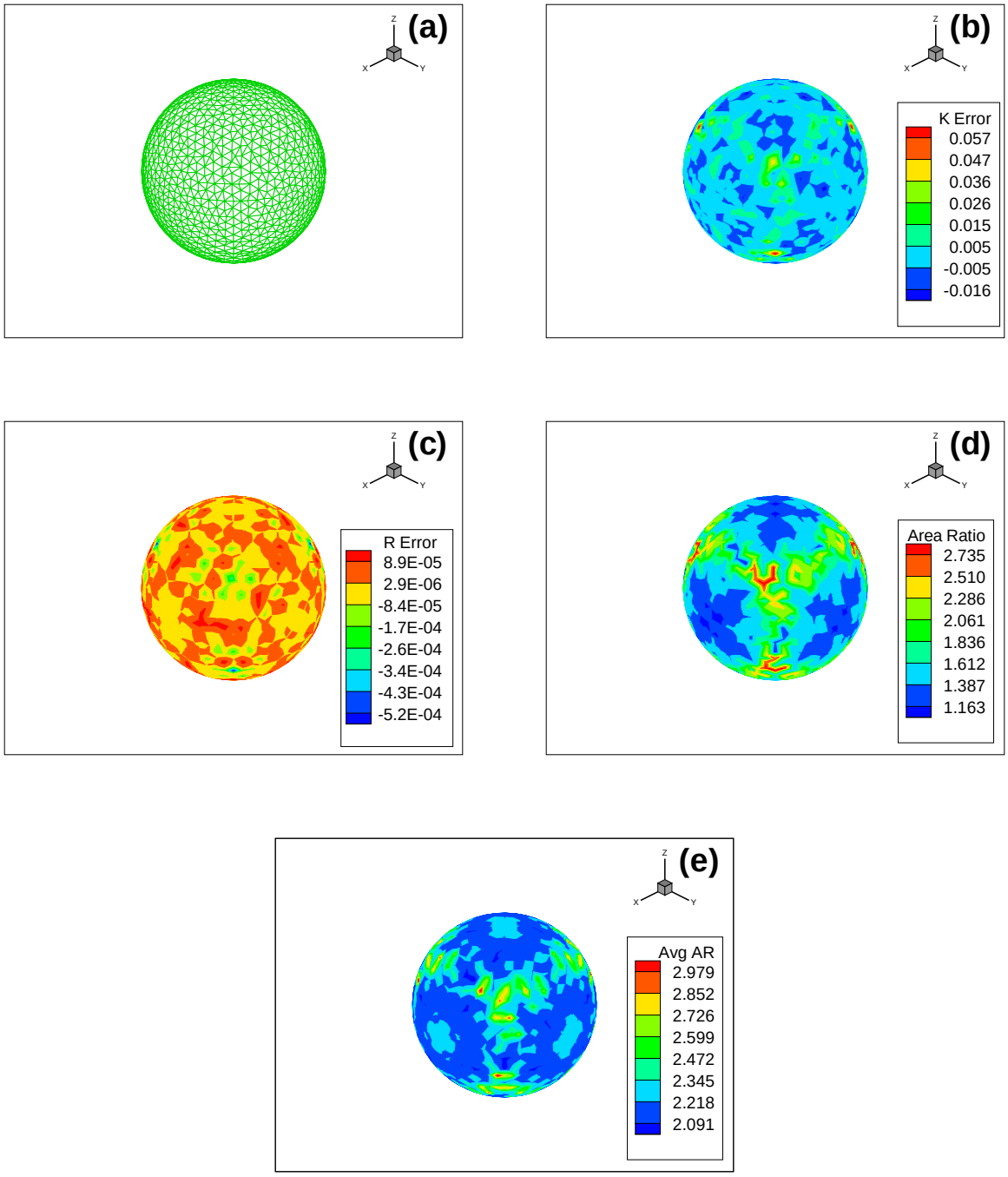



Figure 4.6: Final configuration of a stationary sphere that is refined with an area factor = 0.4: (a) Final surface mesh; (b) Contour of normalized curvature (RMS) error; (c) Contour of normalized radius (RMS) error; (d) Contour of area ratio; (e) Contour of average aspect ratio.

Various methods adopted for evaluating the mean curvature are compared by uniformly expanding a unit sphere. Nodes on the surface are simply advected and there is no mesh refinement taking place in the cases discussed next.

**No Grid Refinement**

Figure 4.7 shows the curvature (% RMS) errors plotted against the number of time steps as a sphere uniformly expands in time. The curvature calculated over the parametric surface, obtained after application of the $C^1$ surface spline algorithm, is fairly accurate. The surface in this case is composed of Bernstein-Beźier triangular patches. Control points of these patches are used to compute derivatives that are required in curvature calculations. Blend ratios of values 0.83 each lead to this result. The error, however, increases with a different choice of ratios. Actually, the blend ratios determine the depth of cuts into the polytope outlined by the control mesh. Smaller cuts result in a surface that follows the input mesh more closely and changes the normal direction more rapidly across the boundary. The choice of the best blend ratios, hence, becomes an issue and the inability to determine them automatically makes the $C^1$ surface spline algorithm unuseable. Mean of normal curvatures and Todd and McLeod methods yield more accurate results. The curvature-error plots obtained from these two methods are almost identical in the absence of any mesh refinement. The curvature error grows in time with the application of the tangent plane intersection method and this method turns out to be highly inaccurate.

The results obtained show that either the mean of normal curvatures or the Todd and McLeod approach should be used for computing mean curvatures on arbitrary interfaces.

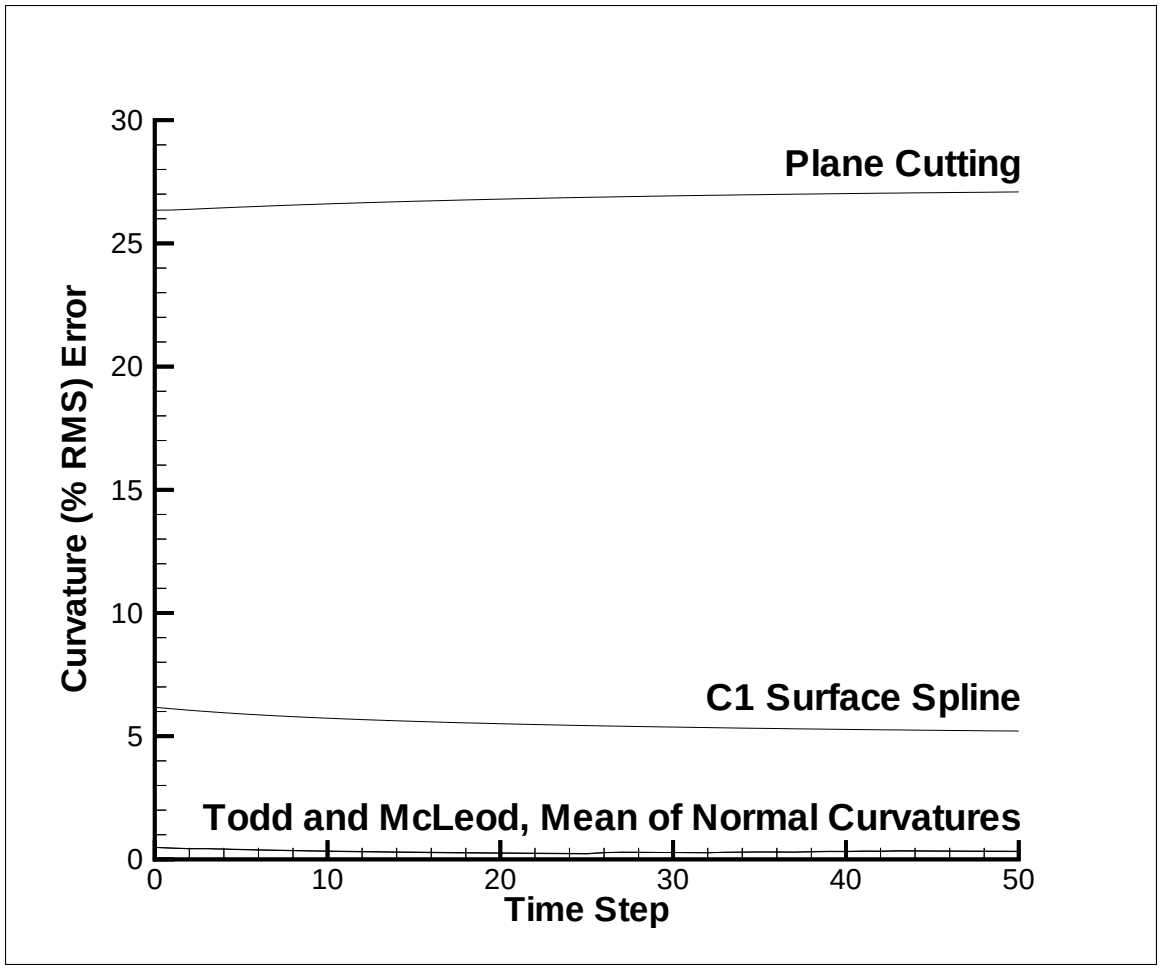



Figure 4.7: Comparison of different methods used for computing curvatures on a sphere when it expands without any mesh refinement.

Table 4.2: Effect of time step size on the radius and curvature (% RMS) errors of a sphere expanding in the absence of any mesh refinement.

| Time increment | Number of time steps | CFL | Curvature (% RMS) error | Radius (% RMS) error |
|---|---|---|---|---|
| 0.025 | 40 | 0.1 | 0.264 | $2.868 \times 10^{-3}$ |
| 0.125 | 8 | 0.5 | 0.264 | $2.760 \times 10^{-3}$ |
| 0.250 | 4 | 1.0 | 0.265 | $2.638 \times 10^{-3}$ |
| 1.000 | 1 | 4.0 | 0.263 | $2.131 \times 10^{-3}$ |

Table 4.2 shows the effect of time step size (CFL) on curvature and radius errors. Curvature, in this case, is computed using the Todd and McLeod approach which is based on the numerical approximation of the Dupin indicatrix. The final configuration of the sphere is same as it expands for unit time in all the cases. From the results obtained, it is clear that time step size does not have any significant effect in the absence of mesh refinement. The radius error is negligible and the curvature error is also insignificantly same in all these cases. These results also indicate that the normals, used to advect the nodes of the mesh, are computed accurately.

**Grid Adaptation**

Figure 4.8 shows the contour plots of components of normals of various shapes. The contours of z components of normals of the initial sphere (Figure 4.8a) and the final sphere (Figure 4.8b) are circles as expected. Figure 4.8c shows the contours of y components of normals of an ellipsoid and Figure 4.8d depicts the contours of z components of normals computed numerically on the final shape.

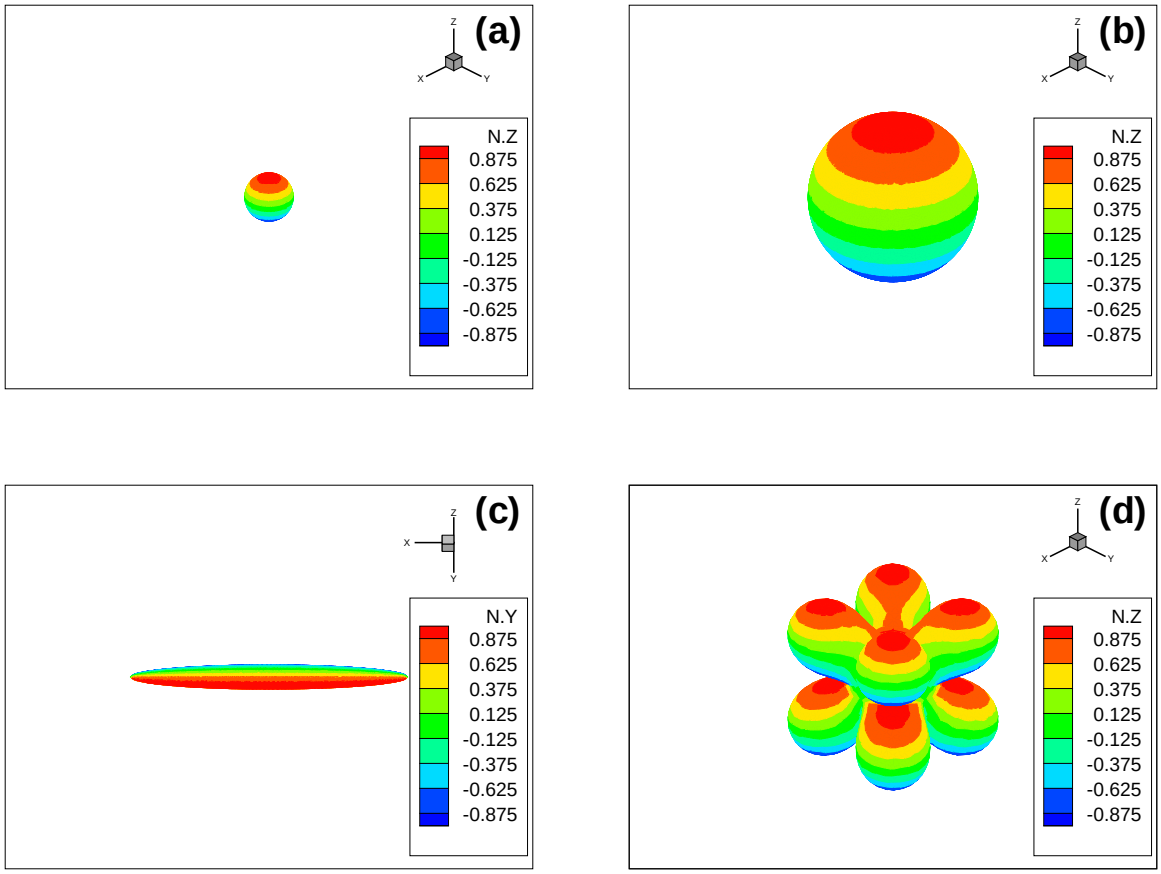



Figure 4.8: Contours of components of normals obtained after mesh refinement: (a) $z$ component of normal of initial sphere; (b) $z$ component of normal of final sphere; (c) $y$ component of normal of ellipsoid; (d) $z$ component of normal of final shape.

Table 4.3: Effect of time step size on the radius and curvature (% RMS) errors of a sphere that gets refined during uniform expansion.

| Time increment | Number of time steps | CFL | Number of nodes | Curvature (% RMS) error | Radius (% RMS) error |
|---|---|---|---|---|---|
| 0.025 | 40 | 0.1 | 1,585 | 0.999 | $7.658 \times 10^{-3}$ |
| 0.125 | 8 | 0.5 | 1,609 | 0.929 | $7.365 \times 10^{-3}$ |
| 0.250 | 4 | 1.0 | 1,613 | 0.842 | $7.311 \times 10^{-3}$ |
| 1.000 | 1 | 4.0 | 1,592 | 1.416 | $7.440 \times 10^{-3}$ |

Table 4.3 shows the effect of time step size (CFL) on the curvature and radius errors of a uniformly expanding sphere. The final configuration of the sphere is the same in all these cases. Area factor = 1.5 here. Curvature is computed using the Todd and McLeod method. The curvature RMS errors show an erratic trend but they approximately have the same value. The randomness in the result is attributed to the quality of the mesh obtained after all these cases. Radius RMS errors, on the other hand, are hardly affected by the time step size.

Figure 4.9 shows the final configuration of a sphere (initial radius = 1 and area factor = 1.5), that gets refined during uniform expansion, after 50 time steps (Time step size = 0.05). The resulting surface mesh is shown in Figure 4.9a. The new point placement is accurate as is evident from the radius error (Figure 4.9c) contour. Area ratio contours of the surface mesh are depicted in Figure 4.9d. A careful observation shows that high curvature error (Figure 4.9b) occurs in the regions where triangles have high aspect ratios (Figure 4.9e).

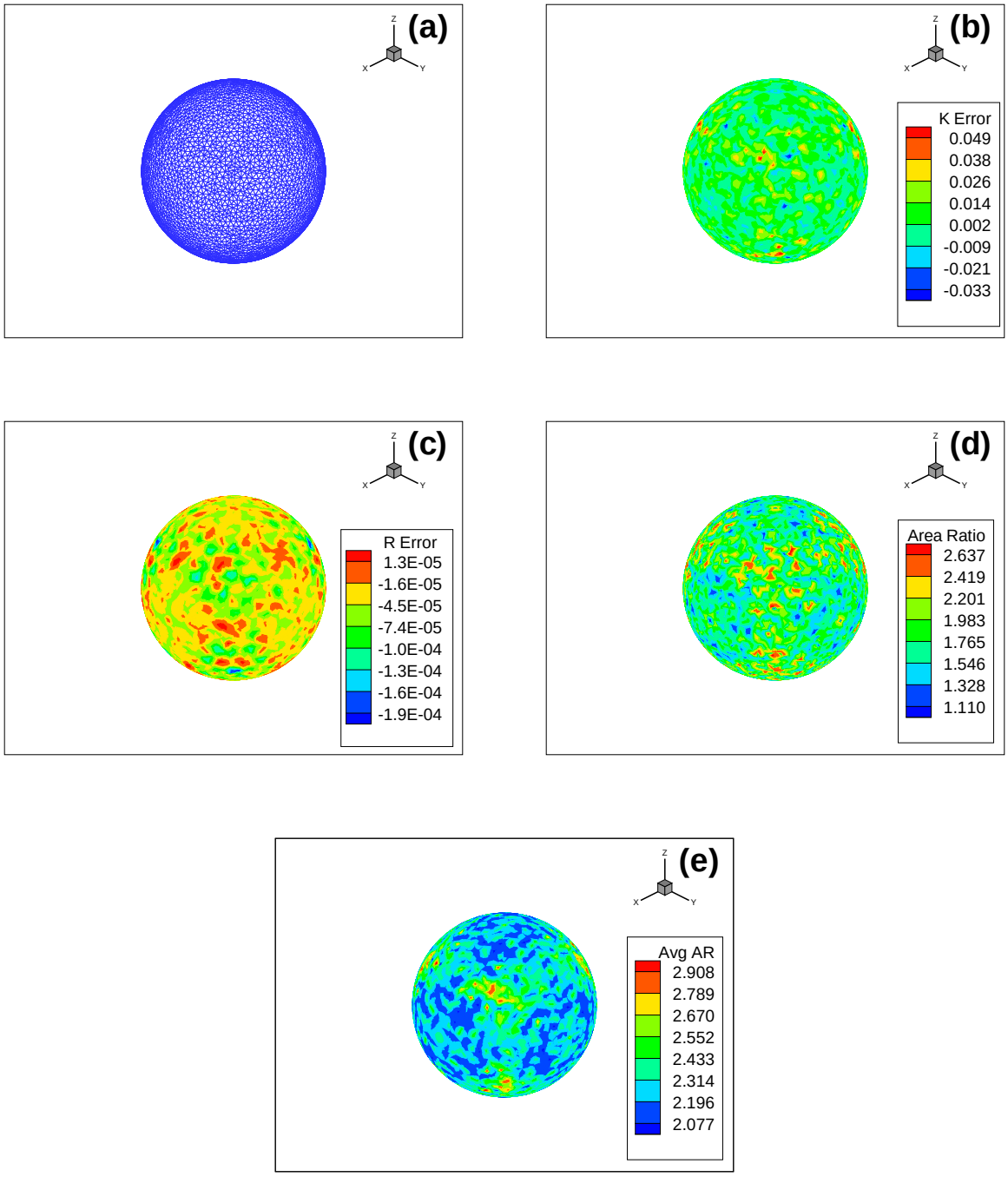



Figure 4.9: Final configuration of a sphere (initial radius = 1), that gets refined during uniform expansion, after 50 time steps (dt = 0.05): (a) Final surface mesh; (b) Contour of normalized curvature (RMS) error; (c) Contour of normalized radius (RMS) error; (d) Contour of area ratio; (e) Contour of average aspect ratio.

Table 4.4: Effect of area factor on the radius and curvature (% RMS) errors as mesh refinement takes place on a sphere expanding uniformly in the outward direction.

| Area factor | Number of nodes | CFL | Curvature (% RMS) error | Radius (% RMS) error |
|---|---|---|---|---|
| 0.5 | 4,733 | 0.37 | 1.235 | $4.179 \times 10^{-3}$ |
| 1.0 | 2,501 | 0.26 | 1.364 | $6.191 \times 10^{-3}$ |
| 1.5 | 1,586 | 0.22 | 0.866 | $7.440 \times 10^{-3}$ |
| 2.0 | 1,242 | 0.19 | 0.971 | $8.459 \times 10^{-3}$ |
| 2.5 | 962 | 0.17 | 1.247 | $8.970 \times 10^{-3}$ |

The effect of the area factor, on the curvature and radius RMS errors, is shown in Table 4.4. The case is that of a uniformly expanding sphere that is refined as it grows in size. The time step size = 0.05 and the number of time steps = 50. Curvature calculations are done using the Todd and McLeod approach. Curvature RMS error remains almost the same. The variations in these values are because of the quality of the mesh obtained at the end. The radius error increases slightly with an increase in the area factor but this error is insignificant because of the accurate placement of the new points.

A comparison between the Todd and McLeod method and the mean of normal curvature method is made below. For a uniformly expanding sphere (Figure 4.10), the mean of normal curvatures computed along a few directions at every node results in more accurate results than those obtained by the Todd and McLeod approach. The initial radius of the sphere = 1, area factor = 1.5, time step size = 0.05 and

the number of time steps = 50. The results in both the cases depend on the choice of points used for computing normal curvatures. It is observed that a pair of points lying opposite to each other with the given point in the middle form a set of three points which results in good normal curvature values. The accuracy decreases if the pair of points lie on the same side of the concerned node. For a sphere, the graphs of Todd and McLeod curvature error and mean of normal curvature error have the same trend. In the latter case, the curvature error peaks are, however, lower and this is because of the averaging involved which smoothens out the curvature error.

Figure 4.11 shows the variation of normalized curvature (% RMS) errors with time steps for an ellipsoid. The ellipsoid is obtained from a sphere (initial radius = 1) expanding with a time step size of 0.05. The area factor = 1.5 here. Theoretically, the $x$ coordinate of a point on an ellipsoid can be expressed in terms of its $y$ and $z$ coordinates as: $x = \sqrt{1 - y^2 - z^2}(1 + \Delta t)^n$. Here $\Delta t$ is the time step size and $n$ is the number of time steps. This relation is used to compute the exact curvature and therefore curvature error of an ellipsoid. The graph shows that the Todd and McLeod method works better for an ellipsoid where normal curvatures, at arbitrary nodes on the surface, change with the tangential directions. The mean of normal curvature is not as accurate because in principle, the mean of the normal curvatures obtained at a few normal directions may not be equal to the average of the maximum and minimum normal curvatures at the concerned point. The error depends on the directions chosen for computing normal curvatures.

As a conclusion, we can say that the Todd and McLeod method should be used for numerically computing curvature over arbitrary moving interfaces.

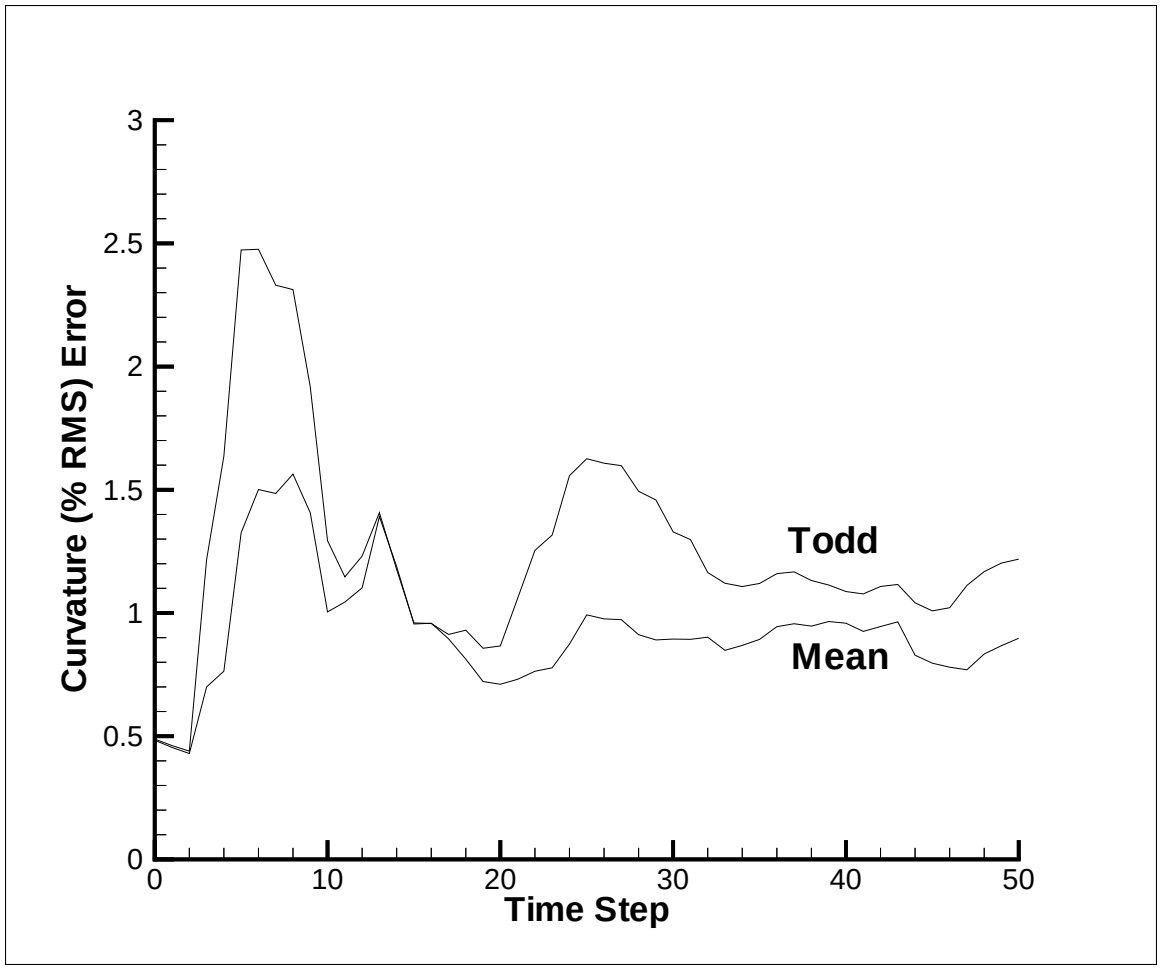



Figure 4.10: A comparison between the Todd and McLeod approach and the mean of normal curvatures method used for numerically computing curvature of a uniformly expanding sphere.

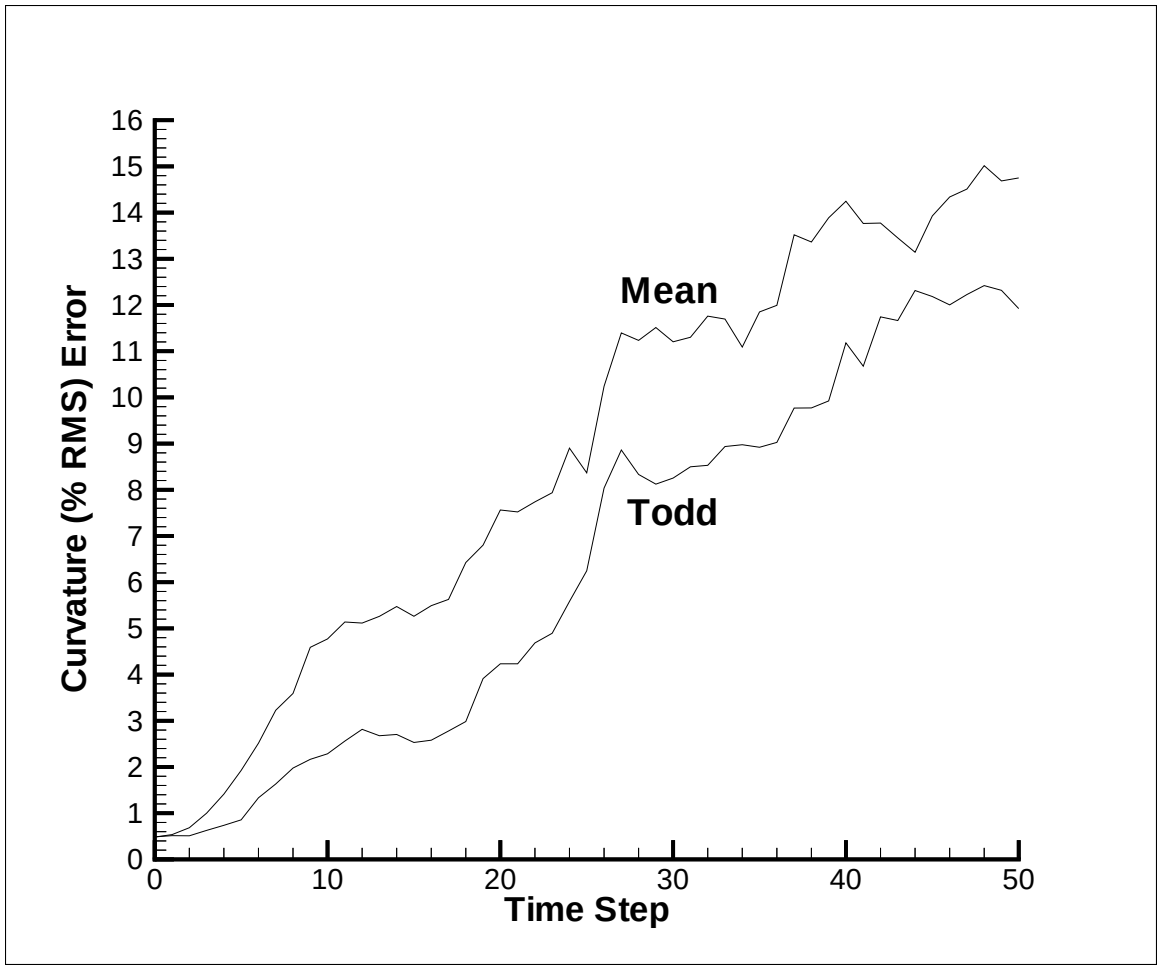



Figure 4.11: A comparison between the Todd and McLeod approach and the mean of normal curvatures method used for numerically computing curvature of an ellipsoid.

Figure 4.12 shows the final configuration of an ellipsoid, obtained from a sphere of unit radius, after 20 time steps (time step size = 0.05). As the ellipsoid grows in time, it gets refined by the point insertion and diagonal swap operations. Point insertion, performed according to the local sphere approximation (Subsection 3), introduces an error in the radial distance of an arbitrary point on an ellipsoid from the origin (we call this a "radius error"). This is shown in Figure 4.12c. The resulting triangulation has good area ratios (Figure 4.12d) and aspect ratios (Figure 4.12e) though. The curvature error of an ellipsoid (Figure 4.12b) appears to be related to the radius errors. A close observation shows that regions of high radius error correspond to zones of high curvature error. This behavior is different from that observed for a sphere where the main contribution to the curvature error came from bad aspect ratios.

We have tried to circumvent the problem arising due to radius errors by projecting points exactly on an ellipsoid. The mesh is refined at every time step by insertion of points according to the local sphere approximation and the new points are exactly placed on an ellipsoid by adjusting their $x$ coordinate according to their $y$ and $z$ coordinates. Figure 4.13 shows the final configuration of such an ellipsoid, obtained by the expansion of a sphere (radius = 1), after 20 time steps (time step size = 0.05). From a visual inspection of the final surface mesh (Figure 4.13a), we observe that the ellipsoid has high area ratios (Figure 4.13d) and aspect ratios (Figure 4.13e) towards its center. The curvature errors (Figure 4.13b) here seem to be directly related to the aspect ratios.

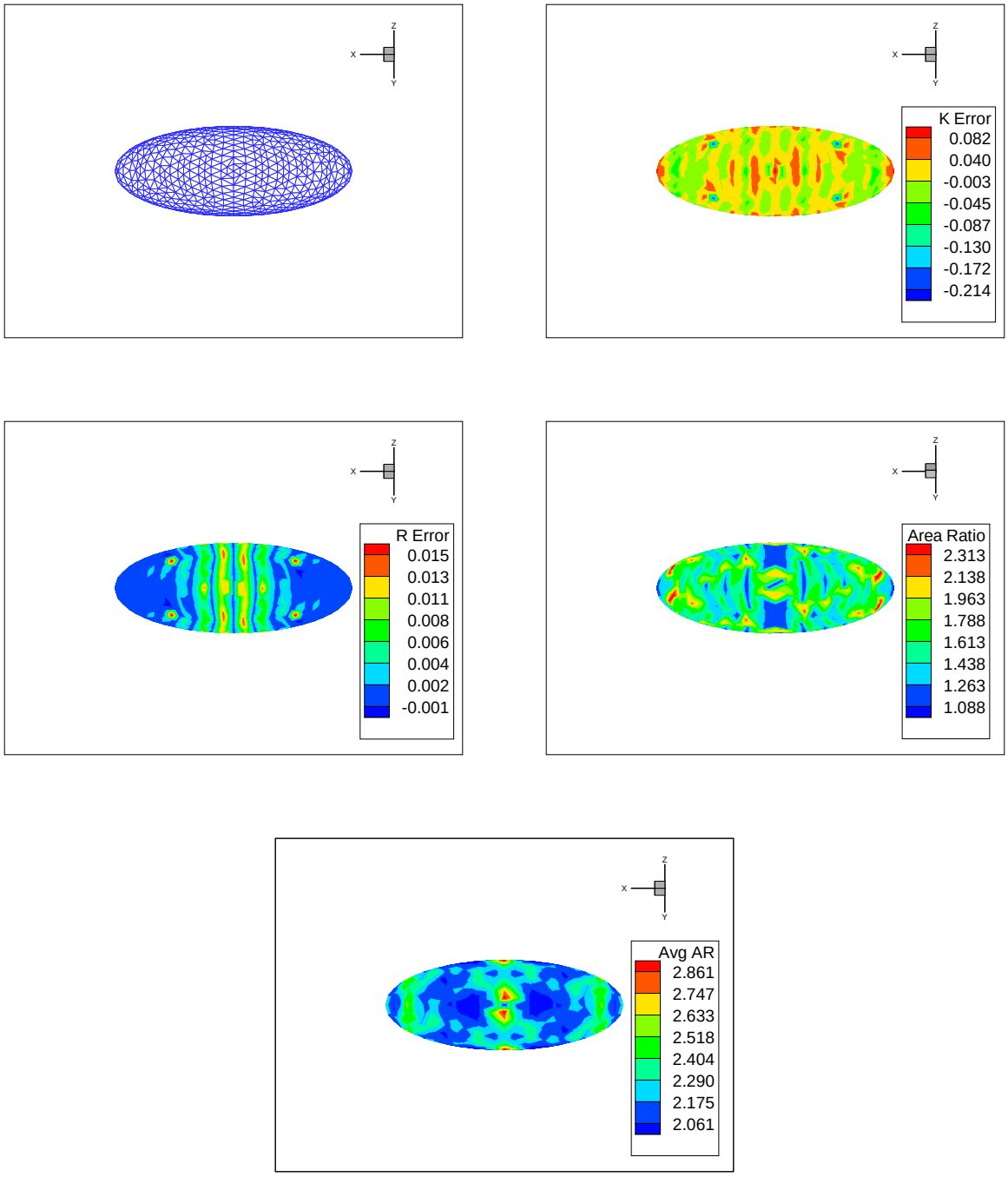



Figure 4.12: Final configuration of an ellipsoid, obtained from a sphere (radius = 1), after 20 time steps (dt = 0.05): (a) Final surface mesh; (b) Contour of normalized curvature (RMS) error; (c) Contour of normalized radius (RMS) error; (d) Contour of area ratio; (e) Contour of average aspect ratio.

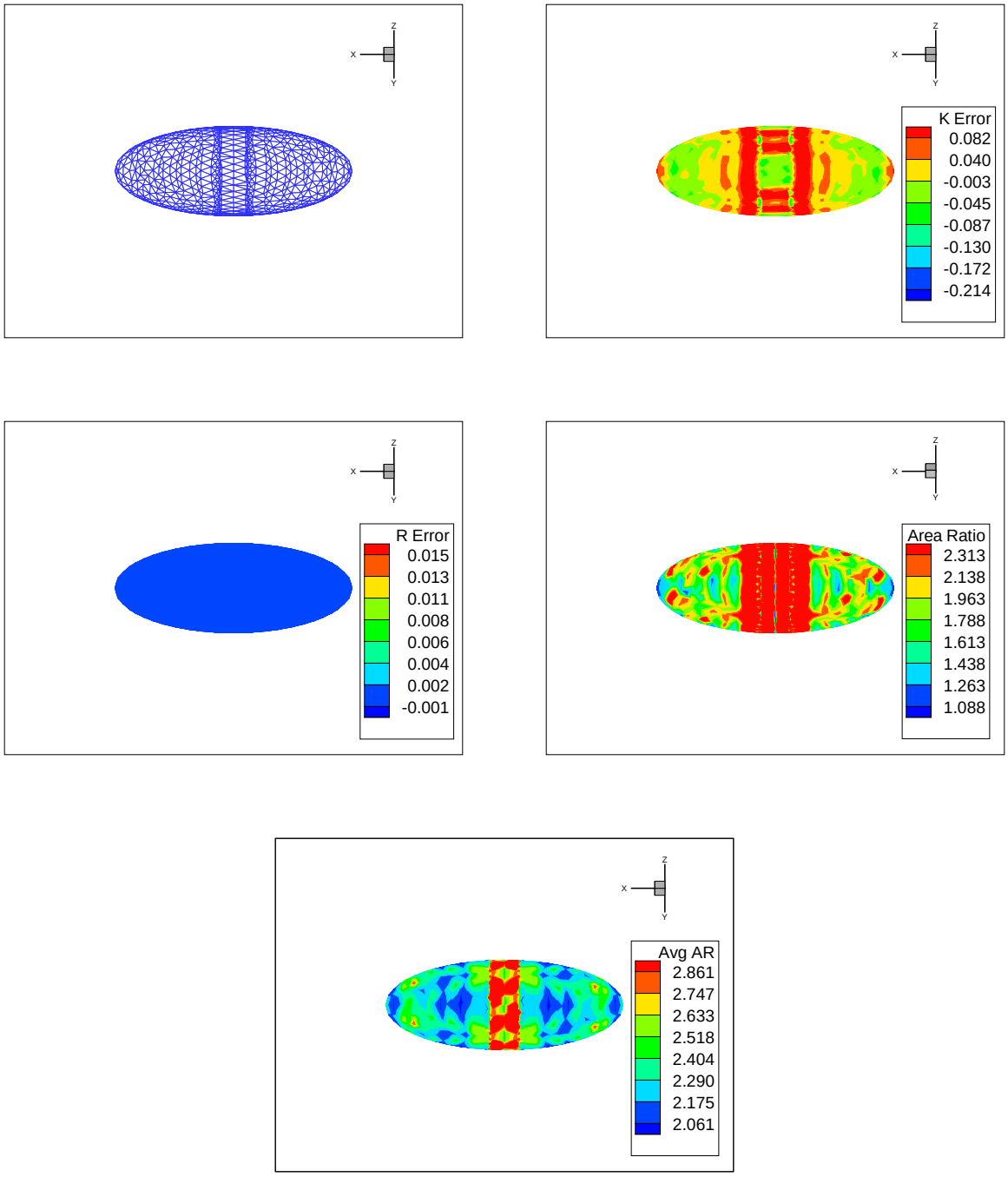



Figure 4.13: Final configuration of an ellipsoid, obtained from a sphere (radius = 1) such that there is no radius error, after 20 time steps (dt = 0.05): (a) Final surface mesh; (b) Contour of normalized curvature (RMS) error; (c) Contour of normalized radius (RMS) error; (d) Contour of area ratio; (e) Contour of average aspect ratio.

# CHAPTER 5
# CONCLUSIONS AND FUTURE WORK

The algorithm for tracking arbitrary interfaces in three dimensions is developed in object oriented C++. Some of the difficulties associated with unstructured grids are related to the numerous bookkeeping operations that frequently need to be performed during node, edge and triangle additions, reconnections and subtractions. It becomes important to design appropriate data structures that optimize both search and storage procedures. Our code takes full advantage of useful data structures such as stacks and linked lists and makes extensive use of the Standard Template Library (STL) for performing operations on them.

The decision of using unstructured grids for representing a moving interface turns out to be a judicious one. Grid refinement, which is possible in this case, is very useful in preserving complex interface features as they develop. Point insertion by the application of the Bowyer-Watson algorithm and diagonal swapping by using the circumcircle criterion results in Delaunay triangulation having fairly good aspect ratios and area ratios.

Calculation of normals and surface curvatures is an important part of the interface-tracking code. Computation of the surface normal by taking the mean of normals of triangles meeting at the concerned point results in accurate values. As far as curvature calculations are concerned, the Todd and McLeod and the mean of normal curvatures

methods give highly accurate results when compared with the values obtained by the plane cutting and $C^1$ surface spline methods. For an arbitrary interface, where the normal curvature value varies with orientation of the normal plane, the numerical approximation to the Dupin indicatrix by the Todd and McLeod approach is found to be more accurate than the mean of normal curvature values obtained along a few directions.

Curvature values are related to the quality of the mesh and significant errors are obtained at locations where triangles having bad aspect ratios meet. Incorrect placement of the new point during the point insertion operation also contributes to this error. The local sphere approximation, used to calculate the coordinates of these new points, leads to placement of the point on a symmetric spherical patch having uniform curvature values along all directions. This technique, therefore, is crude because, in reality, for an arbitrary surface the curvature at the new node location varies with the orientation of the normal plane.

The efforts in the future, thus, should be directed towards insertion of the point at an appropriate location. We need to place the new point on a smooth parametric surface patch whose control points, unlike the $C^1$ spline control points, coincide with the points of the unstructured triangulated surface mesh.

# APPENDIX A
# POINTS AND VECTORS

Points are elements of three-dimensional euclidean (or point) space $\mathcal{E}^3$. They identify a location, often relative to other objects. Examples are the midpoint of a straight line segment or the center of gravity of a physical object. Vectors are elements of three-dimensional linear (or vector) space $\mathcal{R}^3$. They have a magnitude and a direction associated with them.

Although both points and vectors are described by triples of real numbers, there is a clear distinction between them. For any two points $\mathbf{a}$ and $\mathbf{b}$, there is a unique vector $\mathbf{v}$ that points from $\mathbf{a}$ to $\mathbf{b}$. This is produced by componentwise subtraction $\mathbf{v} = \mathbf{b} - \mathbf{a}$. On the other hand, given a vector $\mathbf{v}$, there are infinitely many pairs of points $\mathbf{a}$, $\mathbf{b}$ such that $\mathbf{v} = \mathbf{b} - \mathbf{a}$. For if $\mathbf{a}$, $\mathbf{b}$ is one such pair and if $\mathbf{w}$ is an arbitrary vector, then, $\mathbf{a} + \mathbf{w}$, $\mathbf{b} + \mathbf{w}$ is another such pair since $\mathbf{v} = (\mathbf{b} + \mathbf{w}) - (\mathbf{a} + \mathbf{w})$.

Assigning the point $\mathbf{a} + \mathbf{w}$ to every point $\mathbf{a} \in \mathcal{E}^3$ is called a *translation* and vectors are invariant under translations while points are not. Elements of point space $\mathcal{E}^3$ can only be subtracted from each other and this operation yields a vector. They cannot be added as this operation is not defined for points (it is defined for vectors). However, addition like operations are defined for points and they are called *barycentric combinations* [1]. These are weighted sum of points where the weights sum to one:

[1] Also called *affine combinations*.

$\mathbf{b} = \sum_{j=0}^{n} \alpha_j \mathbf{b}_j$ where $\alpha_0 + \cdots + \alpha_n = 1$. Barycentric combinations generate points from points.

An important special case of barycentric combinations are *convex combinations*. These are barycentric combinations where the coefficients $\alpha_j$, in addition to summing to one, are also nonnegative. A convex combination of points is always "inside" those points. This observation leads to the definition of the *convex hull* of a point set which is the set formed by all convex combinations of a given point set. Figure A.1 gives an example of a convex hull.

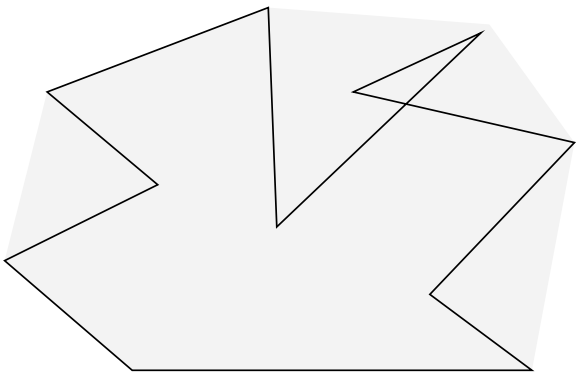


Figure A.1: Convex hull of a point set (a polygon) shown shaded.

The convex hull of a point set is a *convex set*. Such a set is characterized by the following: for any two points in the set, the straight line connecting them is also contained in the set.

# APPENDIX B
# BARYCENTRIC COORDINATES

Consider a triangle with vertices $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$ and a fourth point $\mathbf{p}$, all in $\mathcal{E}^2$. It is always possible to write p as a barycentic combination of $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ as $\mathbf{p} = u\mathbf{a} + v\mathbf{b} + w\mathbf{c}$ where $u+v+w = 1$. The coefficients, $\mathbf{u} = (u, v, w)$, are called *barycentric coordinates* of $\mathbf{p}$ with respect to $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$. If four points $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$ and $\mathbf{p}$ are given, we can find the barycentric coordinates of $\mathbf{p}$ by Cramer's rule:

$$u = \frac{\text{area}(\mathbf{p}, \mathbf{b}, \mathbf{c})}{\text{area}(\mathbf{a}, \mathbf{b}, \mathbf{c})}, \quad v = \frac{\text{area}(\mathbf{a}, \mathbf{p}, \mathbf{c})}{\text{area}(\mathbf{a}, \mathbf{b}, \mathbf{c})}, \quad w = \frac{\text{area}(\mathbf{a}, \mathbf{b}, \mathbf{p})}{\text{area}(\mathbf{a}, \mathbf{b}, \mathbf{c})} \tag{B.1}$$

The Cramer's rule makes use of determinants and they are related to the areas by the identity:

$$\text{area}(\mathbf{a}, \mathbf{b}, \mathbf{c}) = \frac{1}{2} \begin{vmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \\ 1 & 1 & 1 \end{vmatrix} \tag{B.2}$$

# APPENDIX C
# DATA STRUCTURES FOR INTERFACE TRACKING

## Node Data Structure

- Index - a unique number for every node.
- Coordinates - $x$, $y$ and $z$ coordinates of the node.
- Curvature - value of mean curvature at the node location.
- Normal - a unit vector normal to the surface at the node location.
- Velocity - a vector for the velocity of the node.
- EdgeList - list of edges meeting at the node.
- TriangleList - list of triangles meeting at the node.

## Edge Data Structure

- Index - a unique number for every edge.
- Node[2] - nodes of this edge.
- Triangle[2] - triangles sharing this edge.
- EdgeList[2] - lists of edges storing the other edges (in order) of the two triangles of this edge.

- Direction[2] - values to indicate whether the direction of motion in the two triangles is correct or reverse as we move from the first to the second node of this edge.

- State - a number that indicates whether the edge is "alive", "dormant" or "dead" depending on the number of triangles (two, one or zero respectively) this edge is part of.

**Triangle Data Structure**

- Index - a unique number for every triangle.

- Area - area of this triangle.

- FirstEdge - an edge that is a handle to the nodes and other edges of this triangle.

**Surface Mesh Data Structure**

- NumberOfNodes - total number of nodes in the surface mesh.

- NumberOfEdges - total number of edges in the surface mesh.

- NumberOfTriangles - total number of triangles in the surface mesh.

- NodeList - list of all the nodes in the surface mesh.

- EdgeList - list of all the edges in the surface mesh.

- TriangleList - list of all the triangles in the surface mesh.

- MaximumArea - area criteria value beyond which a triangle is bisected into smaller triangles.

- BadEdgeList - list of edges to be swapped through the diagonal swap operation.
- BigTriangleList - list of triangles undergoing the Bowyer-Watson point insertion algorithm.

## REFERENCES


[1] W. S. Shyy, H. S. Udaykumar, M. M. Rao and R. W. Smith, *Computational fluid dynamics with moving boundaries*, Taylor and Francis, Washington, DC (1996).

[2] J. M. Floryan and H. Rasmussen, Numerical methods for viscous flows with moving boundaries, *Applied Mechanics Review* **42**, 323-341 (1989).

[3] S. Osher and J. Sethian, Fronts propagating with curvature dependent speed: Algorithms based on Hamilton-Jacobi formulations, *Journal of Computational Physics* **79**, 12-49 (1988).

[4] J. A. Sethian, *Level set methods: Evolving interfaces in geometry, fluid mechanics, computer vision and materials science*, Cambridge University Press, Cambridge (1996).

[5] C. W. Hirt and B. D. Nichols, Volume of fluid (VOF) method for the dynamics of free boundaries, *Journal of Computational Physics* **39**, 201-225 (1981).

[6] F. H. Harlow and J. E. Welch, Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface, *Physics of Fluids* **8**(12), 2182-2189 (1965).

[7] S. O. Unverdi and G. Tryggvason, A front tracking method for viscous, incompressible, multi-fluid flows, *Journal of Computational Physics* **100**, 25-37 (1992).

[8] H. S. Udaykumar, W. S. Shyy and M. M. Rao, ELAFINT: A mixed Eulerian-Lagrangian method for fluid flows with complex and moving boundaries, *International Journal of Numerical Methods in Fluids* **22**(8), 691-712 (1996).

[9] H. S. Udaykumar and W. S. Shyy, Simulation of morphological instabilities during solidification; Part I: Conduction and capillary effects, *International Journal of Heat and Mass Transfer* **38**, 2057-2073 (1995).

[10] C. S. Peskin, Numerical analysis of blood flow in the heart, *Journal of Computational Physics* **25**, 220-252 (1977).

[11] P. L. George, *Automatic mesh generation: Application to finite element methods*, John Wiley & Sons (1991).

[12] G. Farin, *Curves and surfaces for computer aided geometric design*, Academic Press, New York, $4^{th}$ edition (1997).

[13] L. Piegl and W. Tiller, *The NURBS book, Monographs in Visual Communication*, Springer Verlag, New York, $2^{nd}$ edition (1997).

[14] V. Jayaraman, H. S. Udaykumar and W. S. Shyy, Adaptive unstructured grid for three-dimensional interface representation, *Numerical Heat Transfer*, Part B **32**, 247-265 (1997).

[15] J. Peters, $C^1$ surface splines, *SIAM Journal of Numerical Analysis* **32**(2), 645-666 (1993).

[16] W. Boehm, G. Farin and J. Kahmann, A survey of curve and surface methods in CAGD, *Computer Aided Geometric Design* **1**, 1-60 (1984).

[17] J. Peters, Constructing $C^1$ surfaces of arbitrary topology using biquadratic and bicubic splines, in *Designing fair curves and surfaces*, N. Sapidis (ed.), SIAM, Philadelphia (1994).

[18] J. Peters, Smooth free-form surfaces over irregular meshes generalizing quadratic splines, *Computer Aided Geometric Design* **10**, 347-361 (1993).

[19] A. Bowyer, Computing Dirichlet tesselations, *The Computer Journal* **24**(2), 162-166 (1981).

[20] D. F. Watson, Computing the $n$-dimensional Delaunay tesselation with application to Voronoi polytopes, *The Computer Journal* **24**(2), 167-172 (1981).

[21] N. P. Weatherill, Delaunay triangulation in computational fluid dynamics, *Computers and Mathematics with Applications* **24**(5/6), 129-150 (1992).

[22] P. H. Todd and R. J. Y. McLeod, Numerical estimation of the curvature of surfaces, *Computer Aided Design Journal* **18**(1), 33-37 (1986).

[23] D. J. Struik, *Lectures on classical differential geometry*, Addison-Wesley, Cambridge, MA (1950).

[24] T. J. Willmore, *An introduction to differential geometry*, Oxford University Press, New York (1959).

## BIOGRAPHICAL SKETCH

Abhishek Khanna was born in 1976 in New Delhi, India, where he spent his entire childhood. He did his schooling from Don Bosco School, Delhi, and Delhi Public School, R. K. Puram, New Delhi. He received his bachelor's degree in mechanical engineering from the Indian Institute of Technology, Bombay. After graduation, he came to the United States of America to pursue a master of science degree in the field of engineering mechanics at the University of Florida. As part of his thesis work, he has been developing a three-dimensional tracker code used for tracking interfaces in moving boundary problems.