2-D, 3-D and 4-D Anisotropic Mesh Adaptation for the Time-Continuous Space-Time Finite Element Method with Applications to the Incompressible Navier-Stokes Equations

by

Pascal Tremblay

A thesis presented to the
Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the
requirement for the degree of

DOCTOR OF PHILOSOPHY

in

MECHANICAL ENGINEERING

Ottawa-Carleton Institute for Mechanical and Aerospace Engineering
Department of Mechanical Engineering
University of Ottawa
Ottawa, Ontario, Canada

December 2007

Abstract

A mesh adaptation strategy suitable for unsteady partial differential equations has been developed to control both the spatial and temporal discretization errors in a unified fashion. The aims are to provide a methodology that prevents the accumulation of discretization error associated with time stepping approaches and is also flexible enough to adjust the density of the space-time mesh to varying time scales in the solution domain.

The primary focus of this thesis has been the development of anisotropic meshing algorithms that can operate in 2-D, 3-D and 4-D on unstructured simplicial meshes. The mesh modification operators include edge splitting, edge collapsing, simulated edge swapping, and mesh smoothing and are driven by an anisotropic metric field.

The mesh adaptation methodology has been coupled with a time-continuous space-time finite element flow solver for the incompressible Navier-Stokes equations. The space and time finite element discretizations have been treated in a fully coupled manner using a Galerkin/Least-Squares formulation on a simplicial mesh that covers the entire space-time solution domain. The anisotropic metric field governing the mesh modification algorithms is constructed from an interpolation based error estimate using a modified Hessian of the magnitude of the velocity in the flow field. It provides a specification of the desired mesh size and orientation for the simplicial elements to refine and coarsen the space-time mesh while stretching the elements in preferred directions to reduce the number of mesh points necessary to achieve a solution of a given accuracy.

The anisotropic meshing algorithms have been tested in 2-D, 3-D and 4-D with an analytical metric field and also with a simple heat transfer problem. The resulting element

quality was found to be very high for the 2-D cases, comparable to those produced by methods found in the literature for the 3-D cases, but unsatisfactory for the 4-D cases. The ratio for the number of elements to the number of points in the mesh has been found to grow by a factor of about 3 when increasing the space dimension by one. To the best of our knowledge, this is the first time that mesh modifications were shown to operate in a dimension higher than 3 with the ability to modify the boundary mesh. In contrast, previously existing methods that operate on higher dimensional meshes cannot keep track of the boundary of the domain.

Verifications for the unified space-time adaptive finite element method have been done using manufactured solutions for a linear heat equation and for the incompressible Navier-Stokes equations. The behaviour of the L^2 norm, computed on the entire space-time domain, shows a good agreement between the numerical and the analytical solutions indicating that the unsteady mesh adaptation procedure can control the discretization error in both space and time.

Applications to the incompressible Navier-Stokes problems have been shown with unsteady 2-D flows to demonstrate the ability of the method. Numerical solutions are presented for the flow past a circular cylinder at a Reynolds number of 100, the flow over a backward facing step at a Reynolds number of 800 and the flow in a lid-driven cavity at a Reynolds number of 400. For these test cases, the Picard method with the combined mesh adaptation strategy and solution interpolation, introduced to provide a restart solution for the solver after mesh adaptation, exhibit excellent convergence behaviour.

Acknowledgements

I would like to thank Professor Stavros Tavoularis for serving as my Thesis supervisor and Professor Yves Bourgault for serving as my Thesis co-supervisor. They gave me the leeway to explore my ideas on space-time mesh adaptation and offered me the opportunity to live my passion for numerical simulation in the context of a PhD Thesis.

I am grateful for all the financial support that I have received during the course of my Thesis. More specifically, my thanks go to the Government of Quebec for their funding through the "Fonds Québécois de la Recherche sur la Nature et les Technologies", the Government of Ontario through the Ontario Graduate Scholarship, the Faculty of Graduate and Postdoctoral Studies of the University of Ottawa, the Natural Sciences and Engineering Research Council of Canada, Professor Tavoularis and Professor Bourgault, the National Bank of Canada and my parents.

I am also grateful for the free availability of the Visual Toolkit (VTK), the Portable, Extensible Toolkit for Scientific Computation (PETSc) and the CFD General Notation System (CGNS). The use of these libraries has allowed me to focus on other aspects of this Thesis. Availability of such open source codes makes it possible for people in the academia to explore original concepts on numerical simulations in a way that would otherwise be very difficult in view of the commercial simulation packages that are now available.

I would also like to thank the students with whom I have shared the adventure of graduate studies over these years. My thoughts go to Matthew Doyle, Warren Dunn, Dong Il Chang and several others whom I have met in conferences and who inspired me by sharing their ideas.

Finally, I would like to express my deep appreciation to my family for their unconditional financial and moral support and for encouraging me to persevere to complete this Thesis.

Contents

List of Figures

List of Tables

List of Algorithms

Nomenclature

$\mathbf{A}$	matrix
a	angle between two vectors
$\mathbf{b}$	right hand side vector
C	curvature
$\widetilde{C_i}$	average curvature for the face i
C^0	piecewise continuous functions
c_0	constant
$C\left(t\right)$	time function for manufactured flow solution
D	dilatation matrix
d	space dimension
$\det\left(M\right)$	determinant of the metric tensor M
$\mathbf{e}$	eigenvector
$\mathbf{f}$	body force vector (per unit mass)
F_{ran}	random factor with a real value in the interval $[0, 1]$
$\mathbf{g}$	values of a Dirichlet boundary condition
$\dot{g}$	rate at which energy is generated per unit volume of a medium
G_0	constant for manufactured solution in a heat equation
$H\left(\eta\right)$	Hessian matrix
H_{ij}	the component of the Hessian matrix at row i and column j
H_{ij}^k	discrete approximation of the Hessian at a vertex k
$H^m\left(\Omega\right)$	Sobolev space of functions
H^{1h}	finite dimensional subspace
$\mathbf{h}$	values of a Neumann boundary condition
h	element mesh size
h_e	element mesh size for stabilization parameters in the GLS method
$\mathbf{I}$	identity matrix
K	mesh element
L_{avg}	average Euclidean length of edges incident to a point
$L^\infty\left(\Omega\right)$	space of finite functions
$L_m(P, Q)$	length of a mesh edge measured in a Riemannian space
$L^p\left(\Omega\right)$	space for functions that are L^p-integrable
M	metric tensor
M_{L^p}	metric tensor weighted with an L^p norm related factor
N	linear transformation matrix; also number of time intervals

N_e	number of edges of the element
N_{elem}	number of elements
N_I	number of subdivision intervals along a coordinate axis
N_p	number of points in a mesh
$\mathbf{n}$	unit normal
n	topological dimension of an element
n_{el}	number of element in the finite element mesh
O	origine of coordinate system
$O\left(n^p\right)$	time complexity of order p for an input of size n
P_i	point in a Cartesian coordinate system
P_m	polynomial of order m
p	pressure
p_h	discrete approximation of the pressure
p_j	pressure at node j of the element
Q	element quality
q_h	test functions for pressure
R	rotation matrix
$\Re$	space for real numbers
$\Re^d$	Euclidean d-dimensional space
Re	Reynolds number
r	inscribed radius of an element
S_i	sub-volume for the surface of a face i of a simplex
S_n^e	space-time element
S_n	time slab of a space-time domain
S_p^h	space for trial functions for pressure
S_u^h	space for trial functions for velocities
T	temperature
T_h	finite element mesh; also discrete approximation of temperature
T_i	triangle element i
t	barycentric coordinate; also Cartesian coordinates
t_n	time at interval n
$\mathbf{U}$	solution vector
$\mathbf{U}^k$	solution vector at Picard iteration k
$\mathbf{u}$	velocity vector
$\mathbf{u}_0$	initial solution for the velocity in the space domain
$\mathbf{u}_h$	discrete approximation of the velocity vector
$\mathbf{u}_j$	velocity vector at node j of the element
$\mathbf{u}_+^n$	velocity at time $t = t_n$ on time slab (t_{n-1}, t_n)
$\mathbf{u}_-^n$	velocity at time $t = t_n$ on time slab (t_n, t_{n+1})
$u_{\max}$	maximum magnitude of the velocity
u_q	quadratic function used for inlet flow boundary condition
$u\left(x\right)$	velocity as a function of the coordinate x
V_E	volume of an element in the Euclidean space
$V_{UnitEdge}$	volume of an element with edges of unit length

V_K^M	volume of an element in a Riemannian space
V_u^h	space for test functions for velocities
V_p^h	space for test functions for pressure
$\mathbf{v}_h$	test functions for velocity
v_h	continuous piecewise linear functions
v_h^k	piecewise linear finite element hat function at vertex k
$\mathbf{x}$	vector of coordinates for a mesh point
$\mathbf{x}_c$	inscribed ellipsoid center
$\mathbf{x}_{opt}$	optimal point position
$\mathbf{x}_{prev}$	previous point position
$\mathbf{x}_{trial}$	trial point position
$(x_i)_{prev}$	previous value for point coordinate i
$(x_i)_{\text{rand}}$	randomized location for point coordinate i
x, y, z, t	Cartesian coordinates

Greek Symbols

α	degree of anisotropy; also thermal diffusivity
β	constant for boundary condition in lid-driven cavity
Γ	boundary of the space domain
Γ_D	boundary of the space domain with Dirichlet boundary conditions
Γ_N	boundary of the space domain with Neumann boundary conditions
Δt	length of a space-time element along the time axis
ε	finite element interpolation error
$\boldsymbol{\varepsilon}(\mathbf{u})$	viscous stress tensor
η	exact solution
η_h	numerical approximation of the solution
λ	eigenvalue
μ	shear viscosity of the fluid
ν	kinematic viscosity
ρ	radius of curvature; also density
$\boldsymbol{\sigma}$	total stress tensor
τ	time period
τ_{cont}, τ_{mom}	stabilization parameters for the GLS method
ϕ_j	interpolation functions for pressure
φ^h	interpolation functions
ψ_j	interpolation functions for velocity
Ω	space domain
Ω_0	space domain at time $t = t_0$
Ω_e	element of the the finite element mesh T_h
$\partial\Omega$	boundary of the space domain
ϖ	relaxation factor in the point relocation equation

Other Symbols

2-D	two-dimensional

3-D	three-dimensional
4-D	four-dimensional
$\|u\|$	absolute value
$\|\|u\|\|$	norm of a vector
$\|\|u\|\|_0$	norm in $L^2(\Omega)$
$\|\|u\|\|_1$	norm in $H^1(\Omega)$
$\|\|u\|\|_\infty$	L^∞ norm
$\|\|u\|\|_{L^p}$	L^p norm
(u, v)	inner product in $L^2(\Omega)$
$(u, v)_1$	inner product in $H^1(\Omega)$

Acronyms

AIAA	american institute of aeronautics and astronautics
ALE	arbitrary Lagrangian-Eulerian
BC	boundary conditions
CAD	computer aided design
CFD	computational fluid dynamics
CGNS	CFD general notation systems
CPU	central processing unit
DSD/ST	deforming-spatial-domain / space-time
FDM	finite difference method
FEM	finite element method
FVM	finite volume method
GB	gigabyte
GHz	gigahertz
GLS	Galerkin / least-square formulations
GMRES	generalized minimum residual method
LBB	Ladyzhenskaya-Babuska-Brezzi
MB	megabyte
PDE	partial differential equations
PETSc	portable, extensible toolkit for scientific computation
PLC	piecewise linear complex
RAM	random access memory
STK	simulation toolkit
STL	stereolithography file format
ST_{mom}	stabilization terms for the momentum equations
ST_{cont}	stabilization terms for the continuity equation
TCSTFEM	time-continuous space-time finite element method
VTK	visualization toolkit

Chapter 1

Introduction

1.1 Perspective on the Evolution of CFD

Over the past three decades, CFD has evolved to handle problems of increasing complexity. Numerical methods, including the finite difference (FDM), finite volume (FVM) and finite element (FEM) methods, were first developed to handle flows in fixed geometries. For these methods, the temporal derivatives of the flow variables in the unsteady Navier-Stokes or Euler equations are discretized using finite differences, which, for the FEM and FVM, introduces differences between the discretization techniques in space and time.

The extension of these methods to simulations of flows with moving boundaries has been achieved through the deformation of the mesh to accommodate the moving geometry from one time step to the next. This required the modification of the FDM, FVM and FEM in order to account for the effects of boundary changes on the mass and momentum balances. One of the first methods that have been developed to address this is the Arbitrary Lagrangian-Eulerian (ALE) formulation (Donea et al. (1977), Belytschko et al. (1978),

Hughes et al. (1981)).

As one strives to obtain numerical solutions for problems of increasing complexity, it is also necessary to quantify and control the accuracy of the numerical solution. Mesh adaptation schemes addressing this accuracy have been developed over the past twenty-five years with varying degrees of success (George and Borouchaki (1997), Thompson et al. (1999)). Unfortunately, the vast majority of these schemes are limited to controlling the accuracy of steady solutions in fixed geometries.

In the case of changing geometries, the ALE approach does not allow the connectivity of the mesh to be modified from one time step to the next. Thus, once the discretization approach has been chosen, the discretization error in time can only be reduced by reducing the size of the time step. Although the discretization error in space can be reduced by relocating the nodes, the priority for node movement in this context has to be given to adjusting the mesh to follow the changes in geometry as the solution procedure advances in time. This necessarily limits the possible discretization error reduction that can be accomplished by node movement.

In recent years, a few mesh adaptation strategies have been extended to unsteady problems (Alauzet et al. (2003), Remacle et al. (2005), Alauzet et al. (2007)). In general, they approach unsteady problems by combining a time stepping approach with adapting the space mesh and the size of the time step using the solution computed for the next time step. In the work of Alauzet et al. (2007), the mesh is adapted at the current time step, taking into account the interpolation error in both space and time, until the relative difference between the solution at the current time step and the next time step measure in

the L^1 norm is below a given threshold value. Once the mesh and the solution for the next time step are considered of sufficient accuracy, then the solution procedure advances to the subsequent time step.

Such an approach to mesh adaptation for unsteady problems can greatly increase the robustness and the accuracy of the solution procedure by reducing the discretization error at each time step. However, this cannot prevent entirely the accumulation of the discretization error from one time step to the next, because the discretization error that occurs at one time step is outside the mesh adaptation loop for the subsequent time step. Consequently, the temporal discretization error may accumulate as the solution procedure advances in time if the transient solution varies significantly over one time step.

1.2 Space-Time Formulations

Instead of using a finite difference scheme to discretize the temporal derivatives in the Navier-Stokes equations, a number of authors have presented unified finite element (Shakib et al. (1991), Masud and Hughes (1997), Hand and Lu (1999), N'dri (2001), Pontaza and Reddy (2004), Réthoré et al. (2005)) or finite volume formulations (Zwart et al. (1999)), which are collectively known as space-time formulations. These formulations possess many known interesting properties. They utilize the same formulation for the solution of steady or unsteady problems in fixed domains, or in domains with moving boundaries (N'dri (2001)) and even domains with topological changes (Zwart et al. (1999)).

Available space-time solvers advance in time by extruding the 2-D mesh into time by consecutive discrete amounts equal to the time-step (Jamet (1978), Shakib et al. (1991),

Zwart et al. (1999)) . At the junctions between two layers of elements extruded in time (time slabs), the nodes of the mesh do not need to match one to one. For these reasons, these are known as time-discontinuous space-time formulations (N'dri (2001)).

From the perspective of mesh adaptation, the flexibility allowed by time-discontinuous space-time formulations to control the discretization error in time is limited. For time slabs composed of extruded elements, the mesh is essentially structured in time, but, even if the extruded elements are divided into simplicial elements, once a discretization approach is chosen, the only way to significantly reduce the discretization error in time is to reduce the thickness of the time slabs, which is equivalent to reducing the size of the time-step used in more conventional approaches.

Some methods use the alternative approach of increasing the order of the polynomial basis used in the discretization method; these are generally referred to as p-methods. In this approach, for a given time-slab of a fixed time period, the discretization error can be reduced by some amount by increasing the order of the polynomial basis. Even with p-methods, one still needs to address the issue of connecting these high order polynomial bases between consecutive time-slabs to minimize the discretization error and also that of selecting the time step size. Although consideration of p-methods is beyond the scope of this thesis, it is worth mentioning that they could be combined with space-time formulations along with mesh refinement or coarsening (h-methods). As the accuracy of p-methods depends not only on the order of the method, but also on the density and distribution of mesh points, they would also benefit from mesh adaptation based on h-methods. In the light of the above, the choice was made in this thesis to focus on h-methods, anticipating

that most of the meshing technology developed here can be transferred to higher order methods, provided that a suitable error estimator is employed.

Time-adaptive procedures for space-time formulations are rather rare, but a few such strategies have been found (Li and Wiberg (1998), Karakashian and Makridakis (1999), Feng and Peric (2001), Feng and Peric (2003), Cascon et al. (2006)). Feng and Peric mention the potential for combining spatial mesh adaptation with their time-adaptive strategy, but they did not pursue this approach. Li and Wiberg are the only previous authors who have combined both approaches using essentially two error estimators, one for the discretization error in space and another for the discretization error in time. In this approach, a solution is first calculated for a given time slab and, if the error in space is too large, the space mesh is adapted, in a manner similar to that for a steady problem. At the same time, an error estimator in time is applied to determine whether the thickness of the time slab should be reduced to reduce the discretization error or increased to reduce the computational cost while maintaining an accuracy considered to be reasonable. Once the time slab size is adjusted, the solution is recomputed at this time step until the error estimator criterion is satisfied. Of course, there is an added computational cost to repeating the computation of the solution for several iterations at the same time step, but this can still be beneficial if the initial time steps are chosen to be large and only reduced as required by the error estimator rather than using very small time steps to begin with. These approaches are similar to an adaptive time stepping approach (Alauzet et al. (2007)), but with the added benefit of being able to handle domains with moving boundaries without further modifications.

In view of the above discussion, it becomes evident that complete control of all

sources of discretization error in unsteady flow solutions can only be achieved by devising a method of mesh adaptation suitable for both spatial and temporal formulations. A possible approach is to extend existing mesh adaptation techniques in steady, 3-D flows, by treating time as a fourth dimension and applying an unstructured 4-D mesh to the continuous space-time domain (Tremblay et al. (2003), Pontaza and Reddy (2004), Alam et al. (2006)).

Traditionally, time-discontinuous space-time formulations have been favoured over time-continuous space-time formulations, because the former have lower memory requirements as they treat only one layer of elements at a time, whereas the latter treat the entire space-time domain as one entity (N'dri (2001)). However, when combined with mesh adaptation, solving only one layer of elements at a time prevents the full benefits of mesh adaptation, because the discretization error in time cannot be fully controlled and invariably accumulates as the solution progresses from one time step to the next (Alam et al. (2006)).

An improved approach would be to apply mesh adaptation on completely unstructured 4-D space-time domains in a manner similar to those applied to steady flows in 2-D and 3-D space domains (Vallet (1992), Castro-Díaz et al. (1995), Castro-Diaz et al. (1997), Buscaglia and Dari (1997), Tam et al. (1998), Tam et al. (2000), Fortin et al. (2000), Dompierre et al. (2002), Du et al. (2005)). This approach would permit the computation of time-accurate numerical solutions (see the following definition 1), for geometries with prescribed boundary motions for which the space-time mesh can be refined at any point in the space-time domain. Hence, the resolution of the mesh can be increased not only in space, but in time as well. Similarly, the gain achieved by coarsening the mesh where the

error estimator indicates that the error is lower than in other regions can be also extended with regards to time. This benefit cannot be achieved by standard methods or by time-discontinuous space-time methods, in which each node in the space mesh is duplicated at each time step regardless of whether the solution changes significantly in time or not. In the time-continuous formulation, it is not necessary to maintain a small time step in the entire space-time domain, because the time-step size is adapted locally to the largest value that will produce a solution of a desired accuracy. This method is expected to result in significant savings for the solution of problems in which the characteristic time-scales of various physical parameters vary widely in the space domain.

Definition 1 (TIME-ACCURATE NUMERICAL SOLUTION) *A time-accurate numerical solution of a system of partial differential equations with mathematically appropriate boundary conditions can be defined as a numerical solution for which the difference with the exact solution measured in a suitable norm is bound by a finite constant whose maximum value in the space domain does not increase as function of time.*

A time-continuous space-time formulation requires the computation of all values of the unknown variables in the entire space-time domain rather than separately for each time slab. This may require very large memory capabilities to solve some problems. To reduce memory and computational time requirements, it is possible to combine this approach with a time-discontinuous formulation. The combined method would still utilize time slabs, but each slab would be completely unstructured and would permit mesh refinement so that it could locally have more than one elements across its thickness. The solution would then be re-interpolated from one time slab to the next using existing approaches for time-

discontinuous space-time methods.

Time-continuous space-time formulations are rare in the literature, but a few authors have investigated them in 2-D or 3-D space-time domains for the heat equation (French (1999)), wave equation (French and Peterson (1996)), shallow water equations (Chippada et al. (1998)), nonlinear hyperbolic equations (Chen and Huang (2001)), nonlinear Schrödinger equation (Karakashian and Makridakis (1999)) and a pure advection equation (Perrochet and Azérad (1995)). Pontaza and Reddy (2004) have recently presented results for the incompressible Navier-Stokes equations, comparing a fully coupled space-time formulation (here referred to as a time-continuous space-time formulation), with a space-time formulation using space-time slabs. Based on the few 2-D cases studied, which required 3-D space-time slabs or space-time meshes, these authors concluded that a fully coupled approach is more robust and computationally cost effective compared to a time-slab formulation (Pontaza and Reddy (2004)). Although they did not combine their fully coupled space-time formulation with an error estimator and mesh adaptation procedure, they clearly indicated the possibility of using existing error estimators to control the discretization errors both in space and in time in a unified fashion.

Recognizing the potential benefit of a fully coupled adaptive space-time approach in controlling the temporal discretization error, Alam et al. have developed an adaptive wavelet method for parabolic differential equations that operates on the entire space-time domain simultaneously (Alam et al. (2006)). Although wavelet methods use a different approach to discretize PDE than that used by FEM, it is important to highlight that the idea of using a unified space-time approach is increasingly receiving more attention in several

fields, including the solution of structural problems (Hand and Lu (1999)) and the FEM in general (Réthoré et al. (2005)).

1.3 Motivation and Scope of the Present Study

The motivation for the present study stems from the need to improve the ability of the finite element method to solve unsteady partial differential equations in fixed or changing domains in a time-accurate manner. The range of applications in engineering is very wide, because the FEM has been applied to many types of problems in fluid flow, heat transfer, stress analysis, magnetic fields and quantum mechanics. Of particular interest to mechanical engineers are flows in fluid machinery, including reciprocating, rotary and turbo-machines. Another type of applications of particular interest are blood flows in the cardiovascular system and various prosthetic devices. The recent development of noninvasive medical imaging method, combined with image processing techniques has permitted the reconstruction of 3-D surfaces moving in time (Bonciu et al. (1998), Delibasis et al. (1999), Mcinerney and Terzopoulos (1995), Reinhardt et al. (2000)). Currently, these images are mainly used to visually detect anomalies in the heart, but some projects under development (Abdoulaev et al. (1998)) have focused on the development of virtual reality tools to eventually assist physicians and surgeons in the refinement of surgical procedures and in predicting the effects of surgical interventions (Leval et al. (1996), Migliavacca et al. (1996)). The accuracy of the numerical simulation of hemodynamic flows is important, because it is known that flow features such as separation, recirculation and oscillatory shear stresses play an important role in the development of arterial diseases (Deplano and Siouffi

(1999), Perktold et al. (1998)).

As a first step in the development of this novel approach, the present work will be limited to laminar flows of incompressible fluids and the emphasis will be on the development and validation of the methodology. Nevertheless, the exploration of adaptive unified space-time solution procedures extends to the finite element method in general and can be also applied to other discretization techniques, such as the FVM and wavelet methods (Alam et al. (2006)).

1.4 Objectives and Outline of the Thesis

The primary objective of this thesis has been the development of anisotropic meshing algorithms that can be operated on unstructured simplicial meshes in 2-D, 3-D and 4-D. 4-D anisotropic mesh adaptation makes it possible to extend the benefit of fully coupled space-time formulations to 3-D space domains, while controlling and reducing the discretization error both in space and in time in a unified fashion. This opens up the possibility of exploring the benefits of mesh adaptation for unsteady problems on domains with moving boundaries.

Chapter 2 introduces background information concerning mesh adaptation, highlights basic mesh definition terminologies and more formally presents the objectives of a mesh adaptation procedure. It also presents the rationale for the selection of the Hessian based error estimator used it this thesis, as research on this topic is considered outside the scope of the present study. A brief review of existing anisotropic meshing algorithms is presented with a special emphasis on meshing algorithms that have been extended to higher

dimensions. It is emphasized that existing meshing algorithms that operate in dimensions higher than three are limited to handling meshes with volume meshes only and do not have the ability to manipulate and preserve a boundary mesh in contrast to the method presented in this thesis.

Chapter 3 presents the data-structure developed from the ground up to allow the implementation of meshing algorithms in a dimension independent manner. Although it is a simple extension of concepts found in existing data structures, it is important in the current context because the development of meshing algorithms in higher dimensions is facilitated by implementing them first in 2-D and in 3-D before applying them to 4-D.

Considering that even the most basic engineering simulations require the representation of solution domains that have curved boundaries, such as fluid flow in a circular tube, Chapter 4 presents a simple approach to reconstructing a curved geometry made of second order polynomials using only a surface mesh of triangular elements as an input. Such triangular mesh inputs are frequently used in numerical simulations of medical processes, in which medical images are processed to extract an approximate representation of the boundary of the solution domain. The method presented in this Chapter is simple and can be implemented with a finite element mesh data structure, as long as this structure supports quadratic elements. Although this is a secondary part of the thesis, it is of practical interest for a FEM adaptive solver, because it allows it to modify the mesh while preserving a reasonably close fit to the initial geometry without having to link the FEM solver with a CAD package.

Chapter 5 presents the incompressible fluid flow solver implemented using a Galerkin-

Least-Squares space-time formulation. It is a rather simple classical incompressible flow solver using linear interpolation basis functions and a Picard method, but exposes the salient characteristics of using a fully coupled space-time formulation to handle unsteady fluid flows. It has been fully embedded with the mesh adaptation procedure to form an automatic adaptive space-time FEM procedure.

Chapter 6 presents the anisotropic meshing algorithms developed to operate on 2-D, 3-D and 4-D simplicial meshes. It is a direct extension of an existing edge-based approach that combines edge refinement, edge collapsing, edge swapping and edge smoothing. However, to enable it to operate in higher dimensions, both inside the mesh domain and on its boundaries, an edge swapping algorithm is implemented using a combination of edge splitting and edge collapsing that simulates the effect of classical edge swapping algorithms. A novel mesh smoothing algorithm based on the inscribed ellipsoid is also presented and integrated with the mesh optimization procedure. To the best of the author's knowledge, this is the first extension of such mesh modification algorithms to higher dimensions.

Chapter 7 briefly presents the iterative procedure that combines the solution computation with the FEM solver, the estimation of the discretization error, the mesh adaptation and the interpolation of the previous solution onto the adapted mesh. The fully coupled space-time adaptive procedure presented is functionally the same as those used in previous mesh adaptation procedures for steady problems, except that the present one operates on a fully unstructured space-time mesh that comprises the entire solution domain including the time axis.

Chapter 8 presents the numerical simulations that were performed to verify and

validate the proposed procedure. First, an analytical metric field is used to test the anisotropic mesh optimization procedure on 2-D, 3-D and 4-D meshes on a simple unit box domain. Second, test cases are included for the solution of the linear heat equation, also for 2-D, 3-D and 4-D unit box domains. Third, some classical problems are solved for the unsteady incompressible Navier-Stokes equations using a 3-D mesh. Some test cases include a comparison with analytical solutions to measure the ability of the space-time method to reduce the discretization error in both space and time.

Finally, Chapter 9 presents the conclusions of the present work and briefly highlights possible future research work on unified space-time mesh adaptation methods.

Chapter 2

Background and Literature Review

2.1 Introduction

This chapter presents the basic definitions and terminology that are necessary to discuss in more detail mesh adaptation methodologies. Basic mesh definitions for the FEM are first presented, followed by the objectives of mesh adaptation. A special emphasis is placed on the requirements for mesh adaptation, because these will be used as guiding criteria to select, among the algorithms available in the mesh algorithm literature, the ones that are best suited to form the basis of a mesh adaptation code in four dimensions.

Classical mesh generation and mesh adaptation methods have been described in detail by several authors (George and Borouchaki (1997), Frey and George (1999), Thompson et al. (1999), George (2001)). Therefore, the following literature review will be focused on specific points of interest rather than on general approaches.

2.2 Basic Mesh Definitions

2.2.1 Solution Domain and Boundaries

The finite element method is a method that seeks a numerical solution to a mathematical model expressed in the form of a system of partial differential equations (PDE) with suitable boundary conditions (Cook et al. (2002)). The problem formulation starts by defining a *geometry*, which is the shape of an object. The solution domain is generally bounded by this geometry, on the surface of which boundary conditions are specified. In CFD, the geometry is usually represented as a surface without thickness and the solution extends only in the domain occupied by the fluid. In stress analysis and heat transfer, the solution domain may also include solid geometries with a thickness.

2.2.2 Mesh Points and Elements

The FEM seeks a discrete solution to the PDE in the solution domain and requires that the domain and its boundaries be divided into sub-regions, which are called elements. Inside the elements, the dependent variables of interest vary according to simple functions, most frequently linear or quadratic polynomials. The unknown values of the dependent variables are defined at the nodes of the elements. Nodes may be defined both on the boundary of an element and in its interior.

A finite element mesh is a subdivision of the solution domain in which the arrangement of elements covers the entire domain and has no overlapping elements; furthermore, adjacent elements are connected to the nodes they share across an adjacent face.

Consider a general FEM mesh in d-dimensions ($d = 2, 3, 4$). The elements that

compose the sub-regions of this mesh delimit a closed volume using curved or planar sub-surfaces. Curved surfaces made of quadratic polynomials are most often used in stress analysis, but not in CFD. The simplest element, called *simplex*, is defined using $d + 1$ points, which is the minimum number of points that are necessary to specify a volume in a d-dimensional space using plane faces (George (2001)).

2.3 Mesh Adaptation Definitions and Objectives

Because the numerical solution is not known at the time of mesh generation, the choice of initial mesh density is essentially arbitrary (Habashi et al. (1996), Dompierre et al. (2002)) and left to the discretion of the user of the mesh generation code to the degree permissible by the software package. Mesh adaptation methods were introduced as a means to automate this choice in order to control and minimize the discretization error.

In defining the objectives of mesh adaptation, it is important to specify the types of errors that are affected by this process. In numerical simulations, two types of errors can be distinguished: *modelling errors* and *numerical errors*. Modelling errors arise from the discrepancy between the mathematical model (the PDE and its boundary conditions) and the physical phenomena it attempts to represent. The numerical errors represent the difference between the approximate numerical solution of the PDE and the "exact" mathematical solution, which is usually unknown. The numerical errors can further be sub-divided into three main categories: the *discretization error*, the *iterative convergence error* and the *rounding error*. The rounding error is due to the approximate nature of floating point arithmetic on computers and is usually negligible compared to the other two (Wesseling (2001)).

The discretization error is due to the replacement of the partial differential equations with an algebraic system of equations. Finally, the iterative convergence error is due to inaccuracies in the solution of these algebraic equations by iterative methods (Wesseling (2001)). Mesh adaptation aims specifically at reducing and minimizing the discretization error and has also been found to reduce the convergence error (Tam et al. (1998)). Modelling and rounding errors are not affected by mesh adaptation.

2.3.1 Open-Loop versus Closed-Loop FEM Processes

From a control systems perspective, mesh adaptation can be viewed as the conversion of an open-loop simulation process to a closed-loop one. In the open-loop system, the mesh is considered to be the input, the FEM solver is the process and the solution is the output (Figure 2.1). In the closed-loop system, an error feedback, based on the solution, is used to modify the input mesh (Figure 2.2). Mesh adaptation has been clearly shown to reduce the discrepancy between the numerical solution and the "exact" solution for the Navier-Stokes equations (Habashi et al. (1998), Nithiarasu and Zienkiewicz (2000)). These benefits are not specific to CFD, but applicable to the FEM in general (Zienkiewicz and Zhu (1992)) and correspond to the reduced steady-state error of a transient closed-loop control system.

A closed-loop FEM process has an increased complexity associated with the mesh adaptation procedure and a potential increase in computational time, because the mesh adaptation process must be repeated. After each mesh adaptation cycle, the solution must also be re-computed on the new adapted mesh. On the other hand, the improved positioning of mesh points by the mesh adaptation procedure makes it possible to reduce the number

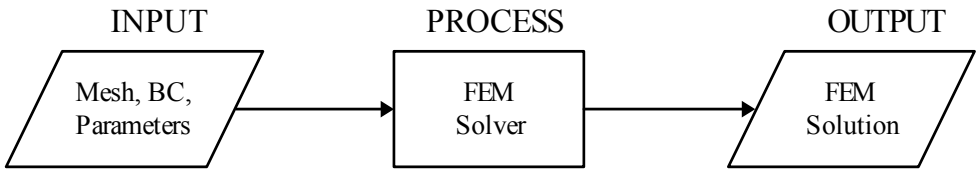

Figure 2.1: An open-loop control system representing the FEM without mesh adaptation.

of mesh points needed to obtain a solution of a desired accuracy, which partially offsets the additional cost of mesh adaptation (Tam et al. (1998)). It is also important to keep in mind that the Navier-Stokes equations are non-linear and require iterative algebraic solvers whose convergence depends on the proximity of the initial solution to the final one, which is unknown *a priori*. Mesh adaptation procedures usually include a re-interpolation of the previous solution, which was used to compute the error estimate, on the new adapted mesh to provide a re-start solution for the solver. If the mesh adaptation process is convergent, then at each cycle the computed solution comes closer to the exact solution. Consequently, the iterative solver tends to converge at a faster rate and to satisfy the convergence criteria in fewer iterations. The entire process also becomes more robust, because, even in cases in which the initial solution was too far from the exact one for the iterative solver to converge, that initial non-converged solution can still be used to adapt the mesh, which results in an increased density of the mesh, even before the desired solution is reached (Tam et al. (1998)). In order to understand fully the benefits and drawbacks of mesh adaptation, one should consider not only the mesh modification phase, but the closed-loop FEM process as a whole.

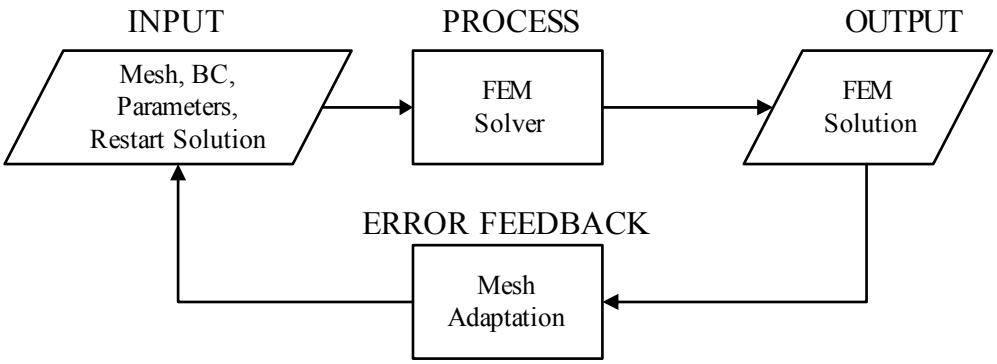

Figure 2.2: A closed-loop control system representing the FEM with mesh adaptation.

2.3.2 Objectives of Mesh Adaptation

In light of the above discussion, the objectives of mesh adaptation can be summarized as follows.

1. To improve the accuracy of the numerical solution by reducing the discretization error using an error feedback mechanism to adapt the mesh; closed-loop systems are known for their ability to control the steady-state error of the transient response of the control system, in sharp contrast to open-loop systems, which do not correct the discrepancy between the actual output and the desired output.

2. To eliminate the dependence of the numerical solution on the user's choice of the density and distribution of mesh points in the solution domain by making the process fully automated and driven by an error estimate.

3. To make the solver and adaptation process more robust; the convergence of iterative algebraic solvers has been found to improve when the mesh is adapted and a re-start solution is used (Tam et al. (1998)).

4. To reduce as much as possible the total computational time necessary to reach a numerical solution at a desired accuracy level; it is obvious that the added complexity and potential cost of mesh adaptation can only be justified if it produces better and faster solutions than indiscriminate mesh refinement would.

2.3.3 Adaptive Time Stepping vs. Mesh Adaptation

It is important to emphasize that, although mesh adaptation procedures have been very successful for steady problems, their applications to unsteady problems is still very limited. At its current state, the closest that mesh adaptation technology has come to handling unsteady problems has been to use adaptive time stepping combined with spatial mesh adaptation at each time step (Li and Wiberg (1998), Remacle et al. (2005), Alauzet et al. (2007)). Although this is certainly an improvement over fixed time stepping methods, these methods still suffer from the accumulation of the temporal discretization error in time because the errors committed at the previous time steps are outside of the error feedback loop (Figure 2.2). In practice, the solution extends over thousands, and even hundreds of thousands, of time steps and the cumulative error may become substantial. Mesh adaptation for steady problems has been shown to drastically improve the capturing of localized flow patterns, such as shock waves and boundary layers, even on meshes that were reasonably refined prior to mesh adaptation. Comparable improvements are to be expected by applying space-time mesh adaptation to unsteady flows with time-dependent localized flow patterns. In such cases, it appears necessary to quantify and control both the spatial and the temporal discretization errors.

2.4 Mesh Adaptation Requirements

2.4.1 Error Estimator

In contrast to many other control systems, the desired output response of the closed-loop FEM process is not known in advance. Hence, it is necessary to estimate the error before adapting the mesh.

Error estimators have been the subject of extensive research (Ainsworth and Oden (2000), George (2001)), so the purpose of this section is not to provide a complete review on the subject, but rather to present the reasoning behind the selection of an error estimator that is suitable for a space-time mesh adaptation procedure coupled to the time-continuous space-time finite element method (TCSTFEM).

Error estimators available in the literature can generally be classified in two broad categories: *residual-based error estimators* and *interpolation-based error estimators* (George (2001)). Residual-based error estimators depend on the residual of the partial differential equation being solved. Although they have a strong mathematical foundation, their direct dependence on the PDE results in coupling the mesh adaptation code with the FEM solver. Furthermore, they generally only provide information about the magnitude of the error in a certain region of the mesh without any directional information. For these reasons, these estimators are mostly limited to isotropic mesh adaptation.

In contrast, interpolation-based error estimators, in the context of FEM, depend only on the finite element interpolation function being used to construct an approximate solution to the PDE, the discrete values of that solution found at mesh points and the coordinates of the mesh points. Their dependence on the specific PDE being solved is

minimal. They usually operate on a scalar field, which changes for different PDE, but remains unchanged for finite element interpolation functions of a given order. This makes them easily applicable to a wide range of PDE, without the need to re-derive and implement the error estimator for each one. Furthermore, interpolation based-estimators have been used to derive anisotropic error estimates that provide not only information about the element size needed to bound the local element error, but also the directions in which the error increases the most and the least. That directional information about the error can be used to create anisotropic meshes with elements that are thin along the direction in which the error increases the most and elongated along the direction in which it increases the least, thereby reducing the number of mesh points needed to keep the error lower than the target level. Interpolation-based error estimators have already been generalized to higher dimensions (George (2001)) and are preferable in the present study because they are directly applicable to the TCSTFEM in 4-D without the need for additional derivations, and also they are already available in anisotropic versions and are independent of the PDE being solved.

The specific error estimator selected for the present work is one that has already been successfully applied to both the incompressible and the compressible Navier-Stokes equations (Frey and Alauzet (2005)). So far, it has only be applied to steady flows or for the spatial discretization error of unsteady problems (Alauzet et al. (2007)). Fortunately, its extension to the full discretization error, comprising both the spatial and temporal discretization errors, is rather straightforward in the context of a unified TCSTFEM formulation.

This error estimator is based on the finite element interpolation error (D'Azevedo and Simpson (1991)), given by

$$\varepsilon = \|\eta(x,y,z,t) - \eta_h(x,y,z,t)\| \leq c_0 h^2 \|H(\eta(x,y,z,t))\| \tag{2.1}$$

where $\|\cdot\|$ is the $L^\infty(\Omega)$ or the $H^1(\Omega)$ norm, η is the exact solution, η_h is the numerical approximation of the solution and the Hessian is given by

$$H(\eta(x,y,z,t)) = \begin{pmatrix} \frac{\partial^2\eta}{\partial x^2} & \frac{\partial^2\eta}{\partial x\partial y} & \frac{\partial^2\eta}{\partial x\partial z} & \frac{\partial^2\eta}{\partial x\partial t} \\ \frac{\partial^2\eta}{\partial y\partial x} & \frac{\partial^2\eta}{\partial y^2} & \frac{\partial^2\eta}{\partial y\partial z} & \frac{\partial^2\eta}{\partial y\partial t} \\ \frac{\partial^2\eta}{\partial z\partial x} & \frac{\partial^2\eta}{\partial z\partial y} & \frac{\partial^2\eta}{\partial z^2} & \frac{\partial^2\eta}{\partial z\partial t} \\ \frac{\partial^2\eta}{\partial t\partial x} & \frac{\partial^2\eta}{\partial t\partial y} & \frac{\partial^2\eta}{\partial t\partial z} & \frac{\partial^2\eta}{\partial t^2} \end{pmatrix} \tag{2.2}$$

It is important to note here that the Hessian is constructed using time as the fourth independent variable, so that the temporal discretization error is taken into account in exactly the same manner as the spatial discretization error.

Because the chosen interpolation functions are piecewise linear, their second derivatives would vanish on each simplicial element. In order to permit an approximate computation of the Hessian, a weak formulation has to be used (Thompson et al. (1999))

$$\int_\Omega H_{ij} \cdot v_h d\Omega = -\int_\Omega \frac{\partial\eta_h}{\partial x_i}\frac{\partial v_h}{\partial x_j} d\Omega + \int_{\partial\Omega} \frac{\partial\eta_h}{\partial x_i} \cdot v_h dS \;\; (i = 1,4,\;\; j = 1,4) \tag{2.3}$$

where H_{ij} is the component of the Hessian matrix at row i and column j, v_h are taken in the set of continuous piecewise linear functions and η_h is the numerical approximation of η. In order to solve that linear problem to recover the Hessian, the mass lumping technique is used to obtain a diagonal system. Thus, the discrete Hessian H_{ij}^k at a vertex k is computed by

$$H_{ij}^k = \frac{\left(-\int_\Omega \frac{\partial\eta_h}{\partial x_i}\frac{\partial v_h^k}{\partial x_j} d\Omega + \int_{\partial\Omega} \frac{\partial\eta_h}{\partial x_i} \cdot v_h^k dS\right)}{\int_\Omega v_h^k d\Omega} \tag{2.4}$$

where v_h^k is the piecewise linear finite element hat function associated with the vertex k.

In practice, this mass lumping gives poor results for mesh points on the boundary. This can be verified by creating a scalar field with a second order polynomial for which the expected second derivatives are known and constant over the domain (Mohammadi et al. (2000)). However, this problem can be circumvented, for the most part, by dropping the surface integral from 2.4 to get (Mohammadi et al. (2000)):

$$H_{ij}^k = \frac{-\int_\Omega \frac{\partial \eta_h}{\partial x_i}\frac{\partial v_h^k}{\partial x_j} d\Omega}{\int_\Omega v_h^k d\Omega} \tag{2.5}$$

2.4.2 Anisotropic Extension and Metric Construction

The most prevalent strategy to extend meshing algorithms to the anisotropic case, whether using a Delaunay based method (Borouchaki et al. (1997)) or other meshing strategies such as edge based modifications (Vallet (1992)), is to employ a metric to modify the measure of distance between points in the usual Euclidean space.

The metric is typically provided at every point in the mesh in the form of a positive-definite symmetric $d \times d$ matrix. In 2-D, this matrix is defined as (Frey and George (1999))

$$M(\overrightarrow{x}) = \begin{pmatrix} a_x & b_x \\ b_x & c_x \end{pmatrix} \tag{2.6}$$

with $a_x > 0$, $c_x > 0$ and $a_x c_x - b_x^2 > 0$.

This metric field defines a *Riemannian structure*, unless the metrics at all points in the field are identical, in which case it becomes a *Euclidean structure* (Borouchaki et al. (1997)).

This metric is constructed from the Hessian matrix 2.2 by computing the eigenvalues and the eigenvectors of the Hessian (George (2001)) and forming the following product of three tensors:

$$M = |H(\eta(x, y, z, t))| = RDR^{-1} \tag{2.7}$$

where

$$D = \begin{pmatrix} |\lambda_1| & 0 & 0 & 0 \\ 0 & |\lambda_2| & 0 & 0 \\ 0 & 0 & |\lambda_3| & 0 \\ 0 & 0 & 0 & |\lambda_4| \end{pmatrix} \tag{2.8}$$

and R is composed of the eigenvectors as columns.

The metric can also be associated to a linear transformation N, with a dilatation transformation matrix D having the square of the eigenvalues as its diagonal and a rotation matrix R containing the eigenvectors as columns:

$$N = \sqrt{D}R = \begin{pmatrix} \sqrt{\lambda_1} & 0 & 0 & 0 \\ 0 & \sqrt{\lambda_2} & 0 & 0 \\ 0 & 0 & \sqrt{\lambda_3} & 0 \\ 0 & 0 & 0 & \sqrt{\lambda_4} \end{pmatrix} R \tag{2.9}$$

The square root of the eigenvalues in the linear transformation corresponds to the inverse of the desired element size along the eigenvectors such that $\lambda_i = h_i^{-1}$. This metric tensor is illustrated in Figure 2.3 by an ellipsoid with the eigenvectors as the minor and major axes and the desired element sizes along these directions.

The metric tensor 2.6 and the linear transform 2.9 are related as follows (George

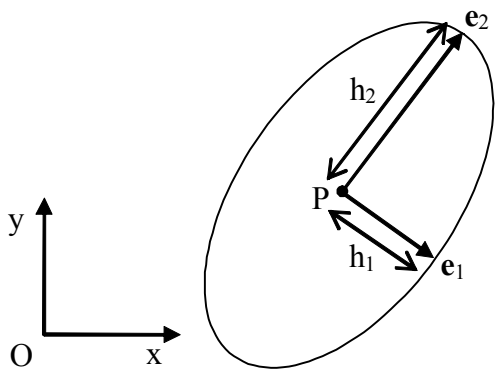

Figure 2.3: Illustration of a metric tensor defined at a point. The eigenvectors corresponds to the minor and major axis of the ellipsoid with the size of the ellipsoid along these directions corresponds to the desired mesh size.

(2001)):

$$M = N^T N \tag{2.10}$$

Hence, for every (column) vector X, the following equivalence holds (George (2001)):

$$X^T M X = X^T N^T N X = (NX)^T NX = \|NX\|^2 \tag{2.11}$$

With this notion of linear transformation, meshing algorithms can be transformed to anisotropic meshing algorithms by measuring the property of elements, such as their length along a particular direction or their volume in the transformed space. Figure 2.4 illustrate how elements are transformed from the Euclidean space to the transformed space.

It is also interesting to note that the ellipsoid associated with the metric tensor becomes a sphere in the transformed space (see Figure 2.5).

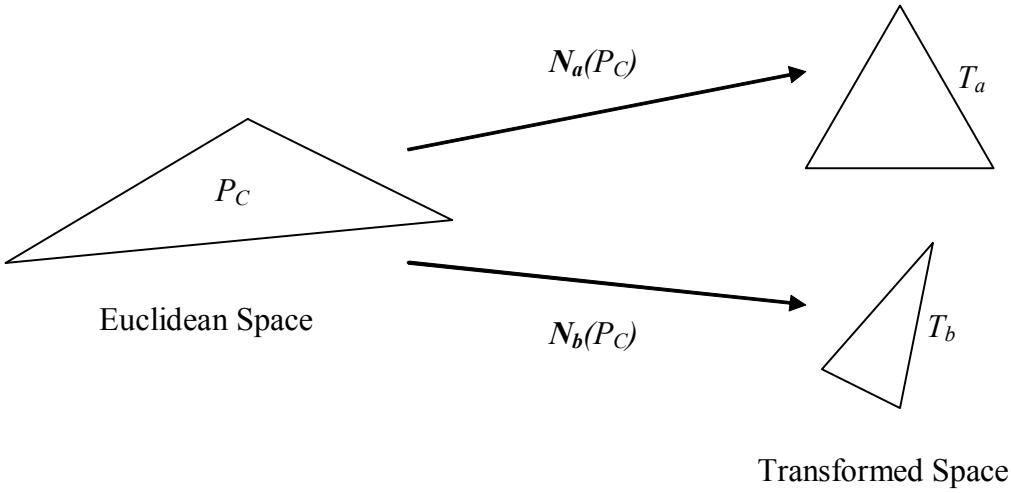

Figure 2.4: Illustration of the transformation of an element to a transformed space using two different linear tranformation.

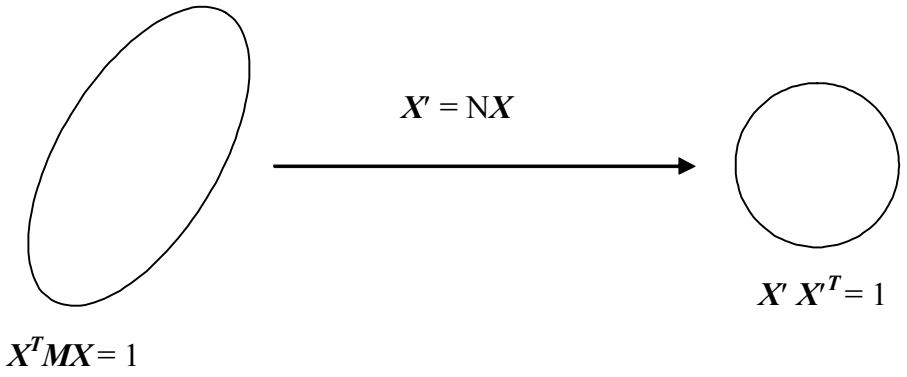

Figure 2.5: Illustration of the ellipsoid associated with the metric tensor and its transformation to a sphere in the transformed space.

Using this symmetric positive-definite metric $M(\overrightarrow{x})$, the length of the edge $[P,Q] = \left(P + t\overrightarrow{PQ}\right)_{0\leq t\leq 1}$ can be measured in the Riemannian space defined by $\left(\Omega, \left(M\left(X\right)\right)_{X\in\Omega}\right)$ with

$$L_m(P,Q) = \int_0^1 \sqrt{{}^t\overrightarrow{PQ}M(P + t\overrightarrow{PQ})\overrightarrow{PQ}}dt \tag{2.12}$$

where $\overrightarrow{PQ}$ is the column vector with origin P and extremity Q, and $M(P + t\overrightarrow{PQ})$ is the metric at point $P + t\overrightarrow{PQ}$.

The volume of the simplex K in the Riemannian space is given by (George (2001))

$$V_K^M = \int_K \sqrt{\det\left(M\right)}dV \tag{2.13}$$

If a metric is specified at each mesh point, its value would vary inside the simplex and the volume would need to be computed by numerical integration. On the other hand, if the metric were chosen at a fixed point, for example the centroid of the simplex, then the determinant of the metric would be a constant and can be taken out of the integral.

Alternatively, if the metric is constant over an element, then each of its points can be transformed using the linear transformation:

$$\mathbf{x}' = N\mathbf{x} \tag{2.14}$$

Then, any property, such as the metric distance along an edge of the element or its metric volume can be computed using the same formula as in the standard Euclidean space after the element has been transformed to the Euclidean metric space using the linear transformation 2.14. This facilitates the implementation of meshing algorithms that can operate isotropically or anisotropically, depending on whether the elements are transformed before measuring any property needed by the particular meshing algorithms.

In practice, the second derivatives for the Hessian can range from zero to very large values, so the corresponding eigenvalues in the metric could lead to element sizes that are not acceptable for a mesh adaptation procedure. Recalling that the absolute values of eigenvalues of the Hessian matrix are related to the inverse of the desired mesh size, they can be restricted between a maximum and a minimum mesh size to avoid problematic extremum cases (George and Borouchaki (1997)):

$$\widetilde{\lambda_i} = \min\left(\max\left(\lambda_i, \frac{1}{h^2_{\max}}\right), \frac{1}{h^2_{\min}}\right) \tag{2.15}$$

Furthermore, as the ratio of the smallest to largest element size specified through the metric increases, it becomes increasingly difficult for the meshing algorithms to satisfy the request for highly stretched elements (Borouchaki et al. (1998)). In this thesis, the degree of stretching is controlled by limiting the range of eigenvalues, which is directly related to the range of mesh sizes. The first step is to compute the *radius* (or range) of the eigenvalues and multiply it by the desired degree of anisotropy α:

$$\widetilde{\lambda}_{radius} = \left|\widetilde{\lambda}_{\max} - \widetilde{\lambda}_{\min}\right| \cdot \alpha \tag{2.16}$$

where $\widetilde{\lambda}_{\max}$ and $\widetilde{\lambda}_{\min}$ are the maximum and minimum values of $\widetilde{\lambda_i}$ computed from equation 2.15 (the absolute value is used to eliminate the possibility of getting zero values with a negative sign due to round-off errors in floating point arithmetic).

The radius of the eigenvalues can then be used to limit the minimum eigenvalues as follows:

$$\widetilde{\lambda_i} \leftarrow \max\left(\min\left(\widetilde{\lambda_i}, \widetilde{\lambda}_{\max}\right), \left|\widetilde{\lambda}_{\max} - \widetilde{\lambda}_{radius}\right|\right) \tag{2.17}$$

If the degree of anisotropy α is set to zero, then the radius of eigenvalues becomes zero and the metric would be isotropic. The maximum degree of anisotropy is obtained

when α is set to 1. Notice that the largest eigenvalue is used as a reference value because it corresponds to the maximum error associated with a given Hessian matrix 2.2; in other words, if the rotation matrix of the linear transform is the identity matrix, then the maximum eigenvalue corresponds to the maximum second derivative of the Hessian matrix in absolute value.

Although this strategy of limiting the desired degree of anisotropy has successfully been used previously (Tam et al. (1998)), it is not sufficient for the present purposes. The error estimator specified by the Hessian has a tendency to cluster too many mesh points in regions of the solution domain where the second derivatives are high. This may cause the solution to miss weaker phenomena which may occur in other regions (Alauzet et al. (2006)). In order to address this problem, the L^p norm of the error is used instead of the infinity norm, following the work of Alauzet et al. (2006), where the L^p norm of a function f is given by:

$$\|f\|_{L^p} = \left(\int_\Omega |f|^p \, d\Omega \right)^{\frac{1}{p}} \tag{2.18}$$

In short, with a suitable choice for the function f related to the interpolation error and the Hessian matrix, Alauzet et al. (2006) contrived a weighting factor for the metric given by:

$$M_{L^p} = (\det |H(\eta(x, y, z, t))|)^{\frac{-1}{2p+3}} \, M \tag{2.19}$$

where $\det |H(\eta(x, y, z, t))|$ is the determinant of the Hessian, p is the order of the L^p error norm used and M it the metric before the scaling.

Another issue concerning the construction of a metric field to drive the meshing algorithm is the notion of mesh gradation control. When the desired mesh size and orien-

tation specified through the metric vary strongly from one point of the mesh to the next, it may become increasingly difficult to construct meshes that have a good quality (Borouchaki et al. (1998), Li et al. (2004)). This problem is most prevalent for compressible flows where shocks occur with sharp variations in flow properties. Because only incompressible flows are considered in this thesis, no mesh gradation control has been used. Although mesh gradation is recognized as being important, its implementation is left as future work.

2.4.3 Element Size and Quality

Although the quality of elements can be measured in many different ways (George (2001)), the important objective for mesh adaptation purposes is to maximize the accuracy of the FEM solution while minimizing its computational cost.

From equation 2.1, it can be seen that, for the interpolation error ε to be bound by the same constant everywhere in the solution domain, the characteristic size of each element h must be inversely proportional to the magnitude of the error measured by the Hessian. The larger the local interpolation error is, the more the elements in the mesh must be refined. On the other hand, at locations at which the error is smaller than the maximum desired value, one may tolerate larger elements, which means that one may be able to remove nodes from the mesh. Removing mesh points can help reduce the computational cost, but can never improve the numerical accuracy of the solution. Therefore, from the viewpoint of accuracy, it is the mesh refinement algorithm which is the crucial element in a mesh adaptation procedure.

Another issue of concern is the effect of the element shape and orientation on the quality of the numerical solution. Certain types of element shapes, commonly called

degenerate elements, are known to cause difficulties for FEM solvers. A complete taxonomy of degenerate elements has been presented by (George (2001)). Succinctly, if the volume of an element or the area of one of its faces approach zero, then the FEM solver would experience numerical difficulties that would compromise the quality of the solution and possibly even the convergence of the iterative solver. Therefore, one has to ensure that such defective elements would not be created by mesh adaptation.

Before anisotropic mesh adaptation became available, it was believed that, if an element were highly elongated (namely if the smallest interior angle of a simplex were smaller than a certain value), then the FEM would experience numerical difficulties. No distinction was made as to the effect of the orientation of such elements. Research in anisotropic mesh adaptation has led to the realization that elongated elements would not adversely affect the quality of the solution if they were properly aligned with the physical patterns of the numerical solution (Vallet (1992)). As an example, consider a boundary layer, in which the flow velocity changes rapidly in the direction normal to the wall, but much more slowly in the direction tangent to the wall. In this case, it is preferable to have a higher mesh density in the direction normal to the wall than in the direction tangent to the wall, which results in a mesh with elements elongated in the direction tangent to the wall. Such elements have been found to give excellent results, albeit at a reduced computational cost compared to approximately equilateral elements. However, if the same elements were rotated to have their longest side in the direction normal to the wall, the quality of the numerical solution would suffer drastically and the convergence of the iteration solver could be compromised. Shock waves are another example in which elongated elements are suitable and cost effective.

Experience in anisotropic mesh adaptation indicates that it is important for the measure of quality of a simplex to take into account the alignment of the element with physical flow patterns (Borouchaki et al. (1997)).

The measure of quality of elements used in a mesh adaptation procedure affects not only the output of the mesh, but also the convergence of the mesh adaptation process itself. Hence, the specific selection of a measure of quality for simplicial elements will be deferred to a later section, after the context of the chosen mesh adaptation algorithm has been clarified. On the other hand, some properties of the element, called *shape criteria*, are known to be important as measures of the quality of an element. Such criteria are based on the following definition (Liu and Joe (1994), Dompierre et al. (1998), George (2001)) [1]:

Definition 2 (SIMPLICIAL SHAPE CRITERION) *A simplicial shape criterion is a continuous function that evaluates the shape of a simplex and is invariant with respect to translation, rotation, reflection and homothety of the simplex. It is said to be valid if it is maximal only for a regular simplex and it is minimal for all degenerate simplices. To facilitate comparisons, valid simplicial shape criteria are normalized in the interval* $[0, 1]$*, with 1 applying to the regular simplex and 0 to the degenerate simplex.*

A few clarifications are in order. In this definition, independence with respect to translation, rotation and reflection means that the shape criterion must be independent of the coordinate system. It is noted that the desired orientation of an element is detected through the Hessian and not by the orientation of the coordinate axes. Anisotropic shape criteria take that into account in ways that will be clarified later when specific shape criteria

[1] Translated from the French.

are formally introduced. Not only must the shape measure be independent of any arbitrary coordinate system, but it must also be independent of units used, which is what is meant by invariance with respect to *homothety*. The requirement that the shape criterion be a continuous function means that, if a simplex is almost regular, then its shape criterion should be almost 1 and, if it is almost completely degenerate, then it should be almost 0. Although the requirement imposed by this definition is straightforward, many algorithms presented in the literature have used ill-defined shape criteria, which indeed caused problems when the resulting elements were used by a FEM solver. To avoid such problems, the choice of shape criterion must be done formally rather than intuitively.

2.4.4 Performance

Mesh adaptation is meant to be a cost effective alternative, in terms of CPU time and memory usage, to indiscriminate mesh refinement. Hence, the primary performance criterion of a mesh adaptation method is the total cost of a numerical solution at a given accuracy level. It is important to keep in mind that the total cost should account not only for the time spent adapting the mesh once, but also for the time needed for the overall mesh adaptation process, which includes re-computing the solution and re-adapting the mesh a few times. Experience in 3-D mesh adaptation has shown that 3 to 5 mesh adaptation cycles are sufficient in most cases, although the number of cycles needed also depends on the choice of initial mesh (Tam et al. (1998)), the particular PDE and the method used to find an approximate numerical solution to the PDE.

Another performance criterion for mesh adaptation procedures is the memory requirement. Normally, this does not impose a stringent constraint, because the mesh adap-

tation procedure is usually executed between solver runs. Therefore, as long as the mesh adaptation algorithm uses less memory than the flow solver, the upper bound on the memory usage by the FEM process would remain unchanged. However, the objective of mesh adaptation is to achieve a solution at a given accuracy with fewer mesh points, which would reduce the overall memory requirement to solve a given problem.

Mesh adaptation procedures can be classified in two broad categories: *re-meshing methods*, which essentially create a completely new mesh at each mesh adaptation cycle, and *mesh modification methods*, which modify the mesh from the previous mesh adaptation cycle. Experience in the use of 2-D and 3-D spatial mesh modification methods (Dompierre et al. (1997), Tam et al. (1998)) has shown that most of the mesh modification happens during the first and second mesh adaptation cycles, whereas mesh modification in subsequent cycles is minimal, compared to the initial ones. Thus, mesh modification methods can potentially lead to significant computation cost savings, compared to re-meshing methods using algorithms of comparable efficiency. For this reason, we will prefer to use algorithms that permit mesh modification.

2.4.5 Mesh Optimization Algorithms in Higher Dimensions

The primary advantage of the time-continuous space-time finite element method over a time-discontinuous approach is the elegance with which an error estimator comprising both the spatial and the temporal discretization errors can be formulated and used as the basis for a space-time mesh adaptation procedure (French and Peterson (1996)). Before applying TCSTFEM to solve unsteady flows of engineering interest in 3-D geometries, it is necessary to demonstrate theoretically that 4-D mesh adaptation is possible.

This task is facilitated by the fact that theoretical studies in computational geometry and mesh generation have proved the existence of meshes in higher dimensions for non-convex input domains defined by a *piecewise linear complex* (PLC), with the constraint of not having "small" input angles in adjacent elements of the PLC. In the current context, a PLC can be defined as a valid finite element surface mesh with elements having an arbitrary number of planar sides. In practice, the PLC can be subdivided to form a simplicial surface mesh.

At least three different types of algorithms have been developed for mesh generation in higher dimensions on PLC, with formal proofs that each algorithm will terminate after adding a finite number of points to the mesh and yielding meshes of quality suitable for the FEM. The first algorithm is a *quadtree based method* developed by (Mitchell and Vavasis (2000)). The basic principle of a quadtree method is to create a bounding box of d-dimensions that contains in its interior all the input points and recursively subdivides that box into smaller boxes until the desired density of points has been reached. Then, the small boxes are subdivided into simplices and the faces of the surface mesh are recovered. The main drawback of quadtree methods is that the boxes created are aligned with the coordinates axes. Removing this alignment requires additional algorithms to modify the mesh. Furthermore, these algorithms have not yet been extended to the anisotropic case, in which elements are elongated along a certain direction (Frey and George (1999)).

A second method available is the *sweep algorithm* (Shewchuk (2000)), which is a one-pass algorithm to generate a mesh from a PLC. It is not suitable for mesh adaptation because it would require the mesh to be completely re-created at each mesh adaptation

cycle. Furthermore, it is not clear whether this algorithm could be governed by a size map to control the concentration of points in the mesh, even in the isotropic case.

A third method available is the *Delaunay refinement method*, first pioneered by Chew (1989), Ruppert (1995), and extended to 3-D by Shewchuk (1997). Li (2000) made a significant contribution by providing an improved version of the algorithm that can generate Delaunay meshes without any defective elements (*sliver free*) and by generalizing the theory to an arbitrary number of dimensions. Delaunay refinement has been shown to perform well in 2-D (Ruppert (1995)) and in 3-D (Shewchuk (1997)) in terms of the quality of mesh they produce. The problem of small input angle in the PLC has also been circumvented in practice for 2-D meshes (Shewchuk (2002)) leading to meshes of good quality that are suitable for the finite element method. However, Miller et al. (2002) mentions that although the algorithm is proved to terminate with a finite number of mesh points, the final mesh can possibly be very large. It remains to be shown in practice that these algorithms can perform reliably with a controllable number of mesh points on 3-D geometries that are of engineering interests.

Although most of the theoretical work aimed at proving that it is possible to construct meshes in higher dimensions was done for a set of mesh points in general positions (Joe (1993)), namely without the ability to conform to the surface mesh of a geometry, it is worthwhile to mention some properties of such volume meshes. First, in the worst case, the number of simplices for a Delaunay mesh is of the order $O\left(n^{d/2}\right)$, where n is the number of input mesh points and d is the space dimension (Joe (1993)). Specifically, this means that, for 2-D, 3-D and 4-D meshes, the number of simplices can reach a maximum of the order

of $O(n)$, $O\left(n^{3/2}\right)$, $O\left(n^2\right)$, respectively. In practice, a quadratic increase of the number of simplices for 4-D simplicial meshes is problematic in terms of performance. However, this is the theoretical worst case among all possible cases, so it remains to be seen empirically how algorithms generating such meshes behave. Second, it has been shown in 2-D that Delaunay meshes have properties that make them highly suitable for simulations based on the finite element method; an example is the maximization of the minimum interior angle, although this property does not scale to dimensions higher than two (Rajan (1994)). Third, it is known that Delaunay meshes in 3-D, although satisfying the Delaunay empty circumsphere criterion, can have tetrahedral elements of zero volume with four points that are cocircular (Frey and George (1999)). Theoretical studies on Delaunay meshes are an interesting starting point for meshing in higher dimensions, but their current state of development is not sufficient to lead to algorithms that are usable for engineering problems.

In this thesis, simplicial meshes will be considered, because they are the easiest to manipulate in 2-D, 3-D and 4-D and their theoretical treatment has proved that modifying meshes in higher dimensions is possible. Methods based on other types of elements, such as hexahedral elements, or even *meshfree* methods, which operate on points themselves without explicitly needing elements, could be considered in future work.

2.5 Anisotropic Mesh Modification Methods

Given that in 4-D the number of simplices increases faster than in 3-D or 2-D, it is imperative to seek to minimize the number of points that are required to compute a numerical solution of a given accuracy. Toward that end, anisotropic mesh modifications

were proved to be successful in 2-D (Vallet (1992), Castro-Díaz et al. (1995), Buscaglia and Dari (1997), Castro-Diaz et al. (1997), Fortin et al. (2000), Dompierre et al. (2002), Belhamadia et al. (2004a)) and have also been extended to 3-D (Tam et al. (1998), Tam et al. (2000), Belhamadia et al. (2004b), Bottasso (2004), Li et al. (2005)). Anisotropic mesh generation methods governed by metric size map have also been developed (Borouchaki et al. (1997), Borouchaki et al. (1997), George et al. (2002), Yamakawa and Shimada (2003), Du and Wang (2005), Frey and Alauzet (2005), Gruau and Coupez (2005)), but the added cost of remeshing at each mesh adaptation cycle is even more problematic for meshes in 4-D, because the boundary recovery procedures that they employ have so far been limited to 3-D (Du and Wang (2004b), Du and Wang (2004a), George (2003)). In this thesis, it was chosen to extend existing anisotropic edge based mesh modifications methods to 4-D, which to the best of our knowledge is used for the first time. The details of the anisotropic meshing algorithms developed are presented in Chapter 6.

Chapter 3

2-D, 3-D and 4-D Parametric Data Structure

3.1 Introduction

This chapter briefly presents the salient features of the parametric data structure created to implement the mesh adaptation algorithms and the finite element method in 2-D, 3-D and 4-D. The emphasis of the presentation is on the mesh data structure, because it is this aspect of the data structure that makes it possible to explore the proposed space-time mesh adaptation procedure in 4-D. It is crucial to the success of the proposed strategy to be able to implement algorithms in a dimension-independent manner, so that each algorithm can be tested first in 2-D, then in 3-D and finally in 4-D. Furthermore, this makes it possible to use the same unified numerical simulation strategy for both steady and unsteady problems.

3.1.1 Finite Element Data

As previously mentioned, the proposed mesh adaptation procedure is based on an unstructured finite element mesh. The type of information that needs to be stored in such meshes is well understood and clearly defined (Cook et al. (2002)). This includes mesh *points,* with a number of coordinates determined by the *space dimension* of the finite element solution domain Ω, and *elements* that constitute a partition of the space domain. Elements determine how an ordered list of points are connected together to define sub-regions of the solution domain. Each element is characterized by its *type* and *topological dimension.* Although the current study focuses on unstructured meshes composed of simplicial elements, a finite element data structure should be flexible enough to support other common element types as well. Elements supported by the Simulation Toolkit (STK) will be presented in the following.

In general, discrete data associated to points and elements also need to be stored in the finite element mesh. An essential set of data that need to be stored at the nodes is the computed finite element solution. In the case of the incompressible Navier-Stokes equations, this includes the velocity vector and the pressure. Material properties varying with location in the space-time domain must also be stored on the elements. A generic finite element data structure requires the ability to store data associated with both mesh points or/and elements.

3.1.2 Design Criteria

After the characteristics of the data to be stored have been identified, the next step is to define appropriate design criteria to guide the process of storing these data so that they can fulfill the desired purpose. In the context of this thesis, the data structure must be suitable for implementing an efficient space-time unstructured mesh adaptation procedure in 2-D, 3-D and 4-D, along with a space-time finite element fluid flow solver. In consequence, the following design criteria were chosen.

Parametric form. It must be possible to conveniently implement mesh modification algorithms, or the FEM procedure, in a dimension-independent or semi-independent manner, as needed. The dimension of the space domain should have to be specified only at runtime, so that the code can be executed for different space dimensions without having to be recompiled or requiring one to keep separate copies compiled for specific space dimensions. It is also necessary to be able to manipulate meshes of two different dimensions at the same time for some algorithms; an example would be mesh extrusion.

Compactness. Considering that the number of mesh points and elements in a FEM mesh typically range from several thousands to several millions, the storage of data should minimize the computer memory requirements.

Efficiency. In order to maximize the performance of the FEM procedure and mesh adaptation code, the data must be accessible in a manner that is independent of the data size (i.e., data should be retrieved and stored in constant time) . This makes it possible to implement algorithms that are proportional to the number of points or elements whenever possible ($O(n)$ in time complexity).

Flexibility. Because the data that need to be stored at runtime for the FEM and mesh adaptation procedure can be significantly different, the data structure must be flexible enough to permit the addition or removal of data at runtime rather than at compilation time. This way, the two procedures can be tightly integrated and the necessary data can be stored only for the period of time that they are needed, thus avoiding excessive memory usage. Moreover, because optimization of such algorithms is often achieved as a compromise between minimizing memory usage and maximizing speed, the flexibility to add or remove data at runtime makes it easier to pursue such optimization. This is very important for a research-oriented code, whose initial use aims at exploring the application of new numerical methods and algorithms.

Resizeability. Mesh modification requires the efficient addition and removal of points and elements. This is usually not necessary for the FEM when the mesh remains fixed during the computation of the numerical solution, but it is vital for the performance of mesh adaptation and mesh generation algorithms. Furthermore, the evolution of FEM codes tends toward a tighter integration of mesh adaptation within the FEM procedure, so it is necessary to have a data structure that provides the necessary functionalities for both.

Simplicity. Simpler data structures are easier to understand, debug and optimize. In general, simplicity in software correlates well with productivity for the developer and the ease with which new programmers can join in during software development.

Portability. The data structure and all procedures implemented on top of it should be able to be compiled on all major operating systems.

The remainder of this Chapter briefly describes the core of the data structure,

Data Structure Aspect	STK	VTK
Application field	FEM and meshing	Visualization
Space dimension	Parametric (2-D, 3-D, 4-D)	Hard coded in 3-D
Deleting points/elements	Explicitly supported	Not explicitly supported
Discrete mesh data resizing	Automatic	Explicit
Element storage and access	By type and id	By id

Table 3.1: Comparison of design criteria for STK and VTK.

which is the unstructured FEM mesh, and its evaluation with respect to the previously presented design criteria.

3.2 Mesh Data Structure

The mesh data structure is a C++ class that groups together the mesh points and elements with their associated data along with methods to manipulate them. Its design is strongly inspired by the Visualization Toolkit (VTK) (Schroeder et al. (1998)), but was adapted to meet design criteria specific to the FEM rather than visualization. By analogy to the source from which it was inspired, the mesh data structure has been named the STK. The primary differences in the design criteria between the two frameworks is summarized in Table 3.1.

The first fundamental difference is that VTK is hard coded to operate on mesh points in 3-D, whereas the current project requires to implement algorithms that can operate in 2-D, 3-D and 4-D. Modifying VTK to meet this requirement would require to branch out the code to create a new incompatible version. Although VTK is open sourced and, in principle, it would have been possible to modify the source code, this approach has been rejected because it would have required an excessive amount of effort and time.

The second fundamental difference is that VTK is not designed to explicitly manage the deletion of points or elements. The general philosophy in VTK is to copy a subset of data if some part of it is no longer needed or needed only for a specific algorithm. This is appropriate for visualization software, because selecting a subset of data to operate on or to display on screen is a common operation. In the context of mesh modification algorithms, however, it must be possible to efficiently remove unneeded points or elements.

The third fundamental difference is that VTK allows data to be added and removed on a mesh, but does not automatically resize these data when the mesh is resized through the addition or removal of mesh points or elements. In VTK, the mesh is either assumed to be fixed or the added data must be explicitly resized when the mesh is resized. However, this approach is highly inappropriate for mesh modification algorithms, because it increases the complexity of the code and requires a strong coupling of different parts of the code to appropriately maintain the coherence of the data with the mesh. Therefore, in STK, when the mesh is resized, its associated data are also automatically resized. This applies to both data added at runtime and at compilation time.

The fourth difference is that VTK stores all elements for its unstructured mesh in a single array and identifies an element in the array by a unique identification integer (id) which is also the index in the array. This makes it necessary to traverse elements of all types even when an algorithm is designed to operate only on elements of a specific type. For example, in a FEM code, it is typical to traverse all volume elements to compute the elementary system, add their contributions to the global system of equations, and only traverse faces on the boundary of the domain to compute fluxes, as in the case of boundary

conditions. Similarly, mesh modification usually proceeds by traversing volume elements and not boundary elements. In STK, elements are stored by type and id to make it possible to efficiently traverse only one type of elements at a time.

The next sections present overviews of the individual components of the mesh data structure, followed by a brief description of the grouping procedure so that they can form the unstructured mesh data structure of STK.

3.2.1 Parametrization: Separating Geometry from Topology

The principle behind the parametrization of the data structure is the separation of the geometry from the topology of the mesh. Geometry deals with the properties of points in space, whereas topology remains invariant under geometric transformations (Frey and George (1999)). In the current context, the geometric properties are functions of the mesh points whereas the topology is a function of the mesh element connectivity. In STK, elements only contain topological information with indirect references to the mesh points used by the elements. Hence, an element contains no information on the space dimension of its mesh points.

By separating geometry from topology, it is possible to manipulate or modify the topology of the mesh purely by manipulating the elements of the mesh without regards for the space dimension. If a part of an algorithm requires knowledge of the geometry, then the mesh points associated with an element can be read and this computation can be isolated in such a way as to make the space-dependent computation part transparent to the rest of the algorithm.

Figure 3.1: Representation of the 1-D contiguous data array with tuples having three components each .

3.2.2 Dynamic Data Arrays

The primary data container in STK are a memory contiguous data array of native types (e.g., float, int, char, bool etc.) that stores a series of *tuples*, each having the same number of components of the same basic type. A specific data entity in the array is identified by the integer id of its tuple and the component id marking the offset from the beginning of the tuple. The array is contained in a C++ class that provides methods to manipulate the data, such as traversing the data, read/write operations, resizing the array, dynamic resizing etc. Figure 3.1 illustrates a contiguous data array with each tuple having three components.

Because the entity in the data array is referenced by id and not by a pointer to a specific memory location, the data array can be copied and moved to a different memory location or even across a network for parallel computing while preserving the ids for its data. The memory allocation scheme in STK leverages this to resize arrays by first trying to extend its last element further in the computer's memory, if possible (a call to realloc), or else allocating a new larger contiguous array in which the data are recopied. Because the pointer to the beginning of the array is hiding inside the C++ array class and all references to the data in the array are done through integer ids, this recopying does not

affect algorithms that reference data in the array.

Contiguous resizeable arrays present several performance benefits, if used properly. First, data entity can be accessed in constant time using the integer ids. Second, traversing a contiguous piece of computer memory is faster then traversing several pieces of memory that may be spread at different locations of the memory, primarily because of the operation of memory caching in computers. Third, it is possible to over-allocate the contiguous data array to minimize the number of resize operations during the execution of an algorithm, such as during mesh refinement. Memory allocation is a very expensive operating system call and meshing algorithms were shown to perform better with pre-allocation memory schemes. Furthermore, the array would always remain contiguous after a resize operation, which prevents memory fragmentation problems that can significantly degrade performance.

In STK this basic data array is considered to be a 1-D array, because the number of components for each tuple must be constant for the array, so that only one index is needed to identify a tuple from the beginning of the array. There is also a 2-D array, which is essentially an array of 1-D arrays with support for tuples. In the 2-D array illustrated in Figure 3.2, the first index is the array index and the second index is the tuple's id for the given 1-D array.

3.2.3 Point Storage

Points are stored in a 1-D array of floats, where the tuple corresponds to the integer point id and the number of components to the space dimension of the points. Although the basic 1-D array of data of STK does not support the deletion of tuples, the management of deleted points is added to the point class. Points that are deleted are automatically skipped

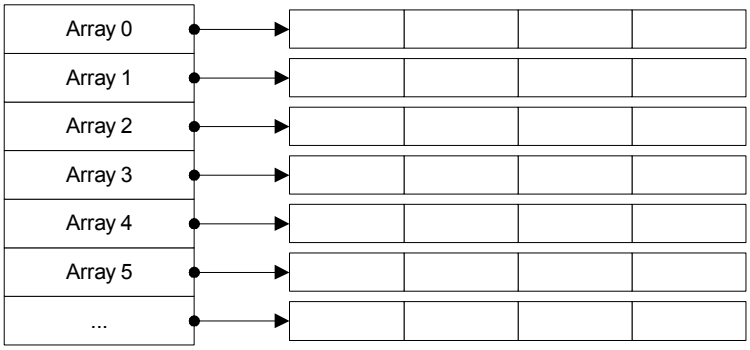

Figure 3.2: Representation of an array of arrays.

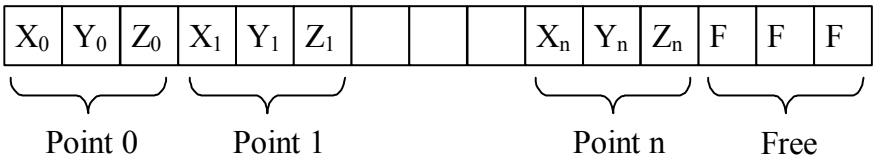

Figure 3.3: Representation of the point storage.

during later traverses of the array. When a new point is added, it replaces a previously deleted point until the deleted points are exhausted, so that the number of unused tuples in the array is minimized.

The point class also supports the dynamic addition and removal of data arrays through the collection of data arrays mechanisms. A collection of data for points is composed of a list of 1-D arrays, each having as many tuples as the number of mesh points. Each array can be of a different basic data type (float, int, char, bool etc.) and have a different number of components per tuple. For example, the velocity at nodes can be stored in a 1-D array of floats with the number of components equal to 3 for a flow in a 3-D geometry and the pressure can be stored in a different array with one component per tuple. The class

managing the collection of 1-D arrays of data for points automatically allocates arrays with a number of tuples that corresponds to the number of points in the array, with an one-to-one correspondence between the point id and its tuple in the data array, and automatically resizes the data arrays as needed when points are added or removed. Therefore, when traversing points of the mesh, it is sufficient to know the point id and a pointer to the class of the added data to retrieve the tuple associated with a particular point and for a given data value. In summary, data of any basic type can be dynamically added or removed on points and retrieved in constant time. Such data arrays are always automatically resized whenever points are added or deleted to maintain the coherence of each point and its associated data.

3.2.4 Element Storage

In STK, the connectivity of elements is stored in a 2-D array of integers in which the first index corresponds to the element type and the second index to the element id. The tuple associated with an element has a number of components equal to the number of points in the element and contains the list of point ids used by that element. Figure 3.4 illustrates the various element types that are supported by STK and Figure 3.5 gives a simplified example of how the element connectivity is stored.

The runtime addition or removal of data for elements follows the same pattern and mechanisms as for points, but the collection of data is composed of 2-D arrays of data, rather 1-D arrays, and tuples are identified through an element key (element type, element id), rather than a point id. This way, the storage of elements is memory compact, making it easy to traverse elements of a given type and whatever data an algorithm has added and needs to retrieve.

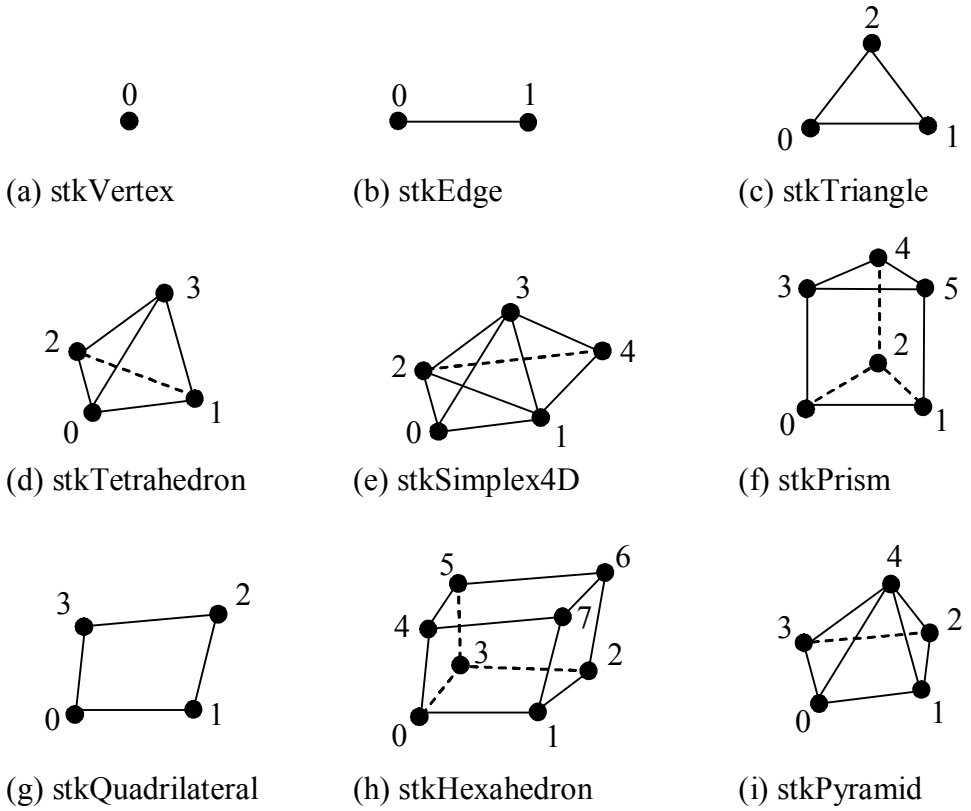

Figure 3.4: Illustration of the element types supported by STK along with their local node numbering.

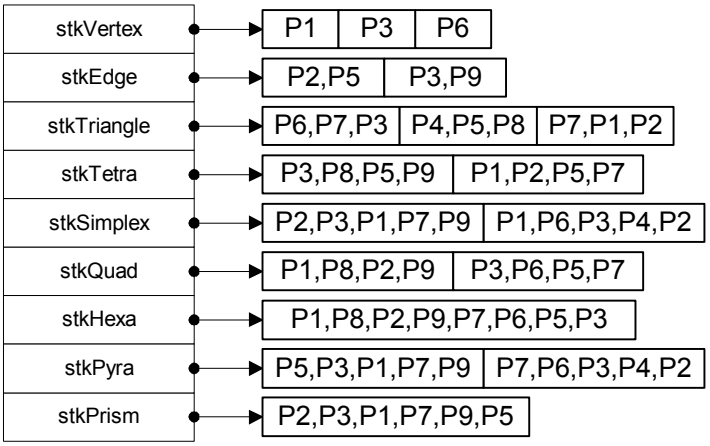

Figure 3.5: Example of how the elements are stored in STK using an array of pointers toward 1-D contiguous data arrays.

As a consequence of storing the element connectivity and its associated information in arrays, the element classes, one for each type of element and all inherited from a common base class, no longer store the information for an element. The element class becomes more like a mediator through which the information associated with an element can be conveniently retrieved. Experience with object oriented design in VTK has shown that this type of storage for elements leads to a more efficient implementation than storing a series of element classes either in a list or an array (Schroeder et al. (1998)).

In the current context, it is important to mention that the element classes contain no information on the space dimension of the mesh points. Elements are only topological entities that reference point ids . Furthermore, methods are provided that make it convenient to manipulate the topology of the mesh without knowing the specific element used.

3.2.5 Unstructured Mesh

The unstructured mesh class groups together the classes for mesh points and mesh elements, along with their respective collection of data mechanisms. It also provides methods to efficiently recover topological information that is frequently used for mesh modification algorithms, such as the list of elements that use a point id or the list of elements that have an element as their neighbour.

Recovering this type of information efficiently requires building an inverse connectivity. Several ways of storing this information exist (Frey and George (1999)), but the one chosen in STK is illustrated in Figure 3.6. In this Figure, the black dots represent points, the circles vertices of elements. The black arrows indicate direct connectivity links going from the vertex of an element to a specific point, whereas the grey arrows indicate inverse

connectivity links. Each point in the mesh has an inverse link pointing to one element that uses that point and to the corresponding vertex in that element. From that vertex, the next element and vertex in the list can be retrieved. Functionally, this is equivalent to a pointer from the mesh points to the list of elements using this point. However, here only the first link of the list is stored on points, while the other links are stored on elements. Although the number of elements using a point varies for unstructured meshes, the number of vertices that an element contributes to the inverse connectivity list is pre-determined by the element type and hence can be allocated at the same time as the elements themselves. This is important because, as the mesh is being modified by being refined or coarsened, the number of elements around points changes, so, if this information was stored in standard lists, one per point, it would require to be resized every time the mesh connectivity were changed. With the chosen scheme in STK, no memory allocation is required by changing the inverse connectivity of the mesh. Convenient methods are also provided in the mesh class to automatically update the inverse connectivity of the mesh when elements are added or removed.

3.3 Implementing Dimension Independent/Dependent Algorithms

A fundamental requirement of STK is to be able to implement dimension independent algorithms that operate on meshes in 2-D, 3-D and 4-D conveniently. Dimension independent codes have been developed by researchers in the computational geometry community (CGAL (2007)), but they all use one element type, a simplicial element, that has

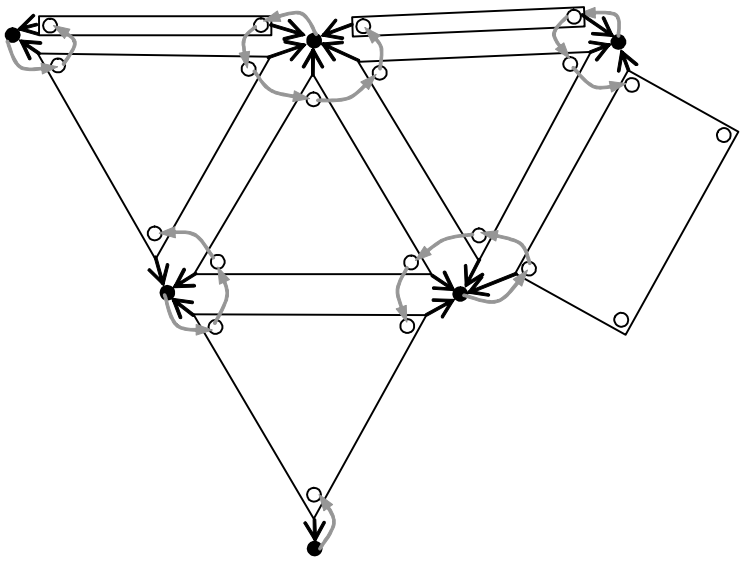

Figure 3.6: Illustration of the unstructured mesh. The dark arrows represent the connectivity from elements to their verticies while the gray arrows represent the inverse connectivity.

$d+1$ mesh points, where d is the space dimension.

This approach is not appropriate for numerical simulations, because of the need to differentiate between volume elements, which have a topological dimension equal to the space dimension, and boundary elements of lower topological dimension that are required to conveniently represent the boundary of the solution domain. For example, in a 3-D domain, tetrahedral elements cover the volume of the domain, triangular elements represent the surface of the boundaries, edge elements represent the delimitation between different adjacent surfaces of the boundary and vertex elements represent discontinuities in boundary curves approximated by a set of adjacent edges. Mesh modification algorithms need to operate differently on simplicial elements, depending on whether they are on the boundary or inside the domain. Therefore, it is appropriate to have different concrete element types

for each simplicial element of topological dimension 0, 1, 2, 3 and 4.

However, the consequence of this approach is that a mesh modification algorithm that operates on volume simplicial elements, for example, would need to operate on triangles in 2-D, tetrahedral elements in 3-D and simplex elements in 4-D, but all those elements are of different concrete types. An elegant method to address this problem without adding cumbersome and expensive conditional (if else) statements in the code is available under the name *visitor pattern* (Gamma et al. (1995)).

The visitor pattern is applied to solve the current dilemma by adding a class called *stkElementVisitor* that has a virtual method for each element type. Adding new functionality is done by inheriting from this base class to implement specific operations. A simple example is the computation of the volume of simplicial elements that differs for simplicial volume elements, boundary elements and non-simplicial elements.

The mechanism to call this element-type-specific method conveniently is implemented by adding a virtual method to the base class element called *AcceptVisitor* and receiving a pointer to the abstract class *stkElementVisitor*. Concrete element classes reimplement the AcceptVisitor method and provide an implementation in which the specific visitors method for the current concrete type is called. For example, the tetrahedral element class will always call the method VisitTetrahedralElement for the element visitor, irrespectively of what that visitor is.

In short, the visitor pattern provides a mechanism for conveniently adding new methods to operate on elements and automatically calling the specific method for the specific element type. Furthermore, it does this without cluttering the element classes with

new methods that might only be useful in some part of the code and not others. New functionalities are added to elements by adding new element visitors without modifying the original element classes.

Using this approach, implementing dimension independent algorithms can be done by hiding the topological dimension dependent part into element visitors. If some part of the algorithm requires knowledge of the space dimension, for example when it operates on mesh points, this information can also be hidden in the concrete visitor. Several algorithms were successfully implemented in a dimension independent manner this way. These will be presented in Chapter 6.

3.4 File Input-Output

The file storage and retrieval of the mesh and its associated data are done through the CFD General Notation Systems (CGNS), which is now a recommended practice by AIAA to facilitate the exchange of CFD data between codes (Poirier et al. (1998), Legensky et al. (2002)). Using CGNS for file input-output in STK makes it possible to import meshes from standard commercial meshers and export solutions to standard post-processing software. Furthermore, CGNS is freely available with source code[1], efficient, cross-platform and capable of storing meshes in 2-D, 3-D or 4-D. It may be noted here that, although it was not intended to support 4-D meshes, the CGNS format is sufficiently flexible to be easily extendable to storing 4-D mesh points and simplicial elements.

[1] http://www.cgns.org/

Chapter 4

Mesh-Based Geometry Reconstruction

4.1 Introduction

The objective of this chapter is to present a geometry reconstruction algorithm that is as simple as possible, yet sufficiently accurate for the purposes of both mesh generation and mesh adaptation for the finite element method using linear interpolation functions. To facilitate its implementation, the geometry is stored and manipulated using the same finite element mesh data structure as the one presented in Chapter 3. Conveniently, this also makes it accessible and suitable for finite element codes that need a lightweight geometry representation to add support for mesh adaptation procedures without having to link with a specific CAD package or more complex geometry library.

The algorithms used in mesh generation and adaptation may require to refine,

coarsen or smooth the surface mesh. In order for these algorithms to preserve the characteristics of the geometry, modifying the elements of the surface mesh requires to take into account the topology of the geometry, while inserting or moving points of the surface mesh requires to project these points on the geometry. Furthermore, it is necessary that the geometry representation be sufficiently smooth compared to the finite element surface mesh such that discontinuities of tangent planes on the geometry does not cause numerical errors in the analysis. The geometry reconstruction algorithm presented in this chapter was designed with these requirements in mind.

Several factors motivate the construction of a geometry rather then interfacing directly with a CAD system. First, in some biomedical applications, the starting point is not a CAD geometry, but rather a medical image that can be processed to get a tesselation (Cebral and Lohner (1999)). In this case, the tesselation is treated to recreate the topology of a surface mesh to represent curves that may need to be preserved or points that must be fixed during mesh generation or mesh modification. Second, CAD geometries are known for having several defects that can be catastrophic for meshing algorithms, for example small gaps between adjacent surfaces or topological inconsistencies that are of no consequence for a CAD geometry, but can cause the meshing algorithms to fail. Third, mesh adaptation requires a geometry to modify the mesh and it is convenient to have a lightweight geometry representation that can eventually be partitioned to run on parallel computers without requiring a CAD license on each processor. Fourth, it is faster for the meshing algorithms to project a point on a simpler geometry representation rather than on the actual CAD model. Fifth, reconstructing a geometry representation purely from a finite element mesh

can facilitate the exchange of finite element meshes and solutions coming from different packages that may themselves depend on a specific CAD package.

The chapter begins with the presentation of the surface mesh topology reconstruction, followed by the quadratic geometry reconstruction and briefly mentions some issues concerning the projection of points to the reconstructed geometry. Pseudo-code for the algorithms is presented at the end of the Chapter.

4.2 Surface Mesh Topology Reconstruction

When the input is a tesselation made of only triangular elements (quadrilateral elements can be easily subdivided into triangles), it is necessary to first reconstruct the topology of the surface mesh using simplicial elements of lower topological dimensions. In order to represent discontinuities between two adjacent topological surfaces, edge elements are used to approximate a curve. Similarly, discontinuities in adjacent curves are represented with vertex elements. Representing the topology of the surface mesh this way makes it easier to preserve that topology during the modification of the mesh, because points added to a topological surface are projected on that surface and are constrained by a perimeter of edges representing curves. Points added to a curve are also confined to that curve and, in the case of an open curve, they are bounded by the end points of that curve represented by vertex elements that are not allowed to move. Figure 4.1 illustrates the topology of a surface mesh where curves are approximated by a set of edges, vertex elements identify the end point of open curves and triangular elements compose the interior of a boundary patch.

The objective of the surface mesh topology reconstruction is to create simplicial el-

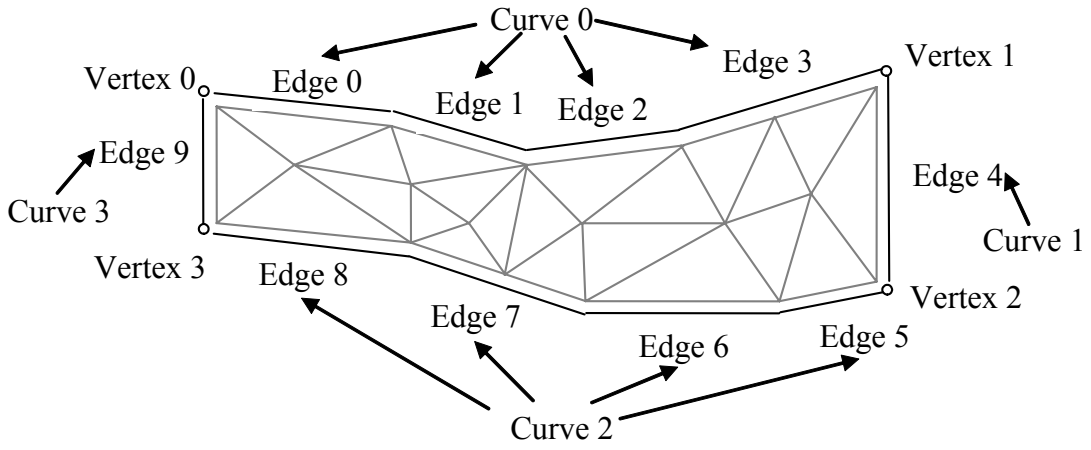

Figure 4.1: Illustration of the topological elements of the surface mesh that forms the boundary of the volume mesh.

ements of lower topological dimension to group elements of the surface mesh into *topological element patches*, which can be defined as:

Definition 3 (TOPOLOGICAL ELEMENT PATCH) *A topological element patch groups elements of the same topological dimension, but not necessarily of the same element type, that are adjacent to each other across common element boundaries not having elements of lower topological dimensions between them.*

The topological element patches created in this process would not be unique and would depend on the criterion used to decide whether adjacent elements of the surface need to be grouped together in the same patch or are separated by an element of lower topological dimension. The approach presented here is relatively simple and uses the angle between two adjacent elements as a topological patch separator criterion (Cebral and Lohner (1999)). More sophisticated methods, based on the estimation of the normals and the curvatures, have also been developed by other investigators (Baker (2004)), but their implementation are

left as future work and are compatible with the rest of the quadratic geometry reconstruction presented here.

The surface mesh reconstruction algorithm begins by adjusting the faces to make sure that the orientation of the normals for faces grouped in the same topological patch is consistent. This is done by marking all faces as not visited, then picking a first face and visiting its adjacent faces to flip them if they are not oriented in the same direction. Notice that this is done by relying on the local numbering of points on the element and not using normals. This way, the ordering only depends on the topology of elements in the patch. Using the normals of the faces for orientation could be misleading if few faces are used to represent a highly curved geometry. After a first face has been checked, it is marked verified and its neighbors are inserted in a list of faces to process. The procedure is repeated recursively until all faces in the list have been treated and no new faces marked unprocessed are found. Since a tessellation can be composed of several topological patches that are disjoint from each other, for example as in the case of the flow around a sphere enclosed inside a bounding box, it is necessary to search for unprocessed topological patches. This is done by searching for a face marked as not processed in the initial tessellation and using it to start the orientation for this new topological patch. The process terminates when all faces are marked as processed.

Once the faces are properly oriented, the normals are computed and stored on the faces until they are no longer needed. Then, a first iteration over all triangles is done to create edge element between two adjacent triangular elements if the angle between their respective normals is greater than a threshold value. After the first pass is completed, as

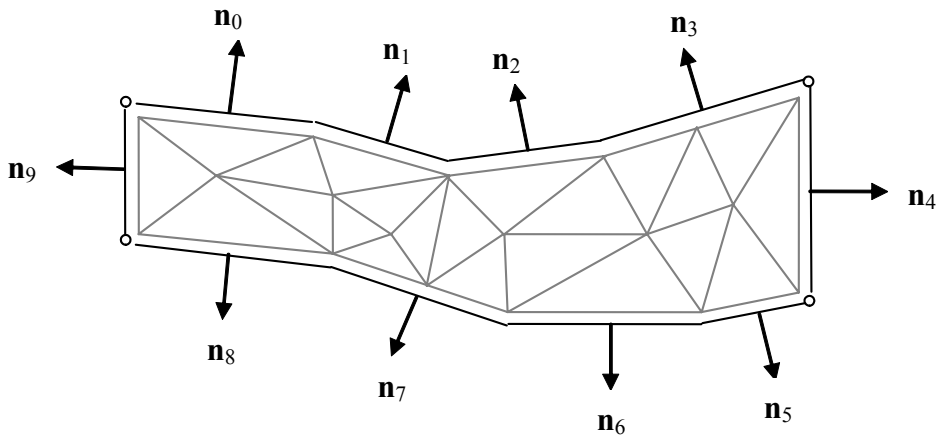

Figure 4.2: Illustration of the boundary face normals.

shown in Figure 4.2, a second pass is performed over the edges to create vertex elements between adjacent edges if their angle is greater than the same threshold value minus 180 degrees (a zero angle between edges corresponds to the two elements being parallel whereas for triangles it corresponds to their normals being parallel). Although this approach is very simple, it works fairly well in practice for surfaces with sharp discontinuities (Owen and White (2003)). However, it may become problematic when the variation of the curvature of a surface is progressive, for example in the case of an ellipsoid, for which methods based on curvatures are more suitable (Baker (2004)).

Once the surface mesh topology is reconstructed using the algorithm 1, it is useful to group all elements of the same topological patch and attribute them a unique topological patch identifier, which is done using the algorithm 3. Following the nomenclature of the CGNS library (Poirier et al. (1998)), this is named a *family*. Family ids are not only useful for the mesh generation or modification algorithms, but also to impose boundary

conditions on a group of boundary elements at the same time. This is especially important in the context of mesh adaptation because the list of boundary face elements on which Neumann boundary conditions are applied and the list of boundary mesh points on which Dirichlet boundary conditions are applied must be regenerated after each mesh adaptation cycle before the finite element assembly procedure. By specifying the boundary conditions on the families and not on specific surface elements or mesh points, any mesh modification procedure only need to preserve family ids as it refines or coarsen the mesh without any knowledge of boundary conditions. This also avoids creating a dependence between the mesh adaptation procedure and any specificity concerning the boundary conditions of a specific FEM solver package.

4.3 Quadratic Geometry Reconstruction

The objective of the quadratic geometry reconstruction algorithm is to build a smoother representation of the surface mesh than the initial ones made of linear simplicial elements. This is necessary to allow points to be added to or relocated on the surface mesh, while preserving a reasonable good fit to the initial geometry. For example, even a geometry as simple as a sphere requires a higher order reconstruction than a simple linear one to be able to move points on the surface and preserve the shape of the geometry.

An important issue concerns the minimal order of the geometry reconstruction that is appropriate for mesh generation and adaptation. In the case of tesselation derived from the processing of medical images, the constructed tesselation is so rough that the order of geometry reconstruction is not the most significant source of inaccuracy of the process.

However, CAD-generated geometries are much more precise (Meek and Walton (2000)) and they require higher order surfaces to accurately represent 3-D surfaces with torsion or saddle points than the simple quadratic one presented here (Walton and Meek (1996)). A practical requirement is that the reconstructed geometry be smooth enough such that a tangent plane along the boundary of adjacent elements in the same topological patch is constant across adjacent elements of the same topological patch (Walton and Meek (1996)), which for triangular elements requires at least fourth order Bezier patches (Piper (1987)).

Here, a simpler quadratic geometry construction will be presented using quadratic finite element interpolation functions for edge and triangular elements. Although this approach might not be sufficient to accurately represent warped surfaces, in practice it may be adequate for FEM using linear shape functions, if the starting point is a tesselation generated by a CAD package in the form of an STL (stereolithography) file. These tesselations are generated in the CAD package by specifying a maximum angle of deviation between triangular elements and the actual CAD geometry, so that warped surfaces will be represented by several triangles. Because the deviation angle is bounded, it seems reasonable for CFD applications to use quadratic finite element functions, as linear functions will be used to compute the solution inside the fluid domain. For other applications, like structural analysis, for which a higher order reconstruction may be necessary, the method presented here can easily be extended to higher order following the work of Walton and Meek (1996), Xue et al. (2004).

The geometry reconstruction algorithm presented builds upon the work of Walton and Meek (1996), in a fashion similar to Owen and White (2003), and requires that the

normals at the points of the surface be known in order to construct higher-order curves and surfaces. When the geometry is reconstructed from a CAD file before passing it to a mesh generation package or FEM software, the CAD system can be queried to return the normal at a point of a given surface. However, the objective here is to make the geometry reconstruction operational for tesselations coming from medical images and also to make it independent of any specific CAD package, so it is not acceptable to call a CAD function. Consequently, the first step of the geometry reconstruction algorithm is the estimation of the surface normals at points assuming that only linear elements are available for the surface mesh.

4.3.1 Point Normal Estimation

Several methods for reconstructing normals from a boundary surface are available (Walton and Meek (1995), Baker (2004)). Because the constructions of the higher-order curves and surfaces rely upon the normals at points, it is important that the determination of normals be as accurate as possible. The salient aspect of the following method is that it takes into account both the topology of the linear surface mesh and an estimate of its local average (isotropic) curvature. This approach permits the reconstruction of curves and surfaces at junctions between different topological patches on which different boundary conditions need to be imposed. As illustrated in Figure 4.3, a normal ($\mathbf{n}_A$) that is evaluated at a point on a curve by summing the normals for the neighboring points on the surfaces adjacent to the curve ($\mathbf{n}_{F_1} + \mathbf{n}_{F_2}$) and normalizing the resulting vector would be incorrect because it would not be tangent to the plane containing the circle shown in Figure 4.3. Although this is typically handled by decomposing the normals on curves to assign a different normal

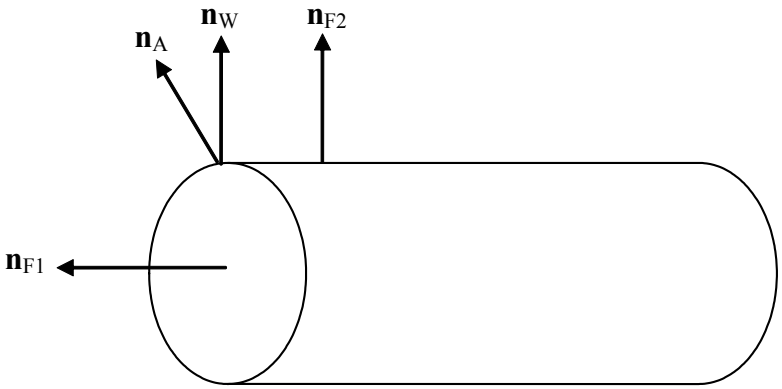

Figure 4.3: Illustration of the difference between the normal at a point on a curve obtained as the curvature-weighted average $\mathbf{n}_W$ of the normals $\mathbf{n}_{F1}$ and $\mathbf{n}_{F2}$ to the adjacent faces and the unweighted average $\mathbf{n}_A$ of the two normals.

to the topological patch corresponding to each surface adjacent to the curve (Owen and White (2003)), the method presented here does not require decomposing normals. Instead, normals are conveniently uniquely defined at boundary mesh points as weighted averages.

The point normals are computed using two different strategies depending on whether the point is on a surface or a curve. If a point is on a surface, it only has incident triangle elements to it and the normal for such point is simply computed as the average of the normals of the faces around that point, as follow:

$$\mathbf{N}_{Psurface} = \sum_{i=1}^{n} \mathbf{n}_i \tag{4.1}$$

where $\mathbf{N}_{Psurface}$ is the normal at a point on the surface (not touching any curves) and $\mathbf{n}_i$ is the unit normal of the face i. The point normals are not normalized at this stage of the algorithm. Instead, it is kept as the last step to normalize both the normals for points on the interior of surfaces and the normals on points touching curves in one iteration for all boundary points.

Going back to the example for the cylinder in Figure 4.3, it can be deduced that the estimation of the point normals on curves can be improved using of the local curvature of the adjacent surfaces. However, at this stage of the reconstruction, the surface is still composed of linear elements for which the curvature cannot be directly computed. Hence, the curvature needs to be estimated from this linear surface mesh, if it is to be used to improve the normal estimation.

A simple approach to estimating the curvature between two points can be done using the normals at the two points, as shown in Figure 4.4, and using basic trigonometry, which leads to:

$$C = \frac{1}{\rho} \simeq \frac{2}{h}\sqrt{\frac{1 - \mathbf{n}_1 \cdot \mathbf{n}_2}{2}} \tag{4.2}$$

where C is the curvature, ρ is the radius of curvature, which is the radius of the osculating circle passing through the two points, $\mathbf{n}_1$, $\mathbf{n}_2$ are the unit normals at the two points and h is the distance between these two points.

Using formula 4.2, the curvature can be estimated between two adjacent faces using their normals and the centroid of each face as the two points. Then, the average curvature for a face is defined as the average curvature computed using all sides of the face (three for the triangles used here). The normals for points on curves of the boundary are then computed as the weighted average of the normals of the faces incident to a particular point, with the weight factor for each normal taken as the average curvature associated to that face divided by the square of the distance between the point and the centroid of the face, which gives:

$$\mathbf{N}_{Pcurve} = \sum_{i=1}^{n} \mathbf{n}_i \frac{\widetilde{C_i}}{h_i^2} \tag{4.3}$$

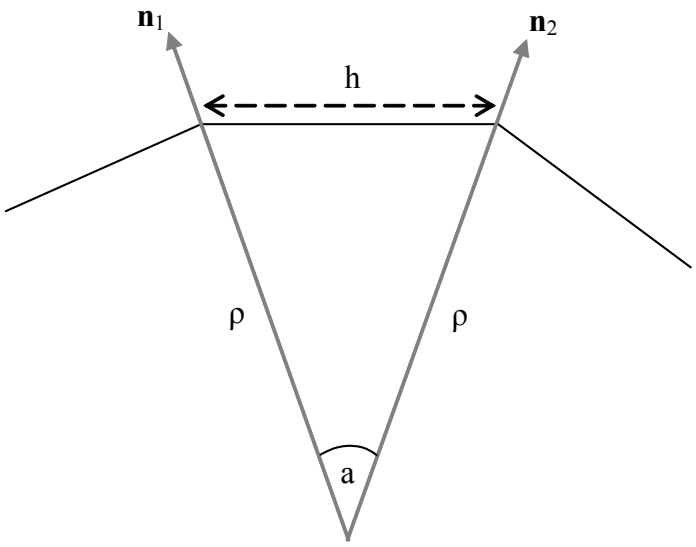

Figure 4.4: Illustration of the point normals and face size used to compute the approximate curvature $(1/\rho)$.

where $\mathbf{N}_{Pcurve}$ is the normal at a point on a curve of the surface mesh, $\mathbf{n}_i$ is the normal for the face i incident at the point, $\widetilde{C_i}$ is the average curvature for the face i and h_i^2 is the square of the distance between the point where the normal is evaluated and the centroid of the face. The last step of the algorithm 4, which is detailed in pseudo-code form at the end of the Chapter, is to iterate over all points to normalize the point normals.

It is important to emphasize that the computation of the normal for the purpose of a geometry reconstruction, even for a geometry as simple as a cylinder, would face serious problems if the normals were simply averaged for point on curves. This can cause significant error in some numerical simulations where boundary conditions are imposed on such surfaces, as it is frequently the case for internal flows. For a cylinder to be properly reconstructed with higher order curves delimiting the plane faces, the normals need to be perpendicular to the circular section, which means that they need to be perpendicular to

the longitudinal axis. Without weighting the normals with the curvature, the normals on the end curves of a cylinder would be a mix between the normals of the triangular elements on the plane end section and those on the circular section of the cylinder. Using a splitting normal approach, it would not be clear for the construction of the curve whether the normal should be estimated using the faces on one side of the curve or the other (a curvature based criteria could be used to make this decision, but this would become very similar to what is presented here). Because the construction of higher-order curves relies on the point normals, incorrect normal estimation would lead to higher order curves at the end of the cylinder that are not in the plane of the end section of the cylinder. The approach presented here does not suffer from this problem and offers the advantage of having normals being uniquely defined at each point, which is simpler to store and manipulate.

It would be interesting to study the accuracy of the presented approach more precisely using a tessellation constructed from a smooth surface, for example a CAD surface, and compare their accuracy with the normals computed from the CAD package using the technique presented by Meek and Walton (2000), but this is left for future work.

4.3.2 Geometric Reconstruction Algorithm

The geometry reconstruction begins by computing the normals as previously described and then uses these normals to construct quadratic edges followed by quadratic triangular elements.

The specification of a quadratic finite element edge requires three points. The first two points are taken as the end points of the linear edge element, while a third, mid-point is computed using a Bezier curve constructed by the method described by Walton and Meek

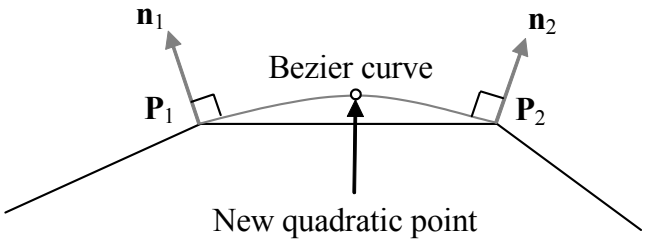

Figure 4.5: Illustration of the Bezier curve used to compute the additional point required to form a quadratic edge on the boundary. The Bezier curve passes through both end points of the linear edge and is perpendicular to the normals at these points.

(1995), Walton and Meek (1996) using the positions of these two points and the weighted normals at these two points, as shown in Figure 4.5.

After all quadratic topological edges have been created, the quadratic triangular elements are created by using the same approach for each of the three edges of a quadratic element shown in Figure 4.6. If the triangle element has a quadratic curve adjacent to one of its sides, then the mid-point from that quadratic edge is recovered and connected to the quadratic triangle. If the triangle element has another element adjacent to it across this side, then the mid-point is recovered from the neighbouring triangular element, if this has already been constructed. Otherwise, it is constructed using a Bezier curve in the exact same manner as for a curve. Details of the algorithm are presented in section 4.7.

4.4 Projection of Points on a Curve or Surface

During a mesh optimization procedure, new mesh points can be added to or moved on the finite element surface mesh. For the surface mesh to remain in good agreement with the geometry, each new point added or moved need to be projected to the closest quadratic

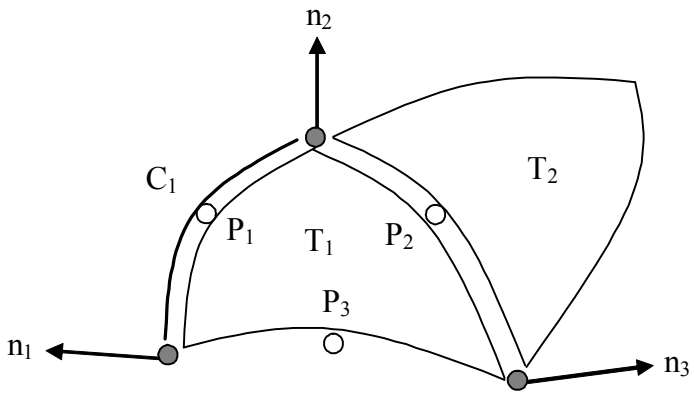

Figure 4.6: Illustration of the contruction of a quadratic triangular element T_1.The normals n_1 and n_2, and their respective points, are used to contruct the Bezier curve to determine the new quadratic point P_1 shared by the quadratic triangle T_1 and the quadratic edge C_1. Point P_2, which is shared by T_1 and T_2, is constructed in a similar manner.

edge or quadratic triangular element.

The first step in the process of projecting a point to the geometry is to locate the closest specific quadratic edge or triangle and then compute the closest point of that entity to the point to project. To facilitate the location of the point on the boundary, an association between the surface mesh used to create the geometry and the finite element surface mesh forming the boundary of the volume mesh used in the finite element analysis is maintained. This association is stored in file and updated during the mesh modification procedure such that an edge always knows its closest quadratic element and a triangular element its closest quadratic triangular element.

The projection of a point on a quadratic edge begins by projecting the point to a linear edge element having the same two end points as the quadratic edge as shown on the left side of Figure 4.7. This projection is done by evaluating the barycentric coordinate of

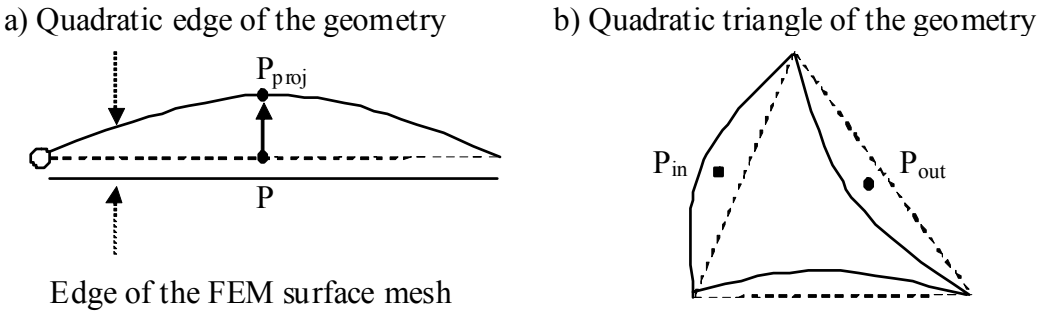

Figure 4.7: Illustration of the projection of a point to a quadratic edge on the left (a) and a quadratic triangle on the right (b). The dotted lines represent the linear elements corresponding to the quadratic entities.

the point to project on the linear edge, shown in dashed in the Figure, to find the closest point of that edge to the projection point. If the projected point is inside that linear element, then that closest point is projected to the associated quadratic curved giving the desired location on the quadratic curve. If the closest point is outside the segment, as indicated by the value of the barycentric coordinates previously evaluated, then the adjacent element in the same topological patch is tested using the same procedure. If the adjacent element is a vertex element, shown as small circle in the Figure, then the end of a topological curve is reached and the point associated to the vertex element is returned as the projected location.

For quadratic triangular elements, it is necessary to modify that method to take into account the curvature of the boundary of the element. This is illustrated on the right side of Figure 4.7, in which the quadratic triangle is shown with its associated linear triangle. In this Figure, the point P_{in} is inside the quadratic element, but outside of the linear element, while point P_{out} is outside of the quadratic element and inside the linear associated one. In the case of triangular elements, the location step is the same as for

the quadratic edge case. The quadratic triangular element that is the closest to the linear triangle element of the FEM surface mesh is retrieved (recall that this mapping is stored) and the point is projected to the linear element associated to that quadratic element, that is a linear element with the same corner points as the quadratic one. Then the search proceeds through adjacent neighbouring elements until a quadratic element is found for which its associated linear element contains the projected point. It is here that an extra step is added compared to the edge case to determine whether the point projected to the linear element is inside the quadratic element or outside of it (again using the barycentric coordinate value as an indicator). If it is inside, then the point evaluated on that quadratic element is returned. If it is outside, the neighbouring element is used instead and the projected point on that quadratic element is returned.

This strategy is efficient because the mapping between the linear elements of the surface finite element mesh and the corresponding quadratic closest elements is updated as the mesh is refined or as points are moved. This mapping is also stored on file when the mesh and the geometry are written and is recovered when they are read. Because a point added to refine an edge or a triangle is always on that entity, it is always close to the initial quadratic element and the location procedure only needs to visit a few elements. Furthermore, the projection always constrains a point to be projected on elements of the same topological patch to avoid deforming the surface mesh. It is also important to highlight that projecting the point to the linear element associated to the quadratic element and travelling along the surface of the geometry avoids problems that could occur when using a spatial location approach, that is a search that considers the entire volume contained by the geometry rather

than travelling along the surface of the geometry, when another part of the same topological patch would be close to itself. A simple example is that of a cylinder squashed to be nearly flat with two parts of the surface of the same topological patch being at a close distance from each other. If a spatial search was used on a squashed cylinder, a point added to split a triangular element on the top surface could possibly end up being projected on the bottom part of the cylinder, which would result in a defective mesh.

4.5 Possible Higher Order Extensions

Although the geometry reconstruction presented here uses quadratic finite elements, it could easily be extended to the use of higher order polynomials or Bezier curves. Such extension has been presented by Owen and White (2003), who have also leveraged the work of Walton and Meek (1996). To accomplish this, one does not need to change the procedures for topological surface reconstruction and normal estimation, which are actually very similar to the procedures used by Owen and White (2003). The main difference is the need to store the reconstructed Bezier curves, or the normals used to construct them. Moreover, each fourth-order Bezier patch, which can be constructed from these Bezier curves following the approach suggested by Walton and Meek (1996), would require the storage of additional control points.

It is worthwhile to mention that the estimation of the normals presented here, which takes into account both the topology and the curvature of the geometry, can also be beneficial for the case of higher-order extensions, because the construction of higher order curves with improper normals, such as in the cylinder example mentioned earlier, would

lead to similar problems as for lower-order elements.

In the present thesis, the decision was made to use quadratic elements, rather than higher-order ones, because the former are simple to implement and sufficient for mesh adaptation using linear finite element interpolation functions. Higher-order elements would have the advantage of being sufficiently smooth for use in higher-order FEM and would also allow an evaluation of the maximum and minimum curvatures of the surface at a point, which would be beneficial in future work on anisotropic mesh generation (see Frey and George (1999) for a different approximate approach).

4.6 Implementation Note on the Mesh-Based Data Structure

In order to minimize the amount of code needed for manipulating the geometry, it is stored with a combination of a mesh class, as presented in Chapter 3, and a list of geometric elements. The mesh class stores the topology of the geometry along with the point coordinates associated with the geometry elements. To each geometric element corresponds a linear element in the mesh class for the geometry. The list of points for each geometric element is ordered such that the first points are the ones corresponding to the linear element. This way, the linear element associated to a quadratic element can be reconstructed simply by reading the first points of the quadratic element and can be used in the projection procedure as explained previously.

The significant advantage of using an FEM mesh class to store the topology of the geometry is that all algorithms implemented to manipulate the topology of the mesh are available to manipulate the geometry simply by using the topological cell key for geometric

elements. This greatly facilitates the implementation of the projection of points to geometric entities.

Furthermore, the implementation already makes it possible to modify the geometry to add or remove geometric elements and dynamically adjust the topology of the geometry. This allows for future improvements, such as optimizing the geometry construction to minimize its deviation from an available CAD geometry, or to minimize the number of elements used in the geometry to represent a surface, which can lead to significant improvement in performance.

4.7 Pseudo-Code for the Geometry Algorithms

This section presents the pseudo-code for the algorithms described in this chapter, which briefly are:

- Algorithm 1: the reconstruction of the topology of a surface mesh from a tessellation
- Algorithm 2: the identification of elements of a same topological patch with a unique family id
- Algorithm 3: the identification of elements that need to be grouped in the same topological patch
- Algorithm 4: the computation of normals on topological curves of the surface
- Algorithm 5: the reconstruction of quadratic edges and triangles from a topologically valid FEM surface mesh

Algorithm 1 Surface Mesh Reconstruction

Input. A tessellation T, made of a set of n triangular elements representing the surface of a geometry, but without any elements of lower topological dimension to represent discontinuities in surfaces and curves. The maximum angle β between two adjacent elements, to indicate that there is a discontinuity in a surface or curve.

Output. A surface mesh M with edge elements to represent discontinuities in surfaces and vertex elements to represent discontinuities in curves.

```
1: Orient faces in each topological patch to have consistent normals
2: for each triangle elements t ∈ T do
3:    for each face f of triangle t do
4:       Find element neighbor E_neighbor of lowest topological dimension across face f
5:       if E_neighbor is a triangle element and has a lower element id than t then
6:          Compute angle θ between triangle t and E_neighbor
7:          if θ is greater than β then
8:             Create new edge element on face f and insert it in T
9: for each edge element e ∈ T do
10:   for each end point p of edge e do
11:      Find element neighbor E_neighbor of lowest topological dimension across end point p
12:      if E_neighbor is an edge element and has a lower element id than e then
13:         Compute angle θ between edge e and E_neighbor
14:         if θ is greater than β then
15:            Create new vertex element on end point p and insert it in T
```

Algorithm 2 Create Families

Input. A mesh M.

Output. A set of n family ids F, stored on the mesh M , with one family flag per element. Each family id uniquely identifies one topological element patch.

```
1: Set number of families N_f to zero.
2: Create an integer data on element to store the family id for each element.
3: Initialize family id on all elements to -1 (not set yet).
4: Initialize FindElementsInTopologicalPatch.
5: for topological dimension td = 0 to SpaceDimension do
6:    while Found a new element E_td of topological dimension td, with the family id not
      set yet. do
7:       Increment N_f
8:       FindElementsInTopologicalPatch(M, E_td, E_patch)
9:       Set family id for all elements in E_patch to N_f
```

Algorithm 3 Find Elements in Topological Patch

Input. A mesh M and an initial element E_i.

Output. The list of elements that belong to the same topological element patch as E_i.

1: Create data on mesh elements to store a Boolean flag $F_{traversed}$ for element that have been traversed.
2: Set all flags $F_{traversed}$ to false.
3: Create an element stack $S_{element}$.
4: Initialize $S_{element}$ with first element E_i in topological element patch.
5: Set $F_{traversed}$ to true for E_i
6: Create an array of element key E_{patch} to store the keys of elements that were found to be in the current patch.
7: Initialize E_{patch} with first element E_i in topological element patch.
8: **while** Stack $S_{element}$ is not empty **do**
9: Remove element E_j off the stack $S_{element}$
10: **for** each face f of element E_j **do**
11: Get the neighboring element $E_neighbor$ of lowest topological dimension for E_j across f
12: **if** the topological dimension of $E_{neighbor}$ equals that of E_j and $E_{neighbor}$ has not been traversed **then**
13: Set $F_{traversed}$ to true for $E_{neighbor}$
14: Add $E_{neighbor}$ to stack $S_{element}$ to check its neighbors
15: Add $E_{neighbor}$ to E_{patch}

Algorithm 4 Point Normal Computation on Curves

Input. A mesh M.

Output. Unit normals at points of the surface mesh.

Allocate data for point normals and set all normals to zero.
Compute and store the normals to each face of the surface mesh.
for each surface element E_s in M **do**
    Retrieve normal E_{normal} for face E_s
    for each point P_i of E_s **do**
        Add E_{normal} to the normal at the point P_{normal}
Reset all normal for points of edge elements to zero.
for each edge element E_{edge} in M **do**
    Find triangular element neighbors $E_{triangle}$ of E_{edge}
    Set the number of weights nw and the weights W to zero.
    for each $E_{triangle}$ **do**
        Compute the average curvature C on $E_{triangle}$
        Compute the distance D between the center of $E_{triangle}$ and E_{edge}
        Set $W_{nw} = C \ / \ D$
        Increment nw
    if the sum of W is smaller than 0.001 **then**
        Reset W to 0.5
    else
        Normalize W with the sum of W
    Set edge normal N_{edge} to zero.
    for each $E_{triangle}$ **do**
        Multiply the face normal of $E_{triangle}$ by W for that triangle and add it to N_{edge}
    Add N_{edge} to P_{normal} for each point of E_{edge}
Make all point normal unit normals.

Algorithm 5 Quadratic Geometry Reconstruction

Input. A mesh M.

Output. A geometry G representing a smoother version of the surface mesh using quadratic polynomials.

1: Compute the surface normals using SurfaceNormalComputation(M, N).
2: Create one vertex geometric element for each vertex element in M.
3: **for** each edge element E_{edge} in M **do**
4: ComputeTopologicallyAdjustedNormals$(E_{edge}, F_{edge}, N_{adjusted})$.
5: ComputeMidPointUsingBezierCurve$(E_{edge}, N_{adjusted}, P_m)$.
6: Insert the mid-point P_m in G.
7: Create a quadratic geometric element using the P_m and the end points of E_{edge} and insert in G.
8: **for** each triangle element $E_{triangle}$ in M **do**
9: Create a new quadratic triangle element $G_{triangle}$ using the points of $E_{triangle}$ and insert it in G.
10: **for** each face F of $E_{triangle}$ **do**
11: Initialize mid-point id to null P_{mid}.
12: Find the neigbor element $E_{neighbor}$ across face F.
13: **if** $E_{neighbor}$ is an edge **then**
14: Recover the geometry edge G_{edge} associated to $E_{neighbor}$
15: Set P_{mid} to the mid-point of G_{edge}.
16: **else if** $E_{neighbor}$ is a triangle and element id of $E_{triangle}$ is less than that of $E_{neighbor}$ **then**
17: Recover edge face F_{edge} of $E_{triangle}$
18: ComputeTopologicallyAdjustedNormals$(E_{triangle}, F_{edge}, N_{adjusted})$
19: ComputeMidPointUsingBezierCurve$(F_{edge}, N_{adjusted}, P_m)$.
20: Insert new point in G using coordinates P_m and set new point id to P_{mid}.
21: **else if** $E_{neighbor}$ is a triangle and element id of $E_{triangle}$ is greater than that of $E_{neighbor}$ **then**
22: Recover the mid-point $P_{mid-face}$ of the face of the triangle $E_{neighbor}$ that corresponds to F.
23: Set P_{mid} to the mid-point of G_{edge}.
24: Set P_{mid} for face F of $G_{triangle}$.

Chapter 5

Governing Flow Equations and Space-Time Finite Element Discretization

5.1 Introduction

This chapter presents the governing equations for laminar incompressible unsteady flows and the selected finite element discretization. In the context of the current research, the emphasis for the solver was on the selection of an existing time-discontinuous space-time formulation that could easily be extended to a time-continuous space-time formulation to investigate the proposed unsteady mesh adaptation method. For this thesis, the application examples will be limited to incompressible laminar flows. It is for these cases that a time-continuous space-time formulation has been developed and compared to an available time-

discontinuous formulation using spectral finite elements (Pontaza and Reddy (2004)).

The present approach could potentially extend to other partial differential equations treatable with the finite element method, provided that a suitable time-continuous formulation and an error estimator can be derived. In that respect, it has been observed that the time-discontinuous Galerkin formulation for unsteady problems has mathematical properties that are analogous to that of the steady case (Johnson et al. (1984)). Furthermore, it has been shown that, for the finite element method, the proofs of solution existence, stability and uniqueness for an advective-diffusive problem are the same for time-continuous and time-discontinuous space-time formulations with the exception that, in the latter case, an additional term is present in the variational formulation to account for projection of the solution from one time slab to the next (Onate and Manzan (1999)). Consequently, properly derived and verified finite element formulations for steady problems can readily be extended to the unsteady case using a space-time formulation (Shakib (1988)), either time-continuous or time-discontinuous. In practice, this facilitates the derivation of time-continuous space-time formulations for various partial differential equations. More research is needed on error estimators, especially for the case for which multi-criteria error estimators are needed, but there is sufficient evidence to indicate that the proposed unsteady mesh adaptation method may possibly be extended to all partial differential equations, making it a potent method to explore (French and Peterson (1996), French (1999), Onate and Manzan (1999), Pontaza and Reddy (2004), Réthoré et al. (2005)).

5.2 Governing Equations for Viscous Incompressible Flows

5.2.1 Unsteady Incompressible Navier-Stokes Equations

Consider an incompressible viscous fluid in a space domain $\Omega \in \Re^d$, $d = 2$ or 3, during a time interval $t \in (0, T)$. The governing equations for laminar incompressible flows include the equation for the conservation of mass and the momentum equation (Newton's second law). In Cartesian coordinates, they are written as (Gresho and Sani (1998)):

Conservation of Mass:

$$\nabla \cdot \mathbf{u} = 0 \tag{5.1}$$

Momentum:

$$\rho \left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \boldsymbol{\nabla}) \mathbf{u} + \mathbf{f} \right) - \nabla \cdot \boldsymbol{\sigma} = 0 \tag{5.2}$$

where $\mathbf{f}$ is the body force vector (per unit mass) and ρ is the density. Besides the velocity vector $\mathbf{u}$, the system of the above equations contains the total stress tensor $\boldsymbol{\sigma}$, which is also unknown; to ensure closure, it is supplemented by a stress-strain relationship, also referred to as a constitutive relationship. For the simple case of Newtonian fluids, this relationship is taken to be (Gresho and Sani (1998)):

$$\begin{aligned} \sigma(\mathbf{u}, p) &= -p\mathbf{I} + 2\mu\boldsymbol{\varepsilon}(\mathbf{u}) \\ \varepsilon(\mathbf{u}) &= \frac{1}{2}\left(\nabla\mathbf{u} + (\nabla\mathbf{u})^T \right) \end{aligned} \tag{5.3}$$

where $\boldsymbol{\varepsilon}(\mathbf{u})$ is the viscous stress tensor, μ is the shear viscosity of the fluid and p is the pressure. The resulting momentum equations are known as the Navier-Stokes equations.

In many applications, the equations are presented in dimensionless form by normalizing the variables by appropriate scales. Numerically, this has the additional advantage

of making the solution variables of order unity, which minimizes the round off errors resulting from large differences in flow variable numerical values (Gresho and Sani (1998), Langtangen (1999)).

5.2.2 Space-Time Domain

In a space-time approach, the d-dimensional spatial domain Ω in $\Re^d$, with $d = 2$ or 3 for the Navier-Stokes equations, is extended to include the time axis leading to a unified space-time domain $\Omega \times (0, T)$ of dimension $d + 1$. The dependent variables for the problem and the interpolation functions used in the finite element discretization are therefore a function of both space and time.

In the time-discontinuous case, the time domain $(0, T)$ is partitioned into N time intervals, such that $0 = t_0 \leq t_1 \leq ... \leq t_N = T$. A subinterval of the time domain will be referred to as a slab with $S_n = \Omega \times (t_n, t_{n+1})$, where $0 \leq n \leq N - 1$. Figure 5.1 illustrates a time slab on the left and contrasts it with the time-continuous case on the right where a fully unstructured mesh is used for the entire time interval $(0, T)$.

In the time-discontinuous case, solving an unsteady problem involves computing the solution sequentially for each time slab, starting with an initial solution at t_0, in a fashion analogous to that used by classical time-stepping approaches. In contrast, in the time-continuous case, the entire *space-time* domain is considered at once, which is analogous to the solution of a steady problem in the entire *space* domain.

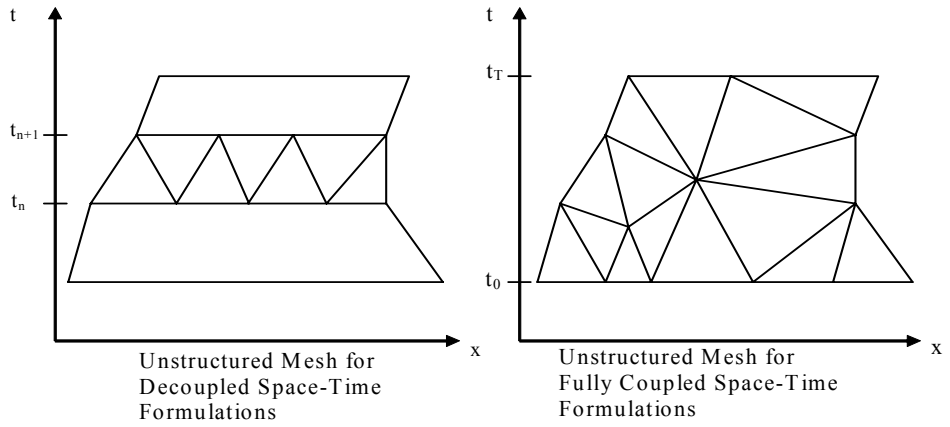

Figure 5.1: Illustration of representative meshes for a space-time slab (left) and a fully coupled, or time-continuous, space-time mesh (right).

5.2.3 Boundary and Initial Conditions

For the problem modelled by the Navier-Stokes equations to be well defined mathematically, proper boundary conditions must be imposed on the boundary of the solution domain along with a compatible initial condition. In this thesis, boundary conditions of the Dirichlet and Neumann type will be applied as shown below:

$$\begin{aligned} \mathbf{u} &= \mathbf{g}, \quad \text{on } \Gamma_D \\ \mathbf{n} \cdot \boldsymbol{\sigma} &= \mathbf{h}, \quad \text{on } \Gamma_N \end{aligned} \tag{5.4}$$

where Γ_D and Γ_N are complementary subsets of the boundary Γ and $\mathbf{n}$ is a normal unit vector pointing outwards of the domain Ω.

Note that the stress-divergence form of the viscous term in the Navier-Stokes equations was chosen because it leads, in the weak formulation shown later, to natural boundary conditions of the Neumann type that represent the physical forces (Gresho and Sani (1998)).

The initial condition for the incompressible Navier-Stokes equations consists of a

divergence-free velocity field everywhere in the domain Ω_0:

$$\mathbf{u}(\mathbf{x}, 0) = \mathbf{u}_0 \tag{5.5}$$

where $\mathbf{u}_0$ satisfies the equation 5.1. Note that there is an infinite number of such fields and the one chosen must satisfy the boundary conditions at time $t = 0$ for the incompressibility condition to be satisfied. An improper initial condition, such as imposing a null velocity field when it does not satisfy the boundary condition for the problem at $t = 0$, violates the incompressibility condition and can compromise the convergence of the solution procedure Gresho and Sani (1998).

In this thesis, in a fashion similar to the suggestion of Gresho and Sani (1998) for time-stepping methods, it is chosen to use a boundary condition that is a function of time to simulate the acceleration of the flow from rest at $t = 0$. This approach ensures that the incompressibility condition is satisfied at $t = 0$ not only inside the solution domain, but also at its junction with the boundary of the domain where the boundary conditions are imposed. The same approach was also chosen by Pontaza and Reddy (2004) for their fully coupled space-time formulation of the incompressible Navier-Stokes equations. Two functions are employed. The first one is a linear ramp:

$$\begin{aligned} \mathbf{n} \cdot \mathbf{u} &= u_{\max} \tfrac{t}{\tau} \quad \text{for } 0 \leq t \leq \tau \\ \mathbf{n} \cdot \mathbf{u} &= u_{\max} \quad \text{for } \tau \leq t \leq T \end{aligned} \tag{5.6}$$

where $\mathbf{n}$ is a normal unit vector pointing outwards of the domain Ω, $\mathbf{u}$ is the velocity vector, $u_{\max}$ is the maximum magnitude of the velocity (with an algebraic sign), t is the time and τ is the time period for which the boundary condition is modified by the ramp function.

The second time function used varies smoothly and includes a hyperbolic tangent function:

$$\mathbf{n} \cdot \mathbf{u} = u_{\max} \tanh\left(\tfrac{t}{\tau}\right) \quad \text{for } 0 \leq t \leq T \tag{5.7}$$

5.3 Finite Element Discretization

5.3.1 Overview of the Galerkin Finite Element Formulation

In the finite element method, a solution to a partial differential equation is sought on a partition of the solution domain, which was previously defined as the finite element mesh. On that mesh, the primary variables for the PDE problem are approximated over an element using piecewise functions, here polynomial expressions are considered. For the Navier-Stokes equations, using a formulation over the space-time domain with polynomial interpolation functions, the primary variables, which are the velocity and pressure, take the forms (Donea and Huerta (2003)):

$$\begin{aligned} \mathbf{u}_h(\mathbf{x}, t) &= \sum_{j=1}^{m} \psi_j(\mathbf{x}, t)\, \mathbf{u}_j \\ p_h(\mathbf{x}, t) &= \sum_{j=1}^{m} \phi_j(\mathbf{x}, t)\, p_j \end{aligned} \tag{5.8}$$

where ψ_j and ϕ_j are the interpolation (or shape) functions for velocity and pressure respectively, $\mathbf{u}_h$ is the interpolated velocity vector on the element, $\mathbf{u}_j$ is the velocity vector at node j of the element, p_h is the interpolated pressure and p_j the pressure at node j. Notice here that the interpolation functions are a function of both the space and time coordinates since a combined space-time discretization is used. This is in contrast to a more classical discretization in space where the interpolation function are only a function of the space coordinates and a different discretization in time is used (frequently a finite difference

discretization in time) (Donea and Huerta (2003)):

$$\begin{aligned} \mathbf{u}_h(\mathbf{x},t) &= \sum_{j=1}^{m} \psi_j(\mathbf{x})\,\mathbf{u}_j(t) \\ p_h(\mathbf{x},t) &= \sum_{j=1}^{m} \phi_j(\mathbf{x})\,p_j(t) \end{aligned} \tag{5.9}$$

In a weighted-residual method, the original PDE is modified by being multiplied by test functions, $\mathbf{v}_h$ for velocity and q_h for pressure and integrating by parts the solution domain Ω such that the weak formulation for the discretization in space becomes (Donea and Huerta (2003)):

$$\begin{aligned} \int_\Omega \mathbf{v}_h \cdot \left(\rho \frac{\partial \mathbf{u}_h}{\partial t}\right) d\Omega + \int_\Omega \mathbf{v}_h \cdot (\rho \mathbf{u}_h \cdot \nabla)\,\mathbf{u}_h d\Omega + \int_\Omega \nabla \mathbf{v}_h \cdot (-p_h \mathbf{I} + \mathbf{2}\boldsymbol{\mu}\boldsymbol{\varepsilon}(\mathbf{u}_h))\, d\Omega \\ + \int_{\Gamma_n} \mathbf{v}_h \cdot \mathbf{h} d\Gamma - \int_\Omega \rho \mathbf{v}_h \mathbf{f} d\Omega + \int_{S_n} q_h \nabla \cdot \mathbf{u}_h d\Omega = 0 \end{aligned} \tag{5.10}$$

The interpolation functions cannot be chosen arbitrarily for the problem to be properly mathematically defined and have a unique solution. The polynomial interpolation functions for the velocity ψ_j and for the pressure ϕ_j are chosen from wider spaces of trial functions that are not necessarily polynomials. In the case of the Navier-Stokes equations, the finite element method requires that these trial functions for velocity and their first derivatives be square-integrable, while it is sufficient for trial function for the pressure to be square-integrable in the solution domain Ω (Donea and Huerta (2003)). This will be briefly formally introduced, but more details can be found in books on the finite element methods such as Donea and Huerta (2003).

First, the space of functions that are square integrable over the domain Ω is noted $L^2(\Omega)$ and defined as:

$$L^2(\Omega) = \left\{ f : \Omega \to \Re \,|\, \int_\Omega f^2 d\Omega < \infty \right\} \tag{5.11}$$

where f is a function of a point $\mathbf{x} \in \Omega$ such that $f(\mathbf{x}) \in \Re$ and the square of the integral of such function is finite. This space is equipped with the inner product :

$$(u, v) = \int_{\Omega} uvd\Omega \tag{5.12}$$

and its induced norm:

$$\|u\|_0 = \sqrt{(u, u)_0} \tag{5.13}$$

Next, the Sobolev space of order 1 for functions and derivatives that are square integrable is defined as:

$$H^1(\Omega) = \left\{ u \in L^2(\Omega) \,|\, \frac{\partial u}{\partial x_i} \in L^2(\Omega) \right\} \; i = 1, ..., d \tag{5.14}$$

and is equipped with the inner product:

$$(u, v)_1 = \int_{\Omega} \left(uv + \sum_{i=1}^{n} \frac{\partial u}{\partial x_i} \frac{\partial v}{\partial x_i} \right) d\Omega \tag{5.15}$$

and its induced norm:

$$\|u\|_1 = \sqrt{(u, u)_1} \tag{5.16}$$

This Sobolev space $H^1(\Omega)$ contains infinitely many functions (Donea and Huerta (2003)). For the current purposes, it is necessary to constrain these possible functions to a set of finite functions that can be used on the partition of the domain Ω previously defined in Chapter 2 as the finite element mesh. Therefore, a finite dimensional subspace $H^{1h} \subset H^1(\Omega)$ is defined here for piecewise-polynomial C^0-continuous interpolation functions over the elements of the finite element mesh as:

$$H^{1h} = \left\{ \varphi^h | \varphi^h \in C^0\left(\overline{\Omega}\right), \varphi^h|_{\Omega_e} \in P_m, \forall \Omega_e \in T_h \right\} \tag{5.17}$$

where φ^h is the interpolation function, Ω_e denotes the element of the the finite element mesh T_h and P_m is a polynomial of order m.

Furthermore, the test functions must properly be chosen for the set of equations generated from the weak form 5.10 to be independent from each other and for the solution to the problem to be uniquely defined. In a Galerkin method, these test functions are chosen from the same space as for the trial functions, but with a small distinction for their values on the boundary of the domain Γ_D where Dirichlet boundary conditions are imposed. The trial functions must be such that they satisfy the Dirichlet boundary conditions on Γ_D, whereas the test functions must vanish on Γ_D. More precisely, for the incompressible Navier-Stokes equations, the trial functions and test functions are chosen here as (Donea and Huerta (2003)):

$$S_u^h = \left\{ u_h | u_h \in H^{1h}, u_h = \overline{u} \text{ on } \Gamma_D \right\} \tag{5.18}$$

$$V_u^h = \left\{ v_h | v_h \in H^{1h}, v_h = 0 \text{ on } \Gamma_D \right\}$$

$$S_p^h = \left\{ p_h | p_h \in H^{1h} \right\}$$

$$V_p^h = \left\{ q_h | q_h \in H^{1h} \right\} \tag{5.19}$$

where S_u^h is the space for trial functions for velocities, V_u^h the space for test functions for velocities, S_p^h the space for trial function for pressure and V_p^h the space for test functions for pressure.

Going back to the Navier-Stokes equations, the elementary finite element system of equations is obtained by substituting the equations for the interpolation functions from equations 5.8 into equations 5.10, the details of which can be found in (Donea and Huerta (2003), Gresho and Sani (1998)).

A global system of algebraic equations for the entire PDE is obtained by assembling the contributions of the elementary system for each element in the mesh assuming that the primary variables are continuous across adjacent elements and that the net flux across adjacent faces vanishes. With the inclusion of proper boundary conditions and initial conditions, an approximate solution to the resulting system of algebraic equations is sought using appropriate numerical methods, to be discussed briefly in the section 5.4. The reader interested in a more detailed discussion on assembly procedures for the FEM can consult Langtangen (1999) among others.

5.3.2 Choice of Elements

The choice of elements in the application of the finite element method to the incompressible Navier-Stokes equations is a very delicate process. Not all choices lead to numerical schemes with a unique and stable numerical solution, even for well-posed mathematical problems (Gresho and Sani (1998)).

One criterion that can be used to select a numerically stable element is the so called *Ladyzhenskaya-Babuska-Brezzi* (LBB) (Ladyshenskaya (1969), Babuska (1973), Brezzi (1974)) or "inf-sup" condition, which is for steady Stokes flow (Donea and Huerta (2003)). The LBB compatibility condition impose requirements on the dimension of the continuous and discrete spaces for the velocity and pressure. In terms of the polynomial interpolation functions used this translates to the requirement that the order of the polynomial for the velocity be of higher order than that of the pressure. If this requirement is satisfied, then the LBB compatibility condition guarantees the existence and uniqueness of the solution. It is considered beyond the scope of this thesis to discuss this in details, but

the references above can be consulted for a more in depth treatment.

Although the LBB compatibility condition precludes the use of equal order interpolation functions for the velocity and pressure, so called *stabilization* methods have been developed to circumvent the LBB compatibility condition and thereby allow for equal-order interpolations for velocity and pressure (Donea and Huerta (2003)). Furthermore, they have also been extended to the incompressible Navier-Stokes equations and such stabilization methods are now considered standard practice for the finite element method (Donea and Huerta (2003)).

For the stabilization method selected in the next section, linear finite element interpolation functions are chosen because they are known to be of sufficiently high order for the incompressible Navier-Stokes equations (Donea and Huerta (2003)), have been widely studied (Franca and Frey (1992), Tezduyar and Behr (1994)), and have been also well tested with the Hessian based error estimator briefly presented in the Chapter 2.

5.3.3 Galerkin / Least-Squares Formulation

In the context of the proposed space-time mesh adaptation, the primary selection criterion for a stabilized formulation of the incompressible Navier-Stokes equations is its applicability to unsteady problems and its extensibility to a time-continuous formulation. Among the stabilization methods available, the category of methods referred to as Galerkin / Least-Squares formulations meet these requirements, although they are usually used in a time-discontinuous framework (Donea and Huerta (2003)).

The formulation presented here follows the work of Tezduyar and Behr (1994), Tezduyar et al. (1992), which was introduced under the name "Deforming-Spatial-Domain

/ Space-Time", but is considered to be part of a wider family of stabilization methods generally referred to a Galerkin / Least-Square formulations (GLS). Although the GLS method was shown to be equivalent to the mini-element (N'dri et al. (2002)), the GLS method was chosen over the mini-element approach because it is simpler to implement in a fashion that is dimension independent. Furthermore, during the FEM assembly process, the GLS method only requires that the first derivatives of the interpolation functions be computed once per element. In contrast, the macro-element used in the approach of N'dri (2001), requires that first derivatives be evaluated for the pressure interpolation functions and for each of the sub-elements for the velocity interpolation functions. As that requires $d+1$ evaluations of the first derivatives per element (recall that d is the space dimension), as opposed to one evaluation per element for the GLS method, it appears that the GLS method is more computationally cost-effective, especially in higher dimensions.

The GLS (or DSD/ST) method of Tezduyar and Behr (1994) can be summarized as follows. For a slab element S_n, given $\mathbf{u}_-^n$, one needs to find $\mathbf{u}_h \in S_u^h$ and $p_h \in S_p^h$, with $\forall v_h \in V_u^h$ and $\forall q_h \in V_p^h$, such that:

$$\int_{S_n} \mathbf{v}_h \cdot \left(\rho \frac{\partial \mathbf{u}_h}{\partial t}\right) d\Omega dt + \int_{S_n} \mathbf{v}_h \cdot \left(\rho \mathbf{u}_h \cdot \nabla\right) \mathbf{u}_h d\Omega dt + \int_{S_n} \nabla \mathbf{v}_h \cdot \left(-p_h \mathbf{I} + \mathbf{2}\boldsymbol{\mu}\boldsymbol{\varepsilon}\left(\mathbf{u}_h\right)\right) d\Omega dt \tag{5.20}$$

$$+ \int_{\Gamma_n} \mathbf{v}_h \cdot \mathbf{h} d\Gamma dt - \int_{S_n} \rho \mathbf{v}_h \mathbf{f} d\Omega dt + \int_{S_n} q_h \nabla \cdot \mathbf{u}_h d\Omega dt$$
$$+ \int_{\Omega_n} \mathbf{v}_+^n \cdot \rho \left(\mathbf{u}_+^n - \mathbf{u}_-^n\right) d\Omega dt + ST_{mom} + ST_{cont} = 0 \tag{5.21}$$

where the stabilization terms for the momentum equations ST_{mom} and for the continuity

equation ST_{cont} are

$$ST_{mom} = \sum_{e=1}^{n_{el}} \int_{S_n^e} \frac{\tau_{mom}}{\rho} \left[\rho \left(\frac{\partial \mathbf{v}_h}{\partial t} + \mathbf{u}_h \cdot \nabla \mathbf{v}_h \right) + \nabla q_h - \nabla \cdot \left(\boldsymbol{\mu} \boldsymbol{\varepsilon} \left(\mathbf{v}_h \right) \right) \right] \tag{5.22}$$

$$\cdot \left[\rho \left(\frac{\partial \mathbf{u}_h}{\partial t} + \mathbf{u}_h \cdot \nabla \mathbf{u}_h \right) + \nabla p_h - \nabla \cdot \left(\boldsymbol{\mu} \boldsymbol{\varepsilon} \left(\mathbf{u}_h \right) \right) \right] d\Omega dt \tag{5.23}$$

$$ST_{cont} = \sum_{e=1}^{n_{el}} \int_{S_n^e} \tau_{cont} \nabla \cdot \mathbf{u}_h \rho \nabla \cdot \mathbf{v}_h d\Omega dt$$

and the stabilization coefficients are

$$\tau_{mom} = \left[\left(\frac{2}{\Delta t} \right)^2 + \left(\frac{2 \left\| \mathbf{u}_h \right\|}{h_e} \right)^2 + \left(\frac{4\nu}{h_e^2} \right)^2 \right]^{-1/2} \tag{5.24}$$

$$\tau_{cont} = \frac{h_e}{2} \left\| \mathbf{u}_h \right\| z \tag{5.25}$$

with $z = \mathrm{Re}/3$ if $\mathrm{Re} \leq 3$ or $z = 1$ otherwise. The local Reynolds number used to compute z is defined as

$$\mathrm{Re} = \frac{\left\| \mathbf{u}_h \right\| h_e}{2\nu} \tag{5.26}$$

where $\left\| \mathbf{u}_h \right\|$ is the average velocity on the element S_n^e, h_e is the element size defined below (see equation 5.27) and ν is the kinematic viscosity.

Notice that this stabilized discretization is performed over the space-time domain $S_n = \Omega \times (t_n, t_{n+1})$ and can be extended to the entire space-time domain $\Omega \times (0, T)$, not just a time slab, since nothing precludes the slabs from being refined and containing several layers of elements. Furthermore, if time-continuous approximations are chosen, then the term $\int_{\Omega_n} \mathbf{v}_+^n \cdot \rho \left(\mathbf{u}_+^n - \mathbf{u}_-^n \right) d\Omega dt$ that was necessary to transfer the solution from one time slab to the next now vanishes (Onate and Manzan (1999)). However, with a finite element method using piecewise-polynomials that are C^0-continuous, this imposes the requirement that the mesh points between the time slabs matches one to one. In this thesis, rather

then using fixed time slabs with continuous in time approximations, a fully unstructured space-time mesh comprising the entire space-time domain $\Omega \times (0, T)$ is chosen because of the flexibility with which such a space-time mesh can be adapted. As mentioned before, such a space-time mesh is of dimension $d + 1$, which requires a 4-D mesh to address PDE's solved on a 3-D space domain.

The stabilization coefficients τ_{mom} and τ_{cont}, which weigh the contributions of the stabilization terms ST_{mom} and ST_{cont} with respect to the rest of the formulation, are functions of Δt and h_e, which are, respectively, the temporal and spatial lengths of the element. The values of these characteristic lengths are known to have a strong effect on the behaviour of the stabilized formulation (Mittal (2000)). For very small values of these lengths, the formulation can exhibit oscillations in the pressure field, as for the case of a standard Galerkin formulation, or oscillations in the velocity field at high Reynolds numbers. To avoid excessive stabilization, it has been suggested that, for anisotropic simplicial meshes, the most appropriate choice of spatial length h_e would be the minimum edge length for the simplex (Mittal (2000)). However, in this thesis a more conservative approach was chosen, in which the characteristic length was taken as

$$h_e = V_E^{\frac{1}{d}} \tag{5.27}$$

where V_E is the volume of the simplex and d is the space dimension. Further investigation would be needed in order to assess what is the most appropriate choice, but this is considered future work that is beyond the scope of this thesis. It is recognized that over-stabilization could lead to numerical solutions that are affected excessively by numerical viscosity, thus corresponding to lower Reynolds numbers than the one computed from the

physical parameters for the problem (Gresho and Sani (1998)). This is a drawback from which the mini-element of N'dri et al. (2002) does not suffer (there is no stabilization coefficient to set in this case), but this is without loss of generality for the unified space-time approach considered in this thesis.

5.4 Solution of Nonlinear Equations Using the Picard Method

The presence of the nonlinear convective term in the Navier-Stokes equations requires an appropriate treatment. Among the several available methods (Turek (1999)), it was chosen in this thesis to use the simplest one, namely a simple Picard iteration method (Langtangen (1999)), which will be briefly presented here.

The discretization of the Navier-Stokes equations leads to a nonlinear set of algebraic equations of the form:

$$\mathbf{A}\left(\mathbf{U}\right)\mathbf{U} = \mathbf{b} \tag{5.28}$$

This can be linearized by lagging the solution vector by one iteration to solve the linear system

$$\mathbf{A}\left(\mathbf{U}^{k}\right)\mathbf{U}^{k+1} = \mathbf{b} \tag{5.29}$$

iteratively by substituting $\mathbf{U}^{k} \leftarrow \mathbf{U}^{k+1}$ after each iteration. This process starts from a specified initial solution $\mathbf{U}^{0}$ and is repeated until either convergence is reached or the number of iterations reaches a maximum value. In this thesis, the convergence criterion was chosen as:

$$\frac{\left\|\mathbf{U}^{k+1} - \mathbf{U}^{k}\right\|}{\left\|\mathbf{U}^{k}\right\|} \leq 10^{-4} \tag{5.30}$$

As mentioned previously, the present approach uses the null velocity field (and pressure field) with suitable time-varying boundary conditions. Notice that, for the incompressible Navier-Stokes equations and the null initial solution, the first Picard iteration corresponds to solving a Stokes problem with the same boundary conditions. Because the boundary conditions are chosen to simulate the acceleration of the fluid from rest, then this choice seems congruent with the physical problem.

It is known that the convergence of the Picard method can become progressively slower as the Reynolds number is increased, and even suffer from oscillations. Nevertheless, this should not be a problem for the cases considered in this thesis, which correspond to relatively low Reynolds number flows ($Re \leq 800$). Faster and more sophisticated methods can be considered in future extensions of this work (Turek (1999), Lohner (2001)).

Chapter 6

Anisotropic Mesh Optimization Algorithms in 2-D, 3-D and 4-D

6.1 Introduction

This chapter presents the mesh optimization algorithms used in the mesh adaptation procedure to directionally adjust the number and locations of mesh points in order to better satisfy the error estimator.

The present approach makes two original contributions, which distinguish it from previous 3-D anisotropic mesh modification procedures (Tam et al. (1998), Tam et al. (2000), Belhamadia et al. (2004b), Bottasso (2004), Gruau and Coupez (2005), Li et al. (2005)). First, the mesh modification algorithms have been designed to operate in spatial domains of dimensions 2, 3 and 4, which paves the way towards the extension of mesh adaptation to the time-continuous space-time formulation in 4-D. Second, a novel mesh

smoothing algorithm based on the inscribed ellipsoid was conceived and shown to be successful in improving the quality of the mesh without compromising its anisotropy.

The presentation of the algorithms begins with the anisotropic measure of quality integrated in the mesh optimization procedure. The next section describes the fundamental mesh modification algorithms employed, followed by explanations of the manner by which they are combined together to form a mesh adaptation procedure driven by the error estimator, which was presented in Chapter 2. The mesh smoothing algorithms are presented afterwards and the chapter ends with the presentation of the pseudo-code for the meshing algorithms.

6.2 Anisotropic Element Quality Measure

Several of the mesh optimization algorithms presented in the remaining of this chapter utilize an anisotropic measure of quality for mesh elements. The choice of element quality measure is important, because it is used in the edge collapsing, edge swapping and point relocation operations, such that these mesh modifications are accepted only if they locally improve the quality of the mesh. As mentioned in Chapter 2, an appropriate element quality measure should be able to detect all types of problematic elements.

Since the primary objective of the mesh optimization presented subsequently is to modify the density of the mesh by adjusting the metric edge length, a quality measure was chosen, among the ones available in the literature (George (2001)), that includes the metric edge length among the quantities used to measure the quality. More specifically, the

element quality measure is defined as:

$$Q = \frac{1}{V_{UnitEdge}^2} \frac{\det(M) \cdot V_E \cdot |V_E|}{\left(\frac{1}{N_e} \sum_{i=1}^{N_e} L_{m_i}\right)^{2n}} \tag{6.1}$$

where n is the topological dimension of the element, $\det(M)$ is the determinant of the average metric on the element, V_E is the Euclidean volume given and $|V_E|$ is its absolute value, L_{m_i} is the metric edge length and N_e is the number of edges of the element. The measure of quality Q is normalized such that the minimum quality is zero and the maximum quality is 1.0; note that the Euclidean volume of a simplex of unit edge length is given by:

$$V_{UnitEdge} = \frac{\sqrt{n+1}}{n!\sqrt{2^n}} \tag{6.2}$$

and the signed volume of a d-dimensional simplex in a d-dimensional space is given by (Heckbert (1994)):

$$V_d = \frac{1}{d!} \det \begin{bmatrix} (\mathbf{x}_1 - \mathbf{x}_0) & (\mathbf{x}_2 - \mathbf{x}_0) & \ldots & (\mathbf{x}_d - \mathbf{x}_0) \end{bmatrix} \tag{6.3}$$

where each column of the $d \times d$ determinant is the difference between two vertices of the simplex. The non-signed volume of a simplex of topological dimension $n \leq d$ is computed by (Heckbert (1994)):

$$(V_n)^2 = \left(\frac{1}{n!}\right)^2 \det \begin{bmatrix} v(1,1) & v(1,2) & \ldots & v(1,n) \\ v(2,1) & v(2,2) & .. & v(2,n) \\ \ldots & \ldots & \ldots & .. \\ v(n,1) & v(n,2) & \ldots & v(n,n) \end{bmatrix} \tag{6.4}$$

where $v(i,j) = (\mathbf{x}_i - \mathbf{x}_0) \cdot (\mathbf{x}_j - \mathbf{x}_0)$. Note that equation 6.4 is used only for a simplex of topological dimension lower than the space dimension ($n < d$) and does not provide any information concerning the sign of the volume, in contrast to equation 6.3.

Also, the sign of the volume of the simplex V_E is preserved in equation 6.1 through the use of the absolute value to ensure that the measure of quality can detect elements that may become inverted during intermediate stages of the mesh optimization process (in which case the edge collapsing, edge swapping or point relocation would be rejected to prevent the creation of elements with negative volumes).

The metric used to evaluate the element quality, through equation 6.1, is a constant metric for the element taken as the arithmetic average of the metrics stored at the points of the element. This choice was made because it is faster to compute the quality using one metric rather than several (recall from Chapter 2 that computing the volume with a metric that varies on the element requires numerical integrations). The question can be raised as to whether a constant metric is sufficiently accurate for the current purposes. In the mesh optimization algorithms presented in the following sections, the decision to refine or coarsen the mesh is based on the measure of the metric length of edges, not on the basis of the element quality, which is only used to prevent collapsing an edge if this would result in deterioration of the mesh quality (this is discussed in more detail in the sections to follow). Furthermore, as the mesh adaptation process is repeated for several cycles, and the FEM solution is recomputed between cycles, the mesh would become more refined in regions where the error estimator requires it. Therefore, the metric variation over the element would tend to diminish, which would also reduce the difference between the quality measured with a constant average metric and that from a varying metric over the element. Consequently, as a compromise, it was chosen to use a constant metric for the evaluation of the quality, although a precise measurement of the impact on the accuracy of the measured

quality on the mesh optimization procedure is left as future work.

Following the technique presented by George (2001), the variation of this measure of quality for a triangle in a 2-D space has been plotted in Figure 6.1. In order to assess how the measure of quality varies when an element is deformed, a simple experiment was performed with the triangle having two fixed vertices located at $(0.0, -0.5)$ and $(0.5, 0.0)$ in the $x - y$ plane and a third vertex moving on the nodes of a 100×100 mesh within the intervals $0 \leq x \leq 2$ and $-1 \leq y \leq 1$. Isocontours of the element quality for different positions of the movable vertex have been plotted in Figure 6.1. As expected, when this vertex is located at (1.0, 0.0), the triangle is equilateral and its quality is equal to 1, whereas, when it is on the $y-$axis, the element has a zero volume and its quality takes the value 0. The quality of the element diminishes progressively when this vertex moves away from the location at which the triangle is equilateral.

Two variations of the same measure of quality were compared. The first one, corresponding to the left side of Figure 6.1, was computed with equation 6.1, whereas the one on the right was computed with the square root of the same quantity. By inspection of the plots for these two measures of quality, the quality for the one on the left shows a more rapid change as the moving vertex moves away from the location $(1, 0)$ for which the quality has a maximum value of 1. Therefore, the quality measure given by equation 6.1 was chosen instead of its square root because it is more sensitive to the variation of the shape of the element.

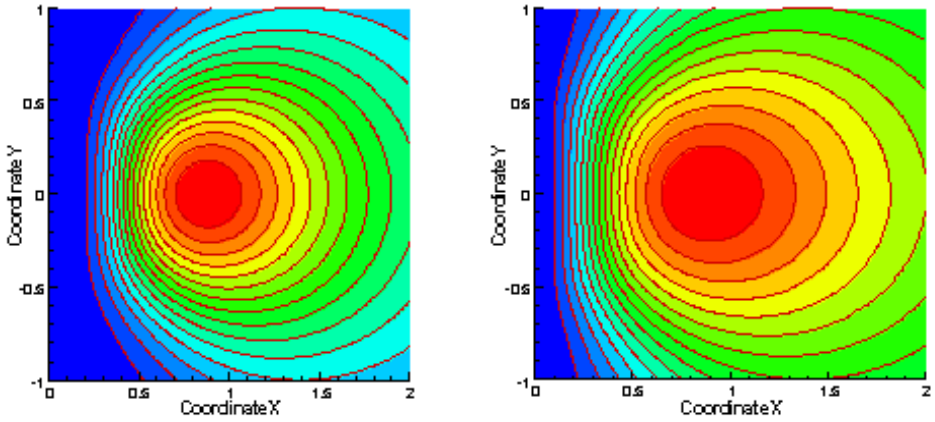

Figure 6.1: Isocontours of the anisotropic measure of quality given by equation 6.1 (left) and its square root (right). The quality is measured for a triangle with two fixed vertices at $(0.0, -0,5)$ and $(0.0, 0.5)$ and a third vertex at the position (x, y).

6.3 Mesh Modification Operators in 2-D, 3-D and 4-D

6.3.1 Edge Splitting

In order to increase the density of mesh points, a simple edge splitting procedure is used. Conveniently, edge splitting is very easy to implement in a dimension independent way to operate in 2-D, 3-D, and 4-D, both inside the mesh and on its boundaries.

The key idea to implement the edge splitting in an arbitrary dimension is to realize that all elements using the two points of the edges need to be split into two copies, as shown in Figure 6.2. The first element copy has the same mesh point ids as the original element to be split, but with the point id corresponding to the first edge point that is replaced with the point id for the new splitting point. Recall from Chapter 3 that each element stores its point ids in a tuple, so the point id to replace, here the point id for the first point of the edge, can be found simply by searching through the tuple of point ids for the element

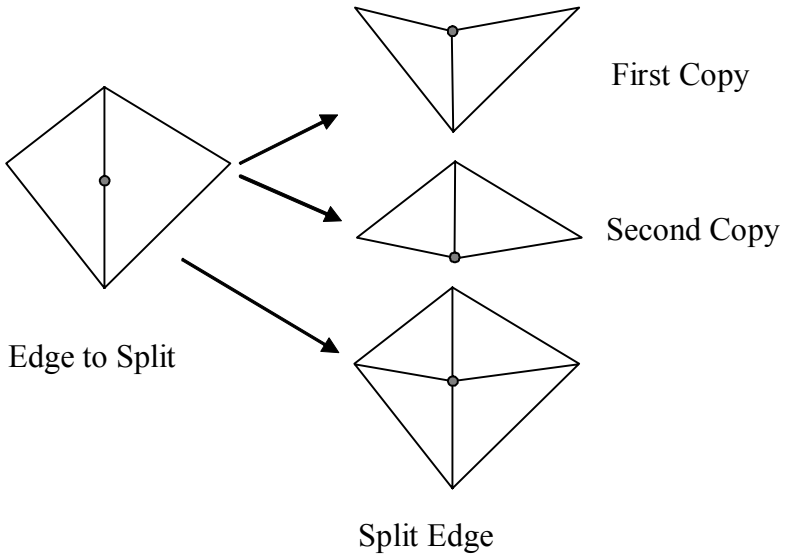

Figure 6.2: Illustration of edge splitting procedure by making two copies of each element to split and replacing one edge end point by the split point for each copy.

(which contains at most five points for a simplex in 4-D). Similarly, the second element copy has the same mesh point ids as the original element to be split, but the mesh point id corresponding to second edge point id is now replaced by the new splitting point id. Hence, with this simple approach splitting an edge in arbitrary dimensions is reduced to finding the list of elements that are neighbours of the edge, making two copies of each element, replacing the point id corresponding to the first edge point in one copy with the split point id and replacing the point id corresponding to the second edge point in the second copy. Conveniently, this procedure works equally for edges that are inside the mesh domain or on its boundaries and properly handles elements of all topological dimensions (recall that the topology of the mesh is stored using edges in 2-D, edges and triangles in 3-D and edges, triangles, and tetrahedra in 4-D).

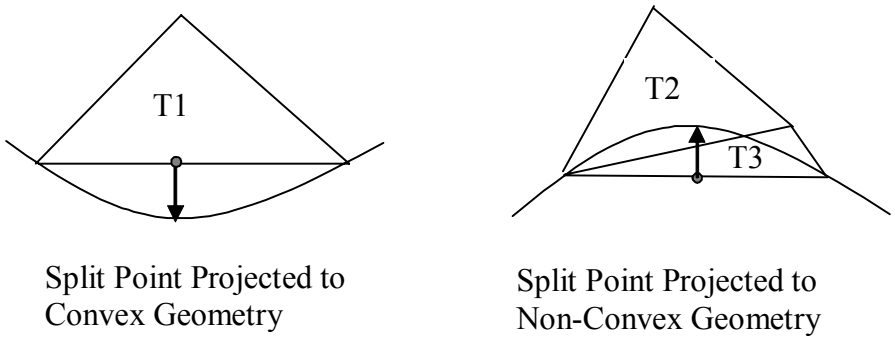

Figure 6.3: Illustration of splitting an edge on the boundary mesh. On the left, the split point is projected to the boundary of a convex geometry. On the right, splitting the triangle T3 near the boundary of a concave geometry by point projection would result in inverted elements.

When edge splitting is performed on the boundary of the mesh, this simple procedure needs to be enhanced to ensure that the new splitting point is located as closely as possible to the physical boundary of the geometry. For convex geometries, the splitting point is simply projected to the geometrical boundary, as shown on the left side of Figure 6.3, using the projection procedure that was described in Chapter 4 and the splitting procedure explained previously.

When applied to non-convex geometries, however, simple projection of the splitting point on the geometrical boundary may produce interior elements that are inverted such that their volume would become negative. This problem is illustrated in Figure 6.3, in which projecting the splitting point to the non-convex geometry after splitting the triangle T3 would result in inverted elements. Some authors have addressed this problem by first splitting all edges that need to be split, then projecting the splitting point to the geometry

and finally correcting the shapes of the problematic elements (Li et al. (2005)).

Here, a different approach is used, in which, before it is completed, splitting is first simulated to determine whether it would lead to invalid elements. Recall that element splitting operates by making two copies of each element 6.2. The locations of the points for the first element copy can be simulated by temporarily displacing the second point of the edge to the splitting point location. The element quality is measured for all elements, using the formula in equation 6.1, in this configuration and if any elements have a quality below a specified threshold, then the split is rejected and the edge end point is restored to its initial location. The second copy can be simulated by moving the first edge end point, after the first one has been restored to its original location.

As explained in a subsequent section, the splitting is applied to longer edges first. It was found by testing this splitting simulation strategy on non-convex geometries that this strategy had a tendency to prevent the splitting of edges too frequently on non-convex geometries whose curvatures were large by comparison to the local mesh size. In other words, when the longest edges were not split first, the simulated approach was found to prevent the surface mesh from being sufficiently refined (such that most edges would be below a specified metric edge length threshold).

6.3.2 Edge Collapsing

The edge collapsing algorithm, illustrated in Figure 6.4, requires more attention than the splitting algorithm, because it needs to be constrained to avoid damaging the topology of the boundary mesh in some cases and also to avoid creating inverted elements inside the domain as well as near the boundary.

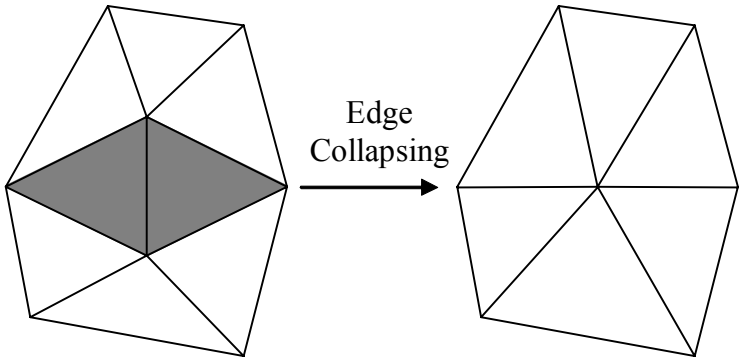

Figure 6.4: Illustration of edge collapsing.

Definition 4 (POINT TOPOLOGICAL DIMENSION) *The topological dimension of a point in a finite element mesh is the topological dimension of the element*[1]*, among the elements using that point, that has the minimum topological dimension.*

The collapsing algorithm begins by determining whether the edge collapsing is topologically admissible. Two tests are made. First, the topological dimension of the end point (as defined above) of the edge to be removed is compared with that of the end point to be retained. If the topological dimension of the point to be removed is lower than that of the point to be retained, edge collapsing is rejected. Illustrations of an edge that is not topologically collapsible and one that is topologically collapsible are shown in Figure 6.5.

When the topological dimensions of both end points of an edge are equal and lower than the space dimension, an additional test must be performed to determine whether the edge is on the boundary mesh or in the interior of the mesh. In this case, edge collapsing can

[1]The topological dimension of an element can be defined as the number of independent parameters needed to locate a point on that element. Hence, the topological dimension of a vertex is 0, an edge is 1, a triangle is 2, a tetrahedral element is 3 and a simplex in a 4-D space is 4.

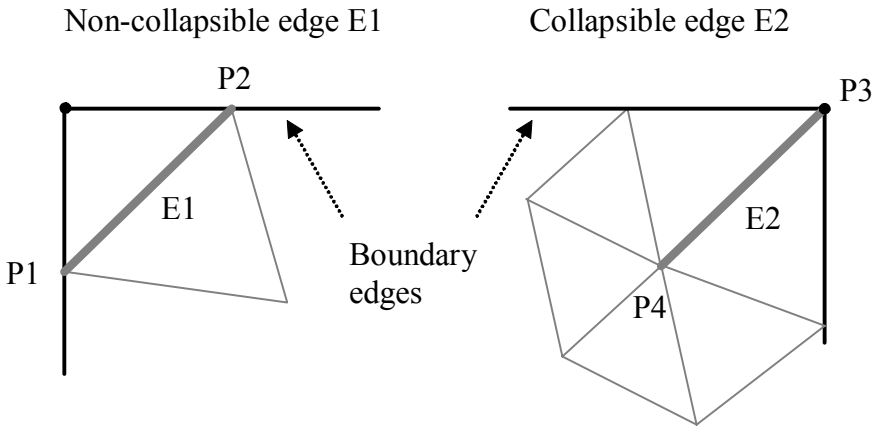

Figure 6.5: Illustrations of an edge that is not topologically collapsible and an edge that is topologically collapsible.

only be attempted if both end points of the edge belong to the same topological patch (please refer to the definition of topological patch in Chapter 4). This condition can be verified by searching the list of elements that are neighbours of the edge under consideration for collapsing to find an element of topological dimension equal to that of the point. If such an element is found, then the edge would be inside the topological patch, which means that it is on the boundary of the mesh and edge collapsing can proceed to the next step. If no such element is found, then the edge collapsing must be rejected because this is an edge that, although within the volume of the mesh, has end points that are located on the boundary mesh. Collapsing a volume edge that has end points belonging to the boundary mesh would alter the topology of the mesh and must be rejected, as illustrated on the left side of Figure 6.5.

After collapsing of an edge is found to be topologically valid, an additional test needs to be made to ensure that collapsing this edge is not going to lead to inverted elements

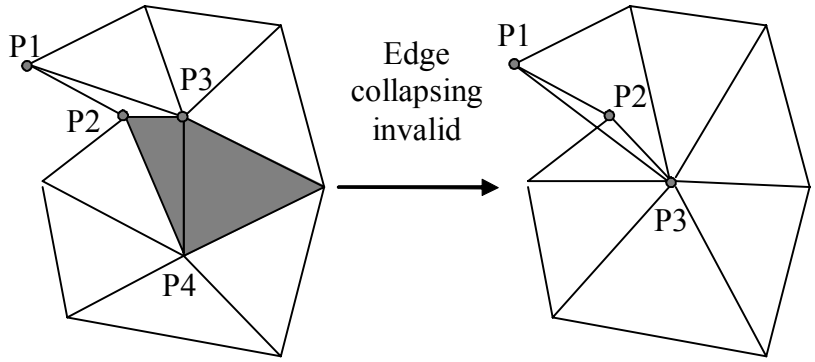

Figure 6.6: Illustration of an edge (P3,P4) that cannot be collapsed because it would result in the triangle (P1,P2,P3) being inverted.

or elements of unacceptably low quality. Inverted elements can result from collapsing an edge when the envelop formed by all the elements using at least one point of the edge is non-convex, as shown in Figure 6.6. The approach chosen here is to move the points of the edge to the location where they would be left after the edge collapsing and loop over elements that are not to be deleted (namely the ones containing either of the end points of the edge but not both) to compute their quality in this position, which simulates the configuration that would result after the collapsing. If one element has a quality that is below the desired threshold, then the collapsing is rejected and the points are restored to their original locations.

Conveniently, this collapse simulation not only prevents the creation of inverted elements, but also offers several other benefits. It can be used as a condition to accept the collapse only if the anisotropic quality of the elements satisfies a prescribed threshold or if the quality of the mesh improves (by using the minimum quality among the elements affected by the collapsing as the threshold). Furthermore, this strategy for verifying that

edge collapsing leads to elements of an acceptable quality can be easily applied to curved geometries. The point to preserve after the collapsing is simply projected to the geometry before the simulated collapsing is performed to verify that the quality is acceptable. Finally, notice that this edge collapsing algorithm scales to higher dimensions without any special modification that would be dimension specific.

6.3.3 Simulated Edge Swapping

The most challenging algorithms to scale to higher dimension are algorithms used to improve the topology of the mesh. In the present work, the choice was made to use an *edge swapping algorithm*. To facilitate the implementation of such an algorithm in 4-D, it is important to realize that, unlike fundamental mesh modification algorithms, an edge swapping algorithm can be decomposed into a sequence of two simpler mesh modification algorithms, more specifically into an edge splitting algorithm followed by an edge collapsing one. This compound algorithm is illustrated in 2-D in Figure 6.7 and in 3-D in Figure 6.8.

The first step in the simulated swap procedure is to split the edge to swap at its mid-point to create a temporary split point, which is labelled PT in figures 6.7 and 6.8. This temporary split point PT is connected to several edges, two of which have among their respective list of points the two points of the edge to swap. As collapsing either of these two edges would recover this initial configuration before the edge was split with the addition of point PT, only the other edges are possible candidates for collapsing for the combined edge splitting and collapsing to be equivalent to swapping the initial edge.

Considering that the objective of an edge swap is to improve the topology of the mesh, this operation is performed only if a combination of split-collapse is found such that

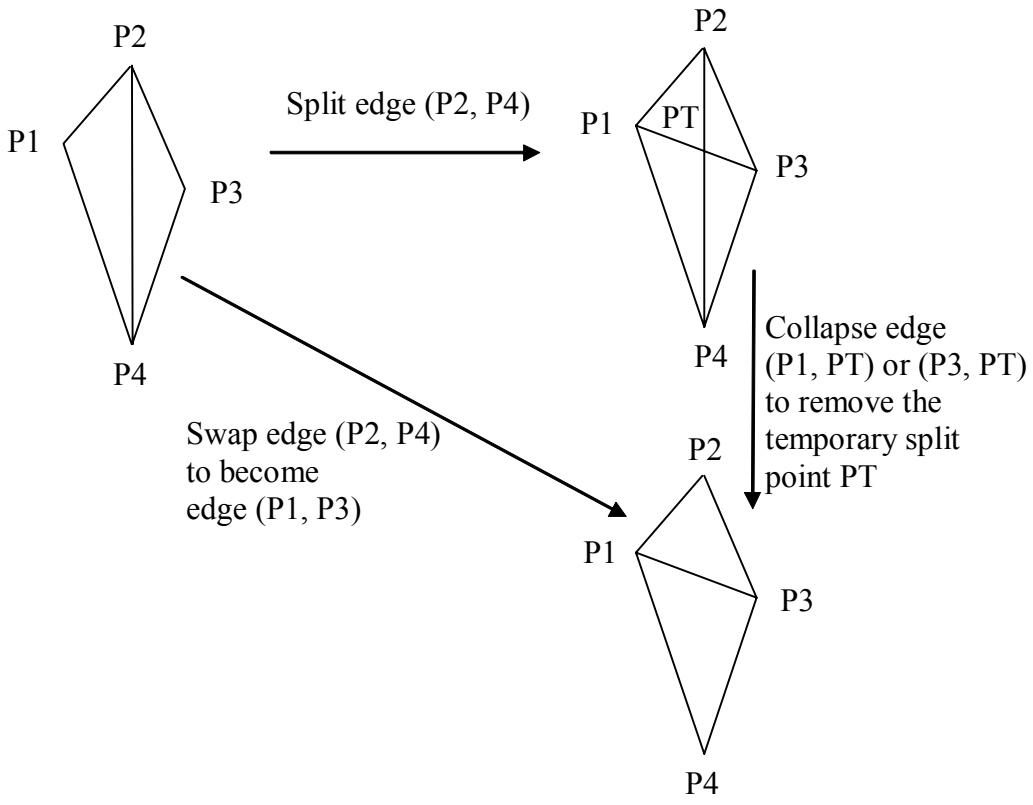

Figure 6.7: Illustration of a simulated edge swap in 2-D.

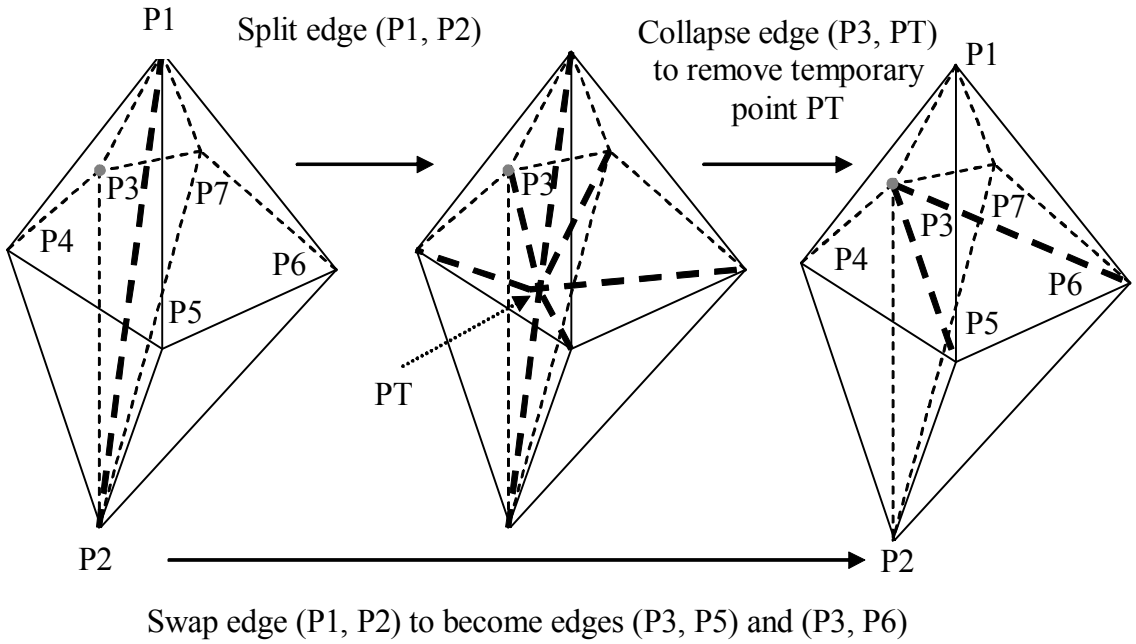

Figure 6.8: Illustration of a simulated edge swapping in 3-D for edge (P1, P2) by splitting edge (P1, P2) and collapsing edge (P3, PT) to remove the tempory split point PT.

the quality of the initial mesh is improved. In order to measure this, the minimum quality of the elements that are neighbours to the initial edge to swap is first computed. Then, an attempt can be made to collapse any of the potential candidate edges around the temporary points with the exception of the ones that would recover the initial configuration before the edge split.

The list of elements to test for the edge collapsing differs depending on whether edge collapsing is performed by leaving the end point to preserve at its initial location or whether this end point is relocated to the mid-point of the edge to collapse (point relocation). If the end point is not relocated, then the list of elements to verify for quality improvement includes all elements attached to the temporary point, except for the ones that are also connected to the point to preserve (the elements that could be deleted do not need to be tested). If the end point is moved to the mid-point of the edge, then the list of elements to verify would be the same as for the edge collapsing presented in the previous section. It was found by experimenting with the test cases presented in Chapter 8 that swapping with point relocation is more likely to successfully improve the quality of the mesh than without it.

It is necessary to reject the edge swapping if it does not improve the quality of the mesh. Consequently, the temporary point that was inserted in the mesh during the simulated edge swap must be removed for the mesh to recover its initial configuration. This is accomplished by collapsing one of the edges formed by the temporary point and either of the previous end points of the initial edge to swap. The choice of end point is not important as long as the associated edge is topologically collapsible, which is always the case (because

the temporary splitting point is added on the edge to swap so it can only be of a topological dimension higher than or equal to that of the point with the lowest topological dimension). Furthermore, the test for quality improvement in the edge collapsing is skipped in this case, so it cannot prevent the edge collapsing from recovering the initial configuration (there is no need to prevent it because the objective of this edge collapsing is to recover the initial configuration that was of better quality than what would result from the rejected edge swap).

Because edge splitting and edge collapsing are implemented to scale to an arbitrary dimension, this simulated edge swapping is also scaled to an arbitrary dimension. Furthermore, because both edge splitting and edge collapsing are implemented to work in the interior of the volume mesh or its boundary, it follows that the simulated swap is also capable of modifying the boundary mesh. To the best of our knowledge, no previously presented meshing algorithms have been shown to be able to modify a boundary mesh in a dimension higher than 3. The existing meshing algorithms that operate in higher dimensions are usually restricted to operate on a set of points with a mesh that contains only volume elements of topological dimension equal to that of the space dimension, and they do not keep track of a boundary mesh (see Chapter 2).

The computation cost of the previously presented simulated edge swap would clearly be higher than that of specialized operators that work only in 2-D or 3-D, because the former requires extra steps, but the present objective was to ensure that this operation applies equally to 2-D, 3-D and 4-D without further modification. By implementing the edge swapping this way, it was found possible to test it first in 2-D and then in 3-D and

finally to run it in 4-D without problems (some examples of 4-D meshes will be shown in Chapter 8).

It should be noted that a face swap can also be implemented with a similar strategy. This would require first to split the face and then to collapse an edge not having a point belonging to the face. As for the case of the simulated edge swap, the simulated face swap could be rejected if it is found not to improve the quality the affected mesh elements. If the quality does not improve, then an edge formed by a temporary split point on the face and any of the previous points used by the face can be collapsed to recover the initial configuration. This simulated face swap also scales to arbitrary dimensions and works on the boundary mesh as well. It was implemented and tested, but not used in the current version of the mesh optimization procedure presented in the next section.

6.4 Mesh Optimization Procedure

6.4.1 Overview

The objective of the mesh optimization procedure is to adjust the density of the input mesh while maintaining a mesh quality and an agreement between the boundary mesh and its associated geometry that are suitable for the finite element method (or other numerical simulation approaches that can use a finite element mesh). The density of the mesh is measured locally along the edges by measuring the length of the edge in the transformed space. The quality of element is also measured in the transformed space so that this procedure results in meshes that are anisotropic when the metric field is anisotropic. The mesh optimization procedure proceeds by splitting edges whose length exceeds some

threshold and collapsing edges whose length is shorter than some other threshold, while improving the mesh quality using a combination of edge swapping and mesh smoothing. The subsequent sections discuss in more detail the rationale for each of the stages of the mesh optimization procedure.

6.4.2 Target Mesh Size

This first step in the mesh optimization procedure is to determine the final target metric edge length used to specify a threshold for the edge splitting and edge collapsing operation, which indirectly controls the mesh size. In the context of mesh adaptation, the target mesh size is driven by the requirement of the error estimator from which a metric is computed at each point of the mesh, as previously explained in Chapter 2. Note that in the context of mesh generation, a mesh could be optimized using a metric field constructed from the curvature of a geometry rather then an error estimator and the procedure would remain the same (Borouchaki et al. (1997)).

For mesh adaptation, the metric constructed from the error estimator prescribes a relative desired mesh size and orientation through the metric at each mesh point, but not an absolute mesh size. What is meant by relative is that regions of the mesh with a higher interpolation error need to be refined to reduce that error, whereas regions of low error can be coarsened to reduce the number of mesh points. This would result in a more uniform distribution of the error throughout the mesh, but does not specify the global target error which will control the final mesh density. This is a recognized weakness of mesh adaptation, because it requires the user of a mesh adaptation code to specify an arbitrary parameter. Further research is needed on the subject (Alauzet et al. (2006), Alauzet et al. (2007)).

In this thesis, a simple error reduction approach is chosen, in which the average metric edge length is computed on the mesh, without scaling the metric field with a constant factor (Dompierre et al. (2002)), and the target metric edge length is set as the average metric edge length multiplied by a reduction factor varying in the interval $]0, 1]$. With this strategy, it is convenient to start the mesh adaptation process with a coarse mesh, on which the solution can rapidly be computed, and reduce the discretization error progressively as the mesh adaptation increases the mesh density and the FEM solver computes a more accurate solution on the adapted mesh. The coupling of the mesh adaptation and the FEM procedure is described in Chapter 7.

Another problem that needs to be addressed to complete the specification of the target metric edge length concerns the compatibility between this requirement for the edge splitting and edge collapsing procedures. More specifically, an edge that is considered too long and split could create two smaller edges that could be found to be too short and need to be collapsed. This kind of oscillation between refining and coarsening results in a loss of efficiency and could even prevent the overall procedure from terminating if an additional stopping criteria was not used (such as a maximum number of times that the edges of the mesh are traversed to determine if they need refinement or coarsening).

This problem can be significantly reduced by setting the threshold for refinement to a metric edge length that is slightly more than twice the threshold for collapsing. This way, an edge split is only performed if it does not produce a metric edge length that is short enough to necessitate edge collapse. In practice, some destructive interference between the two algorithms may still exist, because, as the edges are split, the metric at the split point

is interpolated from the background mesh and the metric lengths of the two segments are not exactly half of the previous segment. In any case, some interference between splitting and collapsing would always occur, because edges are swapped and points are moved and that also modifies the metric length of edges in the mesh.

In order to give priority to regions of the mesh which require refinement or coarsening the most, the thresholds for refinement and coarsening are progressively adjusted at each global iteration of the mesh adaptation procedure. This is done following a simple heuristic strategy, in which the starting value for the threshold for collapsing is taken to be close to the minimum edge length in the mesh and the starting value for the threshold for refinement is taken to be close to the maximum edge length and these threshold values are then progressively modified to approach the final target threshold as Figure 6.9 illustrates.

The specific approach chosen is to linearly reduce the threshold for edge splitting and linearly increase the threshold for edge collapsing during the first half of the specified maximum number of global iterations over the edge splitting, edge collapsing, edge swapping and mesh smoothing phases. For the second half of the global iterations, the final target thresholds for both the refinement and coarsening are used. Several other strategies could be devised and a more in-depth investigation could be done to quantify the impact of this strategy on the mesh adaptation procedure. Some experiments using this strategy are discussed for the test cases of this thesis in Chapter 8. The number of global iterations required to reach a state at which no edge needs to be split or collapse is problem dependent; this is discussed further in Chapter 8.

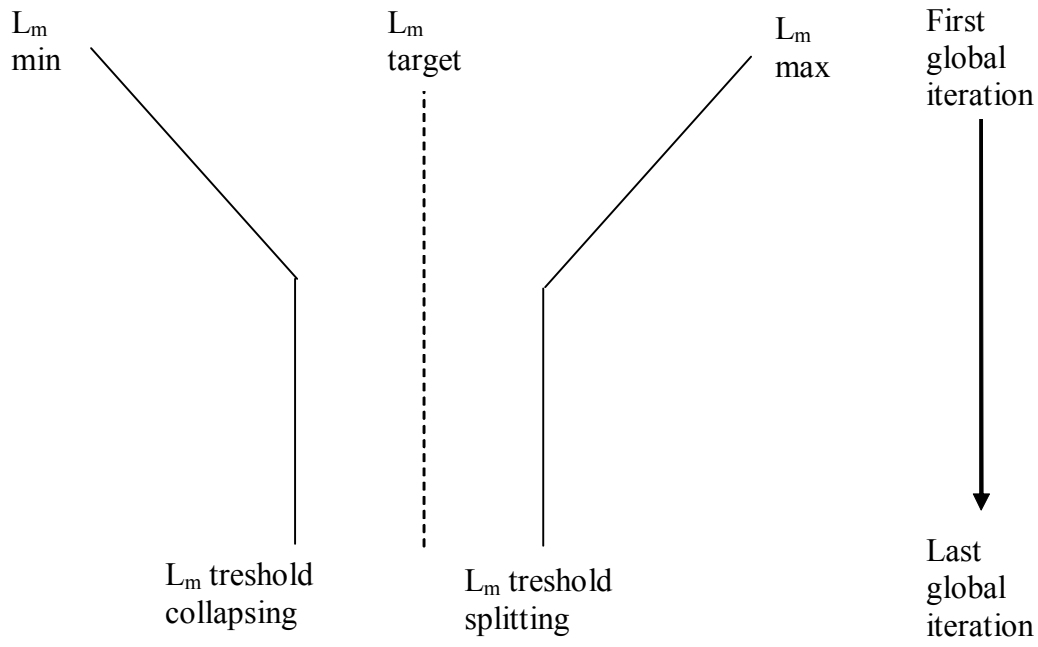

Figure 6.9: Illustrations of the variations of the metric edge length treshold for collapsing, on the left, and refinement, on the right, as the global adaptation iteration progresses from the first iteration on the top to the last iteration at the bottom.

6.4.3 Refinement

Several combinations of refinement, collapsing, swapping and smoothing could be employed. As edge splitting can create unacceptably small edges, namely leading to elements with a very low quality, edge splitting is performed first and then it is followed by edge collapsing to ensure that small edges are removed before an attempt is made to improve quality by swapping edges.

Furthermore, because the location of the splitting point only takes into account the distance from the end points of the edge and not distances from the other neighbouring points, it was decided to relocate the splitting point, using the procedure described in section 6.4.6, right after the edge split in order to improve the local mesh quality as much as possible. This approach was found to reduce the number of poor quality elements that may be created when point splitting results in points of an element being nearly coplanar, colinear or cocircular (in 3 and higher dimensions recursively splitting edges can produce a simplex with points that are lying on the same circle). Reducing the creation of poor quality elements during the split phase through the use of local mesh smoothing subsequently reduces the amount of work needed by the edge swapping algorithm to improve the mesh quality. The smoothing algorithm used is presented in section 6.4.6.

As previously mentioned, the splitting is prevented on the boundary of non-convex geometries if splitting this edge would lead to elements of very poor quality or negative volumes. In the context of the entire mesh optimization algorithm, splitting the longer edges first is important because, for non-convex geometries, problems may arise when the boundary edge is much longer than edges of the neighbouring elements (see Figure 6.3).

Splitting interior edges before the boundary edges is more likely to introduce elements of very low quality, and, for this reason, it is preferable to create smaller edges on the boundary mesh before creating them in the interior of the mesh. Recall that when edges of the boundary mesh are split, the split point is projected on the geometry, which affects the shape of the elements that were split. The longer the distance between the splitting point before and after its projection to the geometry is, compared to the length of the other edges of the elements that are split, the more likely it is that inverted or low quality elements will be created.

Rather than considering for splitting all candidate boundary edges first and then inner edges afterwards, it was decided to sort the edges that are longer than the current threshold for splitting in decreasing order of metric length. This strategy of sorting edges to split the longest first was found to work well in practice, for example on the test case for the flow behind a cylinder presented in Chapter 8, when combined with the simulated split to prevent the creation of inverted elements. The sorting is done using a radix sort (Terdiman (2000)), which is linear in the number of entities to sort, so the linear time complexity of the mesh optimization algorithm is not lost by adding the sorting step.

It is also important to highlight that splitting edges by priority assigned to their length has a favourable impact on the quality of the mesh, compared to the same mesh optimization procedure without the sorting step. If all edges of the mesh are sorted and the longest edge is split first, then this longest edge would not only be the longest in the mesh globally, but it would also be the longest locally. Furthermore, the longest edge of a simplex lies, in general, opposite to the largest solid angle. Consequently, when this edge is

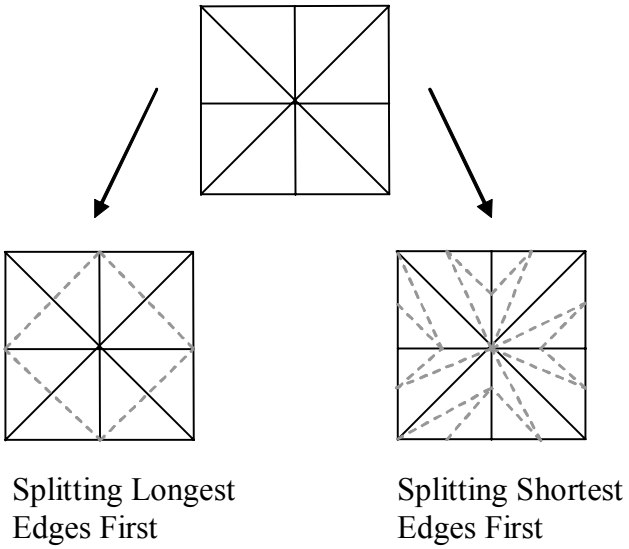

Figure 6.10: Illustration of the impact of splitting the longest edge first and splitting the shortest edge first.

split, the largest angle receives at least one new incident edge and splitting the longest edge results in reducing the largest solid angle of the simplex, which necessary gives a better mesh quality than the case of generating a new incident edge (edges) for a smaller angle. The impact of sorting on element quality can be illustrated by a simple example, shown in Figure 6.10.

Refinement strategies based on inserting points on the longest edges first have been explored previously for 2-D isotropic meshes in combination with Delaunay based methods (Rivara and Hitschfeld (1999)) and edge refinement improvement methods (Plaza et al. (2004)) and both approaches were found very effective in improving the quality of the mesh. Here, a similar idea was used, but in the context of an anisotropic edge based method in 2-D, 3-D and 4-D and using a sorting approach rather than locally searching the mesh for the longest edge to split. Further improvements are certainly possible along this

direction, but simply sorting the list of edges to split was found to be significantly beneficial, especially if the initial mesh contains very small interior angles that would otherwise make it more difficult for the edge swapping procedure to improve the quality of the mesh. It also worth noting that performing a radix sort on the edge list is very fast in comparison to other aspects of the mesh optimization procedure; consequently, this step does not add significantly to the computational cost.

To prevent unnecessarily excessive mesh refinement, it is important to create a list of edges that are longer than the current refinement threshold and only split those edges before performing other mesh operations. Splitting the edges of the mesh and dynamically adding newly created edges that are also longer than the threshold to the list of edges to split could lead to over-refining the mesh in some region of the domain before other mesh modification algorithms (edge collapsing, edge swapping and mesh smoothing) have an opportunity to improve the quality of the mesh.

6.4.4 Coarsening

Mesh coarsening is enacted as the second stage of the overall procedure to remove excessively small edges that might have been created by the edge splitting algorithm or that might have been present in the initial mesh.

As explained previously, edge collapsing needs to be constrained to prevent the creation of elements that are inverted or of very poor quality. It is also important to realize that the operation of first splitting edges and then collapsing edges has similarities with the simulated edge swap that operates by first splitting an edge and collapsing another one to perform an edge swap. Recall that such edge swaps are desired only if they improve element

quality, so it seems advisable to also constrain collapsing, such that it is applied only if it improves quality. However, this condition may become too restrictive when it is applied to regions with very high mesh quality. To avoid this, the following simple heuristic strategy is used. The quality of the initial configuration of elements affected by edge collapsing is measured and the threshold to accept the edge collapsing is defined as the minimum of this initial quality and a low quality threshold, whose value has been set to 0.1 (the threshold 0.1 was chosen after experimenting with some of the test cases of this thesis, but no claim is made that this is optimal). Consequently, if the element quality in the original configuration is lower than 0.1, then the collapsing is performed only if the quality of the element is improved. However, if the quality of the element in the original configuration is higher than or equal to 0.1, then the edge collapsing is accepted if no newly created element has a quality that is lower than 0.1. With this simple modification, the collapsing avoids excessive deterioration of the mesh quality and is given enough opportunity to adjust the density of the mesh. Note that it is important to have a quality threshold significantly higher than 0.0 to avoid the generation of nearly inverted elements by edge collapsing.

Another problem that may arise from multiple edge collapses in a region of the mesh is creating a mesh in which the number of edges incident to one particular point is much higher than the average in the mesh. This results in the elements incident to that point having very small interior angles and therefore a poor quality. In order to reduce this possibility, a marking strategy that prevents both end points of an edge from being removed in the same iteration on the list of edges to collapse has been employed. This strategy is similar to one presented by Li et al. (2005), which evaluates all edges terminating at each

mesh point and collapses the shortest one, if it is found to be shorter than the threshold. In the present approach, however, it was chosen to build a list of edges that are in the range for collapsing and sort that list to collapse the shortest edge first. When an edge is collapsed, the end point of the edges incident to the point left after the collapse are marked as blocked for this iteration. After one complete pass on the list of edges to collapse has been performed, then the point marking is reset and the edges that have been blocked from collapsing in the first pass are considered for collapse again following the same blocking strategy. This strategy effectively reduced the creation of small angles during the edge collapsing phase.

In contrast to edge splitting, edge collapsing affects other edges of the mesh by elongating them. To avoid excessive coarsening of the mesh, the length of edges that are removed from the list of edges to collapse are reevaluated to ensure that they still fall in the range for collapsing (they might have been stretched sufficiently, after they were inserted in the list, by the impact of collapsing other neighbouring edges). Finally, locally smoothing the mesh by relocating the point that remains after the edge collapsing was found to interfere with the marking strategy, so such smoothing was not used in this case, in contrast to the edge splitting case. Notice that the edge collapsing includes a test to reject it if it would deteriorate the mesh quality too much, as described previously, whereas the edge splitting contains no such tests for interior edges.

6.4.5 Topology Improvement

The objective of the topological improvement is to increase the quality of the mesh without affecting its density and with minimal, if any at all, impact on the location of the

points. This is done using the edge swapping algorithm presented earlier.

The topological improvement phase begins by traversing the list of elements to evaluate their quality and identify the ones that potentially need improvement. This way, edges of elements with a low quality can be marked for swapping, while edges of elements for which edge swapping is unlikely to improve the mesh quality are marked as not for swapping.

Because edge swapping is less likely to improve the mesh when the quality is very low, two variants of the edge swapping method are used. If the quality is lower than 0.5, but higher than 0.1, the edge swapping method used follows conventional edge swapping algorithms (Frey and George (1999)), namely the edge collapsing step uses the end point of the edge to collapse as the final point location. If the quality of the element is lower than 0.1, the edge collapsing step of the edge swapping uses the mid-point option. This is functionally equivalent to swapping the edge with the conventional approach, but with a build-in point relocation which is done before evaluating whether the quality of the combined procedure improves the quality of the mesh sufficiently to be accepted. This was found to be more successful in removing elements that are of very low quality by monitoring the minimum quality of the mesh for several test cases with this option turned on or off. However, in the general case, when element quality is not too low, it is preferable to use the conventional edge swapping to avoid moving the node to improve quality, because this affects the metric edge length and may cause an edge whose length was previously close to the target length to fall into the range for splitting or collapsing. Care needs to be taken when combining meshing algorithms to avoid compromising the convergence of the global mesh optimization

procedure.

It is known that the number of possible edge swappings rapidly increases when the number of volume edge neighbours around a point increases (Frey and George (1999)). Seeking among all possible edge swap cases the one which successfully improves the mesh quality or the one that improves the mesh quality the most is very expensive computationally. Hence, it was found preferable to use a simple heuristic procedure to identify one case that is most likely to succeed. More specifically, the end point for the edge to collapse in the second step of the simulated edge swap is chosen as the one among all the possible choices that is the closest to the temporary split point (the distance measured is the standard Euclidean distance, not the one using the metric field). The rationale behind this choice is that the closest point among the set of points around the edge to swap is approximately the one with the largest solid angle. This point is a suitable candidate to receive the newly created edges by the swap procedure, because this operation would be more likely to succeed in improving the quality than using a point having an associated smaller solid angle. This simple approach was found to work well in practice and is significantly less expensive computationally than trying all the possible edge swaps, especially in 3-D and 4-D since the number of possible edge swap increases with the space dimension (there is only one swap possibility in 2-D). Further investigation is necessary to determine if this compromise is sufficient, especially in 4-D.

Another important aspect to highlight concerning the mesh quality for anisotropic meshes is that some anisotropic metric fields can naturally lead to meshes having a large number of edges incident to a point, which is generally considered as a sign of a poor mesh

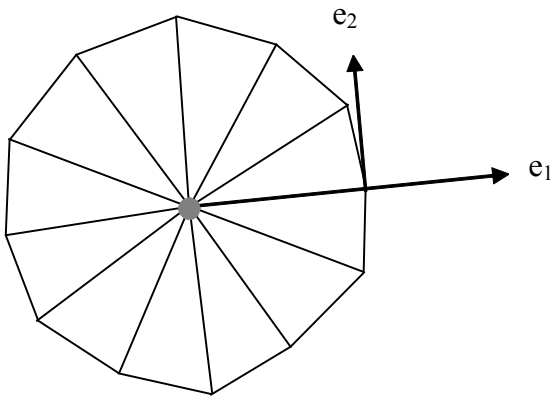

Figure 6.11: Illustration of a simple spoke wheel mesh with a large number of edges incident to the centre point. The eigenvectors for a point on the rim of the wheel are chosen with the first eigenvector in the radial direction and the second eigenvector in the tangential direction.

quality (and leads to increased storage space for sparse matrices, which are frequently used in FEM). This can be illustrated by the simple example of a spoke-wheel-like mesh shown in Figure 6.11.

In this spoke-wheel-like mesh, all the elements are incident to a single point at the centre of the wheel. The edges of the mesh forming the spokes of the wheel are chosen as having a length of one and the metric at the center is the identity matrix. Then, the metric associated to each point on the rim of the wheel can be chosen to have one eigenvector along the spoke of the wheel and the other one tangent to the rim. If the eigenvalue of the metric corresponding to the spoke of the wheel is unity, then the eigenvalue corresponding to the tangent eigenvector can be chosen such that the edges having both points on the rim have a metric length of one. In this example, the number of edges on the rim can be increased arbitrarily but the metric edge lengths in the mesh will always be one. Hence, the

metric quality measured in the transformed space would never detect any need for swapping, because the quality is high (actually, it has the maximum possible value of 1.0) and yet the number of edges incident to the centre of the wheel can become arbitrarily large.

One strategy to avoid this problem that has been employed by previous authors is to adjust the variation of the metric field in the mesh (Borouchaki et al. (1998), Li et al. (2004)). Here it was chosen to perform a second pass of edge swapping on edges that are incident to points having a higher number of incident edges than the average in the mesh (this is generally referred to as degree relaxation (Frey and George (1999))). In 2-D, the number of expected edges incident to a point can be computed from Euler's formula (O'Rourke (1998)). However, it was chosen to compute the average number of edges incident to a point in the mesh and use that as a threshold, because it also works for 3-D and 4-D. A possible improvement of the present approach would be to modify the edge swapping to select as the receiving point for the collapse the point with the lowest degree, but to preserve the anisotropy using the same criterion of quality improvement. One may further speculate that another improvement would be possible by accepting the swapping if it locally reduces the degree of the mesh but without decreasing the mesh quality too much; this possibility is left for future work.

6.4.6 Smoothing

The last phase of the mesh optimization procedure is the mesh smoothing. As described in section 6.4.3, when an edge is split, the point inserted in the mesh at the mid-point of that edge is relocated, using the mesh smoothing technique to be described in the next section, before splitting other edges or modifying the mesh through the edge

collapsing and edge swapping phases. However, the edge splitting may affect only certain regions of the mesh, where the metric edges are too long, and not others. Consequently, a mesh smoothing pass is performed to relocate the mesh points to attempt to locally improve the quality of the elements, which, in case of a point relocation strategy, does not change the topology of the mesh.

6.5 Mesh Smoothing Based on Inscribed Ellipsoid

In the context of anisotropic meshing, previous authors have presented mesh smoothing algorithms that seek to equidistribute the metric length of edges (Tam (1998), Dompierre et al. (2002)). Although this is in harmony with the objective of the mesh optimization procedure that splits long edges and collapses short edges to approach a specified target metric edge length, these smoothing strategies do not improve the quality of the mesh elements and even need to be constrained to prevent the creation of inverted elements (Tam (1998)).

Mesh smoothing based on optimization strategies exist (Freitag (1997), Diachin and Knupp (2006)), but are considered to be too computationally expensive by Bottasso (2004) who compared them with a metric edge based approach. Other strategies based on linear programming paradigm may be more efficient than optimization strategies, but were developed only for isotropic meshes (Amenta (1999)). Several other methods exist (Baker (2002)), some authors have also compared more common methods (Hyun and Lindgren (2001)), but most of them are for isotropic meshes. Smoothing methods that are based on improving the quality of the elements through the solid angle were recently introduced in

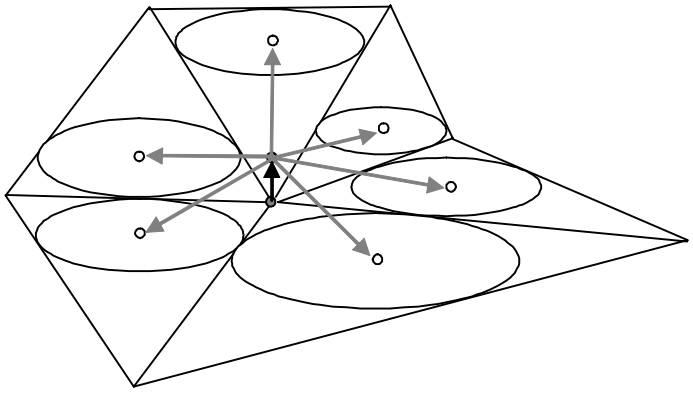

Figure 6.12: Illustration of the point relocation strategy based on the average inscribed ellipsoid of the elements incident to a point. The dark arrow shows the potential displacement from the previous point location to the new trial point location.

2-D (Xu and Newman (2006)), and could possibly extended to 3-D or higher dimensions, but to the best of our knowledge they are currently limited to isotropic meshes (which does not preclude their future extension to the anisotropic cases).

In the context of the present thesis, it was necessary to devise a mesh smoothing strategy that improves the anisotropic quality of the mesh, operates in 2-D, 3-D and 4-D, is compatible with the edge based optimization procedure presented previously, is relatively simple to implement and is efficient in terms of computational cost compared to other mesh optimization algorithms. This anisotropic mesh smoothing method can be summarized as a strategy that seeks to relocate a mesh point to the average location of the inscribed ellipsoid centre of the elements incident to that point. It is illustrated in Figure 6.12.

This smoothing approach based on the inscribed ellipsoid was inspired by the realization that the inscribed radius, defined as

$$r = \frac{d \cdot V}{\sum_{i=1}^{d+1} S_i} \tag{6.5}$$

where d is the space dimension, V is the volume of the simplex and S_i is the surface of the face i of the simplex, is closely related to a measure of element quality that is frequently used, namely the non-dimensionalized volume to surface ratio of the element (George (2001)). The inscribed radius is itself proportional to the ratio of the volume of the element and the sum of the sub-volumes of the faces of that element. Hence, it was postulated that a strategy based on improving the equidistribution of the inscribed radius would also to some degree improve the mesh quality.

Such methods require the computation of the inscribed center of the ellipsoid for the elements of the mesh. Recall that the inscribed sphere of a simplex is the sphere of maximum radius that is tangent to all the planar faces of the simplex and can be computed from:

$$\begin{aligned} \mathbf{n}_1 \cdot (\mathbf{x}_c - \mathbf{x}_0) &= r \\ \mathbf{n}_2 \cdot (\mathbf{x}_c - \mathbf{x}_0) &= r \\ \mathbf{n}_3 \cdot (\mathbf{x}_c - \mathbf{x}_0) &= r \\ \mathbf{n}_4 \cdot (\mathbf{x}_c - \mathbf{x}_0) &= r \end{aligned} \tag{6.6}$$

where $\mathbf{x}_0$ is the coordinate of a first point of the simplex, $\mathbf{x}_c$ is the inscribed center of the sphere, $\mathbf{n}_i$ is the normal of the face opposing the point i of the simplex (pointing toward the inside of the element) and r is the inscribed radius, given by equation 6.5. The solution to this system of linear algebraic equations would give the desired $\mathbf{x}_c$. A solution would only be possible if the points of the element are linearly independent. This might not be the case if points are coplanar or colinear, but for the current purposes the centroid of the element can be used as an approximation to ensure that the method is robust even in degenerate

cases. To find the inscribed sphere centre on a boundary element, this system of equations can be augmented with the equations for the barycentric coordinates of $\mathbf{x}_c$ on that element to constrain the inscribed centre, such as to be a linear combination of the other points of the boundary simplex.

Furthermore, by transforming the points of the element to the metric space, using the metric transform $\mathbf{x}' = N\mathbf{x}$ presented earlier in Chapter 2, the same equation can be used to compute the inscribed centre in the transformed space, after which it can be transformed back to the standard Euclidean space using the inverse transform $\mathbf{x}_c = N^{-1}\mathbf{x}'_c$. A constant linear transform N per element was used by averaging the transforms stored at each mesh point of the element. In the anisotropic case, with a linear transform N that is constant per element, computing the inscribed sphere centre in the transformed space is equivalent to computing the inscribed centre of an ellipsoid in the standard Euclidean space (Li et al. (2005)).

The potentially optimal location for the new point position is determined as the average position of the inscribed ellipsoid centres for all the elements incident to that point and having a topological dimension higher than zero, as follows:

$$\mathbf{x}_{opt} = \frac{1}{N_{elem}} \sum_{i=1}^{N_{elem}} \mathbf{x}_c|_i \tag{6.7}$$

In order to ensure that mesh smoothing always improves the local quality of the mesh, the optimal position is relaxed to be an intermediate value between the previous point position and the optimal position using:

$$\mathbf{x}_{trial} = (1 - \varpi)\, \mathbf{x}_{prev} + \varpi \mathbf{x}_{opt} \tag{6.8}$$

where $\mathbf{x}_{opt}$ is the optimal position given by equation 6.7, ϖ is the relaxation factor, $\mathbf{x}_{prev}$

is the previous point position before smoothing is attempted and $\overrightarrow{x}_{trial}$ is the new trial position used to test if the minimum element quality of the affected element is improved. If the smoothing does not improve the quality of the elements with an initial value of 1.0 for ϖ, then ϖ is reduced by a factor of 0.5 for two other trials. If after three attempts the mesh is not improved, the smoothing is rejected for this point to avoid excessive computational cost and because the displacement would most likely become too small to be significantly beneficial. Furthermore, after completing a smoothing pass on all points, other neighbouring points might have moved, which may enable the movement of a point that was previously rejected.

A small caveat concerning the relocation of points on the boundary of the geometry needs to be highlighted. The trial point position $\mathbf{x}_{trial}$ needs to be projected to the geometry, using the method described in Chapter 4, to remain on the surface of that geometry. However, it is possible that $\mathbf{x}_{trial}$ would be at greater distance from $\mathbf{x}_{prev}$ than another adjacent part of the geometry (for example, near the junction of two adjacent faces of a cube). To avoid problems that this may cause in the location routine of the projection routine, an additional step is performed in which $\mathbf{x}_{trial}$ is projected first to a plane defined by the point $\mathbf{x}_{prev}$ and the approximate normal to the geometry at that point. The approximate normal is computed using the average normal of the neighbouring faces as this is sufficient for this purpose and does not depend on locating the point on the geometry. With this additional step, the modified $proj\left(\mathbf{x}_{trial}\right)$ can be used without problem with the projection routine described in Chapter 4. Notice that this projection is done before the test is made to evaluate whether the point relocation improves the quality of the mesh or

not, thus ensuring that the mesh quality is always improved, or remain the same, as was the case for points inside the volume mesh.

Going back to the mesh optimization procedure, the mesh smoothing presented is used at three different steps of the procedure. First, it is used to relocate each point that is inserted to split an edge just after the edge splitting; second, it is used after the pass on edge swapping; and, finally, it is used after the mesh global iterations on the edge splitting, edge collapsing, edge swapping and edge smoothing phases have been completed to finalize and fine tune the location of the mesh point in an attempt to improve as much as possible the quality of the mesh once its density and topology have been adjusted. It is important to smoothen the mesh after its topology has been improved, and not before, because the mesh smoothing does not modify the topology and its ability to improve the mesh quality can be severely limited if, for example, some mesh points have a very large number of incident edges.

To limit the computational cost of smoothing mesh points after the adjustment of the density of the mesh, a simple heuristic approach is used to select the mesh points to relocate, based on the quality of the element incident to them. This is done by first traversing all volume elements of the mesh to compute their quality and mark as for smoothing all the points corresponding to an element quality that is lower than a specified threshold. After experimenting with the test cases presented in Chapter 8, it was chosen to iterate five times over the mesh points marked for smoothing using a threshold for quality that is progressively reduced for each mesh smoothing pass on all points from 1.0, to 0.8, 0.6, 0.4, and finally 0.3. This progressive reduction of the threshold for smoothing allows for

all points to be initially potentially relocated, if this improves quality, to progressively concentrate the computational expense on elements that need quality improvements the most. Of course, other strategies could be employed in the future with the same inscribed ellipsoid based approach, but this simple one was found to be an acceptable compromise between reducing the computational cost and improving the quality of the mesh.

6.6 Sliver Perturbation by Random Point Relocation

Another point relocation strategy was devised to attempt to improve the quality of elements of very low quality. The algorithm for this is structurally the same as for the ellipsoid based one presented in section 6.5, with a few small exceptions. First, the quality is measured isotropically using equation 6.1, but without the determinant of the metric, such that the point relocation is only accepted if the isotropic quality improves. Second, points are selected for this attempt to improve quality only if they are incident to elements having a very low isotropic quality below a threshold of 0.01. Third, rather than computing the optimal position based on the average ellipsoid using equation 6.7, the following equation is used instead:

$$(x_i)_{\text{rand}} = (x_i)_{prev} L_{avg} F_{ran} \tag{6.9}$$

where $(x_i)_{\text{rand}}$ is the randomized location for the coordinate i of the point under consideration, $(x_i)_{prev}$ is the previous value for that point coordinate, L_{avg} is the average Euclidean length of edges incident to the point multiplied by 0.1 and F_{ran} is a random factor with a real value in the interval $[0, 1]$.

The sliver perturbation by random point relocation is performed at each internal

mesh iteration just before the mesh smoothing based on the ellipsoid. Perturbing elements with a near zero volume can improve the computation of the inscribed radius (recall that computing the inscribed radius requires the face normals which cannot be properly computed if the sub-volume of the face of a simplex is zero). Furthermore, it can help to disturb the mesh points where the quality is very low and thereby increase the probability that swapping edges with be successful in improving the anisotropic quality in subsequent internal iterations of the mesh optimization procedure.

6.7 Pseudo-Code for the Meshing Algorithms

This section presents the pseudo-code for the algorithms described in this chapter, which briefly are:

- Algorithm 6: the splitting of an edge by adding a new mesh point at its mid-point
- Algorithm 7: the collapsing of an edge to remove a point in the mesh
- Algorithm 8: the swapping of an edge through a combined sequence of edge splitting and edge collapsing
- Algorithm 9: the algorithm to decide what edge swap possibility to attempt and call the simulated edge swapping
- Algorithm 10: the smoothing of the mesh element incident to one mesh point
- Algorithm 11: the combined mesh optimization procedure with edge splitting, edge collapsing, edge swapping and mesh smoothing

Algorithm 6 Edge Splitting

Input. The mesh with its associated geometry and the list of split point ids, which are the end point ids for an edge.

Output. The point id created if the split was successful and the list of new elements.

Reset list of new elements to empty state
Create the split point Ps at the mid-point of the edge
Find the list of elements $Es_{neighbors}$ that are neighbors of the edge
if $Es_{neighbors}$ is empty **then**
  Delete split point Ps
  Return false
Find among $Es_{neighbors}$ the element neighbor that is of the lowest topological dimension
Set the topological dimension of the point to that lowest topological dimension
if point topological dimension is lower than space dimension **then**
  Project the split point Ps to the geometry
  Simulate the split to verify if the quality of elements would be sufficient
  if any element during the simulated split has a quality below the treshold **then**
    Delete the split point Ps
    Return false
for each element E in $Es_{neighbors}$ **do**
  Copy the element's point ids $Epids$
  if element E is of topological dimension lower than the space dimension **then**
    Retrieve and store the surface to geometry mapping element key $Egeo$
  Retrieve the family id for the element E
  Delete the element E from the mesh
  for each copy index in the list split point ids **do**
    Retrieve the copy point id in the list split point ids at the copy index
    Retrieve the local element point index for the current copy point id
    Replace the current element copy point id with the split point id
    Insert the new element in the mesh
    Set the family id for the new element
    Set the geometry element mapping if it is a boundary element
    Reset the list of element point ids $Epids$ to its original state for next use
    Insert the created element in the list of new elements

Algorithm 7 Edge Collapsing

Input. The mesh with its associated geometry and the list of split point ids for the edge to collapse.

Output. The point id left after the edge collapse and the list of modified elements.

Reset list of elements to be modified $Es_{modified}$
Verify if collapsing this edge would preserve the topology of the surface mesh
if edge is not topologically collapsible **then**
    Return false
Create the list of elements to be deleted Es_{delete} by finding the element neighbors for the edge
Create the list of elements that survives the collapsing $Es_{modified}$ by finding elements using only one edge point
if use mid-point option and both edge end points are the same topological dimension **then**
    Compute the location of the mid-point of the edge
    Project the mid-point to the geometry if the edge is on the boundary of the mesh
    Set the collapse end point location to the mid-point location
else
    Set the collapse end point location to the edge point id to preserve after the collapse
if collapse only if the quality improves **then**
    Find the minimum element quality among the list of elements to be modified $Es_{modified}$ and deleted Es_{delete}
    if minimum element quality is higher than 0.1 **then**
        Set minimum element quality to 0.1
if mid-point option **then**
    Move both end points of the edge to the collapse end point location
else
    Move the point to remove to the location of the point to preserve
for list of elements to be modified $Es_{modified}$ **do**
    Compute element quality
    if element quality is below acceptable treshold **then**
        Reset points to original location
        Return false
Delete all elements in the list of elements to be deleted Es_{delete}
for each element the list of elements that survives the collapsing $Es_{modified}$ **do**
    if element has the point id to remove **then**
        Disconnect the element from the mesh to remove it from the inverse connectivity
        Replace the point id to remove with the point id to preserve
        Reconnect the element to the mesh to update the inverse connectivity
Update the topological point id for the point to preserve
Delete the point id to remove from the mesh
Update the surface to geometry mapping if necessary

Algorithm 8 Simulated Edge Swapping

Input. The mesh with its associated geometry, the list of point ids for the edge to swap, the list of edge element neighbours and the final point id receiving the edge swapped.
Output. The list of created elements.

1: **if** the edge to swap corresponds to a topological edge **then**
2: Return false
3: **if** swap only if quality improves **then**
4: Compute the minimum element quality for the initial configuration Q_{min}
5: Increase the minimum element quality Q_{min} by 5
6: Compute the temporary split point at the mid-point of the edge to swap
7: Create the split point
8: Split the edge to swap at the mid-point
9: **if** the split was rejected **then**
10: Return false
11: Verify if the collapsing the edge going from the temporary mid-point to the final point id would improve quality
12: **if** quality would improves **then**
13: Collapse the edge going from the temporary split point to the final point id to remove the temporary split point
14: **if** collapse was successful **then**
15: Return true
16: Collapse the edge going from the temporary mid-point to an end point of the initial edge to remove the temporary point and recover the initial configuration

Algorithm 9 Improve Edge Topology

Input. The mesh with its associated geometry and the list of point ids for the edge to swap.
Output. The list of created elements.

1: Find the list of element neighbors to the edge to swap
2: Compute the mid-point of the edge to swap
3: Use the list of element neighbors to find the list of ring points
4: Use the list of ring points compute the point closest to the edge mid-point
5: Try to swap the edge using the simulated edge swap with the final point id as the closest point id
6: **if** not successful and try all possibility is turned on **then**
7: Try to swap using as the final point each point in the list of ring point ids until one is successful

Algorithm 10 Mesh Point Smoothing

Input. A point in the mesh.
Output.

1: Find the list of elements neighbors $E_{neighbors}$ to the point to smooth
2: Compute the optimal location P_{opt} as the average location of the inscribed ellipsoid for the $E_{neighbors}$ including the boundary elements (except for vertex elements)
3: Compute the displacement vector going from the previous point location to P_{opt}
4: Compute the minimum initial element quality of $E_{neighbors}$
5: **for** conformity loop going up to three **do**
6: **if** point to move is on boundary **then**
7: Project P_{opt} to the geometry
8: **for** element in $E_{neighbors}$ **do**
9: Compute element quality and volume (at the same time)
10: **if** element quality is below minimum initial element quality or volume too low **then**
11: Reduce the length of the displacement vector by half and recompute the new P_{opt}
12: Break out of the for loop
13: **if** node movement does not improve quality **then**
14: Restore the point location to its original location
15: Update the surface to geometry mapping if necessary

Algorithm 11 Mesh Optimization

Input. A simplicial mesh with a metric field.
Output. A mesh adapted to the metric field.

1: Create metric edges
2: Compute the target metric edge length to collapse $Lm_{collapse}$ and to refine an edge $Lmrefine$
3: Compute the range for collapsing as the difference between the minimum edge length to the $Lm_{collapse}$
4: Compute the range for refining as the difference between the maximum edge length to the $Lmrefine$
5: **for** iteration 1 to 10 **do**
6: Compute current collapse threshold by reducing range for collapsing by a percentage of 20
7: Compute current refinement threshold by reducing range by a percentage factor
8: Refine edges in the range for splitting by refining the longest in range first and smoothing each inserted point
9: Collapse edges in range by collapsing shortest edges first
10: smooth mesh (five iterations on all points with element having a minimum quality below 1.0, 0.8, 0.6, 0.4, 0.3)

Chapter 7

Space-Time Mesh Adaptation and Solution Procedure

7.1 Introduction

This chapter presents the rationale for developing a fully coupled space-time FEM formulation in the context of unsteady mesh adaptation. The limitations of a decoupled space-time formulations are highlighted first. Then, the fully coupled space-time FEM procedure with an embedded space-time mesh adaptation is proposed as a solution to address these limitations followed by a brief discussion of its associated computational costs. The flow solver algorithms used are also presented along with a short note concerning the solution interpolation necessary to bridge the FEM solver and the mesh adaptation procedure. The chapter concludes with a summary of the overall adaptive space-time FEM solution procedure.

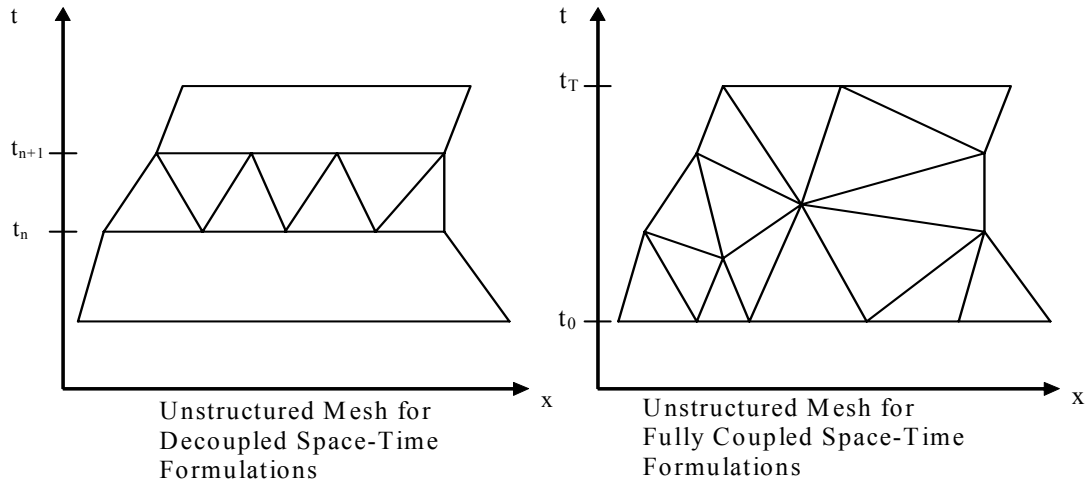

Figure 7.1: Illustration of a decoupled space-time mesh (left side) and a fully coupled space-time mesh (right side).

7.2 Comparison of Decoupled and Fully Coupled Space-Time Formulations

In a fashion similar to classical time stepping approaches, a fully decoupled space time formulation proceeds by computing the solution from one time step to the next. However, instead of using a spatial mesh, in which the mesh points depend only on the spatial coordinates, a decoupled space-time formulation requires a space-time mesh with points which depend on both the space and time coordinates. The sketch on the left side of Figure 7.1 illustrates the progression of a space-time mesh from the time step t_n, at which computations have been completed, to the next time step t_{n+1}, at which computations need to be made. In contrast to the decoupled formulation, a fully coupled formulation can use a mesh that extends to the entire space-time domain, as shown on the right side of Figure 7.1, and hence covers the full range of the time period that is of interest for the simulation.

From the perspective of mesh adaptation, it is possible to add mesh adaptation to a decoupled formulation in a fashion analogous to some adaptive time stepping approaches (Li and Wiberg (1998), Alauzet et al. (2007)). The general strategy is to compute the solution at time t_{n+1} from the solution at t_n (higher order methods would use the solution from additional previous time steps as well), and then estimate the error committed at this time step and determine whether it exceeds an acceptable level (Alauzet et al. (2007)). If the error estimate for the current time step is larger then a specific threshold, the solution can be recomputed with a smaller time step (for p-methods the order of the temporal discretization could be increased, but only methods that adjust the size of the time step are considered here). Once the acceptable error threshold at the current time step is approached, or a minimum time step size is reached, the solution procedure can move to the next time step. This process is illustrated in Figure 7.2 for time stepping methods using only the solution at the previous time step.

It is crucial to note that an adaptive time stepping procedure adjusts the size of the current time step to recompute t_{n+1}, but once this solution is computed it remains fixed for all subsequent iterations because the solutions at times $t_0, t_1, ..., t_n$ are all outside of the error feedback loop. Because, in general, both discretization and convergence errors occur at each time step, it follows that these errors can accumulate from one time step to the next. The accumulation of the discretization error is schematically represented in Figure 7.3.

Clearly this illustration is an over-simplification, because the discretization error does not necessarily accumulate in a linear fashion. The accumulation of the discretization

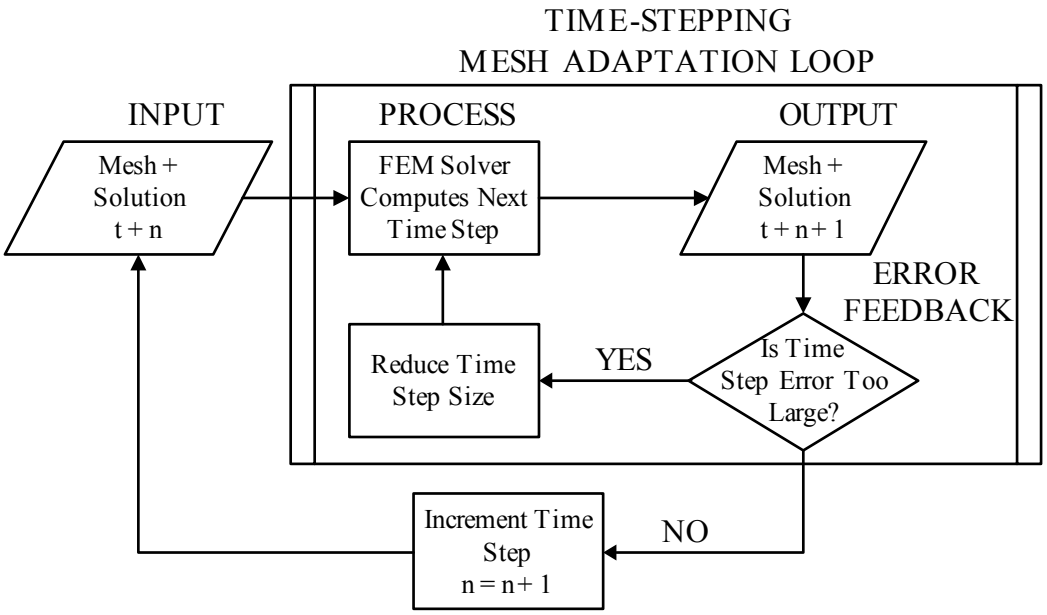

Figure 7.2: Mesh adaptation loop for a time-stepping procedure where the discretization error from the previous time step cannot be controlled by reducing the size of the current time step.

error is not only a function of the order of the decoupled space-time method (or of the time stepping method), but it is also a function of how rapidly the solution changes as a function of time. In certain parts of the space domain, the solution can remain the same for a certain period of time and then go through abrupt changes. In other cases, the solution could be approaching a steady regime for which the temporal discretization can be reduced and even become negligible (some convergence and round-off errors could remain). The key difference is that, for a time stepping approach, the possibility of the accumulation of the discretization error exists, and its assessment is difficult and problem dependent (Hand and Lu (1999)), whereas, for the fully coupled space-time approach, it is possible to refine the mesh in any region of the space-time domain until the error estimator requirement is satisfied (with some constraint on mesh size as explained in Chapter 2). Therefore, the fully coupled

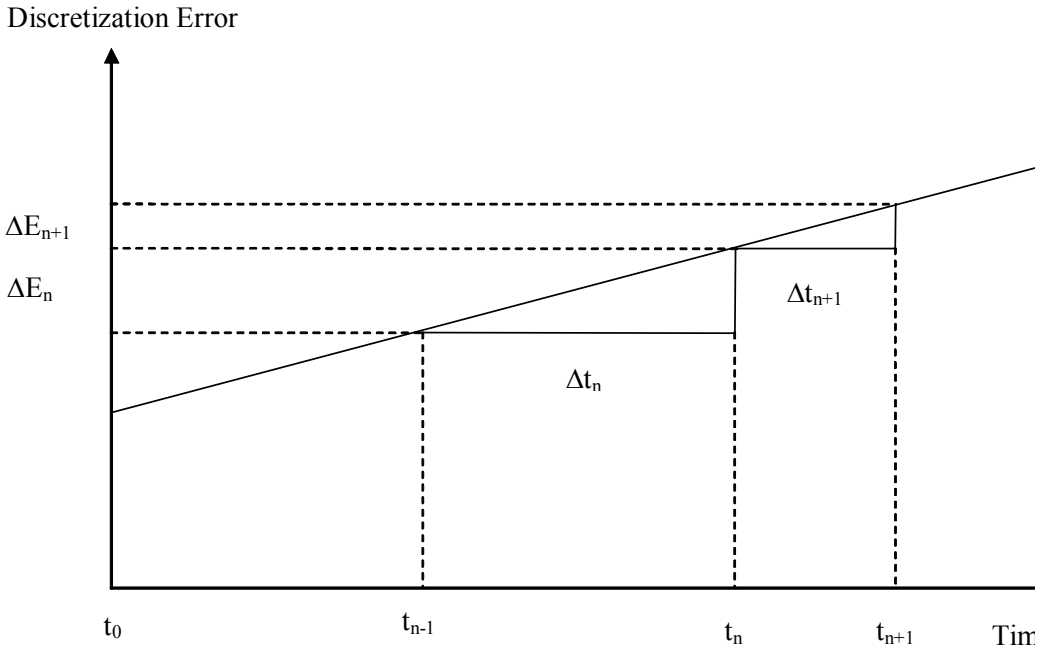

Figure 7.3: Illustration of the accumulation of the discretization error with a time-stepping procedure.

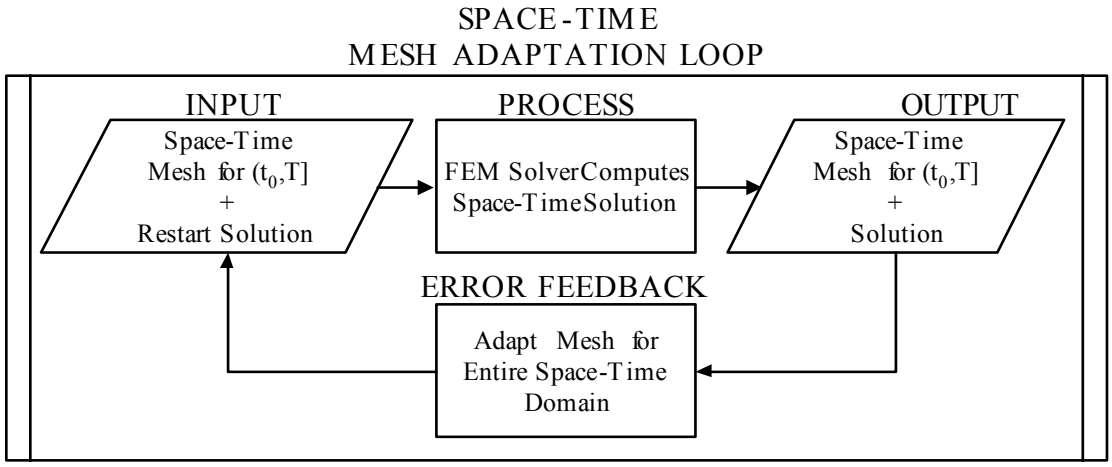

Figure 7.4: Fully coupled space-time mesh adaptation loop, in which the complete space-time solution comprising the entire time range is recomputed at each mesh adaptation iteration.

approach offers the possibility of bounding the maximum discretization error everywhere in the space-time domain, until the resolution of the solution is considered acceptable, in the same manner that mesh adaptation has been used to bound the maximum space discretization error for steady problems. Previous authors, including (French and Peterson (1996), French (1999), Karakashian and Makridakis (1999), Pontaza and Reddy (2004)), have mentioned this possibility, but have not presented such a mesh adaptation scheme. The mesh adaptation loop for the unified space-time approach, in which the entire space-time mesh is used to compute the solution and the space-time error estimate over the entire range of space and time in the solution domain, and to adapt the mesh is illustrated in Figure 7.4.

Hence, the accumulation of the discretization error in time can be avoided, because the solution on the entire space-time domain is recomputed after each mesh adaptation iteration. The space-time solution in the previous iteration is only used as a restart solution

to accelerate the convergence of the non-linear flow solver, which will be discussed briefly in section 7.5. This does not mean that there is no discretization error associated with a fully coupled space-time strategy, but simply that it can be controlled with mesh adaptation throughout the space-time domain in the same manner as it can be controlled throughout the space domain for steady problems.

7.3 Computational Cost of Fully Coupled Space-Time Solution Procedures

Despite their advantages, space-time approaches also have some drawbacks. The first complication is that a space mesh of dimension d needs to be extended in time to become a mesh of dimension $d+1$. For a decoupled formulation, this is relatively simple because the mesh can be extruded in time and subdivided, if a simplicial mesh is used. However, a fully coupled approach requires that the entire space-time domain be meshed, which requires considerably more memory. Nevertheless, even with this limitation, a fully coupled approach is more flexible from the perspective of mesh adaptation, because it allows for the addition of mesh points only where indicated by the error estimator, which accounts for both the spatial and the temporal errors. Consequently, it is possible to start the mesh adaptation process with a very coarse mesh, which does not take much memory and time to compute, and progressively add points as the mesh adaptation procedure iterates (again in a fashion similar to what is done for steady mesh adaptation). In cases in which the solution evolves with a varying time scale in the solution domain, this can lead to a significant reduction in the number of mesh points required, because it is possible to use

elements which are elongated along the time direction in a manner that varies across the space domain. In contrast, a decoupled space-time approach requires that the solution at each mesh point be repeated with the same time step size, regardless of the local time scale.

It remains to be seen whether a fully coupled approach can be competitive in terms of CPU time compared to a time stepping approach for solving 3-D space problems, which require a 4-D mesh. However, for the case of 2-D space problems with a 3-D space-time mesh, Potenza and Reddy, who compared both approaches without the use of mesh adaptation, have reported that a fully coupled approach is cost effective (Pontaza and Reddy (2004)). It is also important to stress that, for the classical benchmark problem of a backward facing step flow, these authors have shown that the decoupled approach leads to a solution that is numerically unstable when the time step size is too large compared to the space mesh, whereas instability did not arise in their fully coupled formulation.

7.4 Flow Solver Algorithm

In order to solve the non-linear Navier-Stokes equations, a simple Picard method, briefly presented in Chapter 5, is used with PETSc as the algebraic solver (Balay et al. (1997), Balay et al. (2001), Balay et al. (2004)), a Jacobi preconditioner and GMRES (Generalized Minimum Residual Method). Although the convergence of Picard method is known to be generally slow, whereas Newton methods can exhibit a quadratic convergence, it offers several advantages (Reddy and Gartling (1994)). First, the Picard method has a larger radius of convergence than the Newton method, which allows for an initial guess to the final solution that is further away from the final solution. Second, it is simpler to imple-

ment, because it does not require programming a Jacobian correction matrix (Langtangen (1999)), which is not only tedious with a GLS method, but also error prone. Third, for the incompressible Navier-Stokes equations, it conveniently allows one to start the solution process with a null velocity field, such that for the first Picard iteration the convective term cancels out and the problem is reduced to solving a Stokes problem with the same boundary conditions. Also recall that, as described in Chapter 5, the boundary conditions to the incompressible Navier-Stokes are chosen to accelerate the fluid from rest progressively by ramping up the value for the boundary conditions, which reduces the problem associated with having an initial guess solution in the Picard method that is too far from the final one. More precisely, at time $t = 0$ the null velocity field is the solution satisfying the incompressible Navier-Stokes equations, but because a fully coupled space-time formulation is used, then the null velocity field is actually the initial solution for the entire space-time domain, whereas for time stepping methods it is usually the initial solution for the space domain only and at time $t = 0$.

Under certain flow conditions, for example strong convective flows, the Picard method may be slow to converge, oscillate and even diverge (Reddy and Gartling (1994)). However, the GLS formulation of the Navier-Stokes equations contains stabilization terms that are designed to dampen out oscillations that could otherwise occur at large Reynolds numbers (Tezduyar and Behr (1994)). Furthermore, the flow solver is not used on its own, but it is embedded within a mesh adaptation loop. In this case, it is possible to start the iteration using a very coarse mesh, which allows the solution to be computed relatively fast and several Picard iterations to be done rapidly. As the mesh is being refined during the

mesh adaptation phase, each iteration of the flow solver does not start from a null solution field, but from the solution obtained at the previous solver iteration interpolated on the adapted mesh. Hence, as the mesh is being adapted and refined, the restart solution passed to the flow solver becomes closer and closer to the desired final solution, assuming of course that the process converges. Therefore, the total number of Picard iterations that would typically be necessary for the convergence of an incompressible Navier-Stokes solver are spread over several solver runs, the first one being on a very coarse mesh and the subsequent ones being on finer and finer meshes, according to the requirements of the error estimator (and available computer memory and CPU time). Consequently, the total computational cost of solving a problem using a Picard method with mesh adaptation can potentially be lower than the one that would incur if the final adapted mesh were used from the start. The computational cost for the flow solver operation would also be lower, compared to the one incurred if a fine mesh were used without mesh adaptation. It cannot be assessed whether the total cost, namely the sum of the costs for flow solver activity and for mesh adaptation, would be lower than without mesh adaptation, because this is highly problem dependent. However, mesh adaptation has been known to greatly increase the robustness of the solution process (Tam et al. (1998)); unless the flow field is uniform, the number of mesh points needed to obtain a solution at a given level of accuracy is significantly reduced (Tam (1998)).

Besides the present simple approach, one may possibly explore the use of sophisticated solution algorithms (Turek (1999)), but these are considered outside the scope of this thesis.

7.5 Interpolation of the Re-start Solution on the Adapted Mesh

Interpolating a solution from one mesh to another usually requires looping over the mesh points of the receiving mesh, locating each mesh point in the mesh with the solution to determine its containing element and then using finite element interpolation functions to evaluate the solution at that point within the element. Various point location procedures can be used and the fastest ones, presuming no prior information is known between the two meshes, have a computational cost that increases at a rate of $O(n \log(n))$, where n is the number of mesh points in the input mesh (Lohner (2001)).

However, in the context of mesh adaptation, this can be improved upon by noticing that it is not necessary to interpolate the solution from an arbitrary mesh to another, but specifically from the initial mesh that was passed on to the mesh optimization process to the adapted mesh resulting from the mesh optimization process. This simpler case can be easily taken advantage of here, because, before the mesh modification begins, a copy of the input mesh is made and used to store the metric field (computed from the error estimator). Each mesh point in the mesh to be adapted (the "foreground mesh") is initialized with the element id of its containing element in the background mesh. Also, during the mesh adaptation process, this containing element in the background mesh for each point in the foreground mesh is continuously updated, so this information is available at the end of the mesh adaptation process. Consequently, to interpolate the solution on the new adapted mesh is simply a matter of reading back the solution associated to the initial non-adapted mesh and evaluating the linear finite element shape function to interpolate the

solution. This is done by calling a walkthrough search routine, which makes use of the linear shape function, with the initial containing element already available. The interpolation of the solution requires the algorithm to loop on all the points of the adapted mesh and to call the search routine once for each point, in order to evaluate the linear finite element shape functions and also to ensure that the containing element is correct. As a result, this interpolation algorithm has a computational cost that increases linearly with the number of mesh points in the input mesh, i.e. it is $O(n)$, rather than $O(n \log(n))$, if a spatial search were used instead.

Although this search routine is not conservative, in the sense that the interpolated velocity field might not exactly satisfy the incompressibility condition for the Navier-Stokes equations, it was found to work very well in practice. Recall that a Picard method is used for the nonlinear Navier-Stokes equations, so that the interpolated velocity field and pressure fields are only used for the first Picard iteration.

Finally, it is worth mentioning that the solution interpolation procedure is the same for steady and unsteady problems, because a fully coupled space-time procedure is used.

7.6 Summary of the Adaptive Solution Algorithm

The overall adaptive solution algorithm, containing both the FEM flow solver and the mesh adaptation procedure, is summarized in Figure 7.5. The process begins by specifying an initial space-time mesh, boundary conditions and values of parameters such as the Reynolds number that are passed on to the flow solver. As part of the flow solver process,

Picard iterations are performed until the following convergence criterion is satisfied. This criterion was the L^2 norm of the difference between the solution from the previous Picard iteration to the current one divided by L^2 norm of the current solution:

$$\varepsilon = \frac{\|\mathbf{U}_{i-1} - \mathbf{U}_i\|_0}{\|\mathbf{U}_i\|_0} \tag{7.1}$$

where $\mathbf{U}_i$ is the solution vector at the Picard iteration i. A condition of $\varepsilon \leq 10^{-4}$ was used for stopping the iterations and a maximum of 10 Picard iterations per solver run were allowed.

Once the Picard iteration has converged or reached the maximum number of iterations, if a specified maximum number of mesh adaptation iterations has not been reached, the solution is passed on to the mesh adaptation process, along with the geometry corresponding to the initial mesh. The error estimator is computed first and the resulting metric tensor field is passed on to the mesh optimization process along with the minimum and maximum mesh size, maximum allowed aspect ratio and metric reduction factor. For the test cases presented in this thesis, typically three or four mesh adaptation passes were used with four or five solver runs for a maximum of 50 Picard iterations in total. Once the mesh optimization process is completed, the solution is interpolated from the initial mesh to the adapted mesh and passed on to the flow solver to recompute the solution again, but this time using the interpolated solution as the restart solution; this way, the Picard method does not start from a null solution field as was the case for the first solver run.

As described in Chapter 6, a simple error reduction technique was used to reduce the average error measured by the error estimator along the edges of the mesh. This was preferred over specifying a target error, because, for an interpolation based error estimator,

the meaning of the value of the target error is not clear and it is problem dependent. Mesh adaptation strategies that normalize the metric, such that the target mesh size as measured by the metric is one, are not easy to use, because they require the scaling factor to be adjusted by trial and error. A wrong guess can result in a mesh that is unnecessarily refined or, even worse, coarsened (Tam et al. (1998), Dompierre et al. (2002)). A mesh reduction strategy is therefore safer and easier to use, although it remains a heuristic approach that can certainly be improved upon. A possible choice could be to use a residual based error estimator, but this is left to future work (Ainsworth and Oden (2000)).

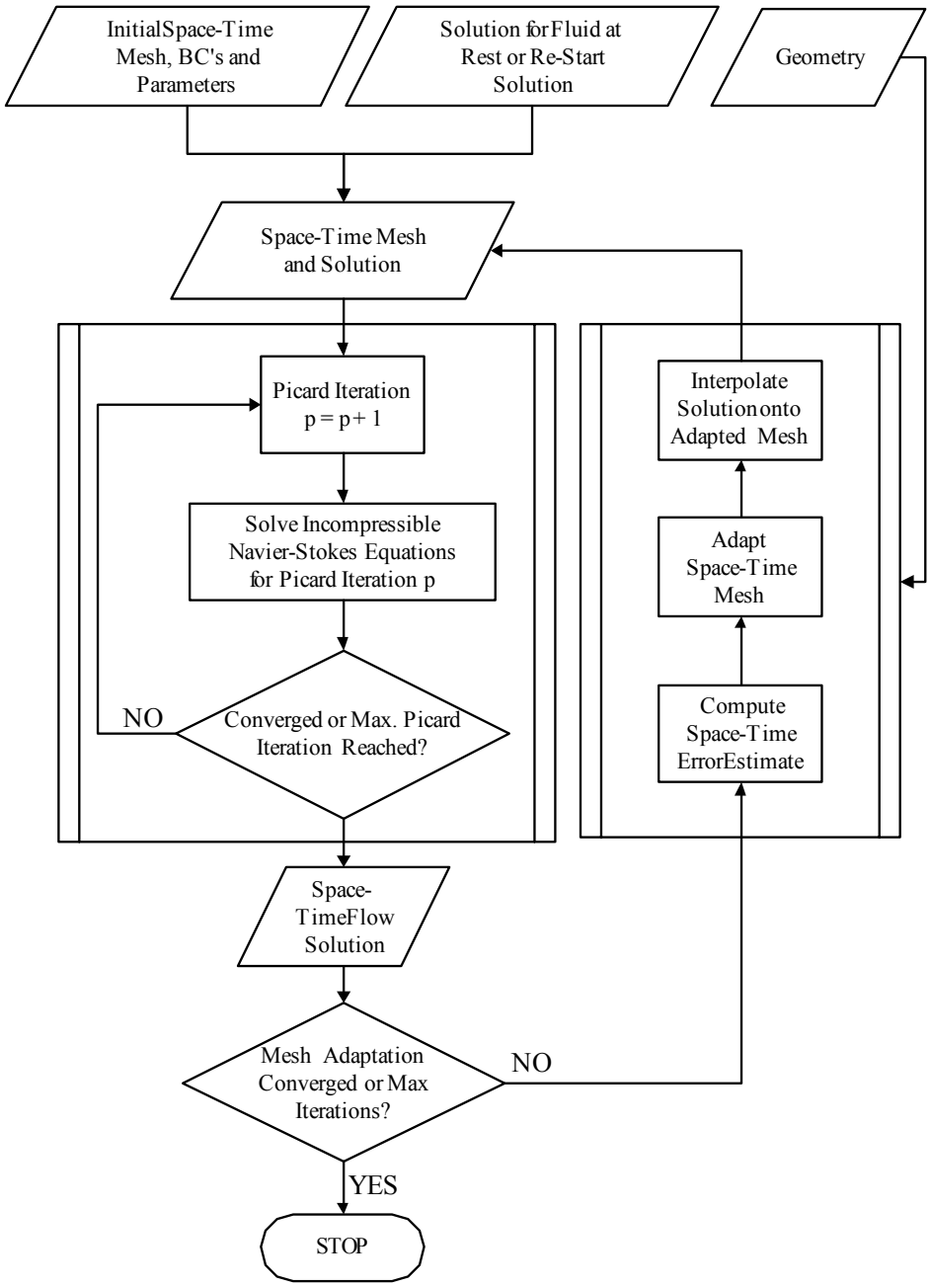

Figure 7.5: Schematic representation of the combined space-time FEM solver and the space-time mesh adaptation procedures.

Chapter 8

Numerical Results

8.1 Introduction

This chapter presents the test cases used to investigate the applicability of the fully coupled space-time adaptive finite element method developed in this thesis. All test cases presented use simplicial meshes in 2-D and 3-D and, in two simple cases, in 4-D. The space-time finite element method is investigated using an unsteady linear heat equation and also the incompressible Navier-Stokes equations for flows with Reynolds numbers from 100 to 800. In summary, the test cases are as follows:

- Section 8.2 assesses the performance of the anisotropic mesh optimization procedure inside unit domains in 2-D, 3-D and 4-D using analytical metric fields. These examples focus on the performance of the meshing algorithms, without giving consideration to the error estimator and the finite element method.

- Section 8.3 is a first test for the adaptive space-time procedure. It considers an

unsteady linear heat equation problem with a manufactured solution used to compare the numerical solution with the expected analytical solution. The L^2 norm of the error on the temperature field on 2-D, 3-D and 4-D space-time meshes inside a unit domain is used for that comparison.

- Section 8.4 describes verification cases for the incompressible Navier-Stokes equations, also using manufactured solutions. A first test is performed for a steady 2-D flow field on a 2-D mesh. A second test is performed with a similar analytical solution, but for an unsteady flow using a fully coupled space-time flow solver and 3-D meshes, in which time is the third dimension. The numerical solutions are compared with the analytical solutions for isotropic uniform meshes as well as for adapted meshes.

- Section 8.5 presents the solution of the classical benchmark problem of flow past a circular cylinder at Reynolds number of 100. Again, a fully-coupled space-time approach is used on a solution domain with a dimensionless time extending from zero to ten units.

- Section 8.6 considers the flow behind a backward facing step at Reynolds number of 800 using the same adaptive space-time procedure as that in the previous example and also for a dimensionless time range from zero to ten units.

- Section 8.7 presents the impulsively started flow in a lid-driven cavity for a Reynolds number of 400 using the same boundary conditions and problem specifications as those in the work of Pontaza and Reddy (2004); however, the present approach uses a fully coupled Galerkin/least-square finite element method combined with an anisotropic

space-time mesh adaptation procedure, rather than a higher order finite element spectral method without any mesh adaptation.

All numerical solutions were computed using a personal computer with a Pentium 4 processor, running at 3.2 GHz and using 1 GB of RAM; the code was compiled using double precision floating point arithmetics.

8.2 Mesh Optimization Using an Analytical Metric with 2-D, 3-D and 4-D Meshes

To access the effectiveness of the mesh optimization procedure on 2-D, 3-D and 4-D unstructured simplicial meshes, tests were performed on meshes with an analytical metric specified as

$$N(x,y,z,t) = \begin{bmatrix} 1/h_0 & 0 & 0 & 0 \\ 0 & 1/h_1 & 0 & 0 \\ 0 & 0 & 1/h_2 & 0 \\ 0 & 0 & 0 & 1/h_3 \end{bmatrix} \mathbf{I} \tag{8.1}$$

where N is a linear transform, as presented in equation 2.9, $\mathbf{I}$ is the identity matrix and h_i is the length along the coordinates axis i, given as

$$\begin{aligned} h_0 &= 0.025, \text{ if } |x-0.5| \leq 0.1 \\ h_1 &= 0.025, \text{ if } |y-0.5| \leq 0.1 \\ h_i &= 0.25, \text{ otherwise} \end{aligned} \tag{8.2}$$

The domain for the mesh spans the region $[0,1]$ for each of the x, y, z coordinates

of the space domain, while the time axis for the 4-D case only ranges between $[0, 0.2]$ to restrict the number of points in the mesh. For the 2-D case, the initial mesh is constructed by dividing the unit square in 10 intervals along each axis and subdividing the quadrilateral elements in two triangles leading to an initial mesh with 200 simplicial elements and 121 points. For the 3-D case, the initial mesh is also constructed by dividing a unit cube with 10 intervals and subdividing the hexahedral elements into six tetrahedral elements leading to a mesh with 6,000 simplicial elements and 1,331 points. For the 4-D case, a 3-D unit cube is divided in six tetrahedral elements, extruded to 4-D and refined with the mesh optimization procedure with a uniform metric with $h_i = 0.1$, which leads to a starting mesh with 684,869 4-D simplicial elements and 42,974 points. The initial meshes were chosen to be of sufficient resolution to capture the variation of the metric field computed with the analytical metric of equation 8.1; they were stored in the background mesh during the mesh optimization procedure.

For the 2-D case, Figure 8.1 shows the meshes before and after 10 internal cycles of mesh optimization, where the density was adjusted followed by 5 cycles of mesh smoothing (the numbers of cycles are the same for all test cases). The histograms for the element quality and the metric edge length are shown in Figure 8.2. The number of mesh points and elements as well as the numbers of edge collapses, edge splits and edge swaps as functions of the number of internal iterations in the mesh optimization procedure are shown in Figure 8.3.

For the 3-D case, Figure 8.4 shows the meshes before and after optimization, while Figure 8.5 shows two details on the mesh. Histograms for the element quality and the metric

edge lengths are presented in Figure 8.6 and the numbers of mesh points and elements as well as the numbers of mesh modification operations are shown in Figure 8.7. Notice that some residual edge splitting and edge collapsing operations are still present after 10 internal iterations of the mesh optimization procedure, but from the top part of Figure 8.7 it can be seen that the numbers of mesh points and elements are not fully stabilized so that this is to be expected. Nevertheless, the numbers of residual mesh operations are small compared to their peak values. Furthermore, each internal iteration includes an attempt to swap edges for elements that have a quality lower than 0.5 and to smoothen the mesh as well. These mesh improvement operations may perturb the metric edges and are also more expensive than splitting or collapsing. Consequently, it is important to stop the internal mesh optimization procedure as soon as possible to minimize its computational cost.

For the 4-D case, the meshes before and after the mesh optimization procedure are shown for the boundary of the domain corresponding to $t = 0$ in Figure 8.8 and for the boundary of the domain corresponding to $t = 0.2$ in Figure 8.9. The faces of the optimized mesh for the hypercube are shown in Figure 8.10 for $x = 0$, $y = 0$, $z = 0$, and $z = 1$. The meshes shown for the 4-D cases were generated by selecting the tetrahedral elements composing the faces of the 4-D hypercube domain (using their topological family id) and writing them into a CGNS file so that they could be processed with a standard CFD post-processor (Tecplot). The mesh points are modified by dropping the time coordinate for faces for which the time coordinate is equal to a constant. For the other faces, the coordinate held constant is removed and the mesh point coordinates are contracted to give a 3-D point that is composed of (y, z, t) or (x, z, t) or (x, y, t) (although it is stored as (x, y, z)

d	MinQ	MaxQ	AvgQ	StdQ	NumPoints	NumElems
2-D	0.5	0.97	0.82	0.08	344	617
3-D	0.03	0.99	0.49	0.21	5,158	26,678
4-D	0.00	0.95	0.058	0.11	124,917	1,927,662

Table 8.1: Analytic metric. Element quality, number of mesh points and number of elements for the mesh optimization procedure.

in the CGNS file to be compatible with the post-processor format). More sophisticated visualization techniques, such as slicing the 4-D fully unstructured simplicial mesh with an hyperplane at a constant time inside the domain rather than simply on its faces, are left for future work. The histograms for the element quality and the metric edge lengths are shown in Figure 8.11 and the ones for the number of mesh points and elements and mesh operations are shown in Figure 8.12.

The trends for the element quality for the 2-D, 3-D and 4-D cases are summarized in Table 8.1. For the 2-D case, the minimum element quality is 0.5 and the average quality is 0.97; considering that the maximum quality value is 1.0, the quality of these elements may be assessed to be excellent. The average element quality for the 3-D case is 0.45, which may be assessed as very good and about the same as for the results of Li et al. (2005), although not for the same metric field. In contrast, the minimum quality for the 4-D case is 0.0 and most of the element have a quality that is below 0.05, which is clearly unsatisfactory. Note that the quality is measured using equation 6.1, which uses the square of the quality, so a quality of 0.03 for the 3-D case would have been 0.17, if the quality were not squared. Furthermore, notice that the squared quality, containing the V_E^2 in the numerator, is proportional to h^{2d}, where d is the space dimension. Hence, the increase in the exponent of h with the increase of the space dimension also accentuates the sensitivity

of the measure of quality in a fashion similar to the one shown by Figure 6.1. It would be interesting to study the variation of the element quality with other measures of element quality across space dimensions to evaluate how the trends are affected by this, but here a commonly used measure of element quality was chosen to facilitate comparison with the work of other authors (e.g., (Li et al. (2005))), even though this might not be an optimal choice for quality comparisons across space dimensions.

Another observation concerns the number of mesh points as a function of the space dimension. Even with almost two million elements for the 4-D case, the adapted meshes shown in figures 8.8 and 8.9 are not fully converged, when compared with the ones for the 3-D case in Figure 8.4. The number of points was increased to high levels in 4-D to investigate how the meshing procedure would behave in 4-D compared to the 2-D and 3-D cases. At this time, these results are considered preliminary and further investigations are necessary to determine the reasons for the degradation of the mesh quality in 4-D for the anisotropic case. However, it is possible that at least a portion of the problem is due to the surface mesh of the 4-D meshes, which is composed of tetrahedral elements that can have zero or nearly zero volumes even if none of their metric edge lengths are near zero (sliver elements). Further improvements on the topological operators to remove those elements are clearly needed. Even so, in 2-D and 3-D, the edge swapping procedure was successful in producing meshes of a quality suitable for use in a finite element method.

Another interesting trend across space dimensions is the ratio of the number of elements to the number of mesh points. This ratio is 1.79, 5.17, 15.43 for the 2-D, 3-D and 4-D cases, respectively. Notice that this corresponds approximately to the number

d	Time (s) per 1000 points	Memory (MB) per 1000 points
2-D	3.04	0.60
3-D	54.26	4.17
4-D	374.9	4.38

Table 8.2: Analytic metric. Execution time and memory consumption per 1000 points for the mesh optimization procedure.

of simplicial elements needed to subdivide a quadrilateral element in 2-D (two simplicial elements) and a hexahedral element in 3-D (five or six simplicial elements). Interestingly, the value of this ratio increases by a factor of approximately 3, when going from 2-D to 3-D, and also by the same factor of approximately 3 when going from 3-D to 4-D.

The execution times for the mesh optimization procedure, not counting the CGNS file reading and writing, and the memory usage per 1,000 mesh points are presented in Table 8.2. The 3-D and 4-D anisotropic mesh adaptation procedures are, respectively, about 18 and 123 times more expensive than the 2-D case. The 4-D case is about 7 times more expensive than the 3-D case. It is important to highlight that the 2-D case started with a rather small mesh with 121 points compared to 1331 points for the 3-D case and 42974 points for the 4-D case. However, the general trend clearly indicates that the cost of anisotropic mesh adaptation drastically increases as the number of space dimension increases and generalizing about what can be obtained with such techniques only by studying 2-D meshes can be very misleading.

The percentages of execution time spent for splitting edges, collapsing edges, swapping edges and smoothing the mesh are shown in Table 8.3. The relative cost of splitting and collapsing compared to other mesh operations seems to decrease as the space dimension increases, whereas the cost of swapping edges and smoothing the mesh increases. Consid-

d	Splitting time %	Collapsing time %	Swapping time %	Smoothing time %
2-D	24.7	25.3	8.9	41.1
3-D	8.1	32.1	13.0	46.8
4-D	1.7	18.4	21.6	58.3

Table 8.3: Analytic metric. Percentage of execution time for each mesh operator for the mesh optimization procedure.

ering that about 50% of the time is spent for smoothing, one may explore the possibility of saving computational time by adjusting this procedure. However, smoothing was found necessary to maintain a good mesh quality, so that reducing the mesh smoothing could result in time saving but at the risk of compromizing element quality. For the percentage of time spent swapping edges, the heuristic strategy of selecting only one edge swap possibility, described in Chapter 6, keeps the cost of edge swapping relatively low compared to the smoothing and is sufficient to maintain a quality that is excellent in 2-D, good in 3-D, but insufficient in 4-D (at least for this anisotropic case with the current measure of quality that was used).

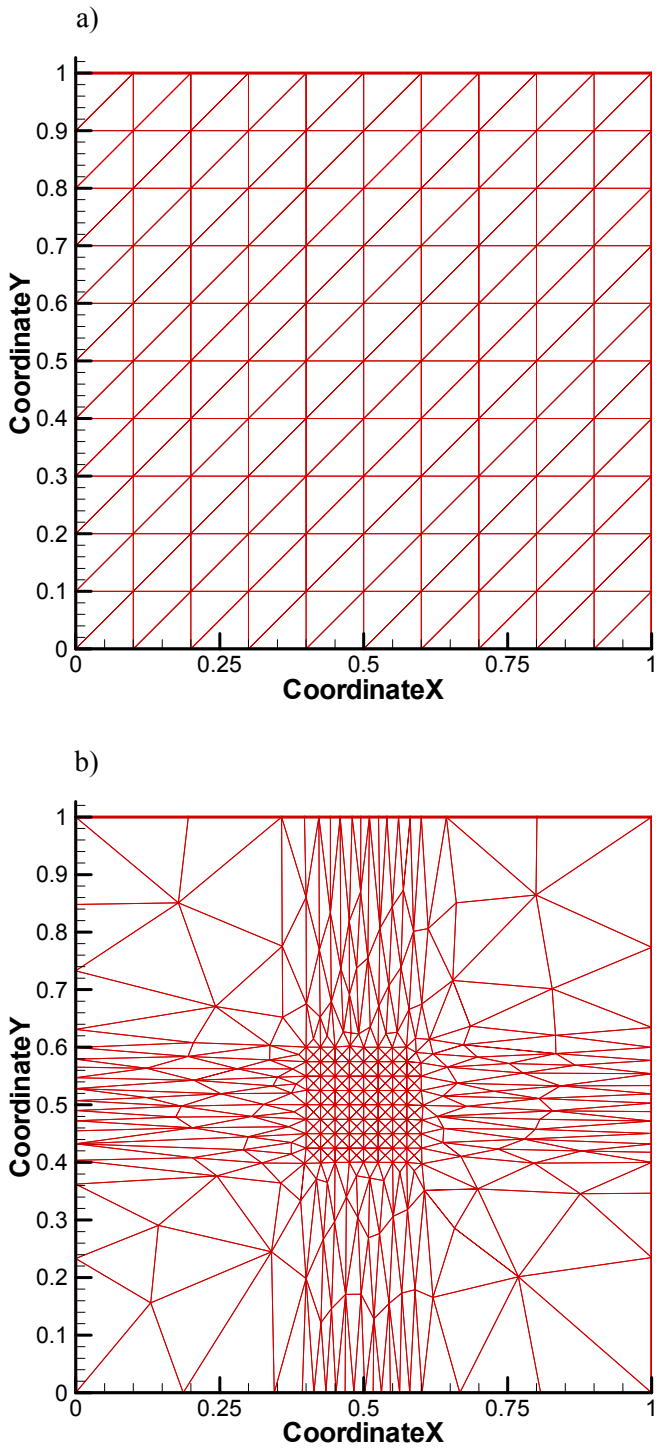

Figure 8.1: Analytic metric 2-D. a) Initial mesh; b) optimized mesh.

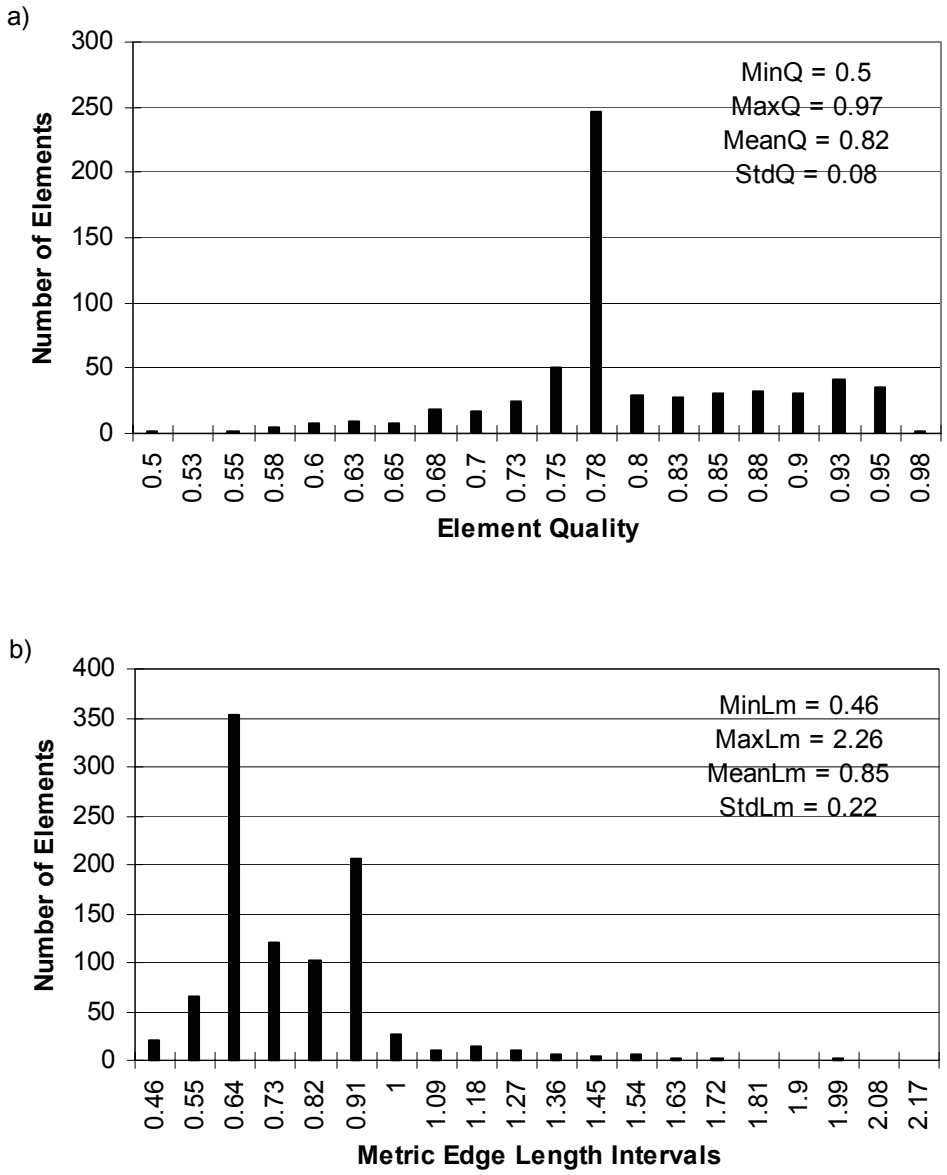

Figure 8.2: Analytic metric 2-D. a) Histogram of element quality; b) histogram of metric edge length.

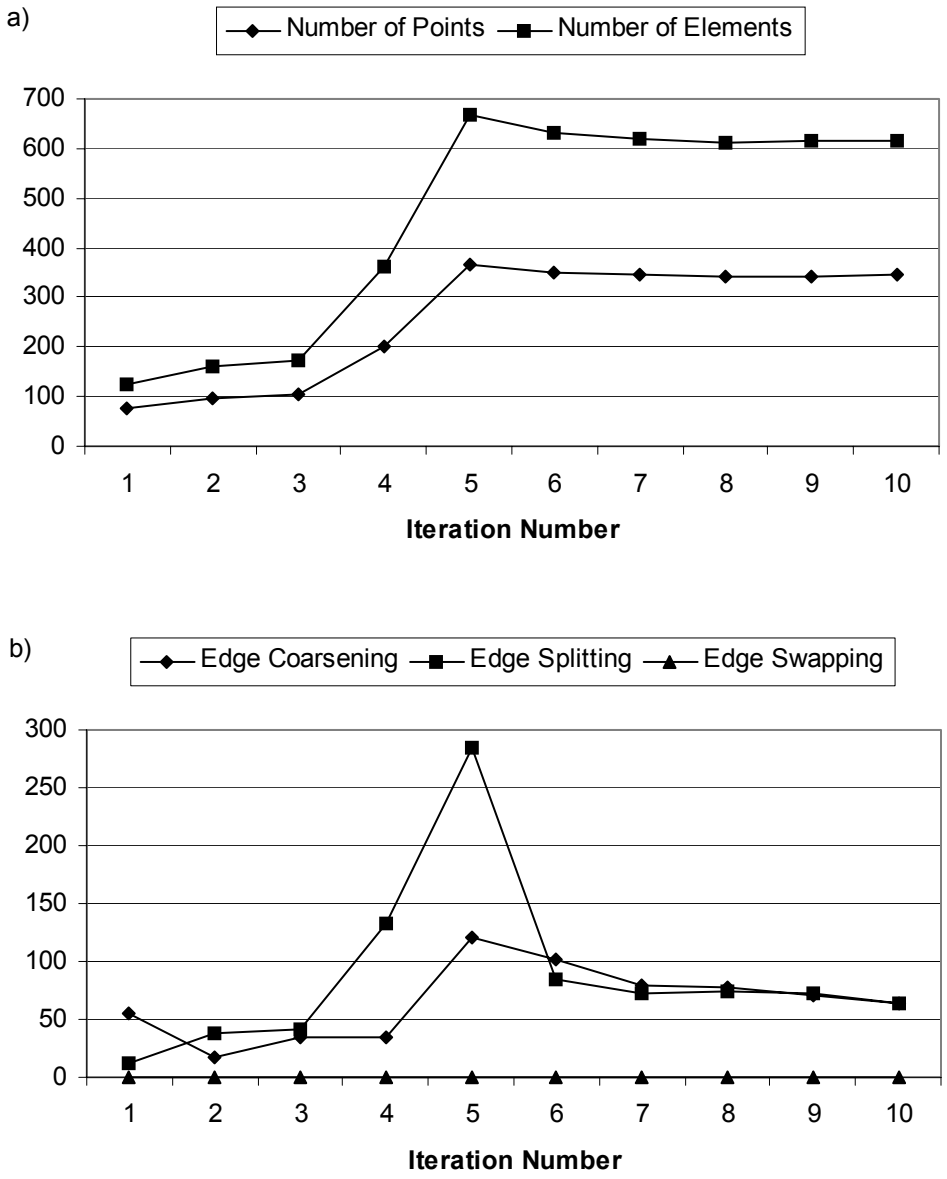

Figure 8.3: Analytic metric 2-D. a) Number of mesh points and elements; b) number of edge coarsening, edge splitting and edge swapping operations per internal mesh adaptation iteration.

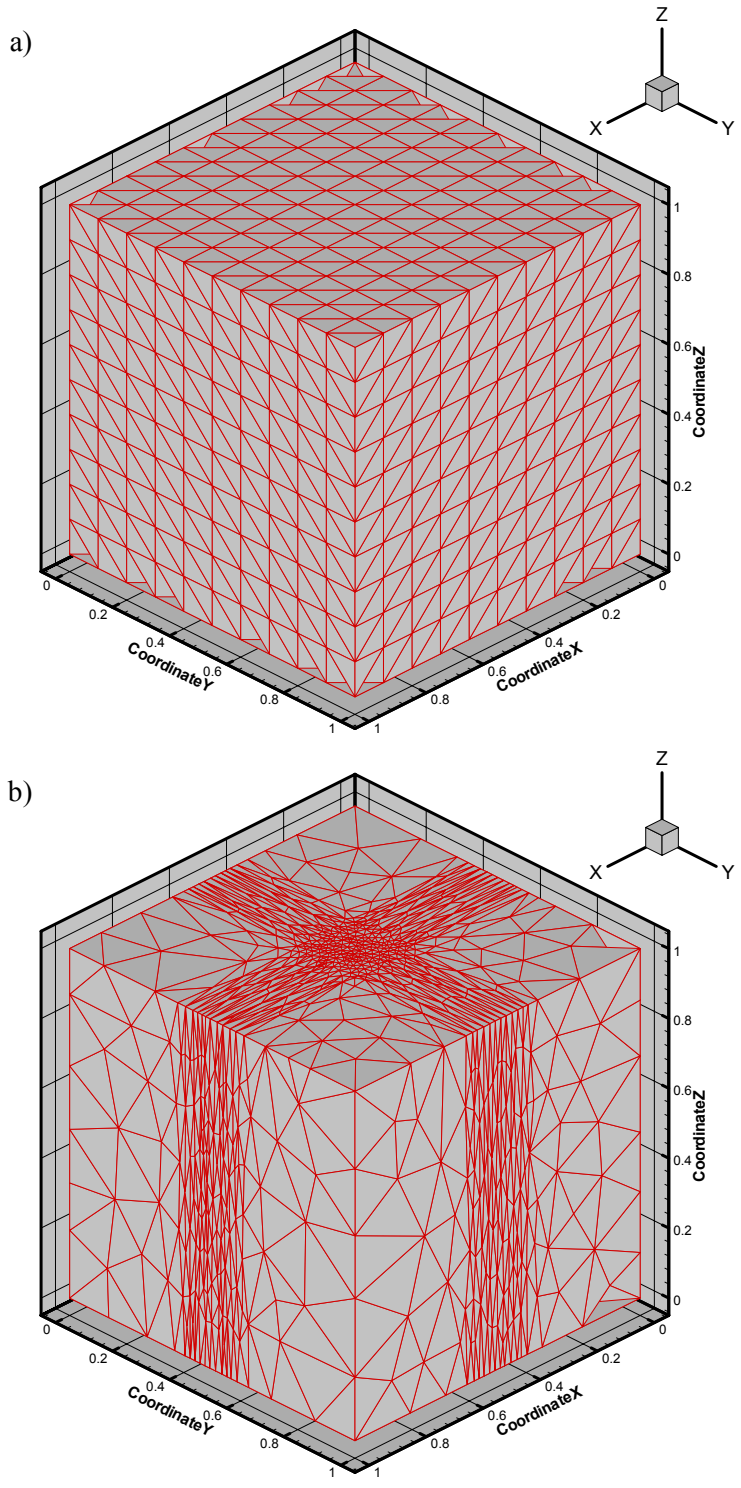

Figure 8.4: Analytic metric 3-D. a) Initial mesh; b) optimized mesh.

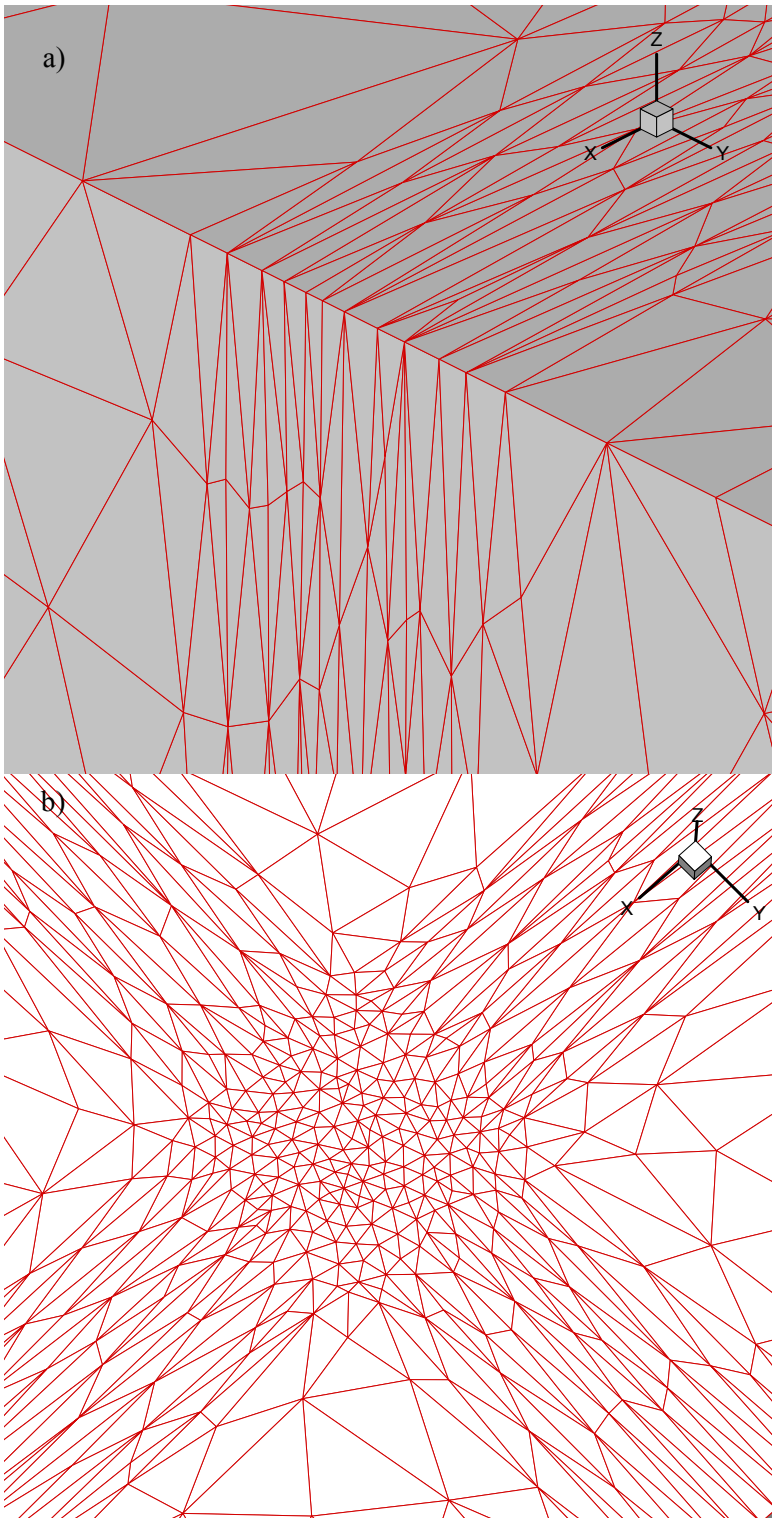

Figure 8.5: Analytic metric 3-D. Details of mesh faces corresponding to a) $x = 1$ and $z = 1$; b) $z = 1$.

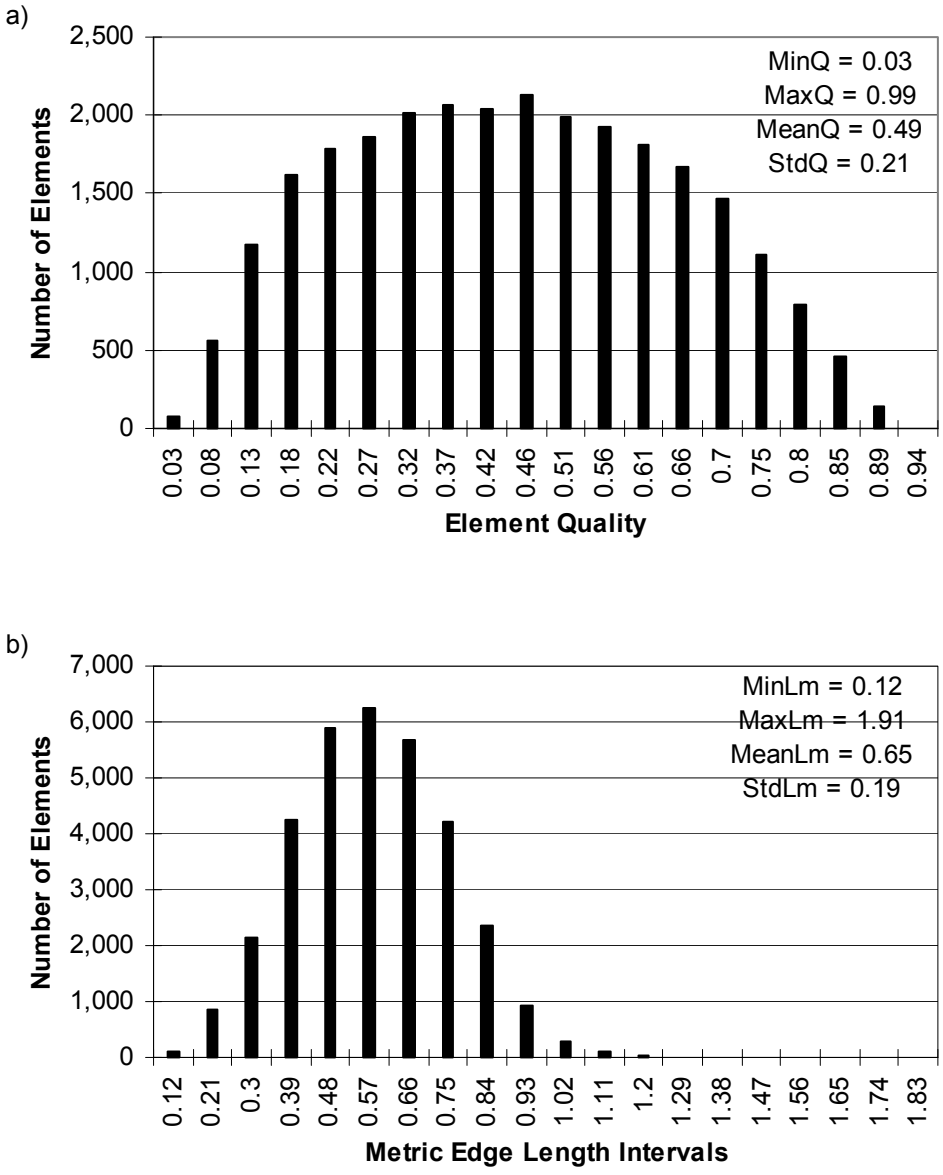

Figure 8.6: Analytic metric 3- D. a) Histogram of element quality; b) histogram of metric edge length.

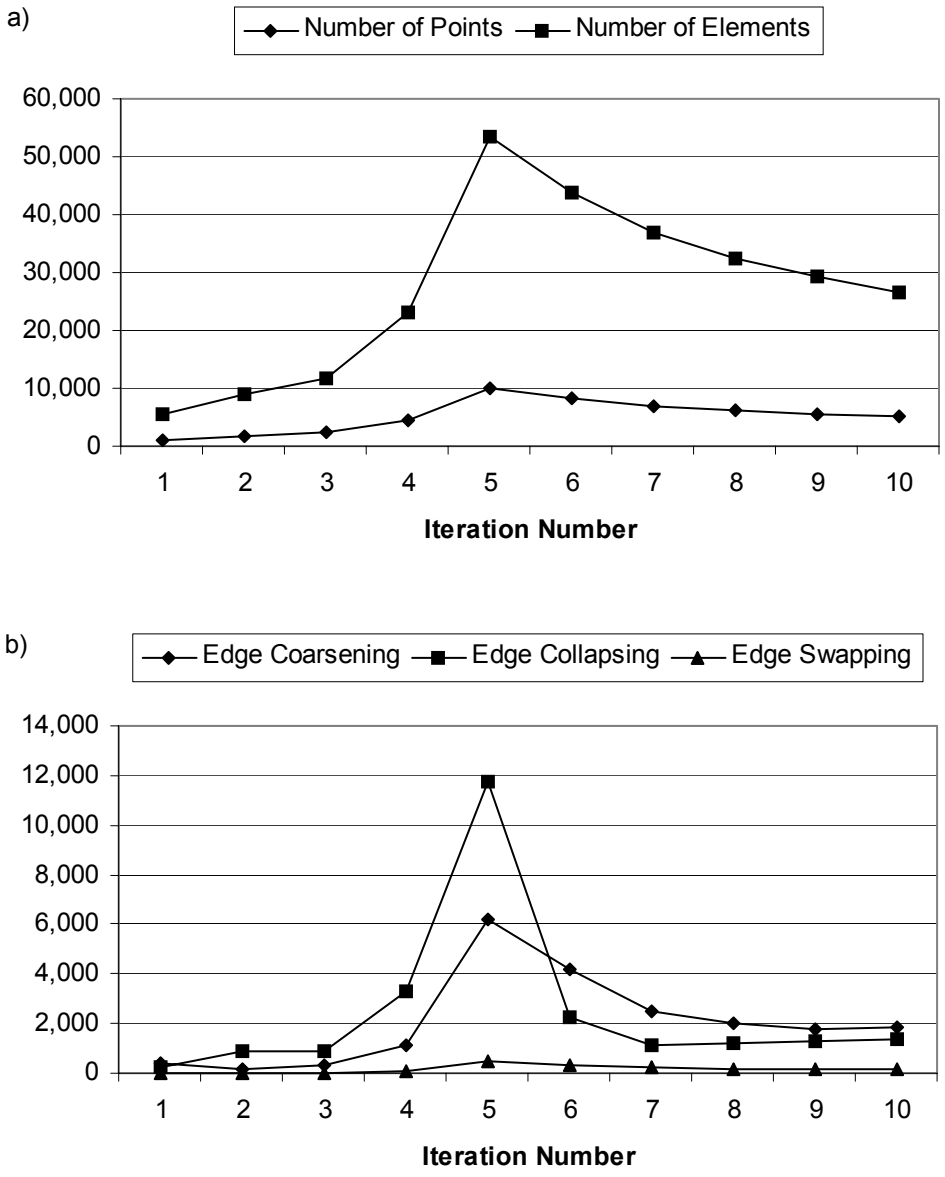

Figure 8.7: Analytic metric 3-D. a) Number of mesh points and elements; b) number of edge coarsening, edge splitting and edge swapping operations per internal mesh adaptation iteration.

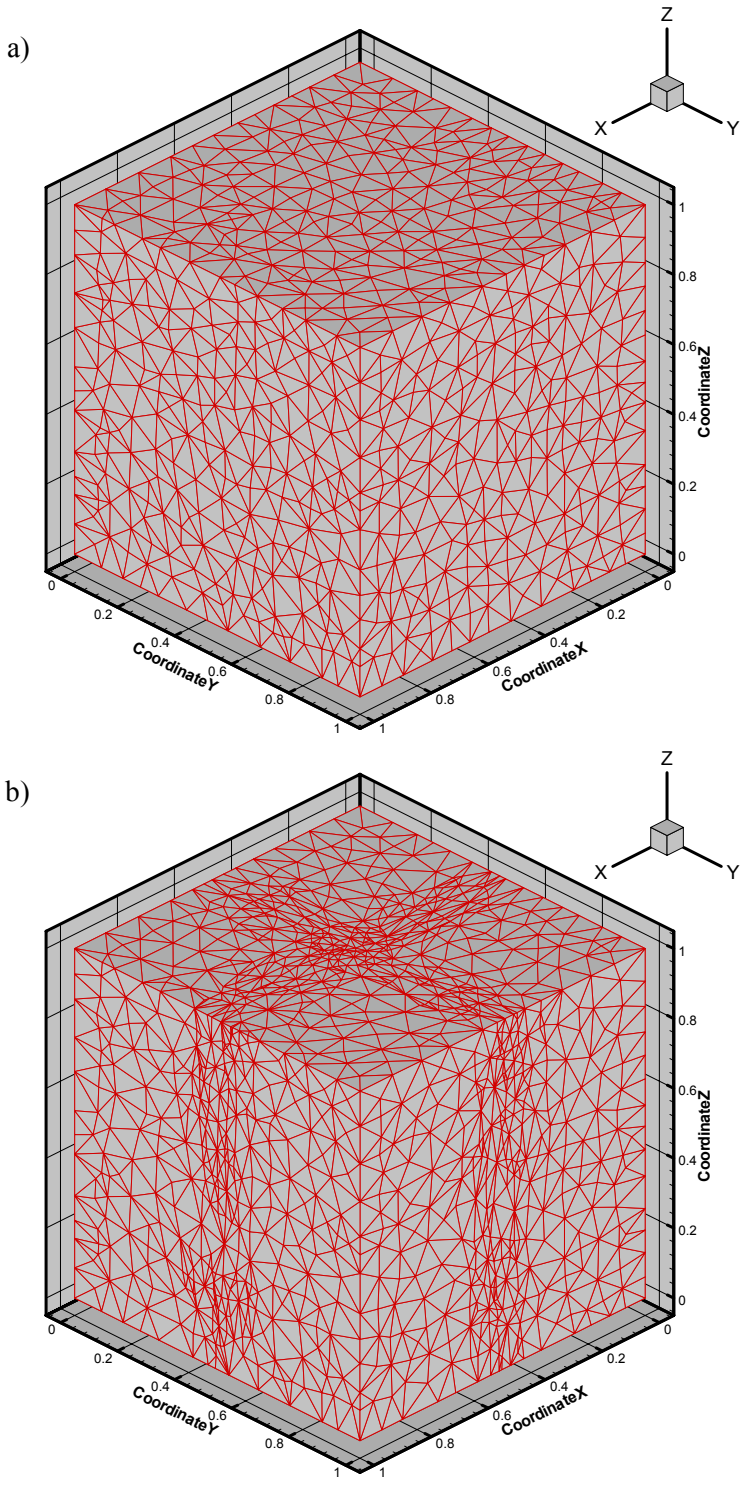

Figure 8.8: Analytic metric 4-D. a) Initial mesh at $t = 0$; b) optimized mesh at $t = 0$.

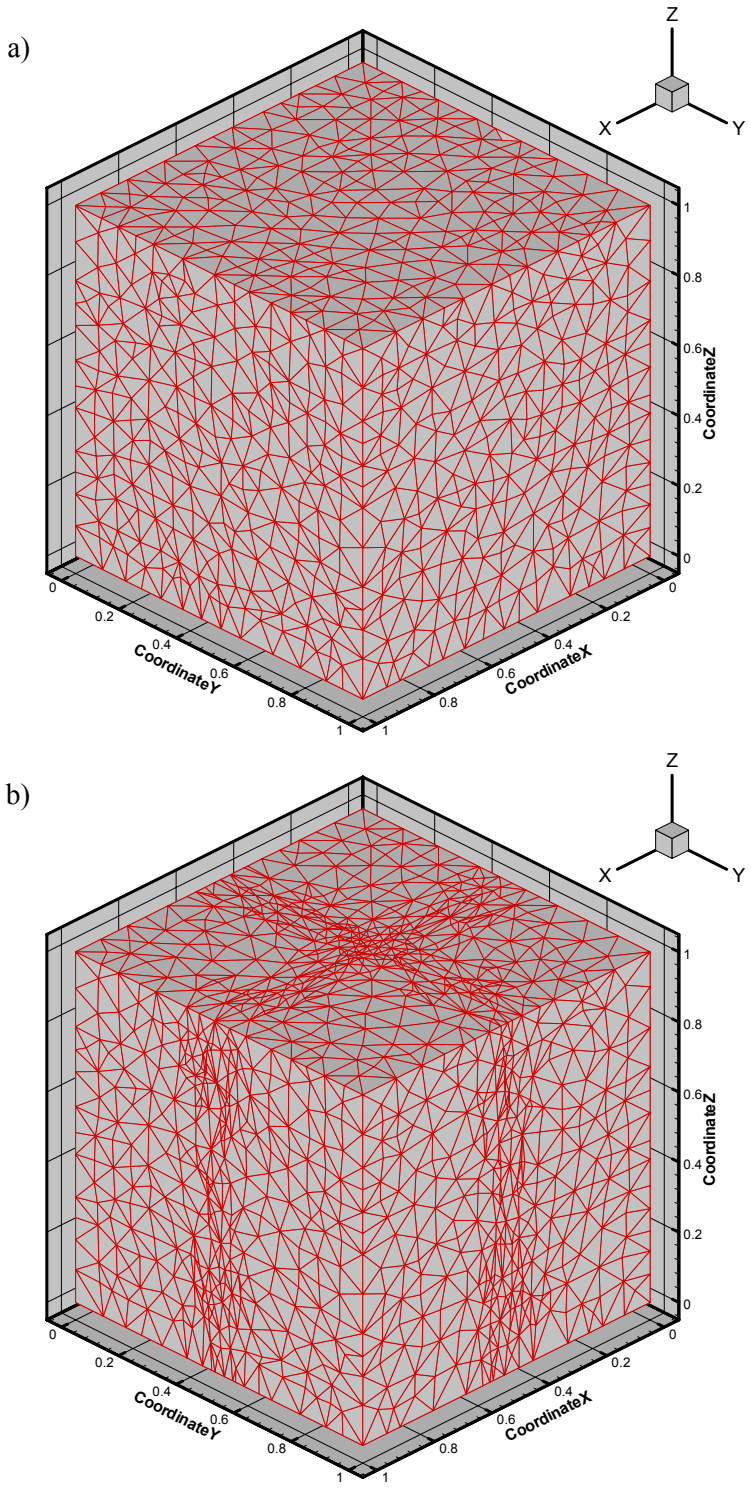

Figure 8.9: Analytic metric 4-D. a) Initial mesh at $t = 1$; b) optimized mesh at $t = 1$.

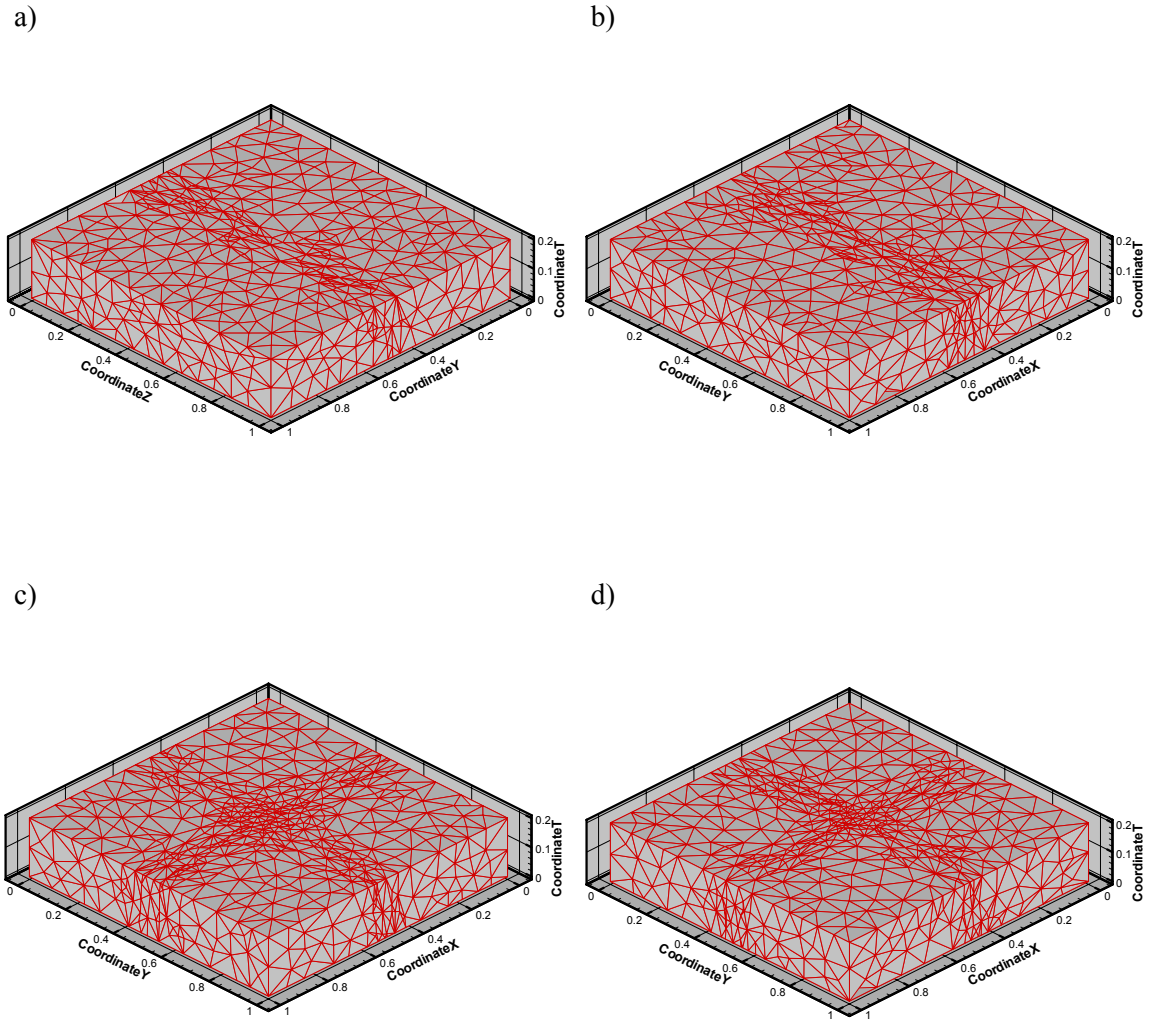

Figure 8.10: Analytic metric 4-D. Adapted mesh for plane a) $x = 0$; b) $y = 0$; c) $z = 0$; d) $z = 1$.

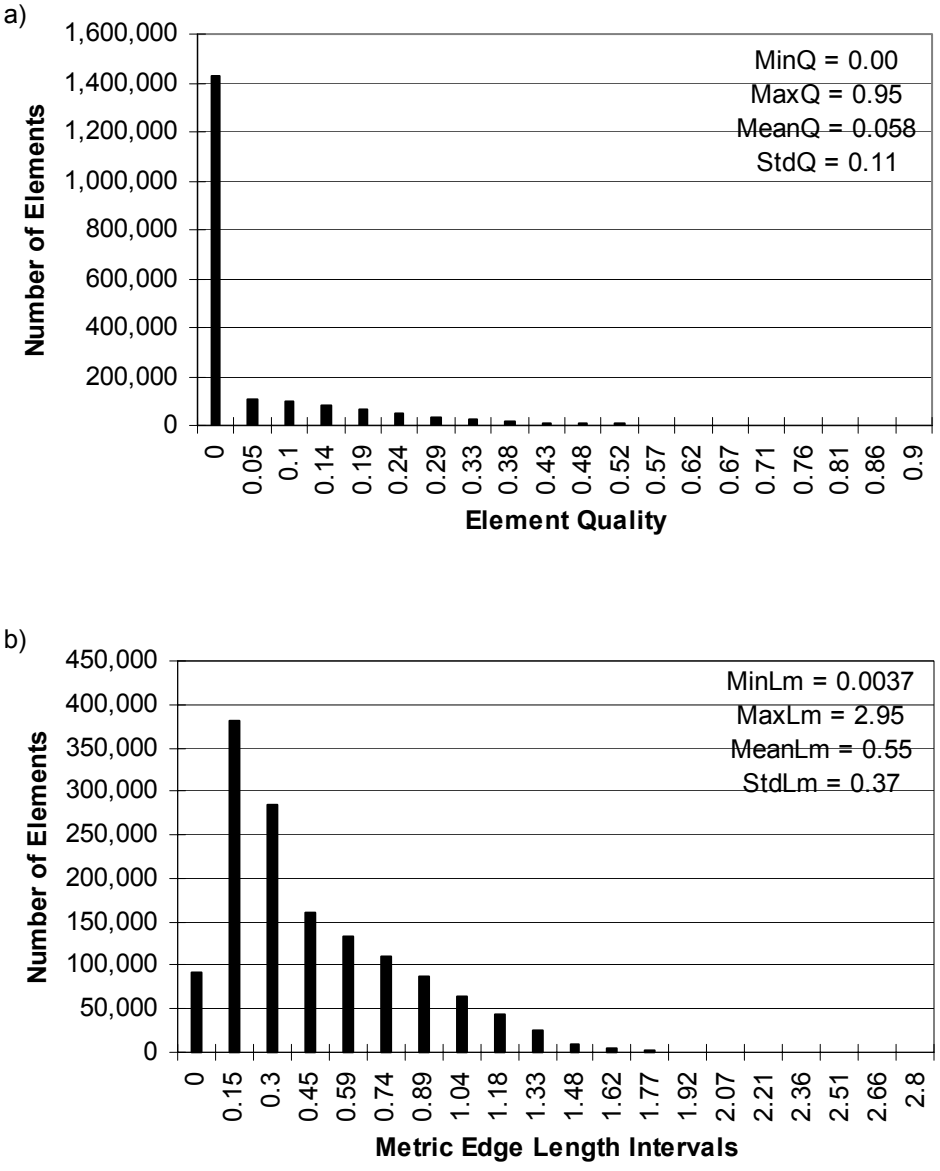

Figure 8.11: Analyic metric 4-D. a) Histogram of element quality; b) histogram of metric edge length.

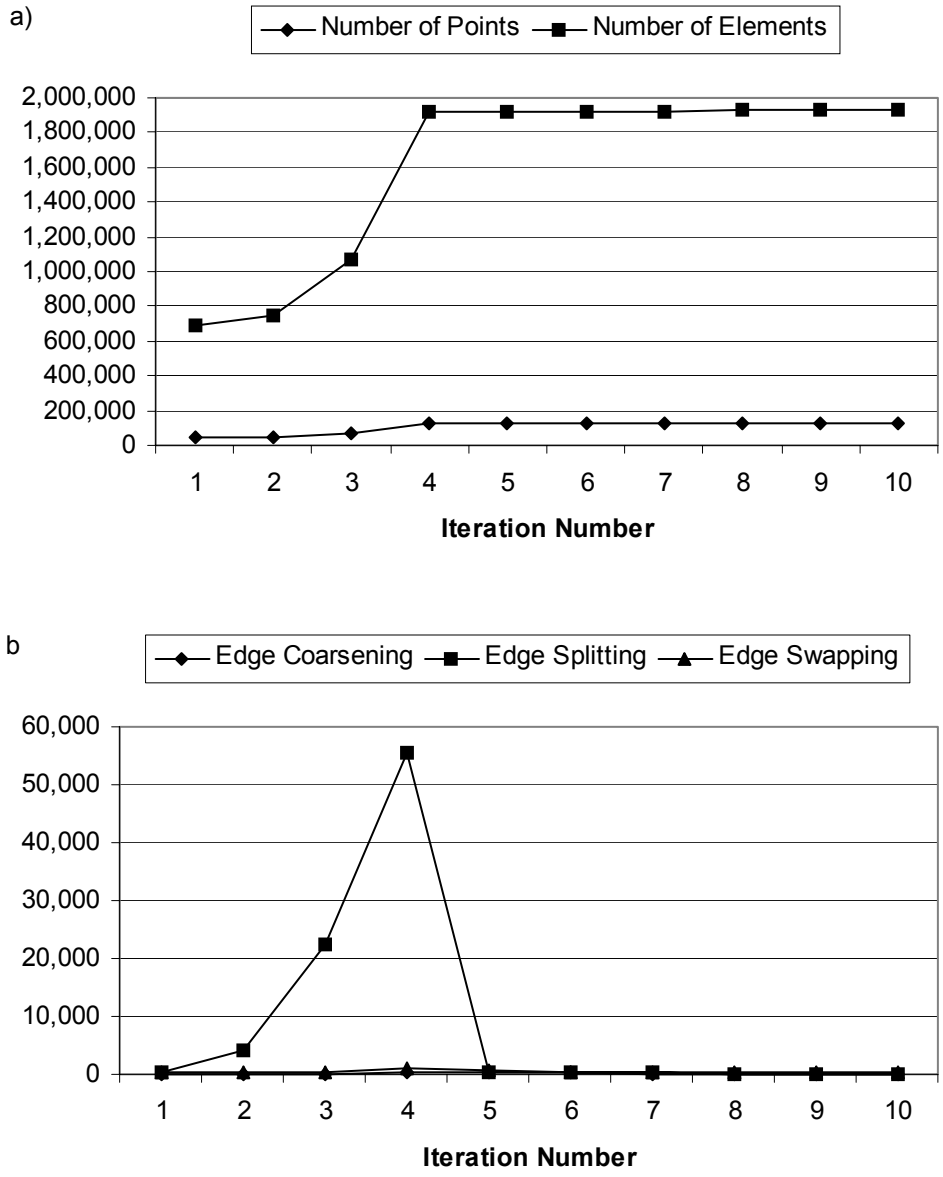

Figure 8.12: Analytic metric 4-D. a) Number of mesh points and elements; b) number of edge coarsening, edge splitting and edge swapping operations per internal mesh adaptation iteration.

8.3 Unsteady Heat Transfer in Boxes with 2-D, 3-D and 4-D Meshes

The first test case of the unified space-time finite element adaptive procedure is the solution of the linear unsteady heat equation

$$\frac{\partial T}{\partial t} = \alpha \nabla^2 T + \dot{g} \tag{8.3}$$

where α is the thermal diffusivity, T is the temperature and $\dot{g}$ is the rate at which energy is generated per unit volume of the medium. For a space-time formulation, the solution domain comprises both space and time, such that $\mathbf{x} \in \Omega \times (0, T) \in \Re^d$, with $d = 2$, 3 or 4.

Following the method of manufactured solutions (Roache (2002)), the analytical solution

$$T(\mathbf{x}) = G_0 \tanh\left(\frac{t}{\tau}\right) \prod_{i=1}^{d} \cos(x_i) \tag{8.4}$$

where

$$\alpha = 1, G_0 = 1, \tau = 0.2 \tag{8.5}$$

is inserted in equation 8.3 to compute the corresponding term $\dot{g}$. The specification of the problem is completed with the use of Dirichlet boundary conditions on the boundary $\partial\Omega \times (0, T)$ using the manufactured solution 8.4, as recommended by Roache (2002).

This test case was repeated in 2-D, 3-D and 4-D space-time domains of unit length along all space and time axes. The initial meshes use 10 intervals along the coordinate axes, as for the previous case of section 8.2, leading to a mesh with 200 simplicial elements and 121 points for the 2-D case and 6,000 simplicial elements and 1,331 points for the 3-D case. The initial mesh for the 4-D was generated as previously described in section 8.2, but with a

mesh size of $h_i = 1.0$ with a target metric edge length size of 0.5 leading to 4,557 simplicial elements and 417 mesh points.

For the 2-D case, 3 cycles of mesh adaptation were performed for a total of 4 solver executions. The error reduction factors for each mesh adaptation iteration were chosen, respectively, as 0.5, 0.5 and 0.8. The initial and adapted meshes after the 4 solver executions are shown in Figure 8.13, while the solutions before and after these operations are shown in Figure 8.14. Two details of the solution field with the mesh super-imposed are shown in Figure 8.15. Figures 8.16 show histograms for the element quality and the metric edge lengths.

The variation of L^2 error norm $\|T - T_h\|_0$ is shown in the lower part of Figure 8.17. The variation of error norm is usually shown in convergence analyses as a function of the characteristics element size h on uniform meshes. In the test cases presented in this thesis, the meshes are adapted anisotropically, so that the mesh size is not uniform throughout the mesh. Because the characteristics element size h is usually taken as the inverse of the number of subdivision intervals along each axis N_I, such that $h = 1/N_I$, and the volume of an element is proportional to h^d, where d is the dimension of the space-time domain, the characteristic size was chosen as $h = 1/N_p^{1/d}$, where N_p is the number of points in the space-time domain.

For the 3-D case, the error reduction factors used were 0.5 and 0.8 with 3 solver executions. Similar figures are shown for the 3-D case, where Figure 8.18 shows the 3-D initial and adapted meshes, Figure 8.19 shows the solutions on the initial and final adapted meshes, Figure 8.20 shows details of the temperature field with the mesh super-imposed,

Figure 8.21 shows histograms of the element quality and the metric edges length, Figure 8.22 shows the variations of the numbers of mesh points and elements vs. the solver iteration number and also the L^2 norm of the error.

For the 4-D case, error reduction factors of 0.6 and 0.8 were used with 3 solver executions. The initial and adapted meshes are shown in Figure 8.23 for the face elements corresponding to $t = 0$ and in Figure 8.24 for the face elements corresponding to $t = 1$. The solutions on the initial and adapted meshes are shown for $t = 0$, $x = 0$, $y = 0$, and $z = 0$ in figures 8.25, 8.26 and 8.27, 8.28, respectively. Histograms for the element quality and the metric edge lengths are shown in Figure 8.29. Notice that the minimum element quality for the 4-D adapted mesh in this case is 10^{-5} and the average quality is 0.061, which is slightly better than that for the previous case with the anisotropic analytical metric shown in section 8.2. By inspection of figures 8.23 and 8.24, it appears that the mesh is relatively coarse and mostly isotropic and in this case the mesh optimization maintains a higher minimum and average element quality than for the 4-D case with the anisotropic analytical metric field presented previously in section 8.2.

Table 8.4 presents the minimum, maximum, average and standard deviation of the mesh quality on the final adapted mesh as a function of the dimension of the space-time domain along with the corresponding numbers of mesh points and elements. The minimum element quality is higher in 2-D than in 3-D and 4-D, but in all cases the minimum quality is sufficient for the finite element method.

Table 8.5 shows $\|T - T_h\|_0$ as a function of the approximate characteristic mesh size $h = 1/N_p^{1/d}$. In general, the numerical solution is in good agreement with the analytical

d	MinQ	MaxQ	AvgQ	StdQ	NumPoints	NumElems
2-D	0.069	1.0	0.77	0.18	121	200
3-D	0.0017	0.99	0.33	0.19	9,805	50,872
4-D	0.000010	0.89	0.061	0.075	49,941	764,056

Table 8.4: Heat equation. Element quality and numbers of mesh points and elements for the mesh optimization procedure.

d	SolverIt	$\|T - T_h\|_0$	$1/N_p^{1/d}$
2-D	1	0.0878	0.0909
	2	0.0246	0.0700
	3	0.00702	0.0354
	4	0.00630	0.0275
3-D	1	0.155	0.0909
	2	0.0222	0.0467
	3	0.00918	0.0313
4-D	1	0.603	0.221
	2	0.187	0.109
	3	0.0925	0.0669

Table 8.5: Heat equation. Norm of the error.

solution and drops rapidly as the number of mesh points increases in the space-time domain. It is important to emphasize that this L^2 norm is computed on the entire space-time domain and therefore accounts for errors in both space and time.

The percentages of the total execution time, including both the mesh adaptation and the solver execution, that is spent only on mesh adaptation are 96.8%, 99.3% and 99.8% for the 2-D, 3-D and 4-D cases, respectively. Clearly, for this simple linear heat equation, the time spent by the solver is negligible compared to the time spent for the anisotropic mesh adaptation procedure.

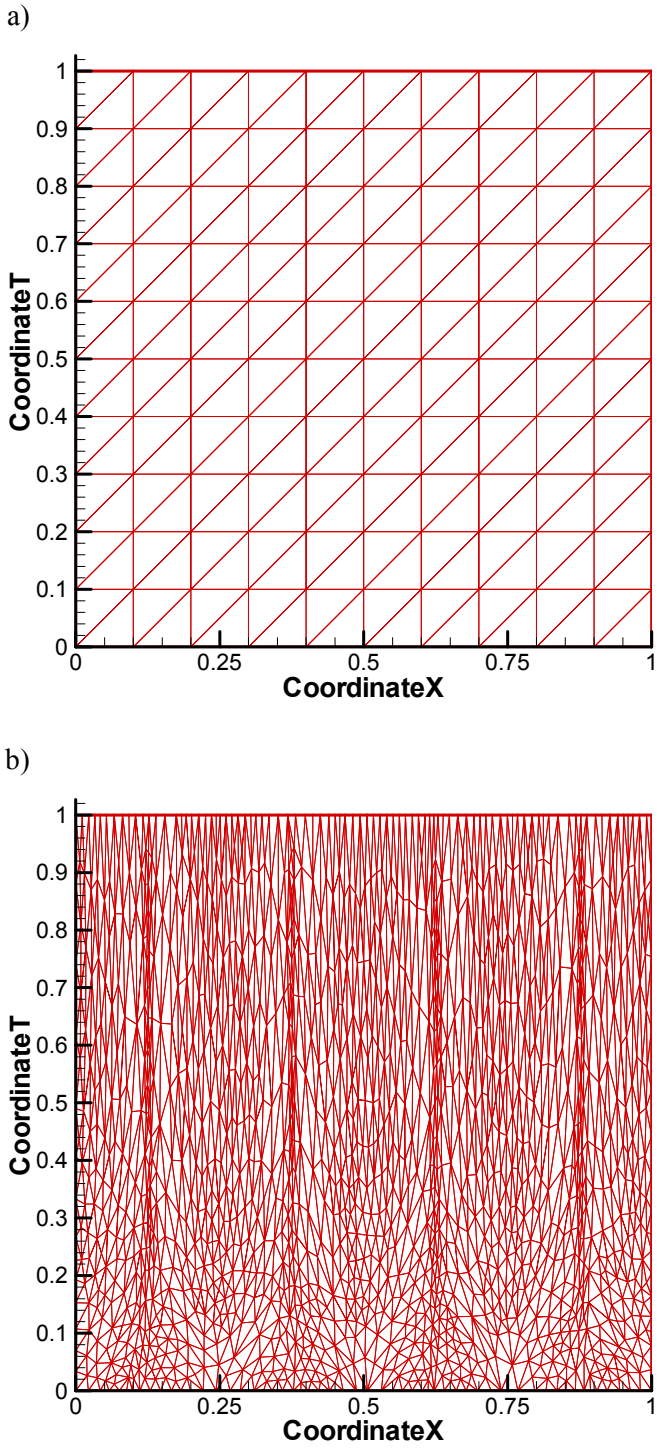

Figure 8.13: Unsteady 1-D heat transfer. a) Initial mesh; b) adapted mesh.

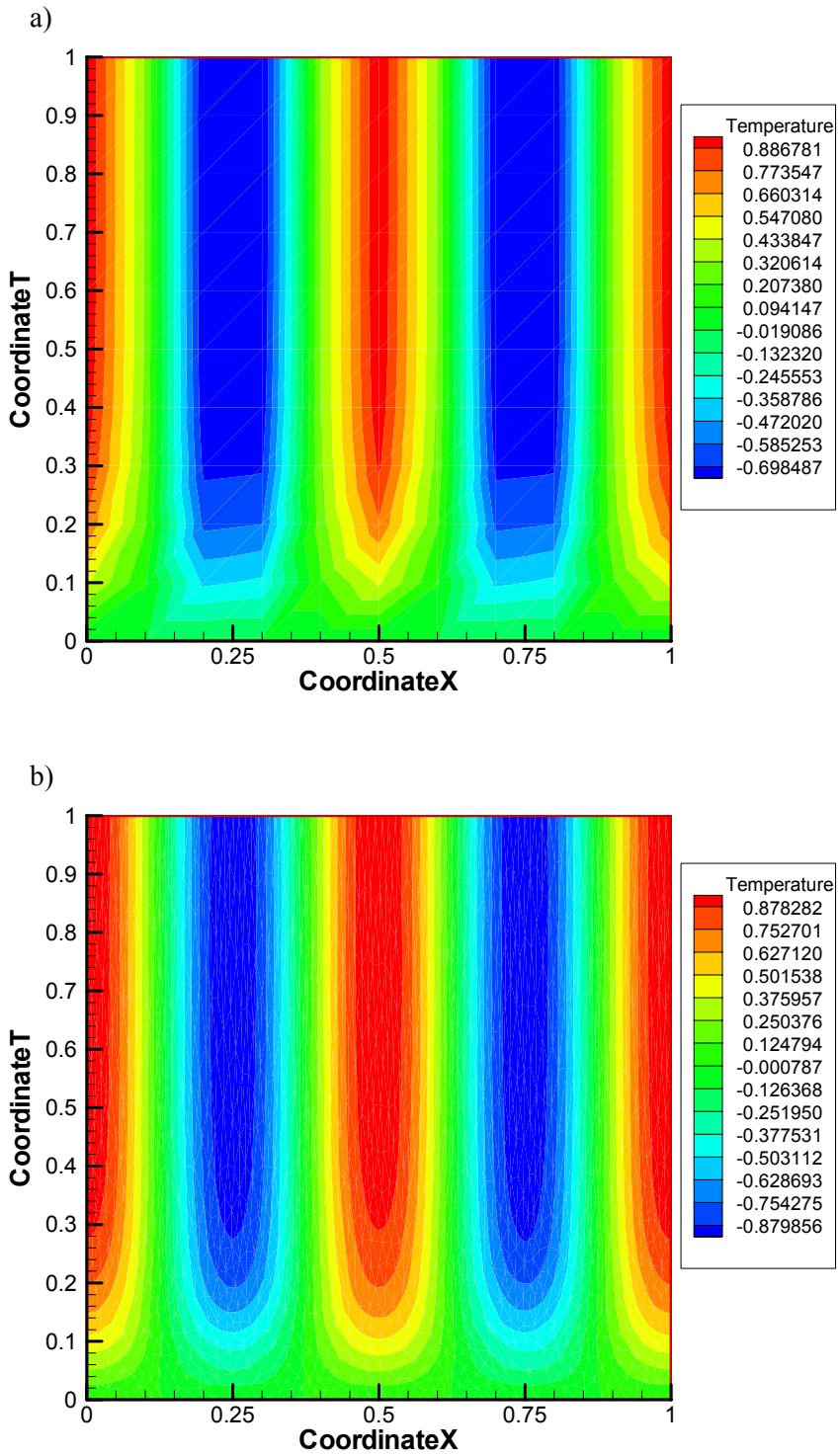

Figure 8.14: Unsteady 1-D heat transfer. a) Initial temperature field; b) adapted temperature field.

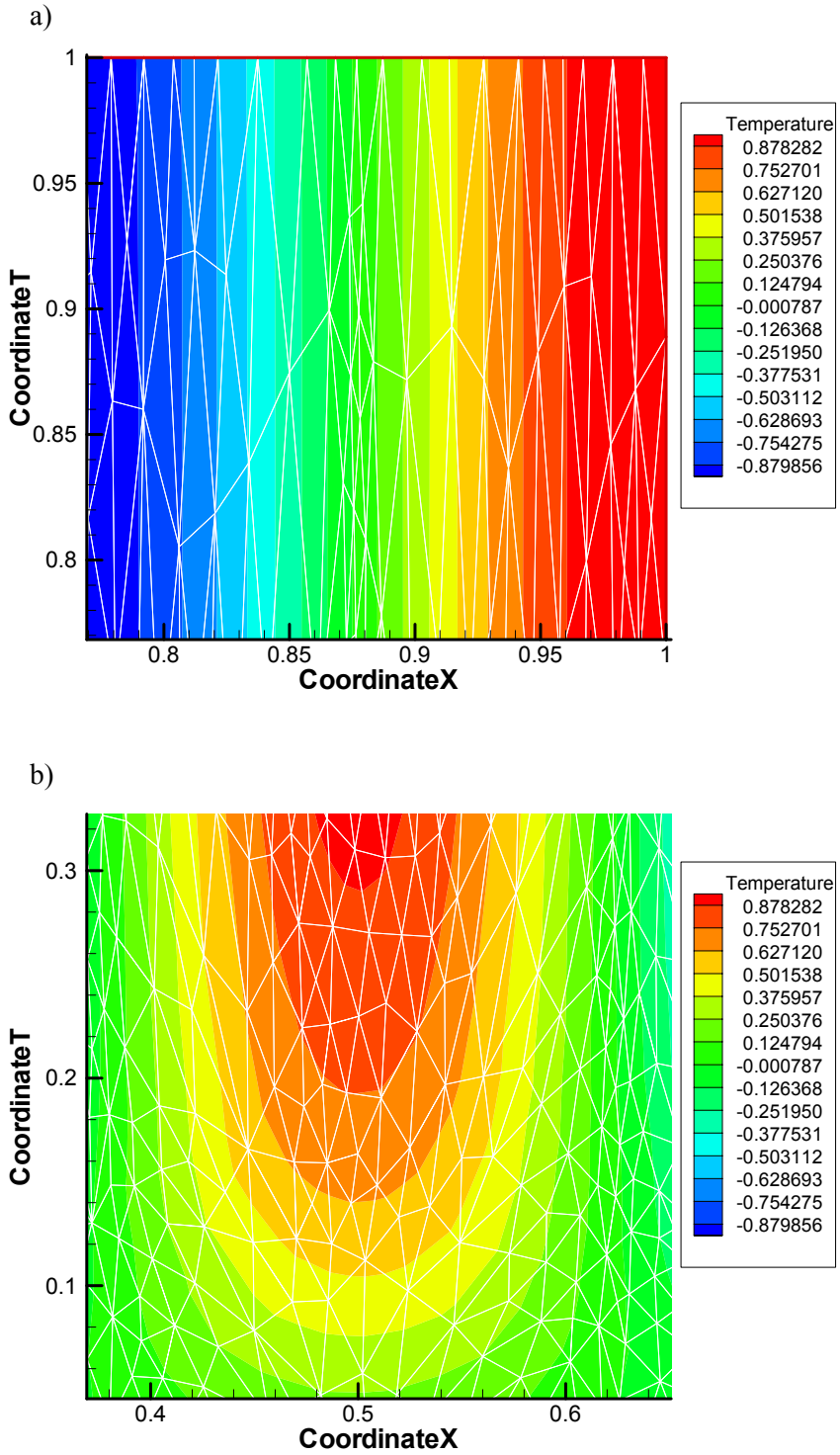

Figure 8.15: Unsteady 1-D heat transfer. a) Temperature field and mesh near corner point (1,1); b) temperature field and mesh near point (0.5,0.2).

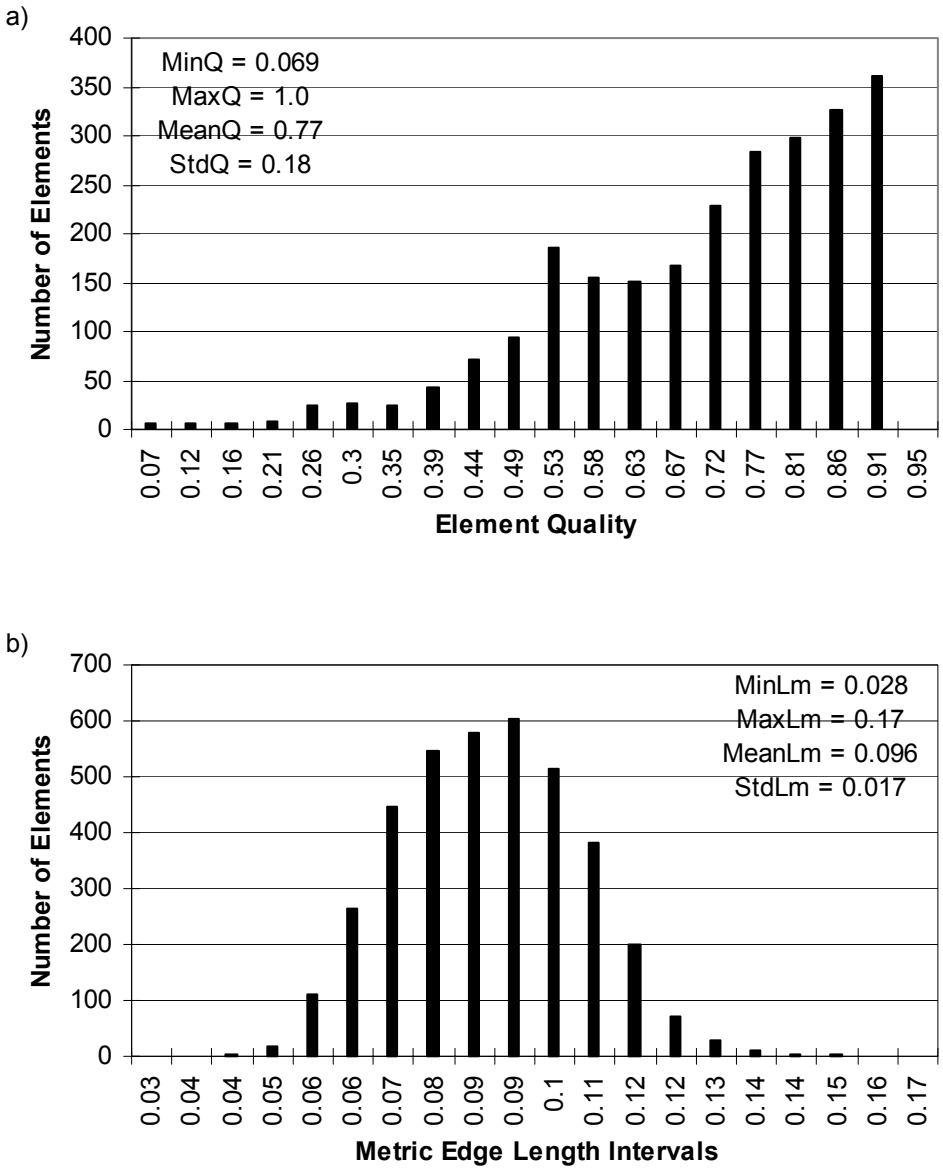

Figure 8.16: Unsteady 1-D heat transfer. a) Histogram of element quality; b) histogram of metric edge length.

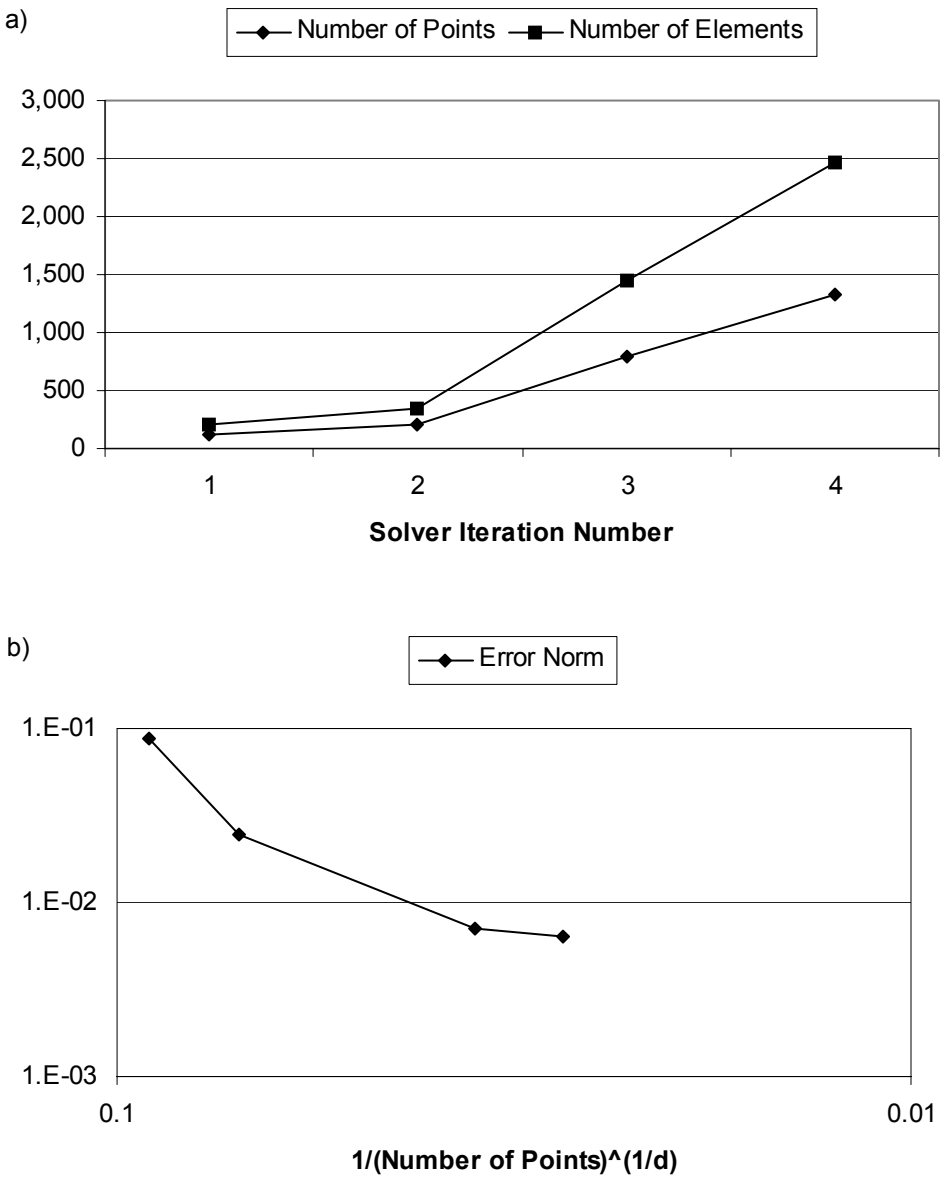

Figure 8.17: Unsteady 1-D heat transfer. a) Numbers of mesh points and elements; b) L^2 error norm as a function of $1/N_p^{1/d}$.

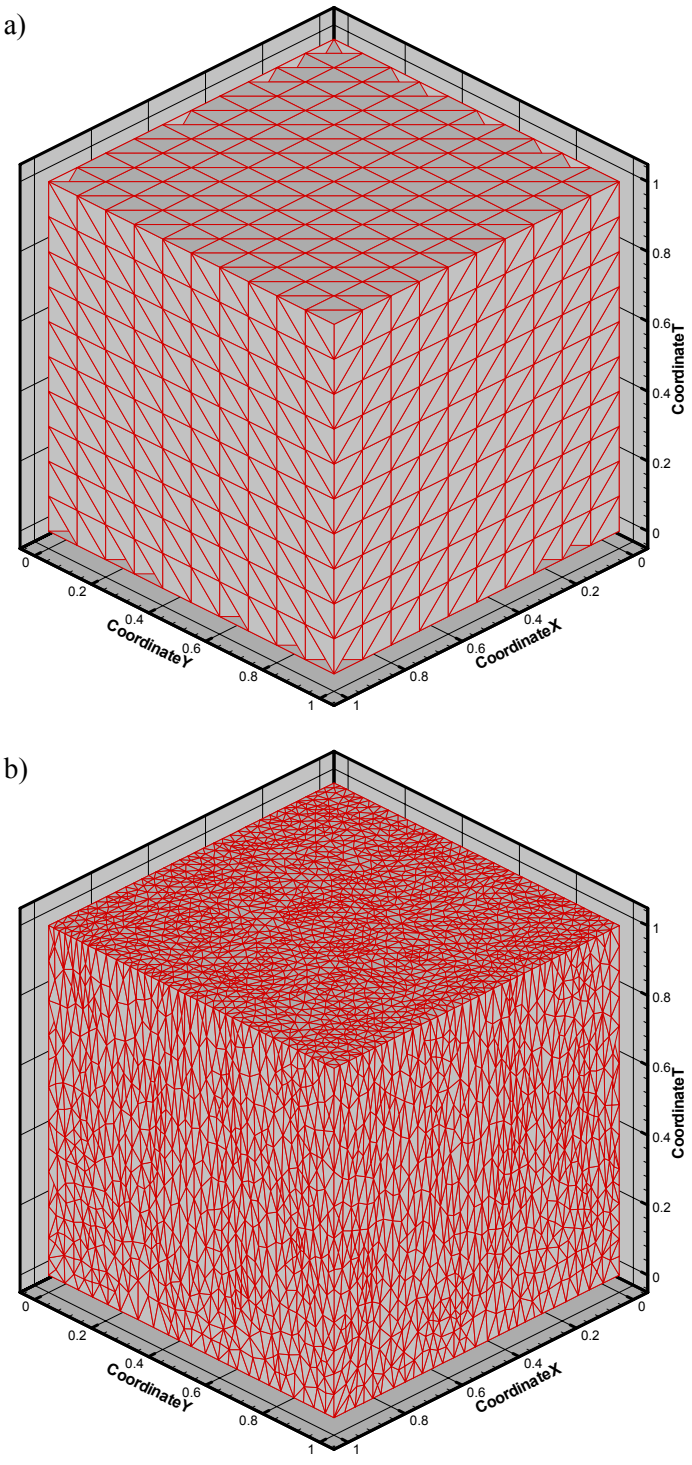

Figure 8.18: Unsteady 2-D heat transfer. a) Initial mesh; b) adapted mesh.

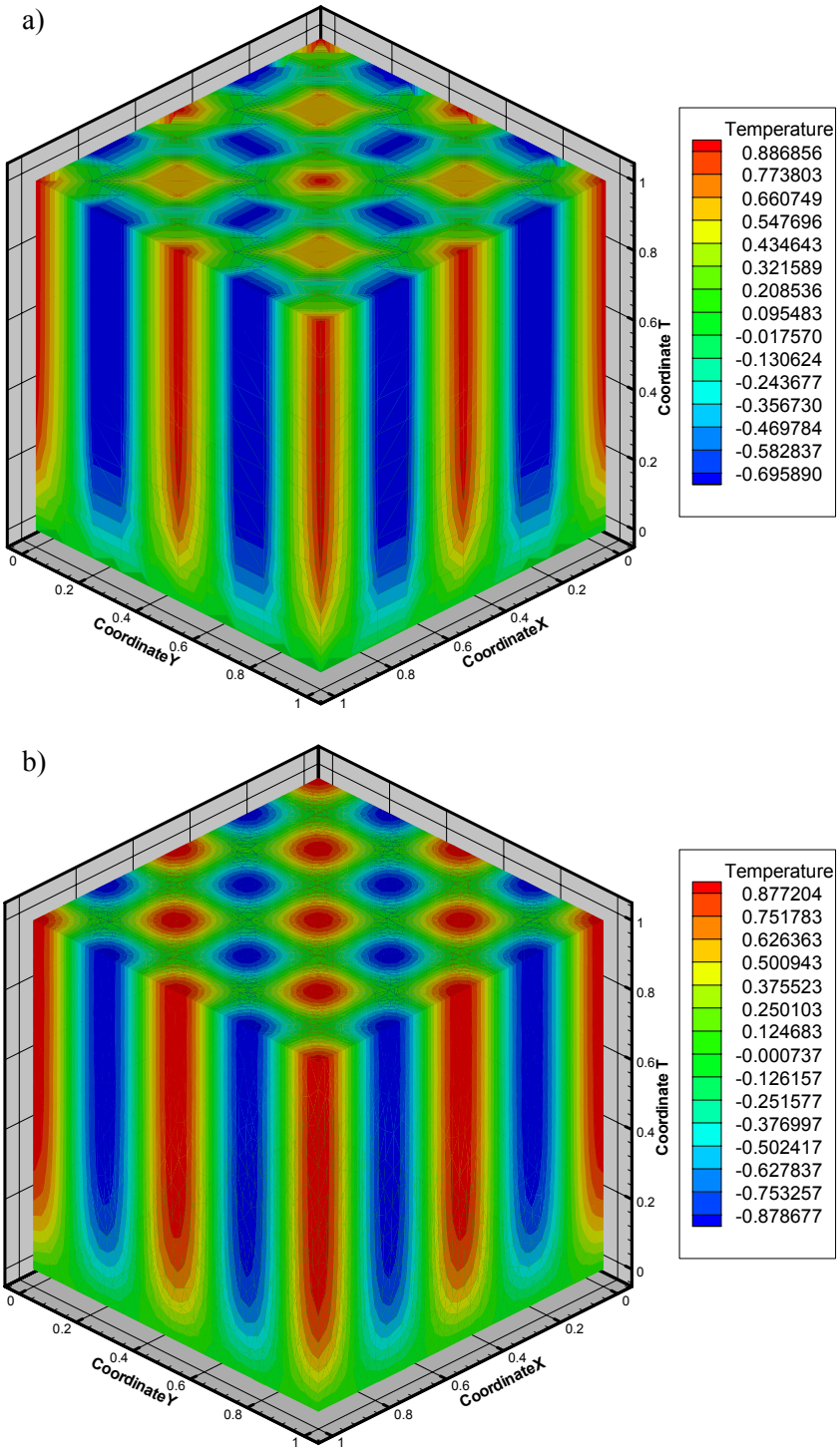

Figure 8.19: Unsteady 2-D heat transfer. a) Initial temperature field; b) adapted temperature field.

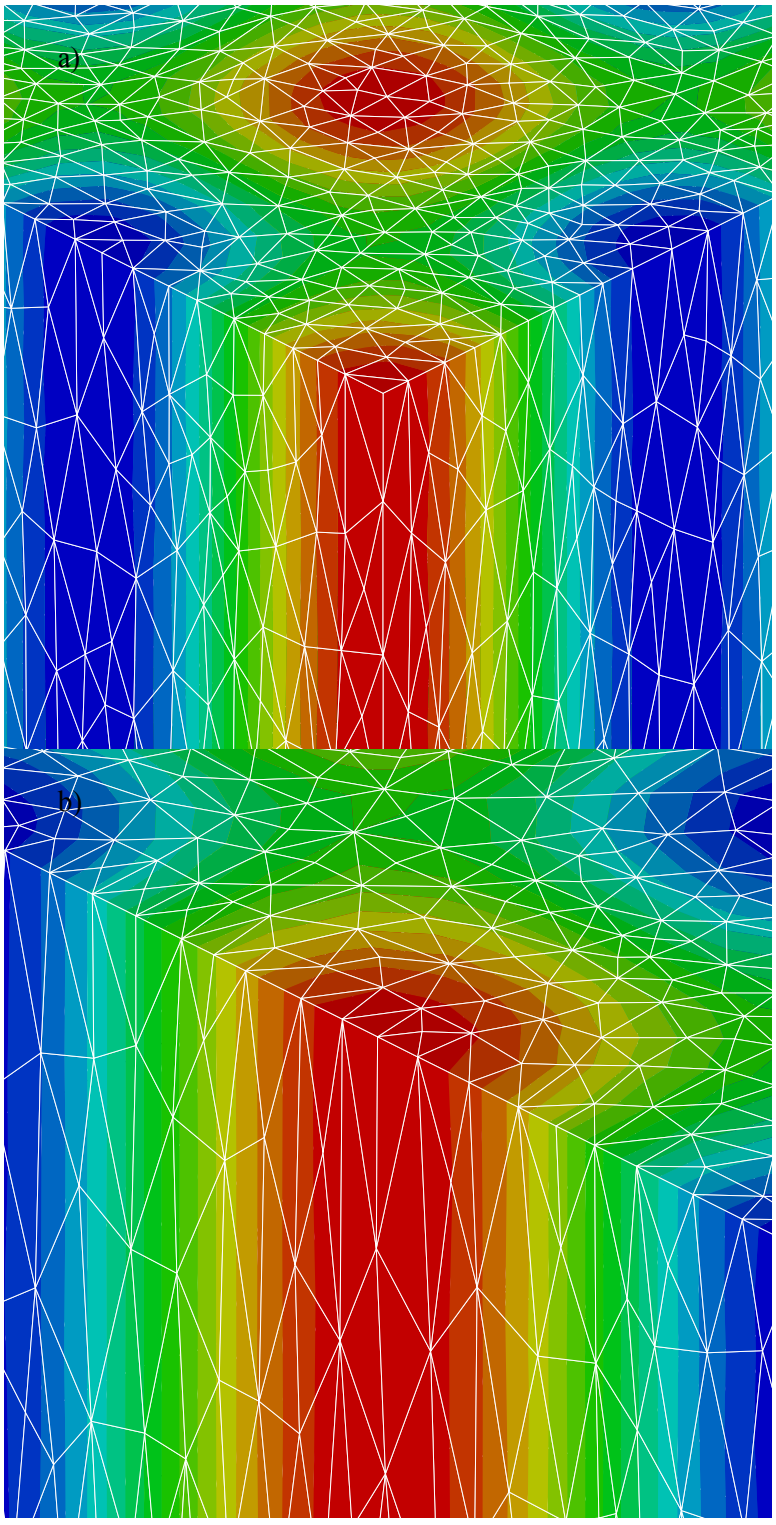

Figure 8.20: Unsteady 2-D heat transfer. a) Temperature field and mesh near corner point (1,1,1); b) Temperature field and mesh near point (1,0.5,1).

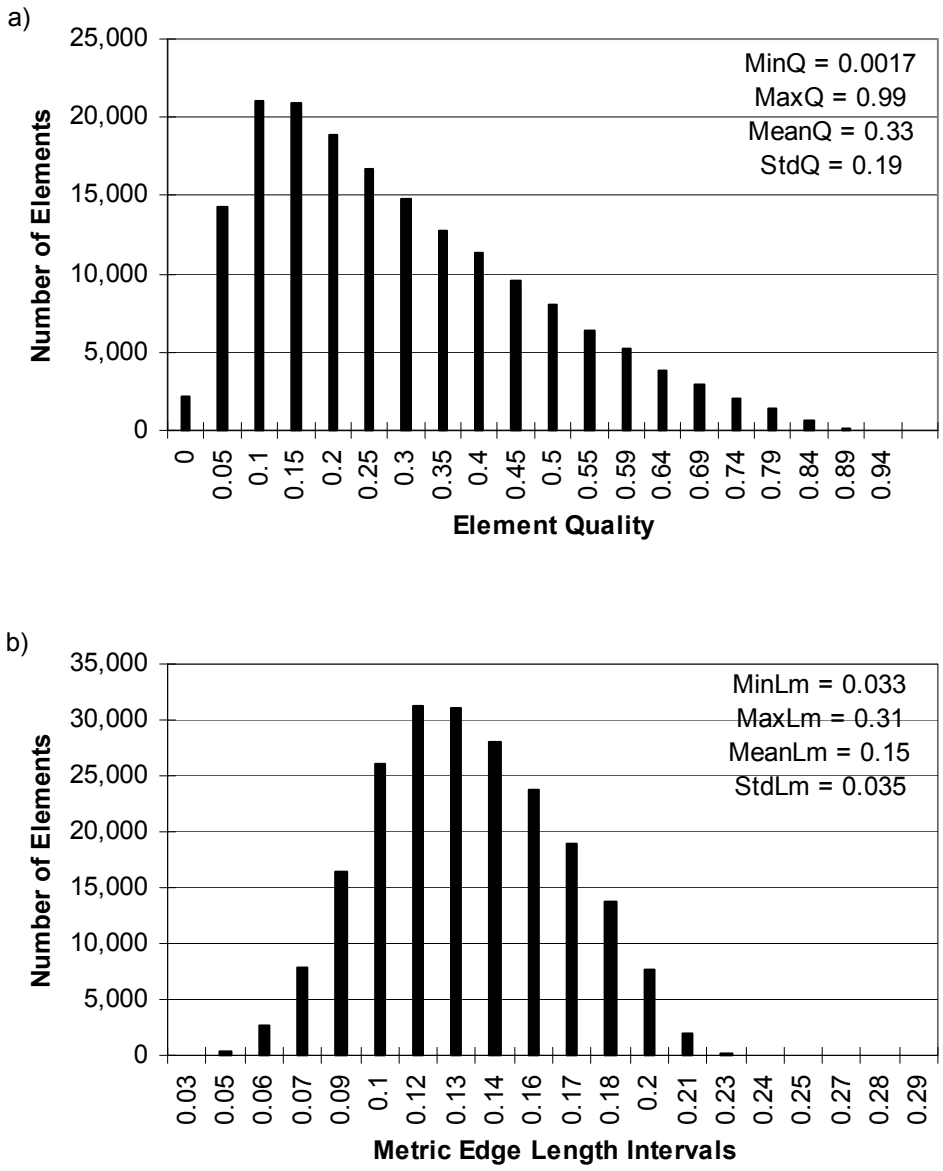

Figure 8.21: Unsteady 2-D heat transfer. a) Histogram of element quality; b) histogram of metric edge length.

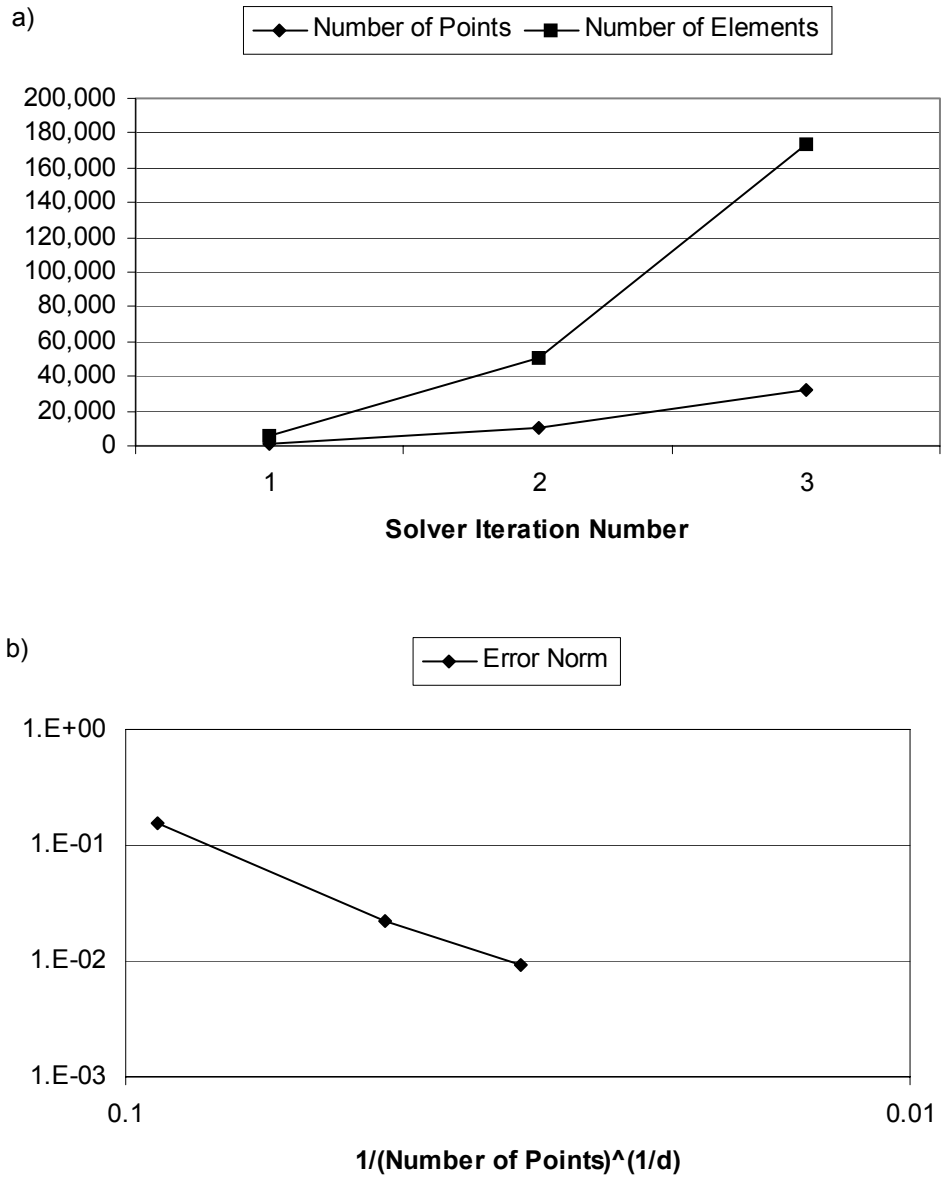

Figure 8.22: Unsteady 2-D heat transfer. a) Numbers of mesh points and elements; b) L^2 error norm as a function of $1/N_p^{1/d}$.

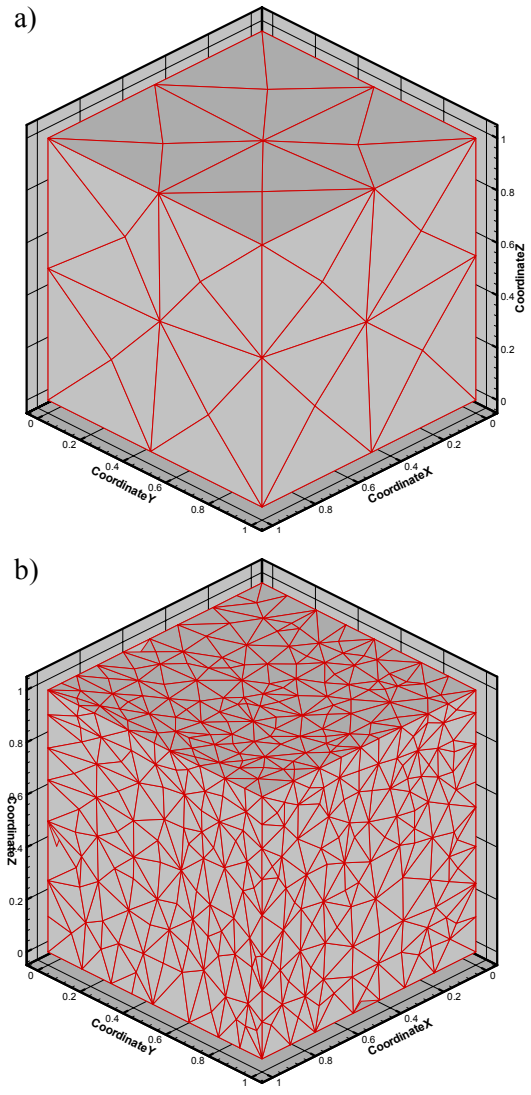

Figure 8.23: Unsteady 3-D heat transfer. a) Initial mesh at $t = 0$; b) adapted mesh at $t = 0$.

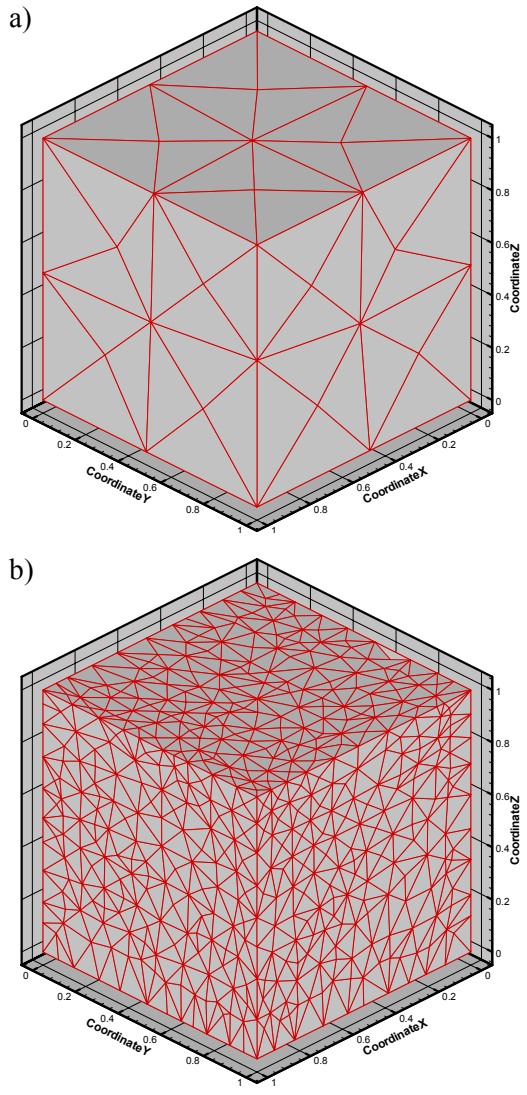

Figure 8.24: Unsteady 3-D heat transfer. a) Initial mesh at $t = 1$; b) adapted mesh at $t = 1$.

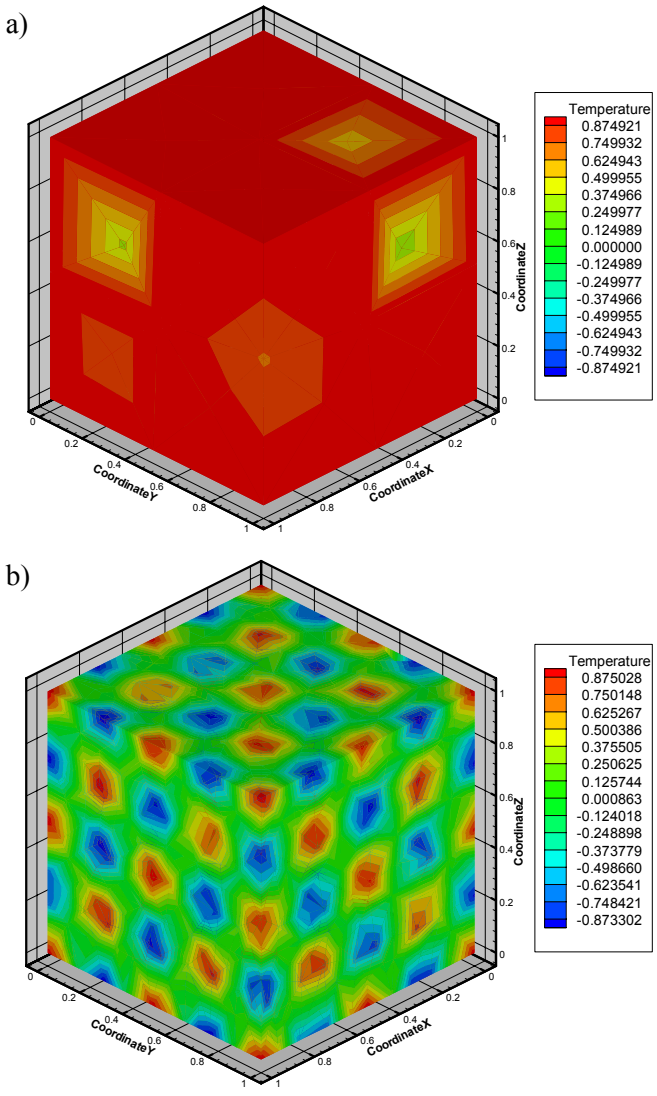

Figure 8.25: Unsteady 3-D heat transfer. a) Initial temperature field at $t = 1$; b) adapted solution at $t = 1$.

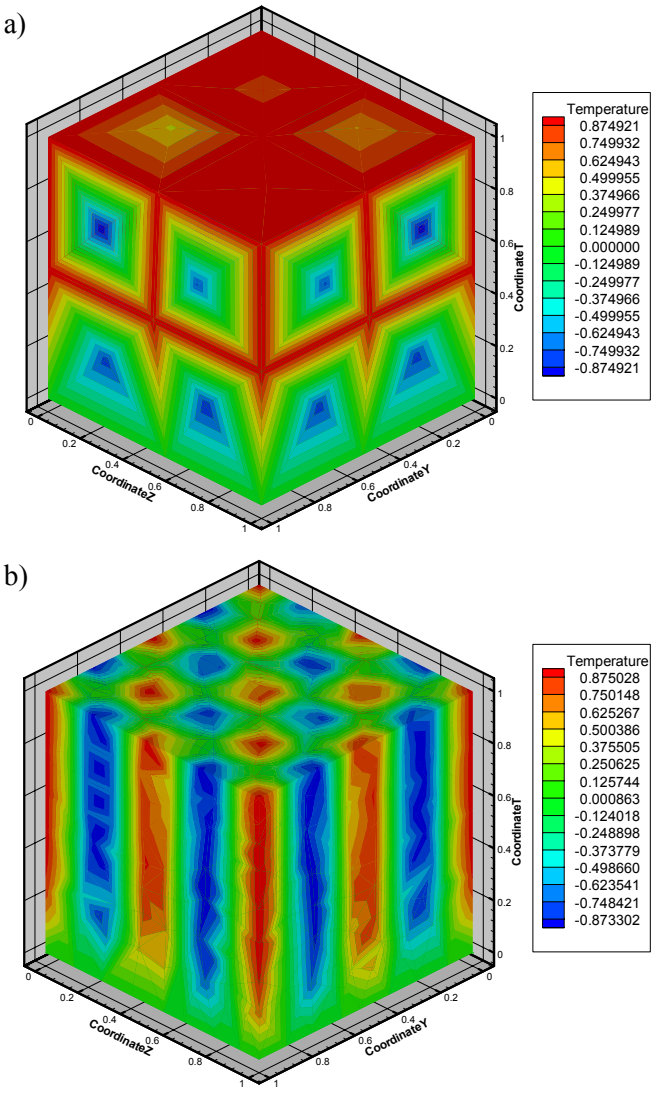

Figure 8.26: Unsteady 3-D heat transfer. a) Initial temperature field at $x = 0$; b) adapted solution at $x = 0$.

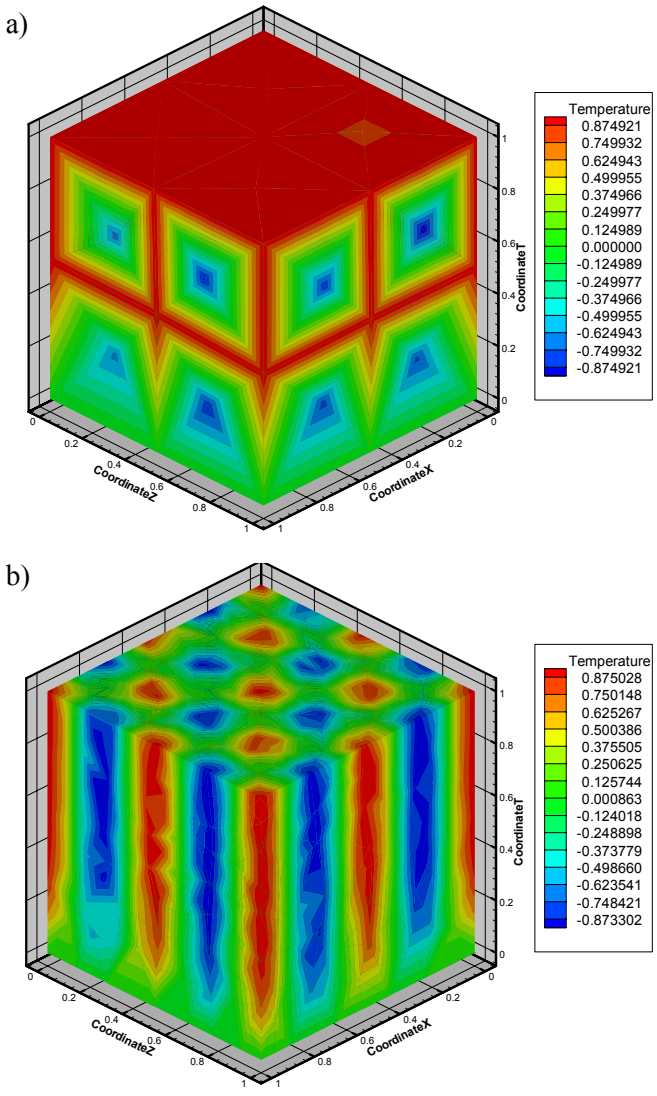

Figure 8.27: Unsteady 3-D heat transfer. a) Initial temperature field at $y = 0$; b) adapted solution at $y = 0$.

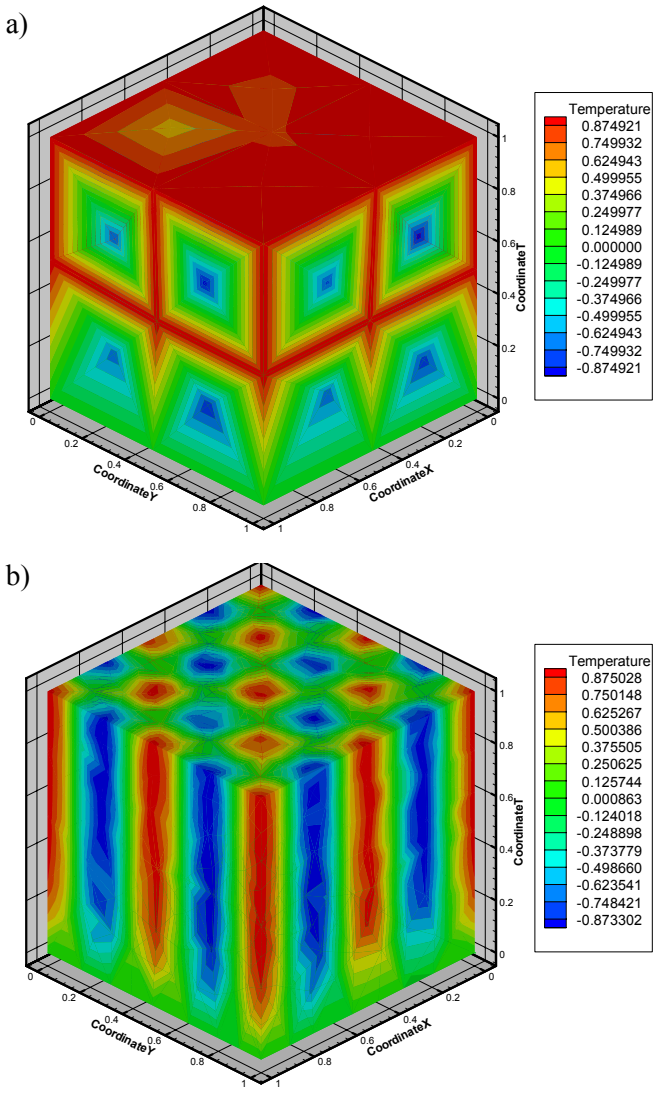

Figure 8.28: Unsteady 3-D heat transfer. a) Initial solution at $z = 0$; b) adapted solution at $z = 0$.

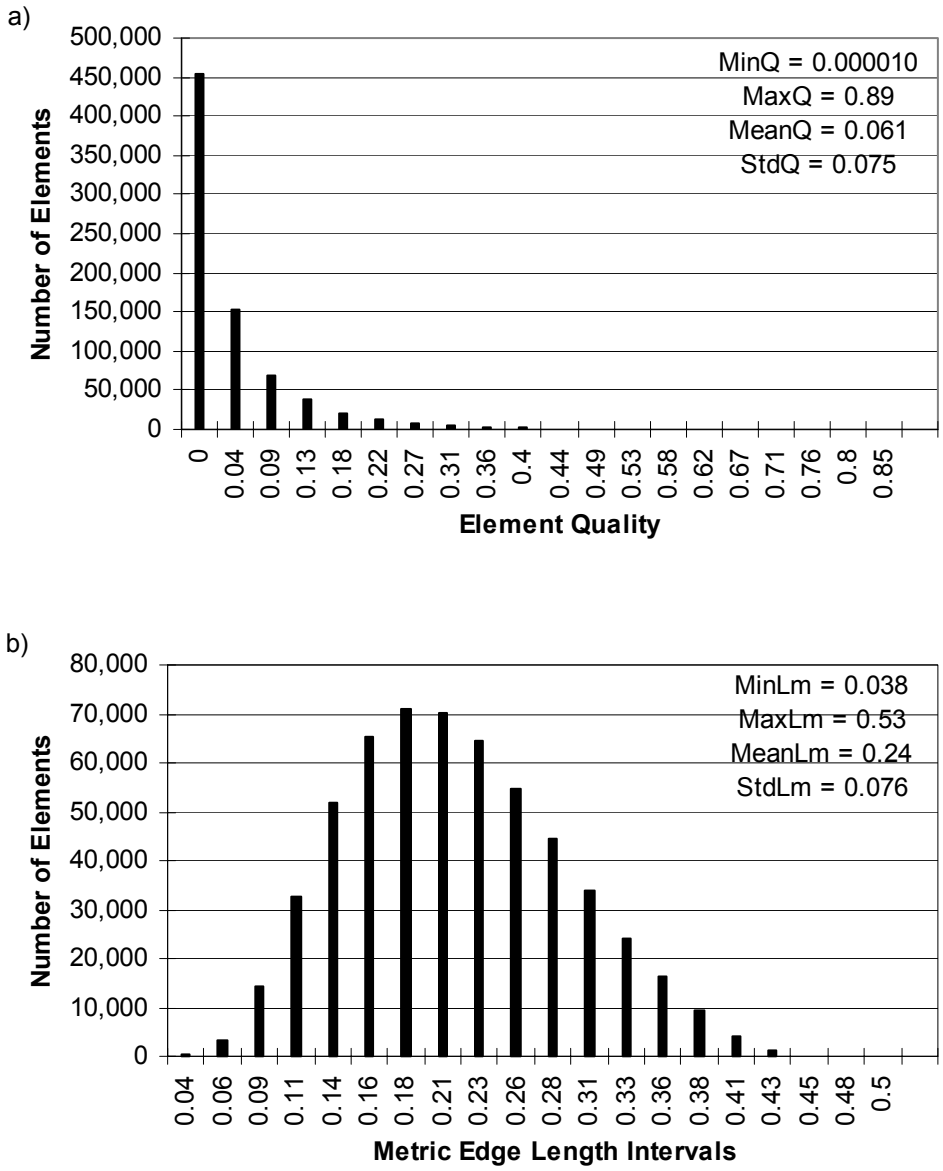

Figure 8.29: Unsteady 3-D heat transfer. a) Histogram of element quality; b) histogram of metric edge length.

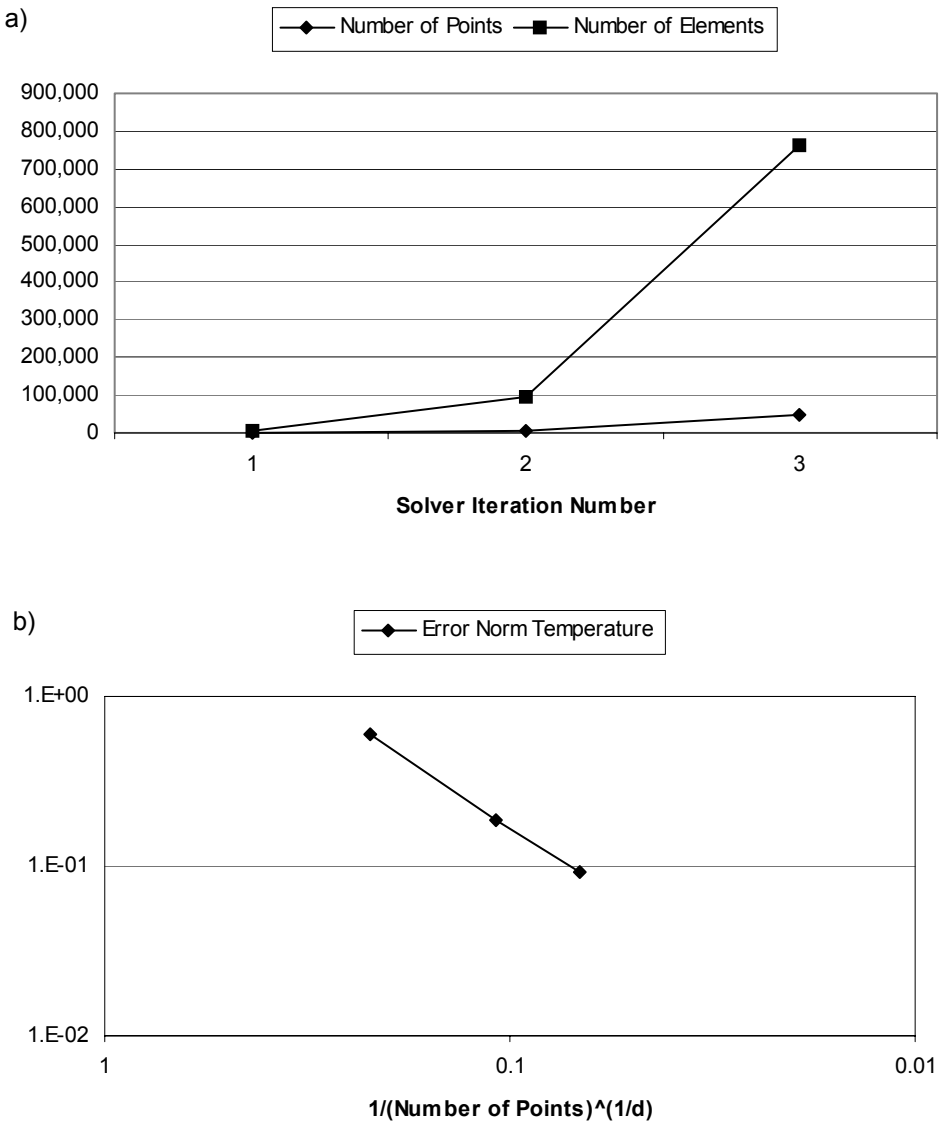

Figure 8.30: Unsteady 3-D heat transfer. a) Numbers of mesh points and elements; b) L^2 error norm as a function of $1/N_p^{1/d}$.

8.4 Unsteady Flow in a Cavity with a Manufactured Solution

The first incompressible Navier-Stokes test case is designed to verify that the solver works properly for steady flows and for unsteady flows with a fully coupled space-time formulation. Following the method of manufactured solutions, an analytical solution is chosen based on the work of Watanabe et al. (1999), but with an added term that is a function of time, as

$$\begin{aligned} C(t) &= \sin(\pi t) \\ u &= C(t)\sin(\pi x)^2 \sin(\pi y)\cos(\pi y) \\ v &= -C(t)\sin(\pi y)^2 \sin(\pi x)\cos(\pi x) \\ p &= -\frac{1}{16}C^2(t)\cos(2\pi x)\cos(2\pi y) \end{aligned} \tag{8.6}$$

Because this expression satisfies the continuity equation, only source terms corresponding to the momentum equations need to be computed by inserting the manufactured solution into the incompressible Navier-Stokes equations. The problem definition is completed by specifying a Dirichlet boundary condition using the analytical solution specified by 8.6 over the entire boundary of the space-time domain and a Reynolds number of 100.

A convergence analysis is first performed for the 2-D steady case and then for the 2-D unsteady case using isotropic meshes in a unit domain with the number of intervals N_I along each axis taken as 10, 20, 30 and 40, respectively. The L^2 norms of the error are shown for u, v and p as functions of the characteristic mesh size $h = 1/N_I$ in Figure 8.31 and in Table 8.6. In both the steady and the unsteady 2-D cases, the numerical solutions are in excellent agreement with the manufactured analytical solution.

d	**SolverIt**	$\|u-u_h\|_0$	$\|v-v_h\|_0$	$\|p-p_h\|_0$	$h=1/N_I$
2-D	1	0.0301622	0.0176131	0.0142572	1/10
	2	0.00664341	0.00444518	0.00299479	1/20
	3	0.00270476	0.00196802	0.0011713	1/30
	4	0.00145283	0.00111379	0.000620702	1/40
2-D+Time	1	0.0105763	0.0136233	0.00505374	1/10
	2	0.00261793	0.00310698	0.00109538	1/20
	3	0.00117717	0.0013415	0.000474425	1/30
	4	0.000668394	0.000745813	0.000266375	1/40

Table 8.6: Flow in Cavity. Error norms for the convergence analysis.

Next, the verification proceeds with the same 2-D steady flow, but in combination with the anisotropic mesh adaptation procedure rather than with isotropic meshes that are progressively subdivided. The initial 2-D mesh used had 10 intervals along each axis, which gives 121 mesh points and 200 triangular elements. The error reduction factors used for the mesh adaptation procedure were 0.5, 0.8, and 1.0 for the 3 adaptive cycles, for a total of 4 solver executions. The initial and adapted meshes are shown in Figure 8.32. The isocontours of u, v and p along with the velocity vector field are shown in Figure 8.33. The streamlines for the adapted mesh and a detail of the streamlines are shown in Figure 8.34. Histograms for the element quality and the metric edge lengths are shown in Figure 8.35. The numbers of mesh points and elements have been plotted vs. the solver iteration number in Figure 8.36. The L^2 norms of the errors for u, v and p, shown in the same Figure, indicate good agreement with the analytical solution.

One is reminded that, in the case of the non-linear incompressible Navier-Stokes, a Picard iteration method has been used. In Figure 8.37, the convergence criterion of equation 5.30, which is the relative norm of the difference between the solution vector at two consecutive Picard iterations, is plotted vs. the cumulative number of Picard iterations

for all the solver and adaptation cycles. The Picard method converges to a value below 10^{-4} in about 12 iterations for each of the solver runs. It can be seen that, after each mesh adaptation cycle, the relative norm between two consecutive solutions in the Picard method rises abruptly. Recall that, after each mesh adaptation execution, the previous solution, also used to compute the error estimate driving the mesh optimization procedure, is interpolated on the adapted mesh. The interpolation procedure, previously described in Chapter 7, only uses linear finite element interpolation functions, so that it is not guaranteed to satisfy the equation for the mass conservation. In spite of this, the Picard method exhibits a rapid convergence after each of these abrupt increases. Notice also that the peaks in the relative norm corresponding to the first Picard iteration after each mesh adaptation cycle get progressively reduced at each solver executions. This indicates that, for this specific case, which corresponds to the relative low Reynolds number of 100, the Picard methods converges rapidly. Considering that the Picard method is known to suffer from oscillations for flows at high Reynolds numbers, this rapid convergence cannot be expected to apply to problems at high Reynolds numbers. Even so, the present results indicate that the chosen method is sufficient for the purposes of the current investigation.

The same verification is performed using a 3-D mesh for an unsteady solution on a space-time domain with a fully coupled space-time finite element formulation. In this case, the flow problem is still 2-D, but the space-time domain now includes the time axis, so that the mesh used is 3-D. This initial 3-D mesh uses 10 intervals along each axis, leading to 1,331 mesh points and 6,000 tetrahedral elements. The error reduction factors used for the 3 mesh adaptation cycles were also 0.5, 0.8 and 1.0. The initial and adapted meshes are

d	MinQ	MaxQ	AvgQ	StdQ	NumPoints	NumElems
2-D	0.15	1.0	0.75	0.18	812	1,524
3-D	0.000033	0.99	0.28	0.20	80,493	444,192

Table 8.7: Flow in Cavity. Element quality and numbers of mesh points and elements for the mesh optimization procedure.

shown in Figure 8.38, where the time axis corresponds to the vertical axis in this Figure. The isocontours for velocity components u and v are shown at the time $t = 0.5$ and for $x = 0.5$ in Figure 8.39 and for the pressure in Figure 8.40. Two details of the streamlines with the isocontours of the pressure at $t = 0.5$ are also shown in Figure 8.41. Histograms for the element quality and the metric edge lengths are shown in Figure 8.42. The numbers of mesh points and elements along with the L^2 norms for u, v and p are shown in Figure 8.43. The convergence criterion for the Picard method is shown in Figure 8.44.

The statistics for the element quality on the final adapted mesh, along with the numbers of mesh points and elements, are summarized in Table 8.7. The minimum quality in 2-D is 0.15, which may be considered as acceptable. In contrast, the minimum quality in the 3-D case is relatively low, at 0.000033, and the average quality is only 0.28.

The L^2 error norms for the velocity components and pressure for the 2-D and 3-D cases are summarized in Table 8.8. The 2-D case approaches excellent agreement with the analytical solution, as the mesh is refined. The 3-D case shows good agreement, but seems to oscillate. It is suspected that this might, at least in part, be attributed to the use of $1/N_p^{1/d}$ as the characteristic mesh size when the mesh is actually anisotropic and its size is not uniform. Another possibility is that the GLS formulation is over-stabilized, as the stabilization factors use a characteristic size that was computed as function of the volume through equation 5.27. Mittal (2000) recommend to use the minimum edge length

d	**SolverIt**	$\|u-u_h\|_0$	$\|v-v_h\|_0$	$\|p-p_h\|_0$	$1/N_p^{1/d}$
2-D	1	0.0302	0.0176	0.0143	0.0909
	2	0.0319	0.0197	0.0143	0.0469
	3	0.00622	0.00663	0.00279	0.0370
	4	0.00358	0.00361	0.00123	0.0351
3-D	1	0.0106	0.0136	0.00505	0.0909
	2	0.0114	0.0143	0.00510	0.0423
	3	0.00272	0.00254	0.000849	0.0287
	4	0.00784	0.00702	0.00127	0.0232

Table 8.8: Flow in Cavity. Norms of the errors.

for elements with high aspect ratio. However, for the present numerical experiments, using the minimum edge length seemed to understabilize the procedure (it was observed that, for a highly stretched mesh in simple Poiseuille flow, oscillations in the pressure field appeared). Further investigations are needed on this subject for highly anisotropic meshes.

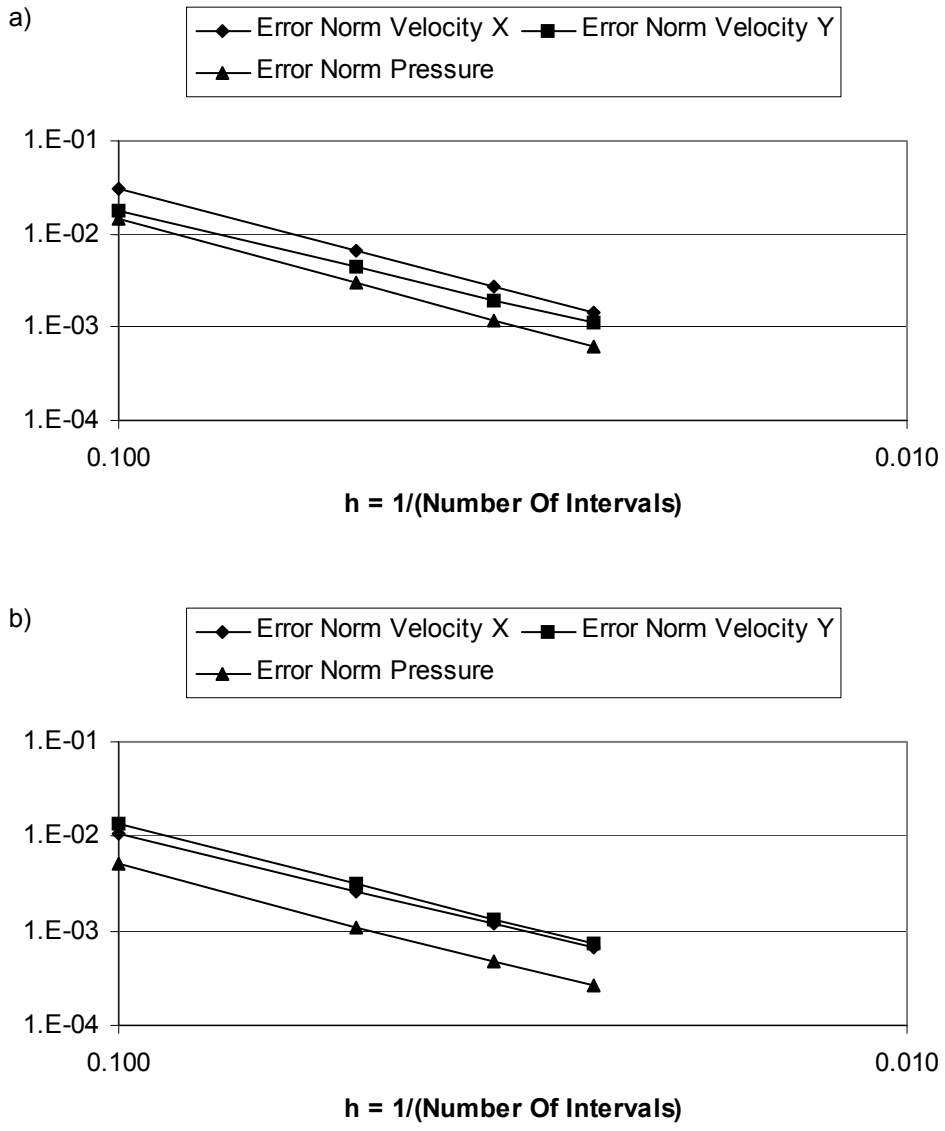

Figure 8.31: Steady and unsteady 2-D flow in a cavity. L^2 error norm as a function of the characteristic mesh size for a) the steady case and b) the unsteady case.

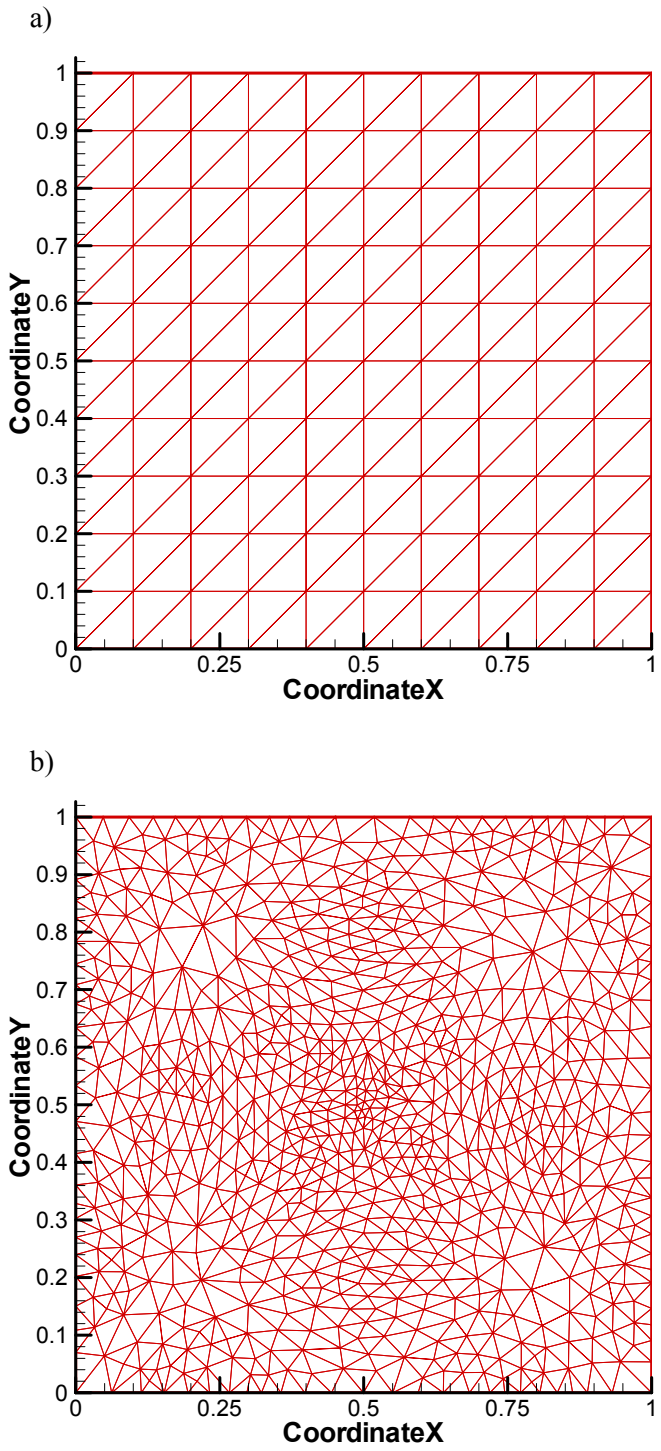

Figure 8.32: Steady 2-D flow in a cavity. a) Initial mesh; b) adapted mesh.

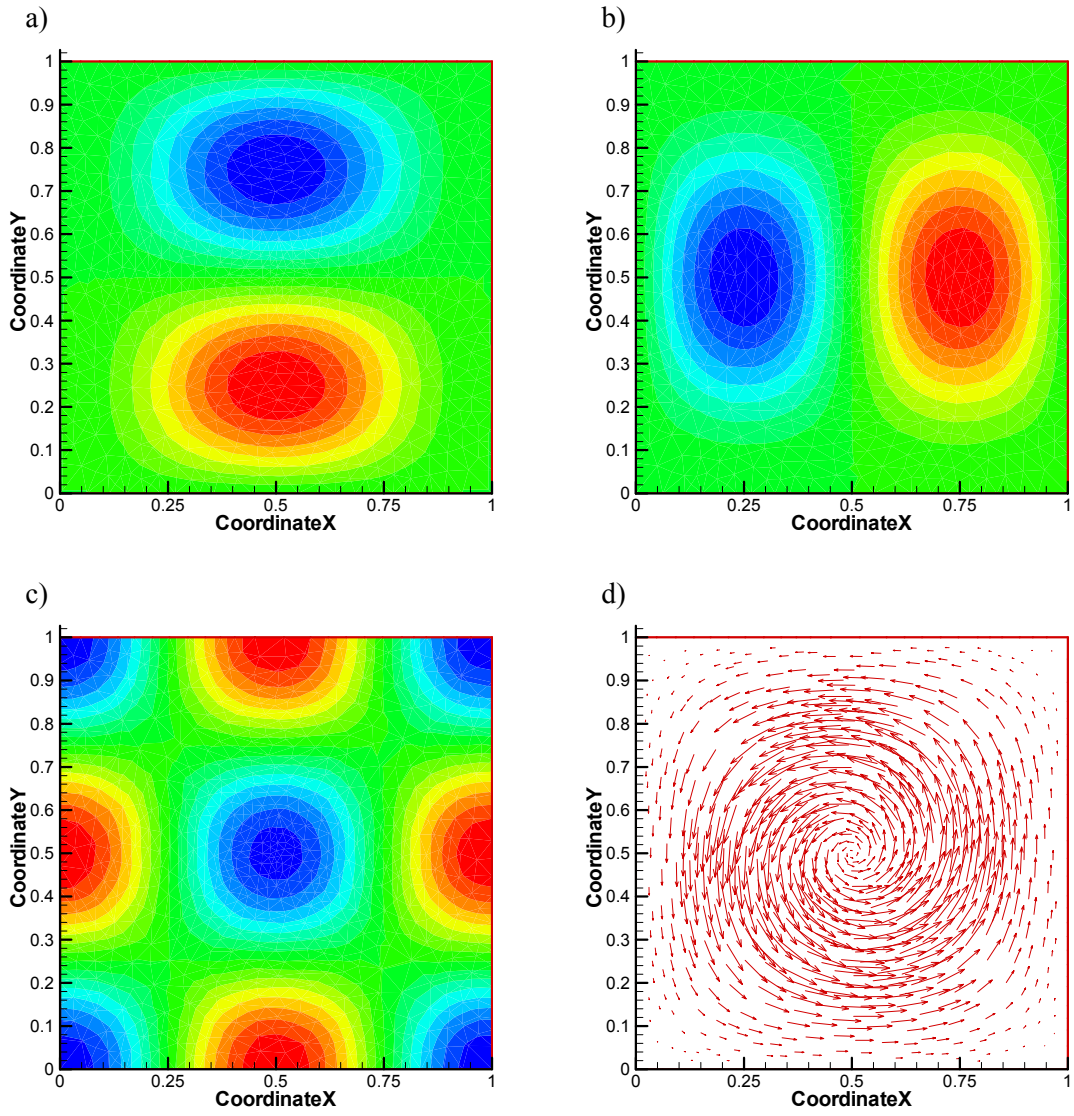

Figure 8.33: Steady 2-D flow in a cavity. a) u; b) v; c) pressure; d) velocity vector field.

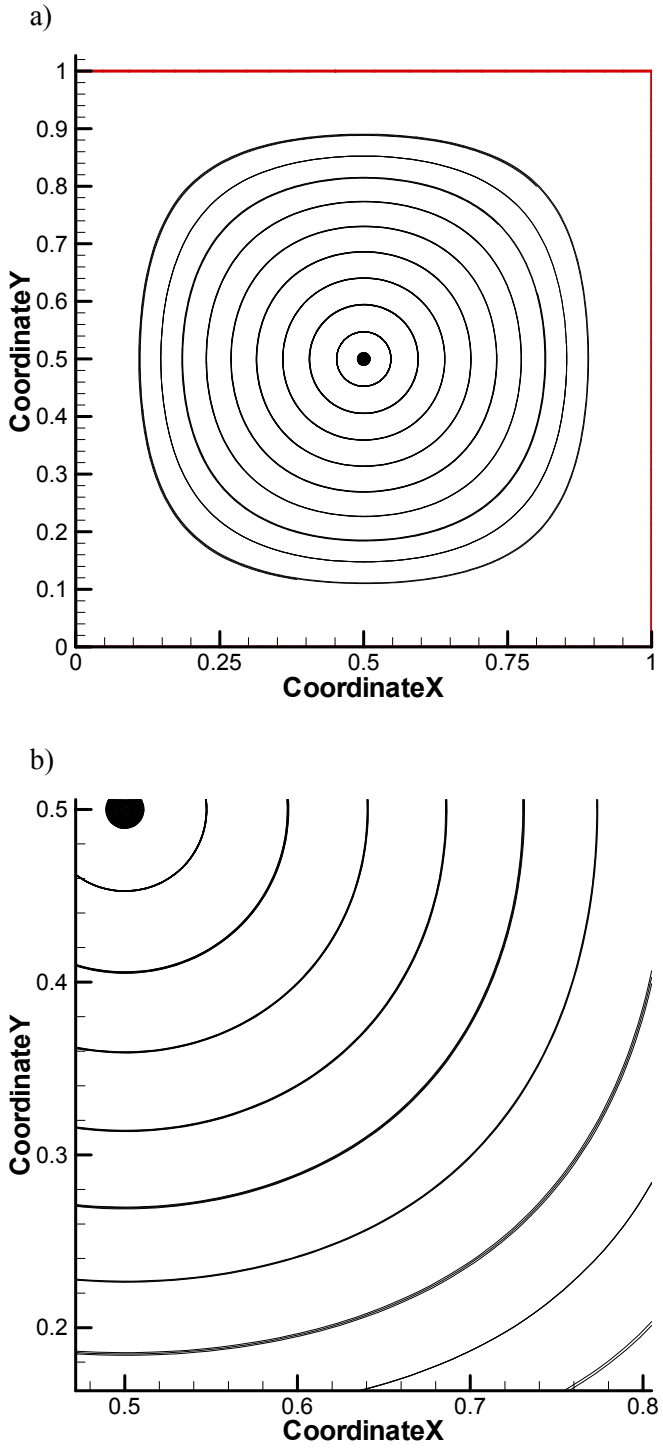

Figure 8.34: Steady 2-D flow in a cavity. Streamlines a) general pattern; b) detail.

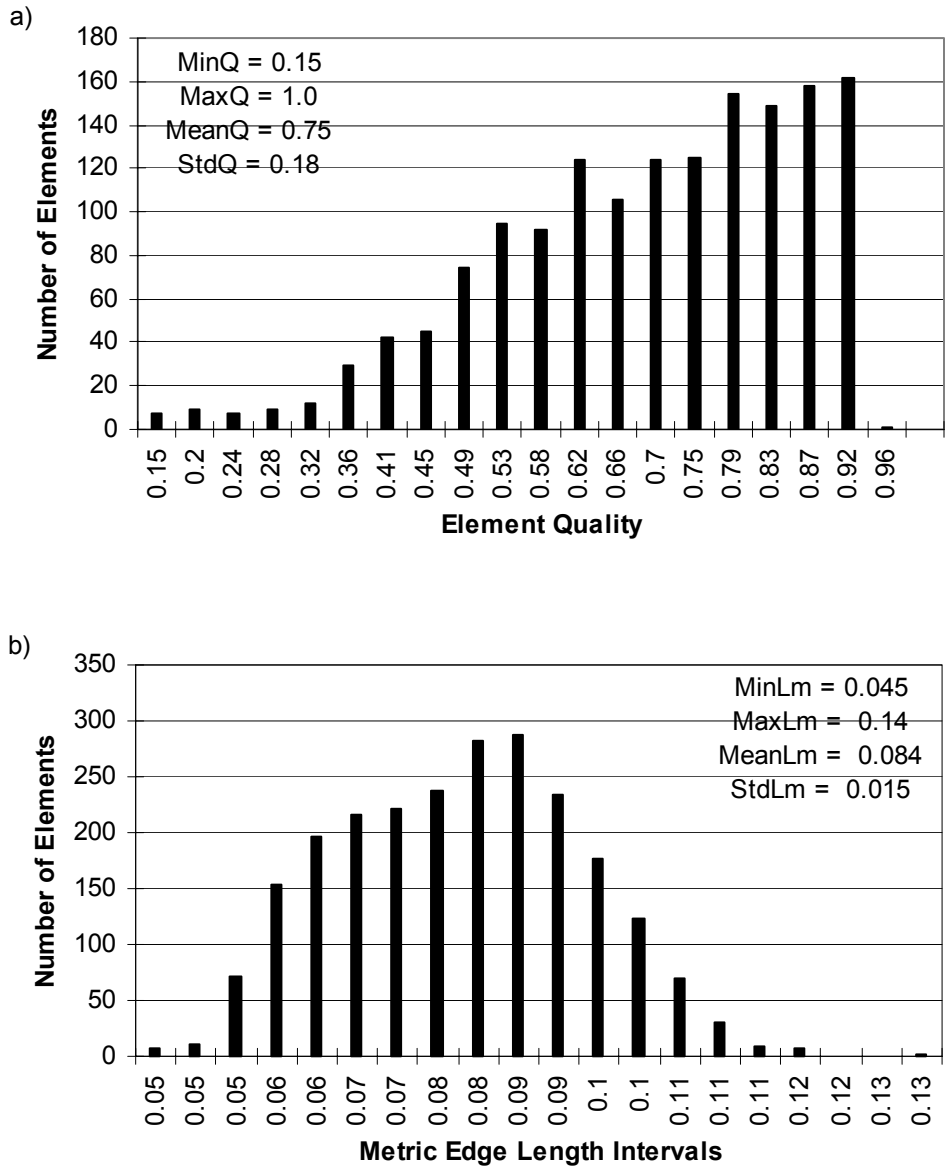

Figure 8.35: Steady 2-D flow in a cavity. a) Histogram of element quality; b) histogram of metric edge length.

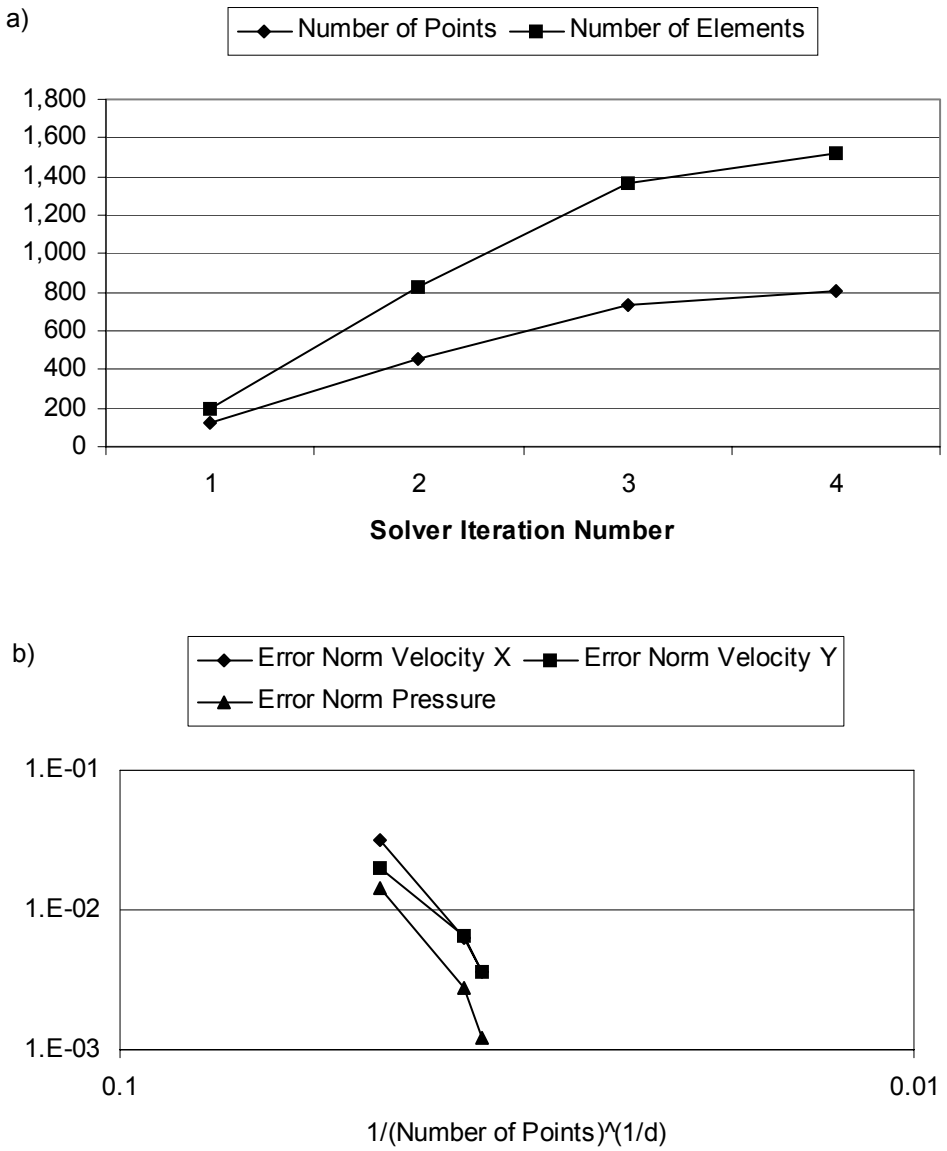

Figure 8.36: Steady 2-D flow in a cavity. a) Numbers of mesh points and elements; b) L^2 error norm as a function of $1/N_p^{1/d}$.

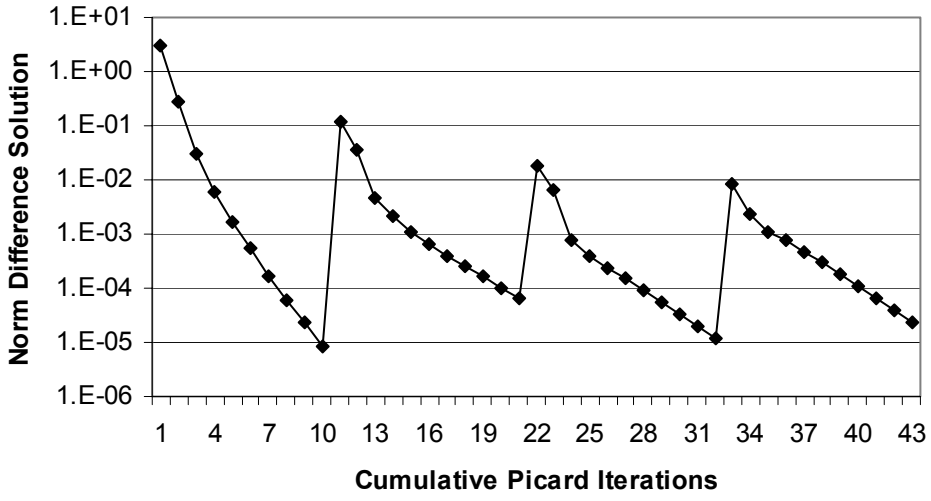

Figure 8.37: Steady 2-D flow in a cavity. Relative norm of the difference between two consecutive solutions as a function of the cumulative number of Picard iterations.

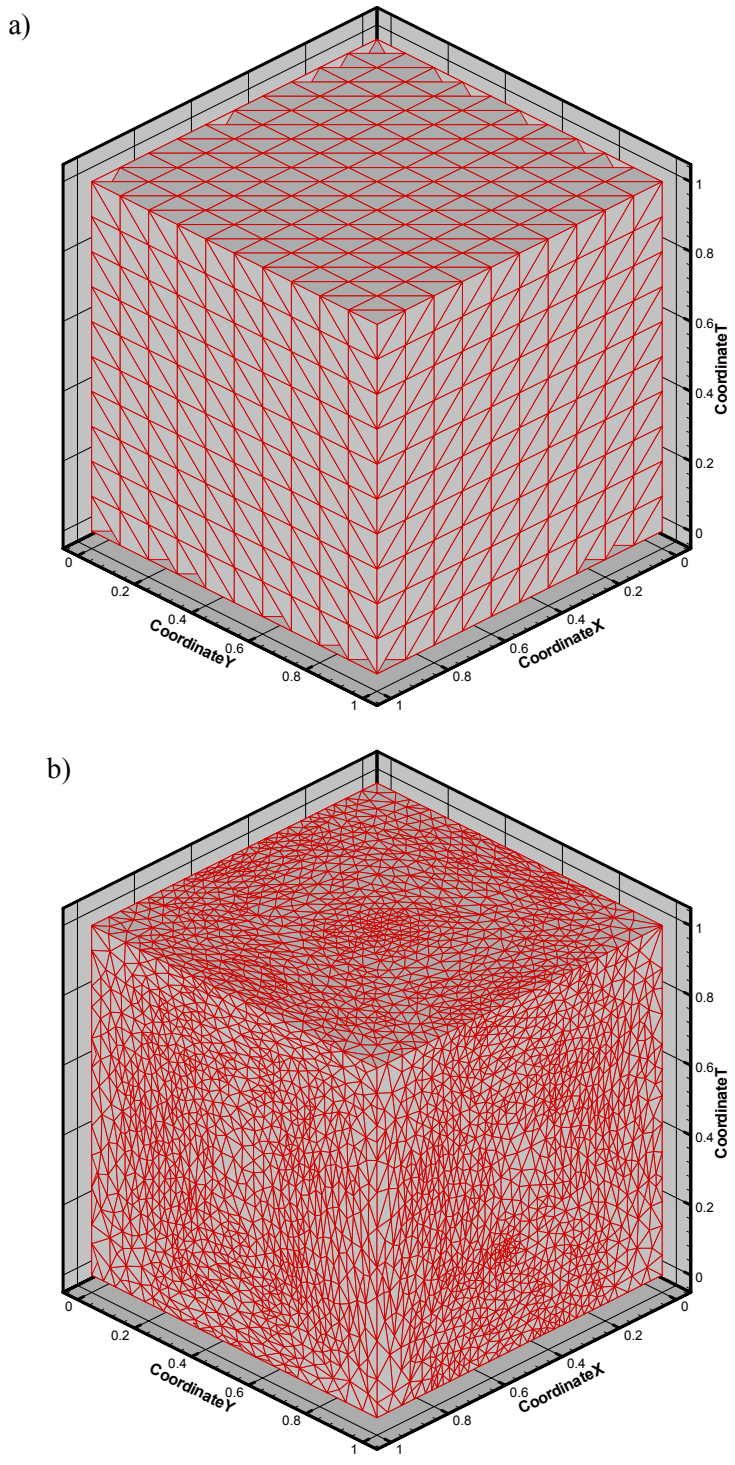

Figure 8.38: Unsteady 2-D flow in a cavity. a) Initial mesh; b) adapted mesh.

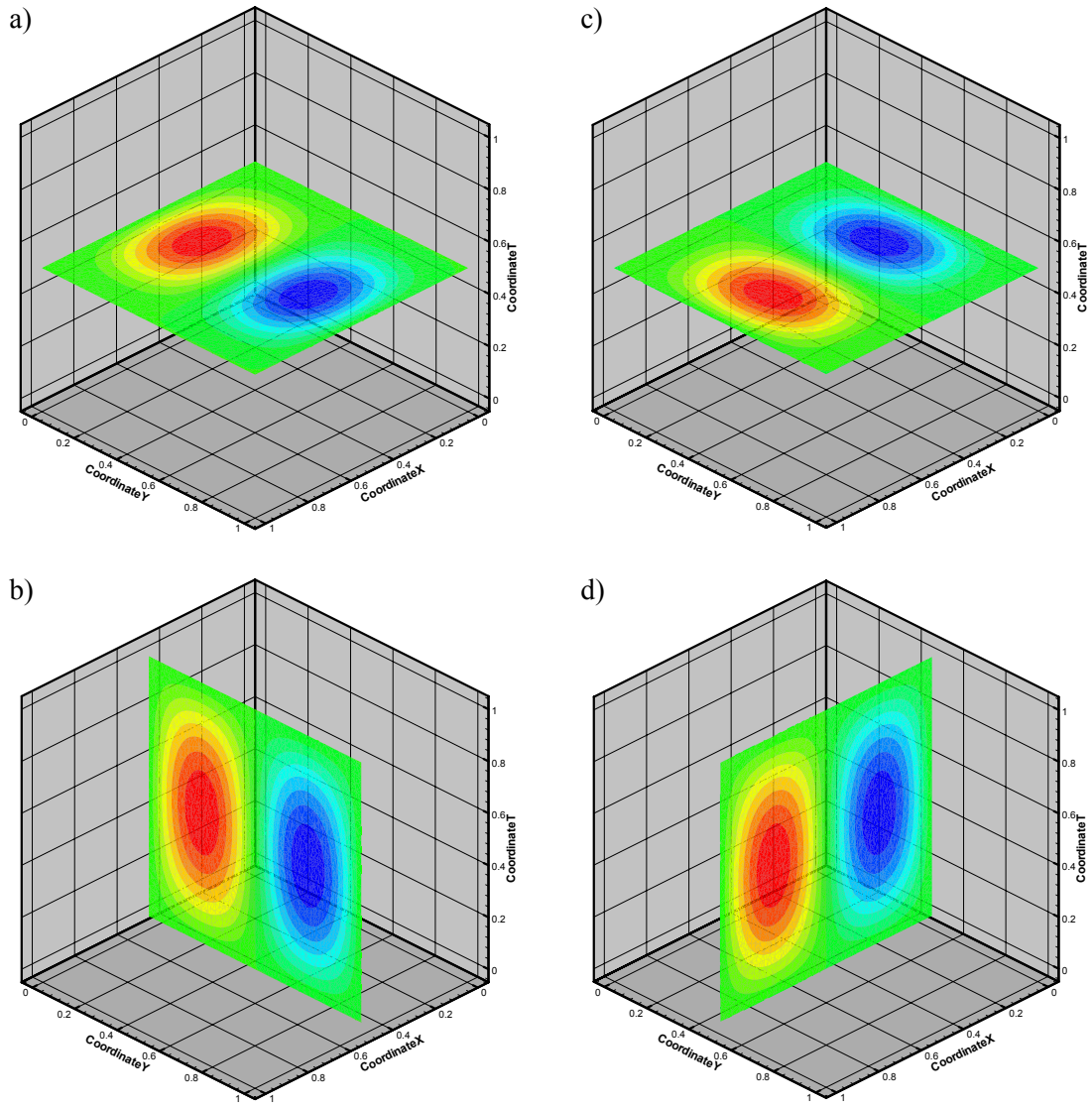

Figure 8.39: Unsteady 2-D flow in a cavity. Isovelocity contours a) u at $t = 0.5$; b) u at $x = 0.5$; c) v at $t = 0.5$; d) v at $y = 0.5$.

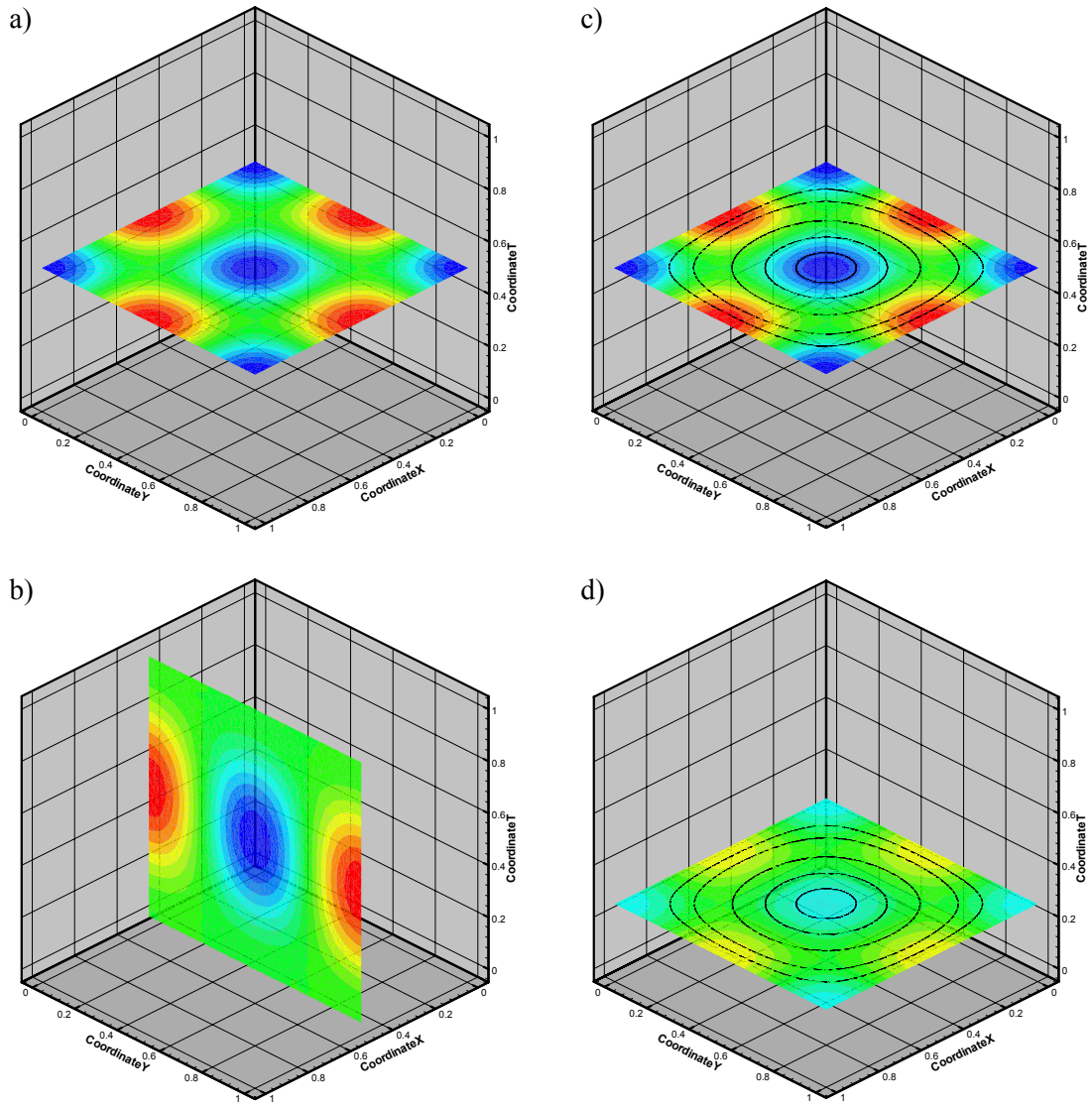

Figure 8.40: Unsteady 2-D flow in a cavity. a) Pressute at $t = 0.5$; b) pressure at $x = 0.5$; c) streamlines and pressure at $t = 0.5$; d) streamlines and pressure at $t = 0.25$.

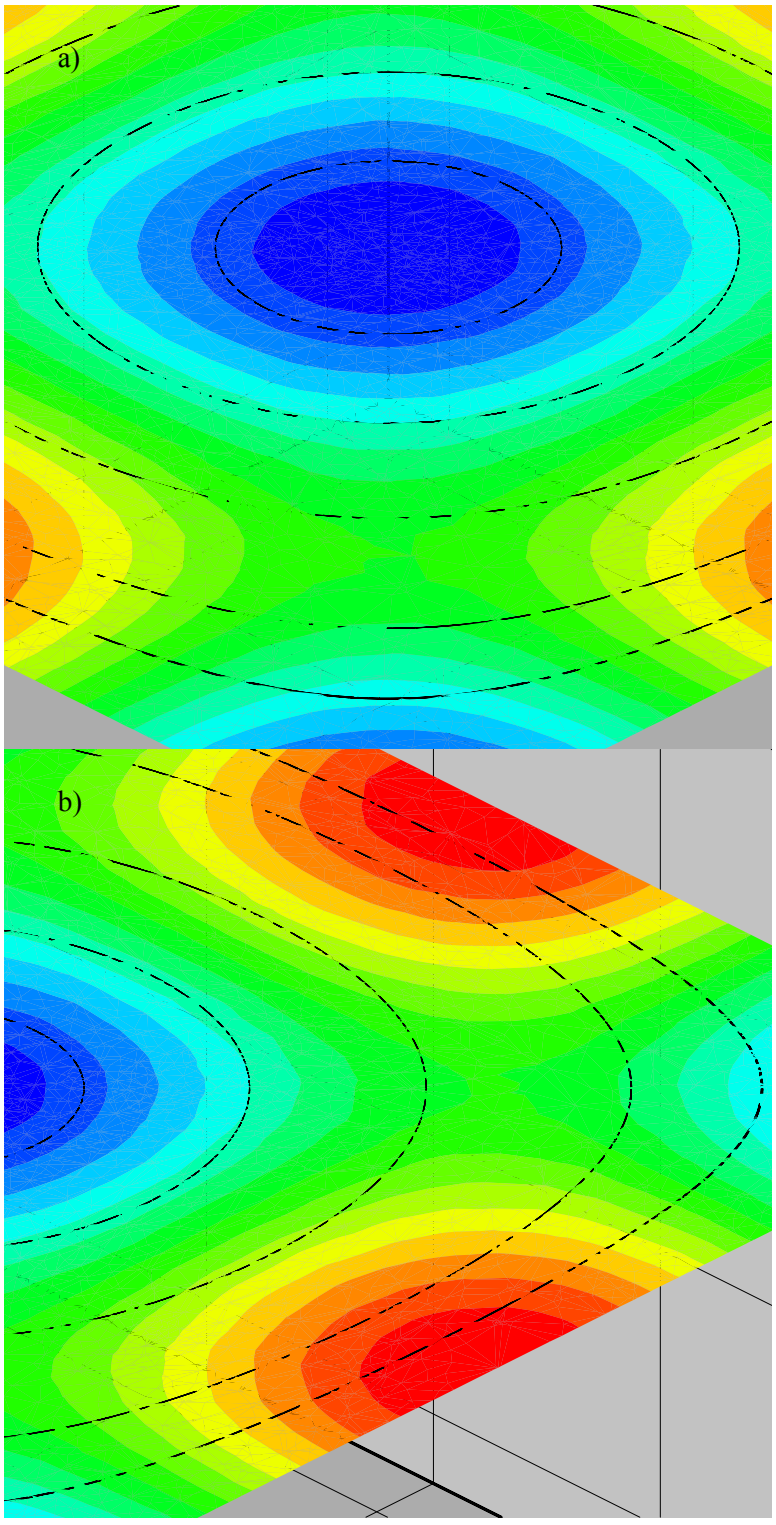

Figure 8.41: Unsteady 2-D flow in a cavity. Streamlines and pressure at $t = 0.5$. a) General view; b) detail near point (0.5,0.5,0.5).

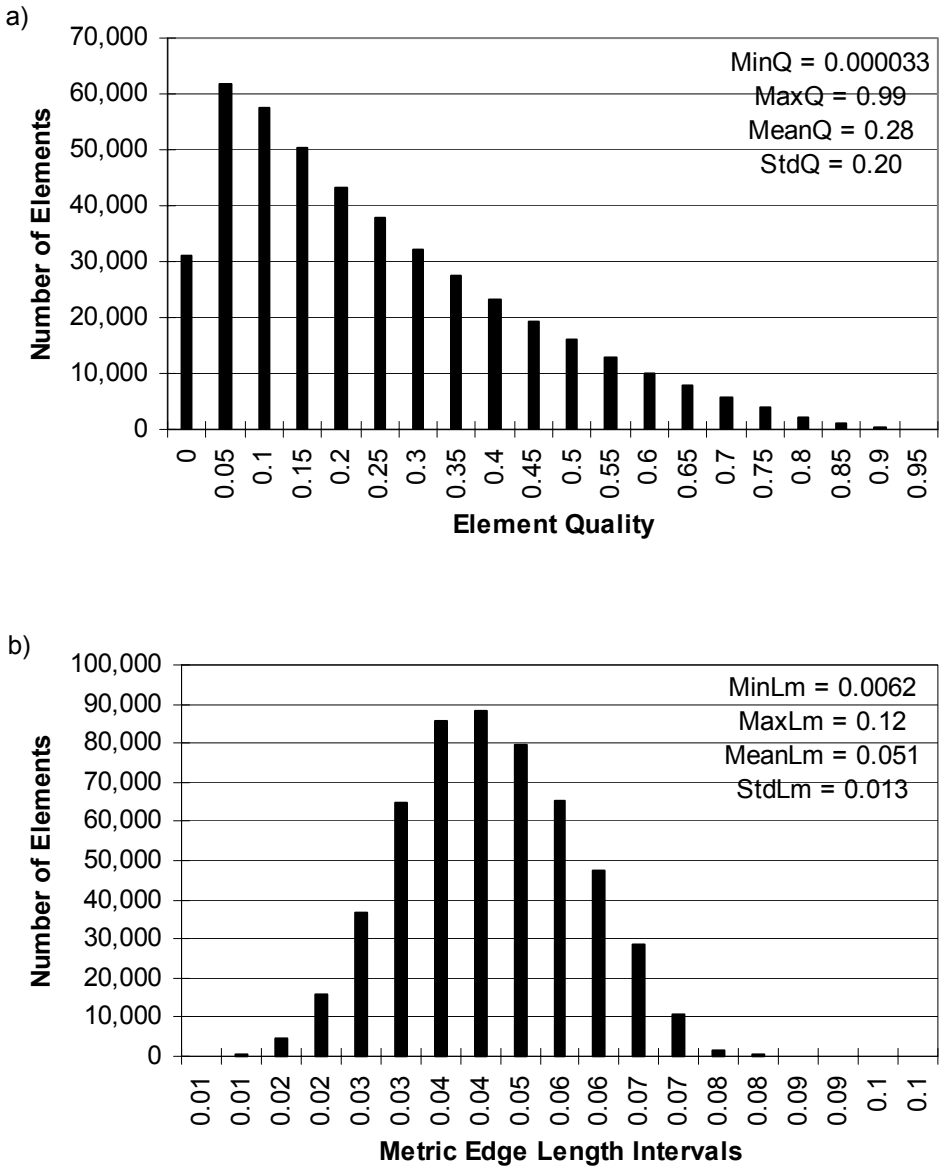

Figure 8.42: Unsteady 2-D flow in a cavity. a) Histogram of element quality; b) histogram of metric edge length.

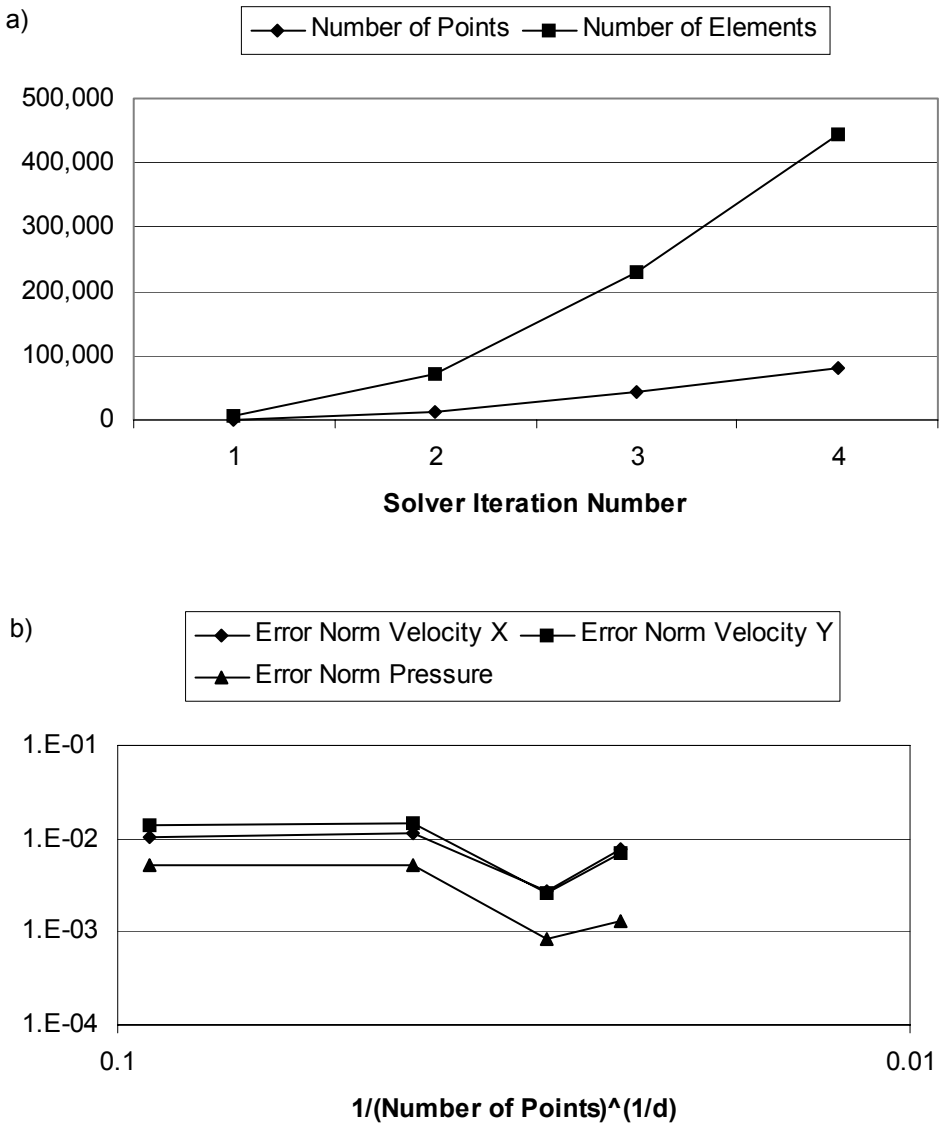

Figure 8.43: Unsteady 2-D flow in a cavity. a) Number of mesh points and elements; b) L^2 error norm as a function of $1/N_p^{1/d}$.

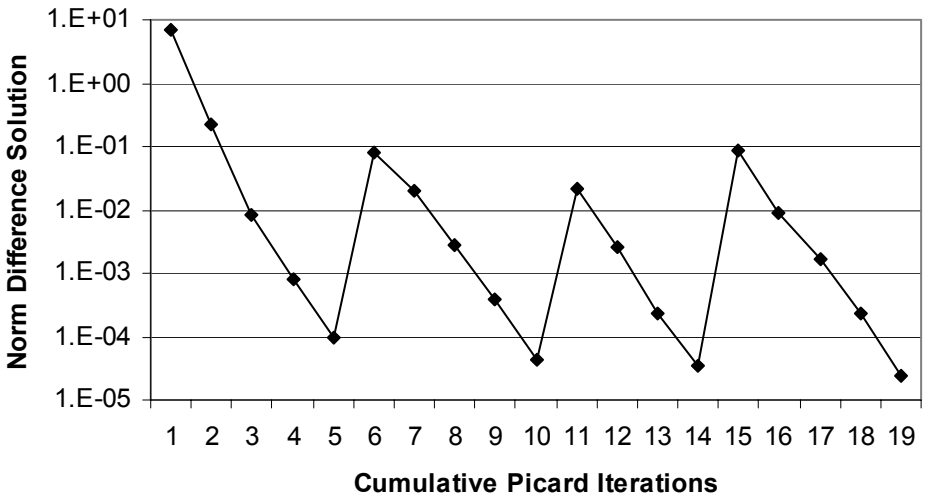

Figure 8.44: Unsteady 2-D flow in a cavity. Relative norm of the difference between two consecutive solutions as a function of the cumulative number of Picard iterations.

8.5 Unsteady Flow Past a Circular Cylinder

The incompressible flow past a cylinder at a Reynolds number of 100 has been chosen to evaluate the performance of the space-time adaptive method on a classical benchmark problem.

In this test case, a cylinder of unit radius centred at the spatial origin is enclosed into the rectangular domain with $-4 \leq x \leq 25.2$ and $-4 \leq y \leq 4$. This 2-D domain is extruded along the time axis for $0 \leq t \leq 10$ to form a 3-D space-time mesh.

The boundary conditions on the planes $x = -4$, $y = -4$ and $y = 4$ are

$$\begin{aligned} u &= \tanh\left(\frac{t}{\tau}\right) \\ v &= 0 \end{aligned} \tag{8.7}$$

with $\tau = 1$ such that, at $t = 0$, $u = v = 0$. The initial condition on the velocity field was chosen to match these boundary conditions with a value of zero at $t = 0$ and to satisfy the incompressibility condition. As explained in Chapter 5, this corresponds to accelerating the fluid from rest at $t = 0$ by rapidly increasing the value of the velocity specified by the Dirichlet boundary condition. Because a fully coupled space-time formulation is used, the solution on the entire space-time domain is computed for each Picard iteration of the non-linear flow solver. The exit boundary condition at the plane $x = 25.2$ is a Neumann boundary condition with a zero value, which corresponds to a zero stress condition, as a stress divergence form of the viscous term was used (see Chapter 5). Also, a no-slip boundary condition is used at the surface of the cylinder.

The initial mesh used, shown in Figure 8.45, has 2,343 mesh points and 10,980 tetrahedral elements. The geometry for that test case, as for all the others except the

ones with 4-D meshes, was constructed from the simplicial surface mesh with the geometry reconstruction algorithm presented in Chapter 4.

Three cycles of mesh adaptation were performed using error reduction factors of 0.5, 0.8 and 1.0, which corresponds to 4 solver executions. The final adapted mesh, shown in Figure 8.46, has 123,313 mesh points and 645,158 tetrahedral elements. A detail of the adapted mesh near the cylinder at $t = 10$ is shown in Figure 8.47.

Isocontours of the velocity components u and v are shown in Figure 8.48. Figure 8.49 shows details of the velocity vector field at $t = 10$ near the cylinder. Figure 8.50 shows the pressure field at $t = 0.2$ and isocontours of u, together with velocity vectors, $t = 9.9$.

Histograms for the element quality and the metric edge length are shown in Figure 8.51, whereas Figure 8.59 shows the numbers of mesh points and elements as functions of the solver iteration number and the convergence history of the Picard method.

The percentage of total computational time, comprising both the flow solver and the mesh adaptation, spent for the mesh adaptation alone was 69%, with the remainder of 31% spent for the flow solver (excluding the CGNS file reading and writing). Although the mesh adaptation time remains the largest component of the total computation time, it is significantly less than that for the heat equation solver, for which about 98% of the time was spent for the mesh adaptation. It would be interesting to consider a non-adapted isotropic mesh of density sufficiently high to have a comparable flow field resolution as the adapted mesh shown here and compare the total time necessary to compute a similar solution with the Picard method. Although this is left as future work, it is still important to consider that mesh adaptation allows for the initial mesh to be coarser than without it and

to progressively refine the mesh during the cumulative Picard iterations needed. This is a beneficial aspect, because the initial Picard iterations are done in a mesh with fewer points than without mesh adaptation and would thus converge faster. In the present low Reynolds number test case, the adaptive space-time solution procedure was found to be very robust, although it remains to be seen how this approach would perform at much higher Reynolds numbers.

The present computation was performed over the relatively short period of 10 non-dimensional time units. It would be interesting to pursue the investigation further to examine the behaviour of the solution over a very long period of time, but due to memory constraints of the available computer this is left for future work. Because of the use of a GMRES method, as mentioned in Chapter 5, a global matrix needs to be assembled for the entire space-time domain with the present fully coupled space-time approach. An element-by-element solution procedure, such as the one used by Pontaza and Reddy (2004), would alleviate this problem, provided that only a portion of the space-time mesh is read from file and used to compute the solution. Even without this improvement, a qualitative assessment of the solution for the time period investigated demonstrates that the flow solution is developing in a manner compatible with expectations for this benchmark problem.

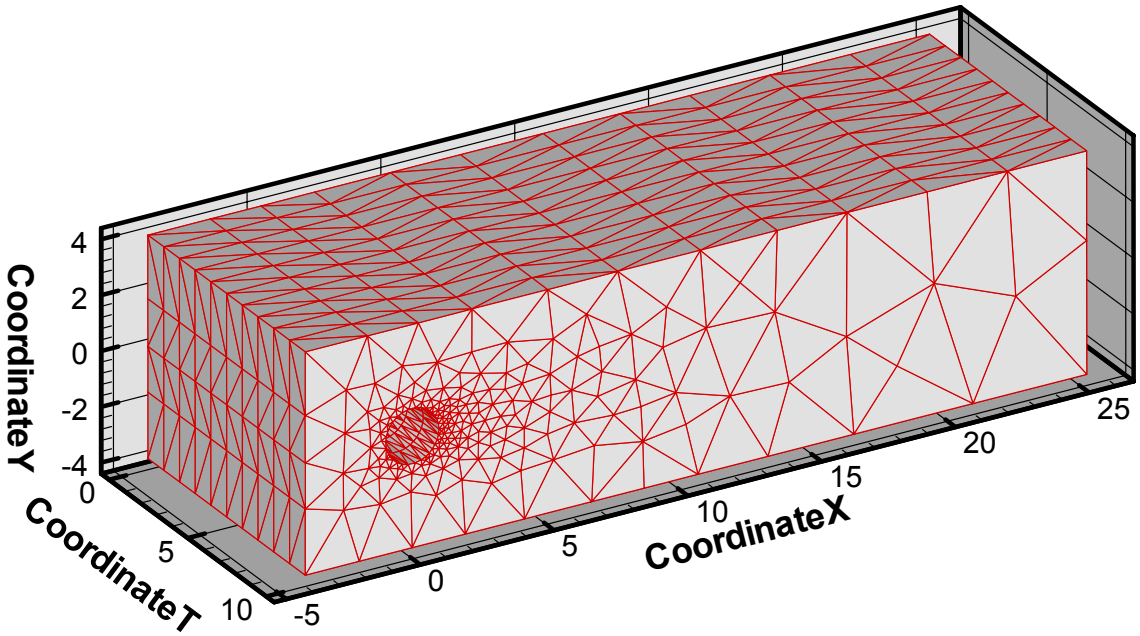

Figure 8.45: Unsteady 2-D flow past a circular cylinder. Initial mesh.

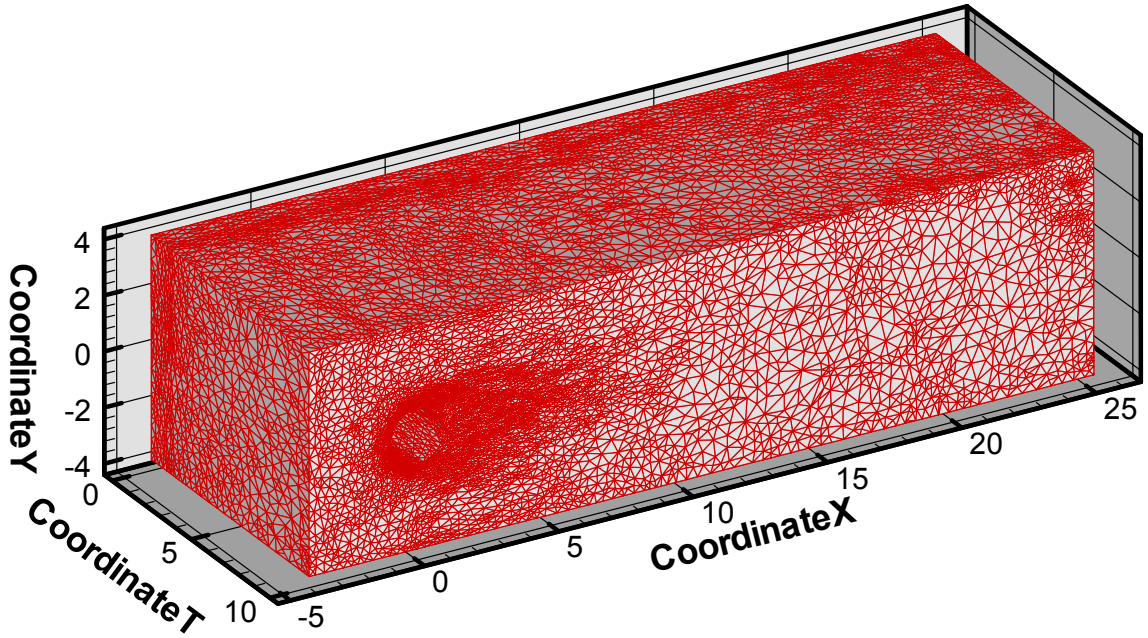

Figure 8.46: Unsteady 2-D flow past a circular cylinder. Adapted mesh.

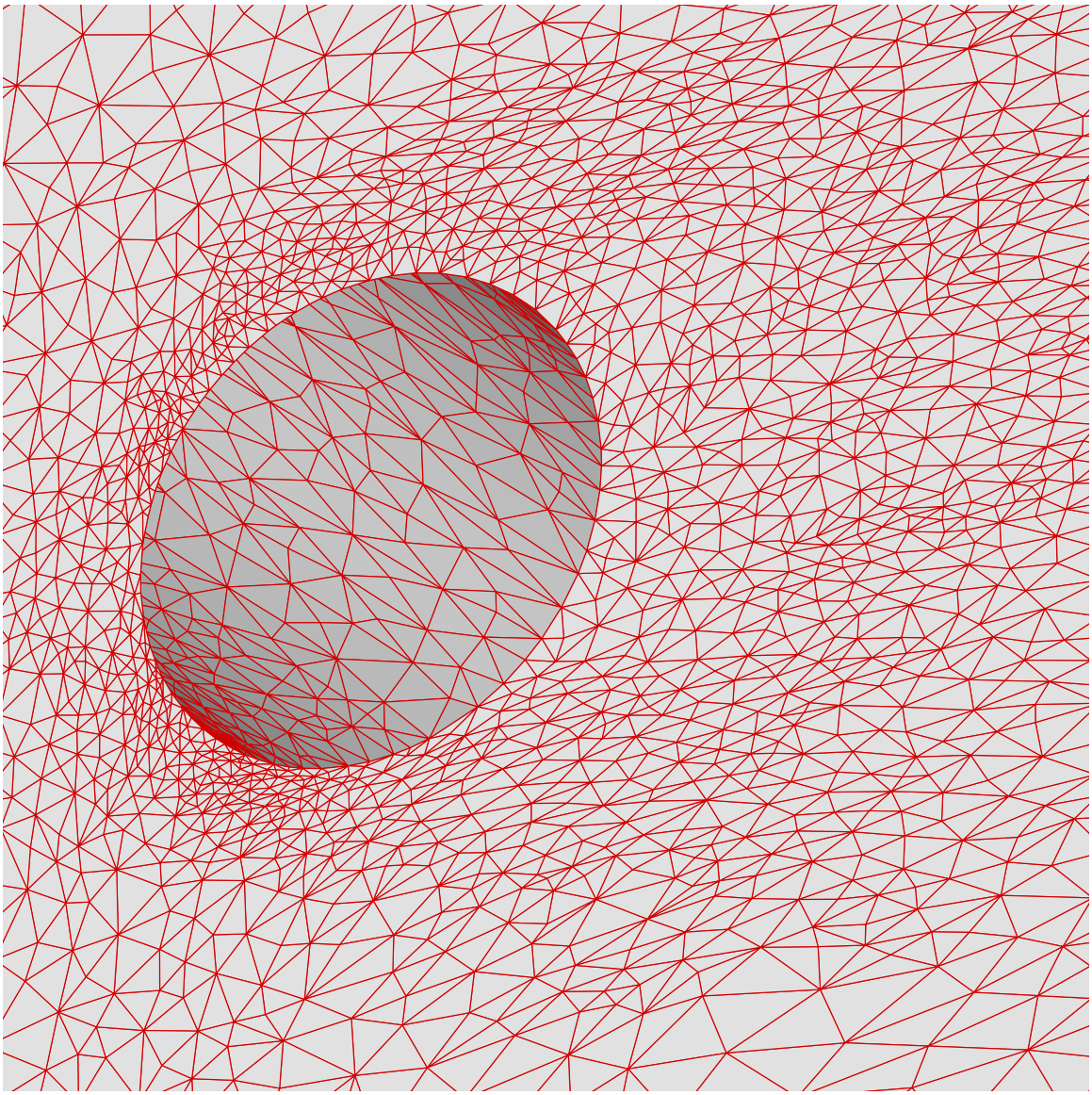

Figure 8.47: Unsteady 2-D flow past a circular cylinder. Detail of the adapted mesh near the cylinder at $t = 10$.

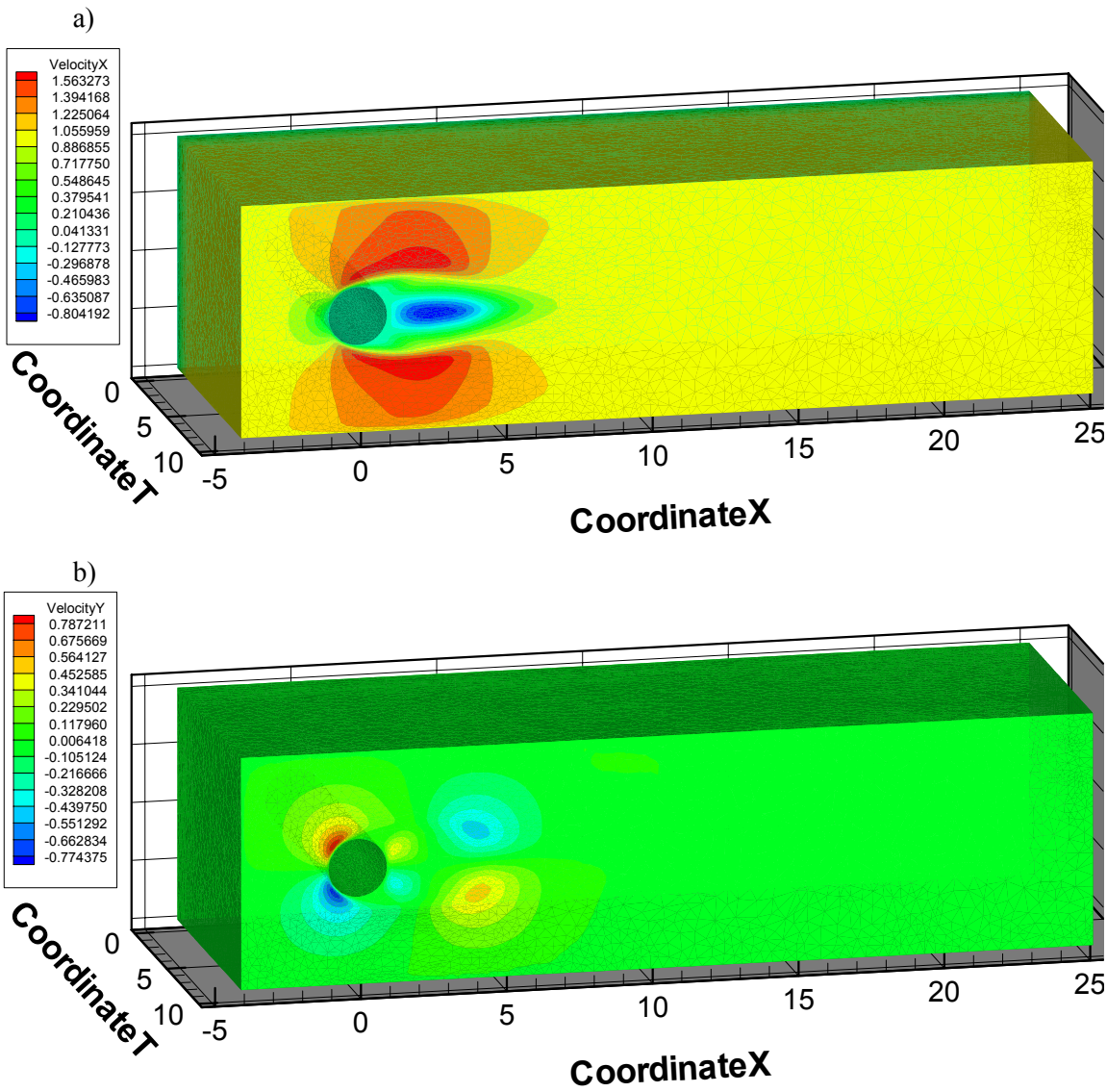

Figure 8.48: Unsteady 2-D flow past a circular cylinder. a) Scalar field for u velocity at $t = 10$; b) scalar field for the v velocity at $t = 10$.

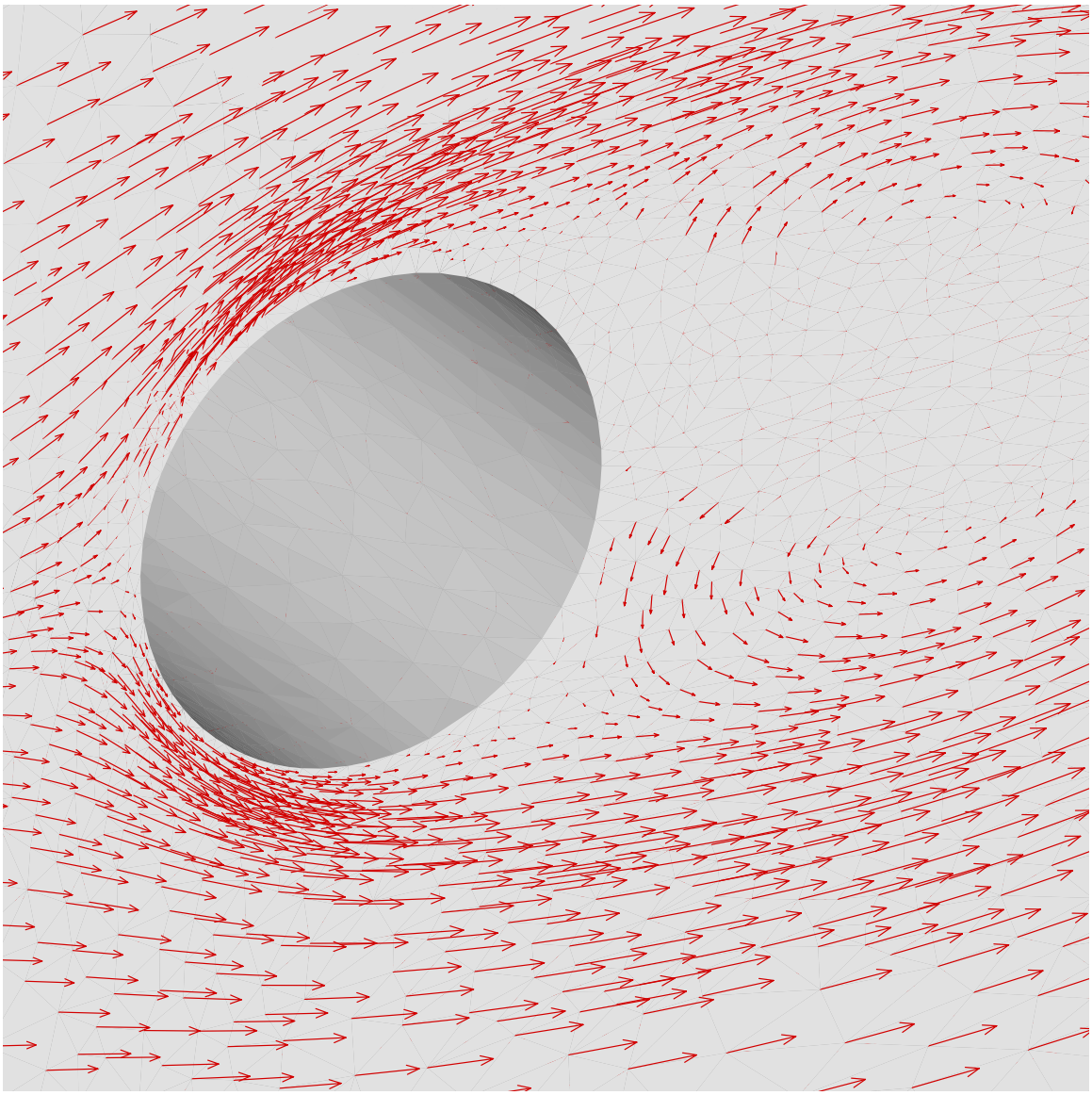

Figure 8.49: Unsteady 2-D flow past a circular cylinder. Detail of the velocity vector field neat the cylinder at $t = 10$.

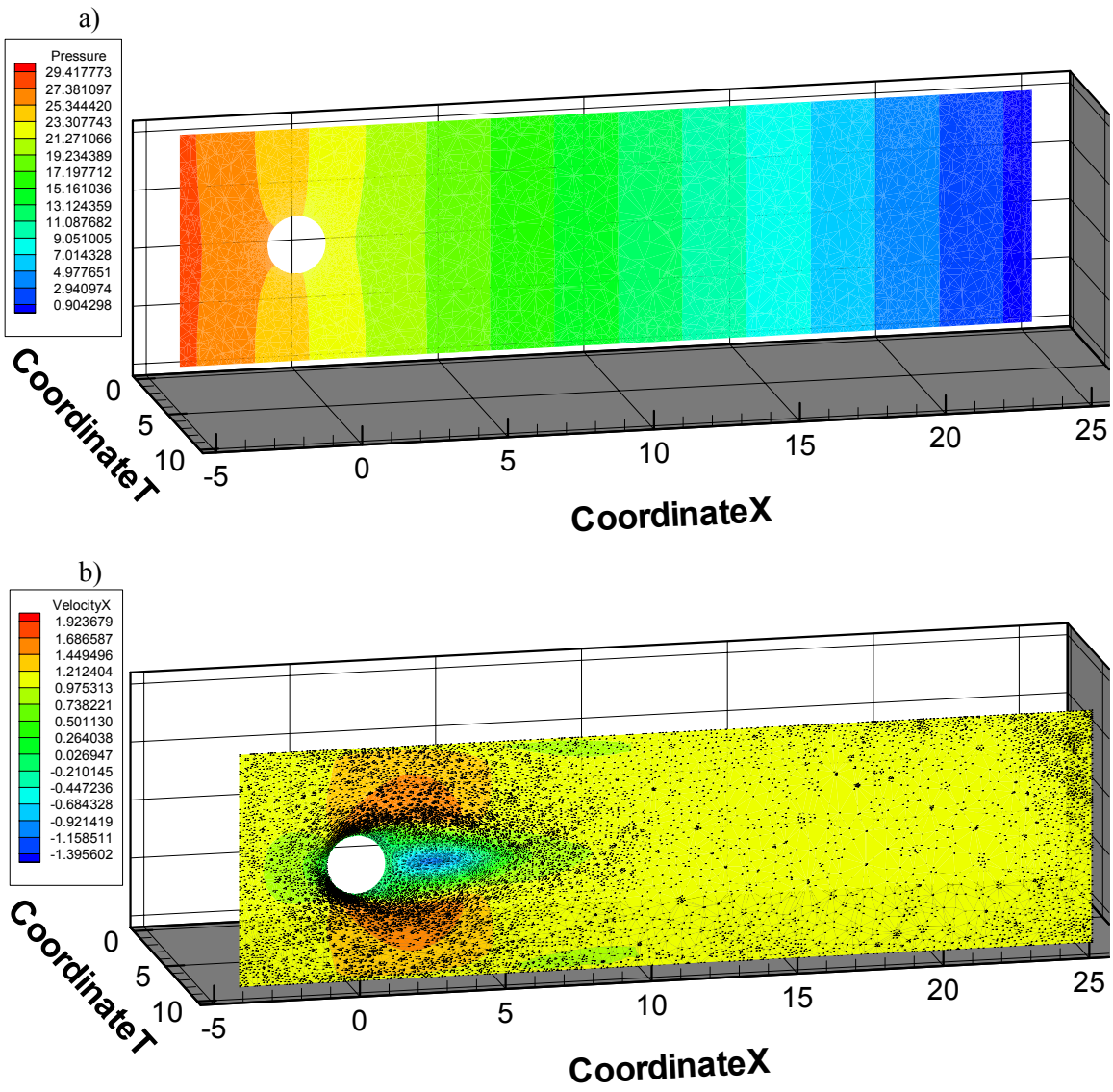

Figure 8.50: Unsteady 2-D flow past a circular cylinder. a) Pressure isocontours at $t = 0.2$; b) isocontours of u and velocity vector field at $t = 9.9$.

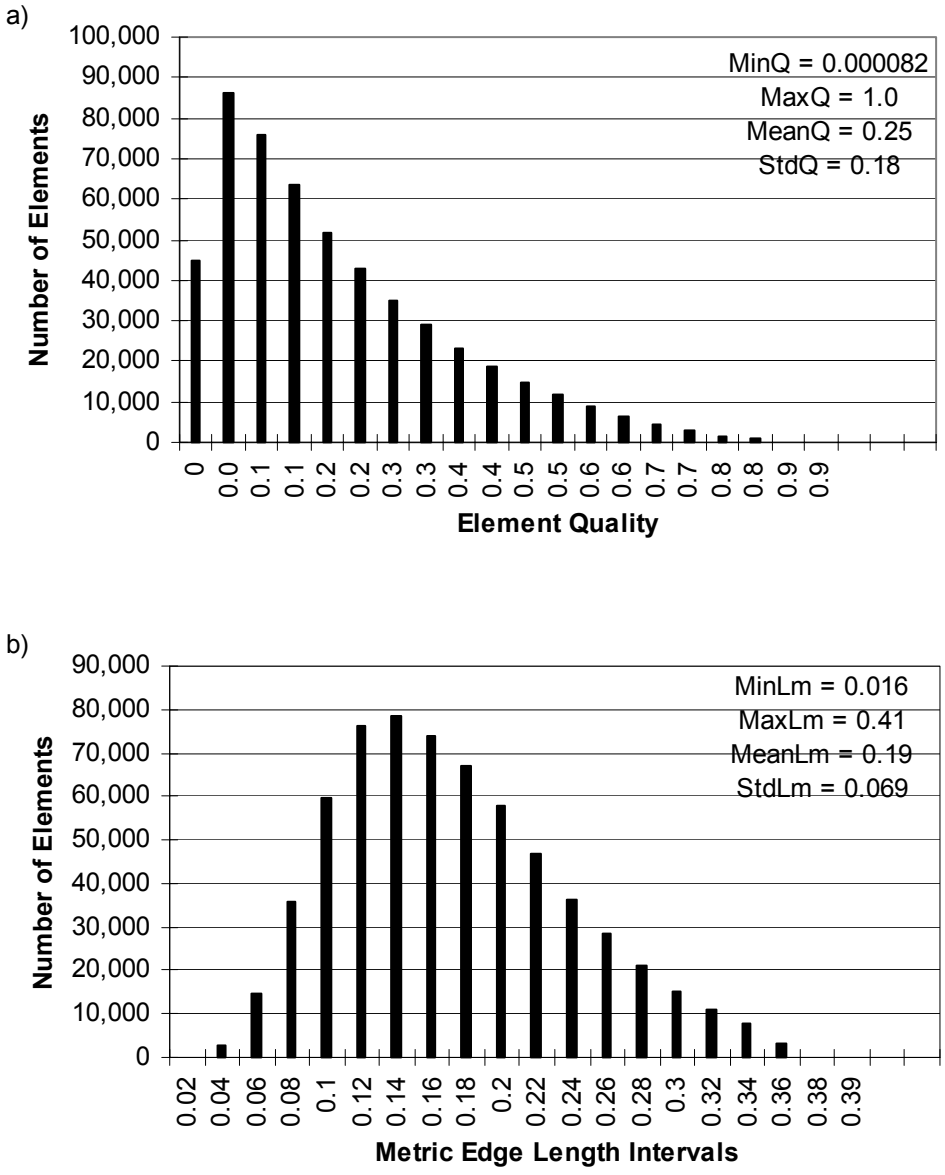

Figure 8.51: Unsteady 2-D flow past a circular cylinder. a) Histogram of element quality; b) histogram of metric edge length.

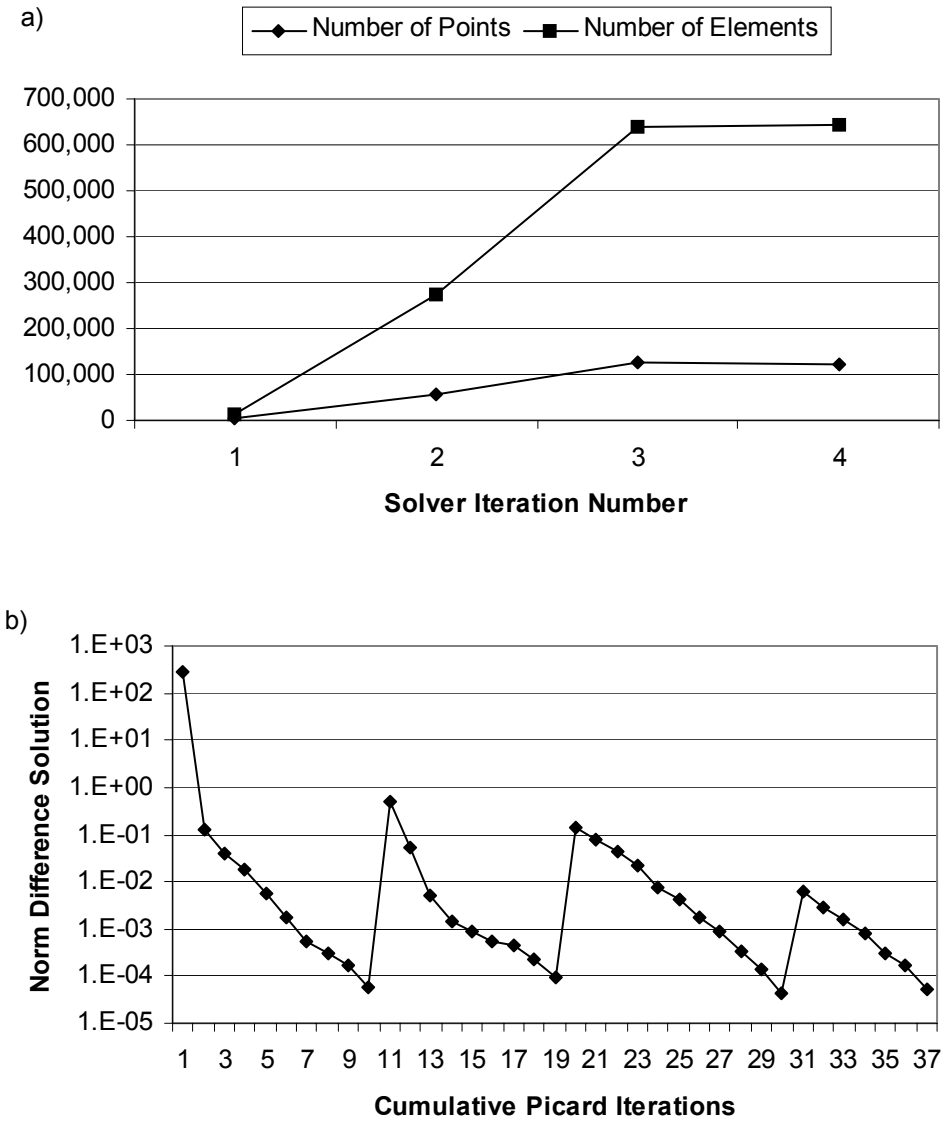

Figure 8.52: Unsteady 2-D flow past a circular cylinder. a) Numbers of mesh points and elements; b) relative norm of the difference between two consecutive solutions as a function of the cumulative number of Picard iterations.

8.6 Unsteady Flow Over a Backward-Facing Step

Another classical benchmark problem is considered here, namely the flow over a backward facing step. The Reynolds number for this flow is 800, based on a velocity equal to 2/3 of the maximum velocity of the quadratic profile imposed at the inlet of the solution domain. The space domain for this problem is shown in Figure 8.53.

The inlet boundary conditions for this problem are:

$$\begin{aligned} u &= \tanh\left(\frac{t}{\tau}\right) u_q\,(y) \\ v &= 0 \end{aligned} \tag{8.8}$$

where u_q is a quadratic function with a maximum of 1 at a $y = 0$ and a zero value at $y = \pm 0.5$. Notice that, although the space-time mesh of the domain is 3-D, the velocity profile is only 2-D. In a fashion similar to the previous test case for the flow cylinder, presented in section 8.5, a null velocity field is used for the initial condition at $t = 0$, no slip boundary conditions are applied on walls, a zero stress boundary condition is used at the exit and the dimensionless time (assuming that $\tau = 1$) spans the range from 0 to 10.

The initial mesh used, consisting of 6,204 mesh points and 28,710 tetrahedral elements, is shown in the top part of Figure 8.54 whereas the adapted mesh, comprising 99,446 mesh points and 522,625 tetrahedral elements, is shown in the bottom half of the same Figure. Three mesh adaptation cycles were used, with corresponding reduction factors of 0.6, 0.75 and 1.0, for a total of 4 solver executions. The adapted solution near the step is shown in Figure 8.55 and the local velocity field is shown in Figure 8.56. Figure 8.57 shows pressure isocontours at $t = 0.2$ and streamwise velocity contours, together with velocity vectors, at $t = 9.9$. Histograms for element quality and metric edge length are presented in

Figure 8.58. The numbers of mesh points and elements as a function of the solver iteration number are shown in Figure 8.59, together with the convergence history of the Picard method.

Qualitatively, the flow fields seems to develop according to expectations. A quantitative analysis, such as finding the precise location of the reattachment point, would be interesting to perform, but this must also take into consideration the fact that the fluid was accelerated from rest very rapidly. Using a more gradual time function (e.g., a ramp function) instead of equation 8.8 would result in a shift in time for the evolution of the solution. A precise analysis would be required to establish the proper initial condition for this flow and this is left for future work. It would also be interesting to extend this solution to a very long time range and to compare it with solutions using conventional approaches to determine whether a difference in time origins can be identified. As previously mentioned in section 8.5, this would first require that the current method be extended with an element-by-element solution method to limit the amount of memory required to determine the solution for long time periods.

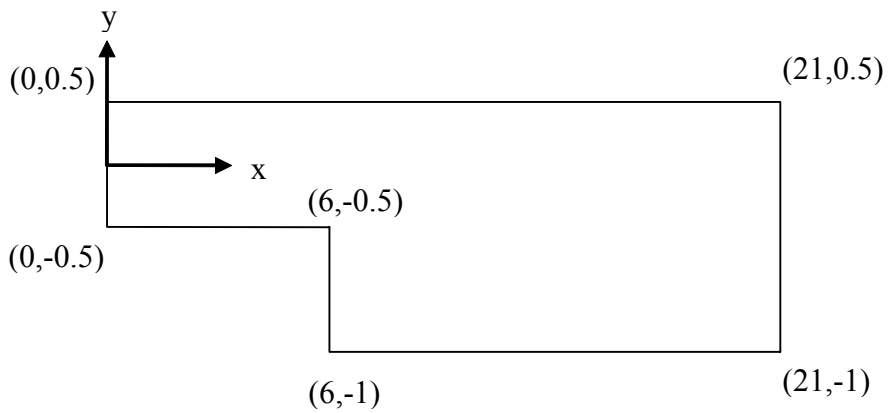

Figure 8.53: Unsteady 2-D flow past a back-facing step. Illustration of the space domain, which is extruded along the time axis for the space-time mesh.

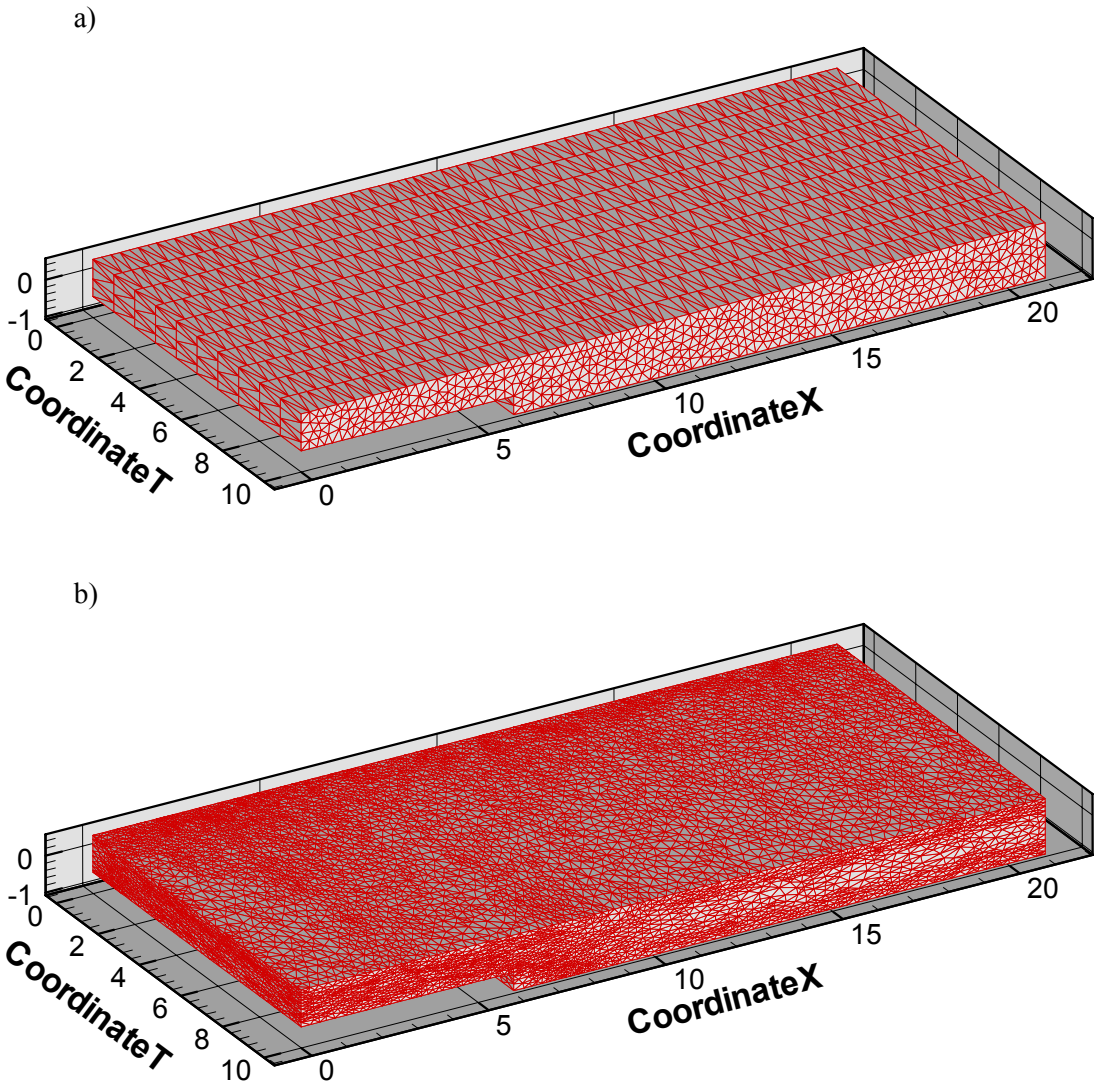

Figure 8.54: Unsteady 2-D flow past a back-facing step. a) Initial mesh; b) adapted mesh.

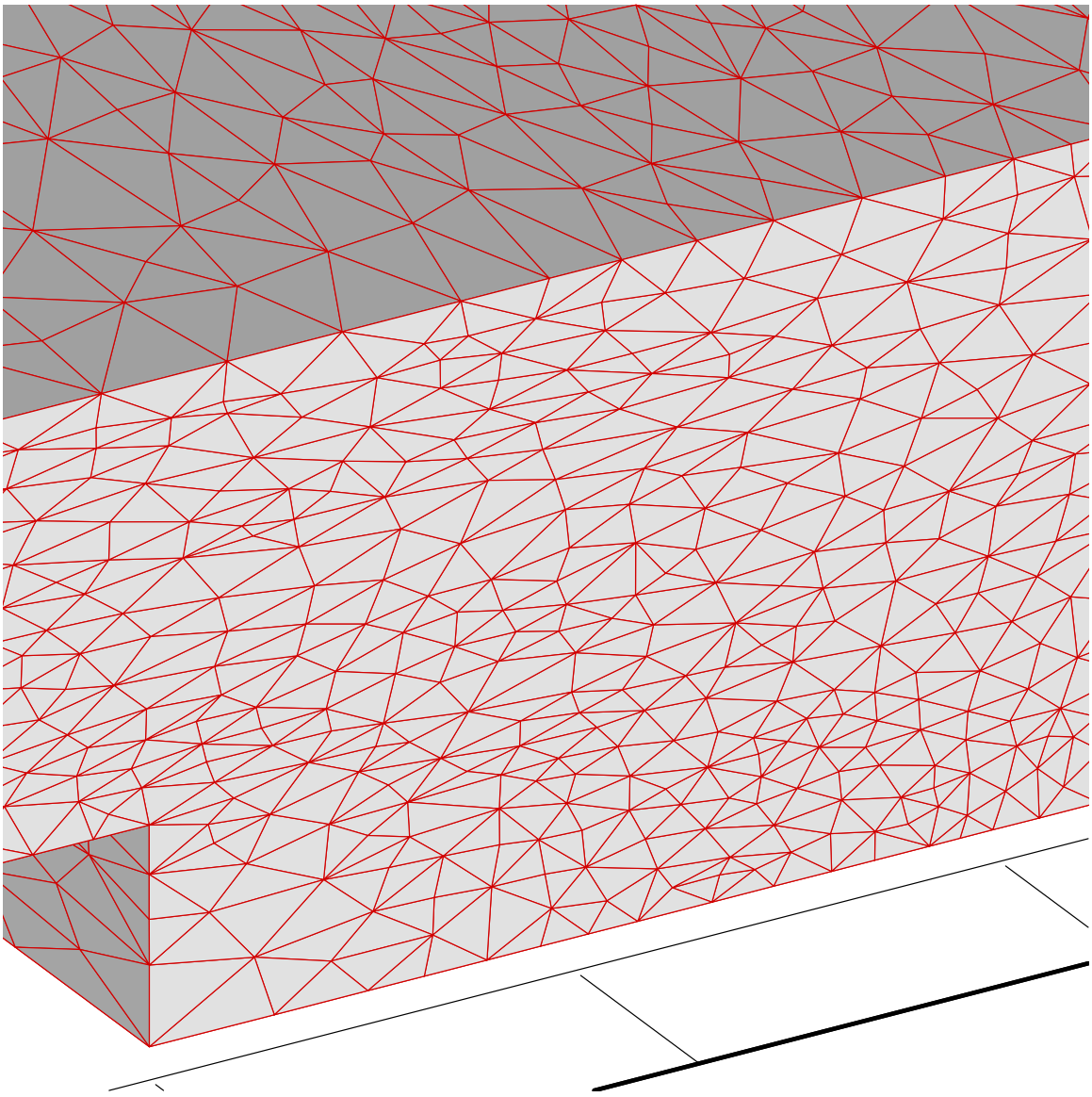

Figure 8.55: Unsteady 2-D flow past a back-facing step. Adapted mesh near the step at $t = 10$.

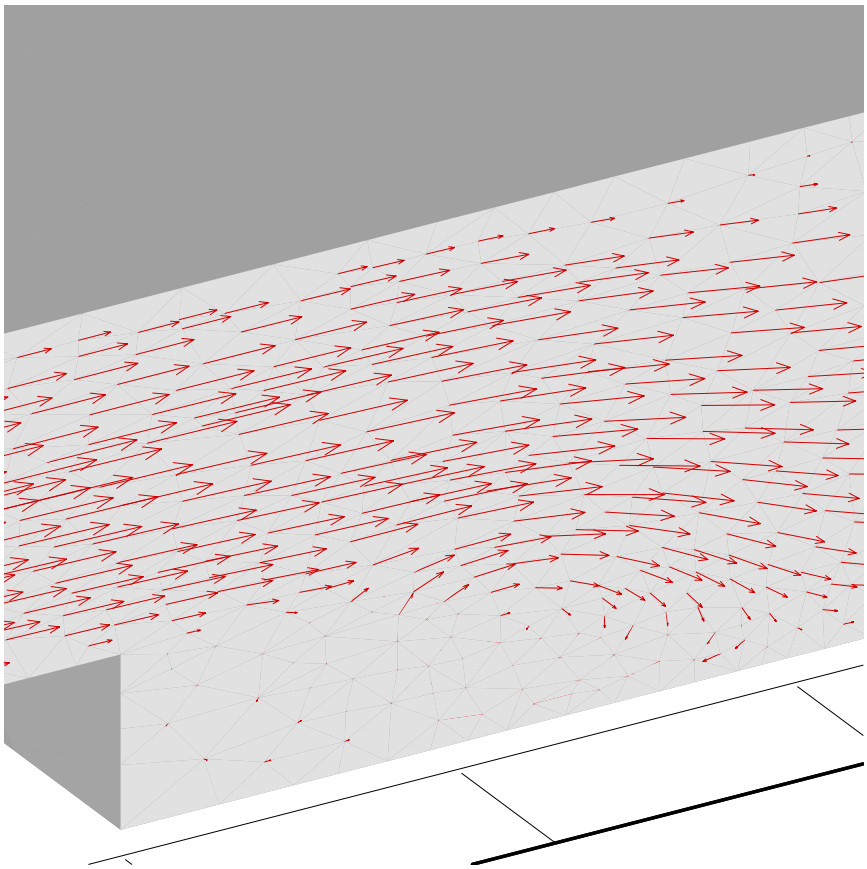

Figure 8.56: Unsteady 2-D flow past a back-facing step.Velocity vectors near the step at $t = 10$.

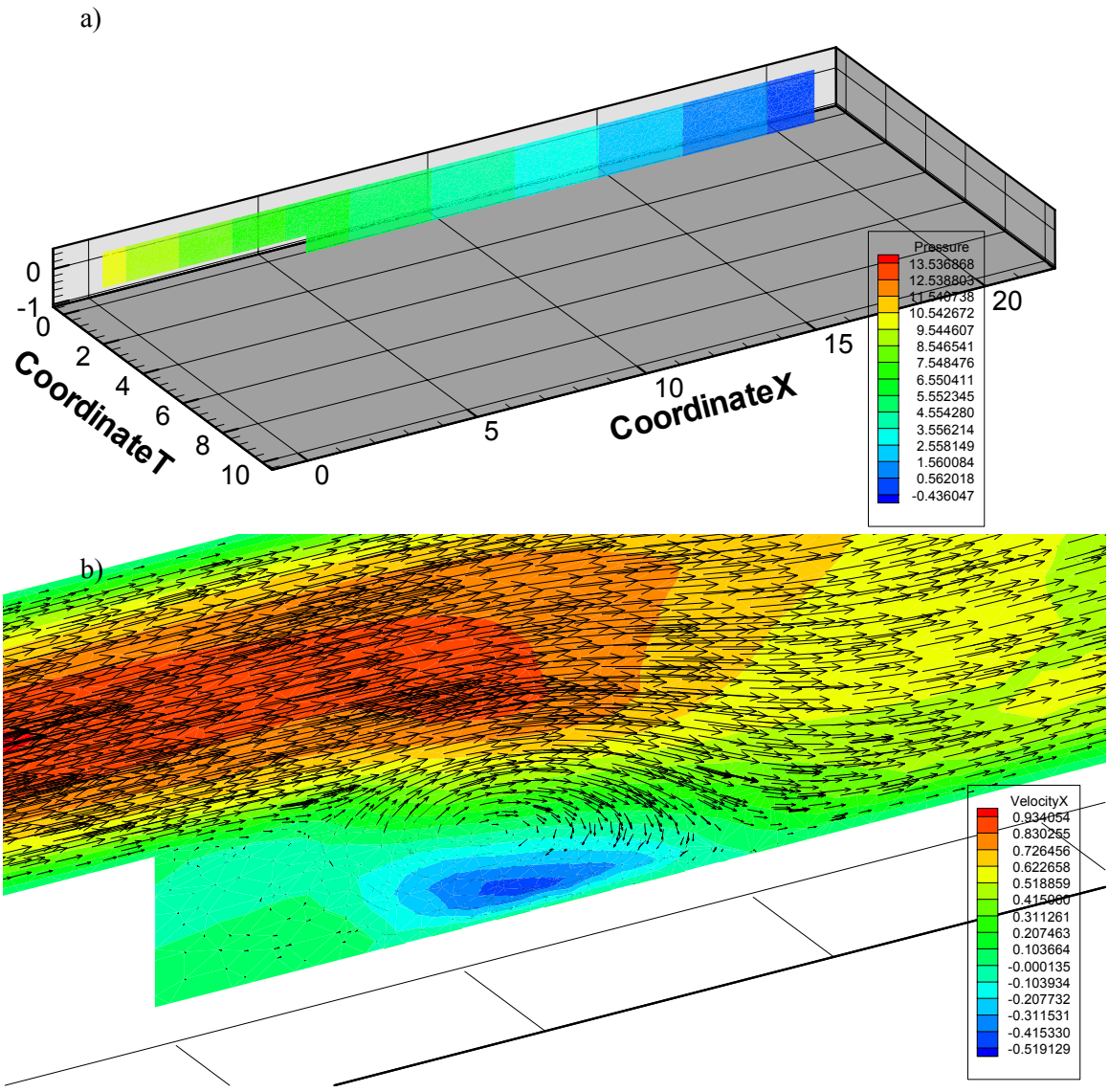

Figure 8.57: Unsteady 2-D flow past a back-facing step. a) Pressure contours at $t = 0.2$; b) u contours and velocity vectors at $t = 9.9$.

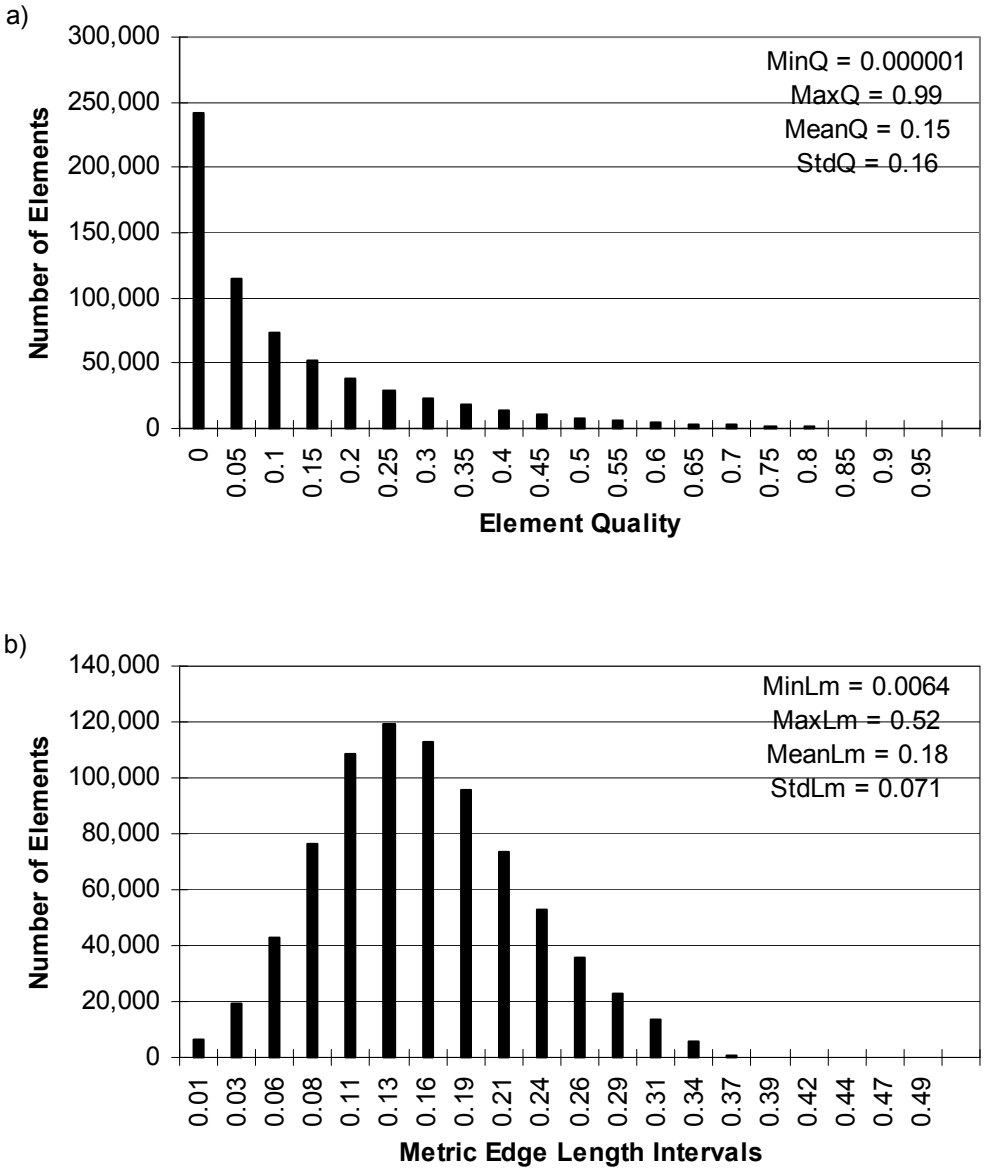

Figure 8.58: Unsteady 2-D flow past a back-facing step. a) Histogram of element quality; b) histogram of metric edge length.

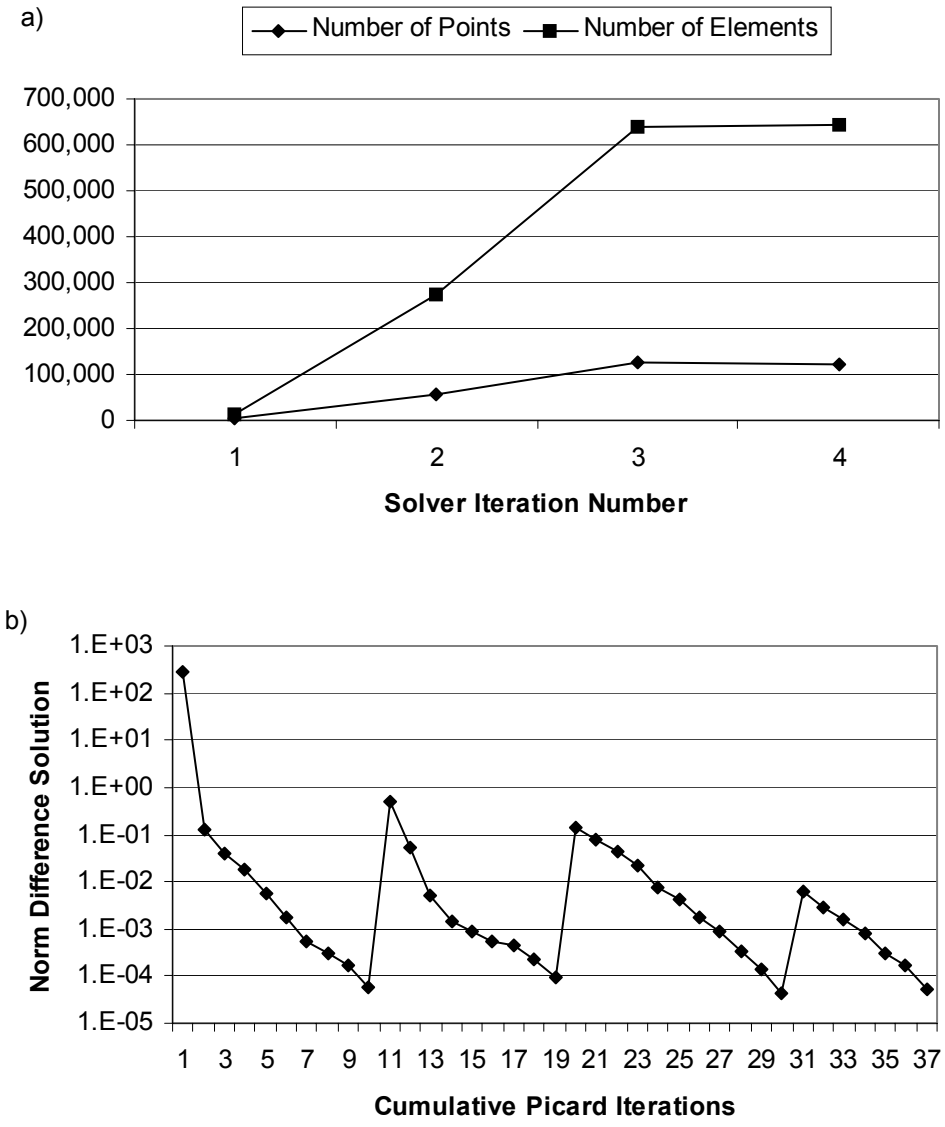

Figure 8.59: Unsteady 2-D flow past a back-facing step. a) Numbers of mesh points and elements; b) relative norm of the difference between two consecutive solutions as a function of the cumulative number of Picard iterations.

8.7 Unsteady Flow in a Lid-Driven Cavity

The incompressible flow in a lid-driven cavity at a Reynolds number of 400 is investigated to facilitate the comparison with the solution obtained by Pontaza and Reddy (2004) using a higher order spectral method. The space domain is a unit square with the x and y coordinates each ranging between 0 and 1. This space domain is extruded in time from $t = 0$ to $t = 6$.

The boundary conditions are the same as the ones specified by Pontaza and Reddy (2004). More specifically, the velocity u at the plane $y = 1$ is given as

$$\begin{aligned} u(x) &= \tanh\left(\tfrac{t}{\tau}\right)\tanh(\beta x) \qquad 0 \le x \le 0.5 \\ u(x) &= -\tanh\left(\tfrac{t}{\tau}\right)\tanh(\beta(x-1)) \quad 0.5 < x \le 1 \end{aligned} \tag{8.9}$$

where $\tau = 1$ and $\beta = 50$. The initial solution at $t = 0$ is the null velocity field and no-slip boundary conditions are used for the other walls of the cavity. Notice that the boundary conditions for the problem are chosen such that the velocity specified at the top of the cavity matches the no-slip boundary condition at the walls $x = 0$ and $x = 1$ (without any sudden jump in boundary conditions specified on the moving lid and the walls, where a no-slip boundary condition is imposed).

The initial space-time mesh for the cavity is shown in Figure 8.60, where the z axis corresponds to the time axis with $t = 6$ at the front. The number of mesh points in this initial mesh is 12,221 and the number of elements is 60,000. Figure 8.61 shows the adapted mesh and Figure 8.62 shows a detail of the adapted mesh.

Streamlines at times $t = 0.2$, 1.0, 2.5 and 6.0 (the same as the ones presented by Pontaza and Reddy (2004) to facilitate a comparison) are shown in Figure 8.63. Details for

u and v are also shown close to $t = 6.0$ in Figure 8.64. Although the solution appears to be qualitatively similar for the most part, some differences remain visible even after 3 mesh adaptation cycles and 4 solver executions of the fully coupled space-time approach. First, at time $t = 0.2$, the point in the centre of streamlines where the fluid is not moving seems to be shifted slightly to the left. Second, at time $t = 6.0$ the recirculation that should appear near the bottom right corner could not be captured. Further investigation is necessary to know the exact cause for this, but it speculated that the GLS with linear shape functions only is not of sufficiently high order to capture these fine details with the current level of refinement for the mesh (having about 100,000 for the space-time domain). Recall that the method of Pontaza and Reddy (2004) is of much higher order. On the other hand, it should be noted that even with their high order method care was taken by these others to refine the mesh further near the walls and close to $t = 0$ to better capture the rapid variation of the solution in these region. This indicates that both these methods, a higher order method and a mesh adaptation method, could benefit from being combined together.

The histograms for the element quality and the metric edge lengths are shown in Figure 8.65. The Figure 8.66 shows the number of mesh points and elements for each solver execution and the convergence criteria as a function of the cumulative Picard iteration.

The error reduction factor used were 0.6, 0.75, and 1.0 corresponding to 3 mesh adaptation cycles and 4 solver executions. Notice that in Figure 8.66 the number of mesh points levels out after the second mesh adaptation cycles because the maximum number of mesh points allowed for the refinement was set to 100,000 to avoid excessive memory consumption. This is a case where the arbitrary choice for the error reduction factor is

problematic and needs manual adjustment depending on how much computer memory and CPU are available for the simulation.

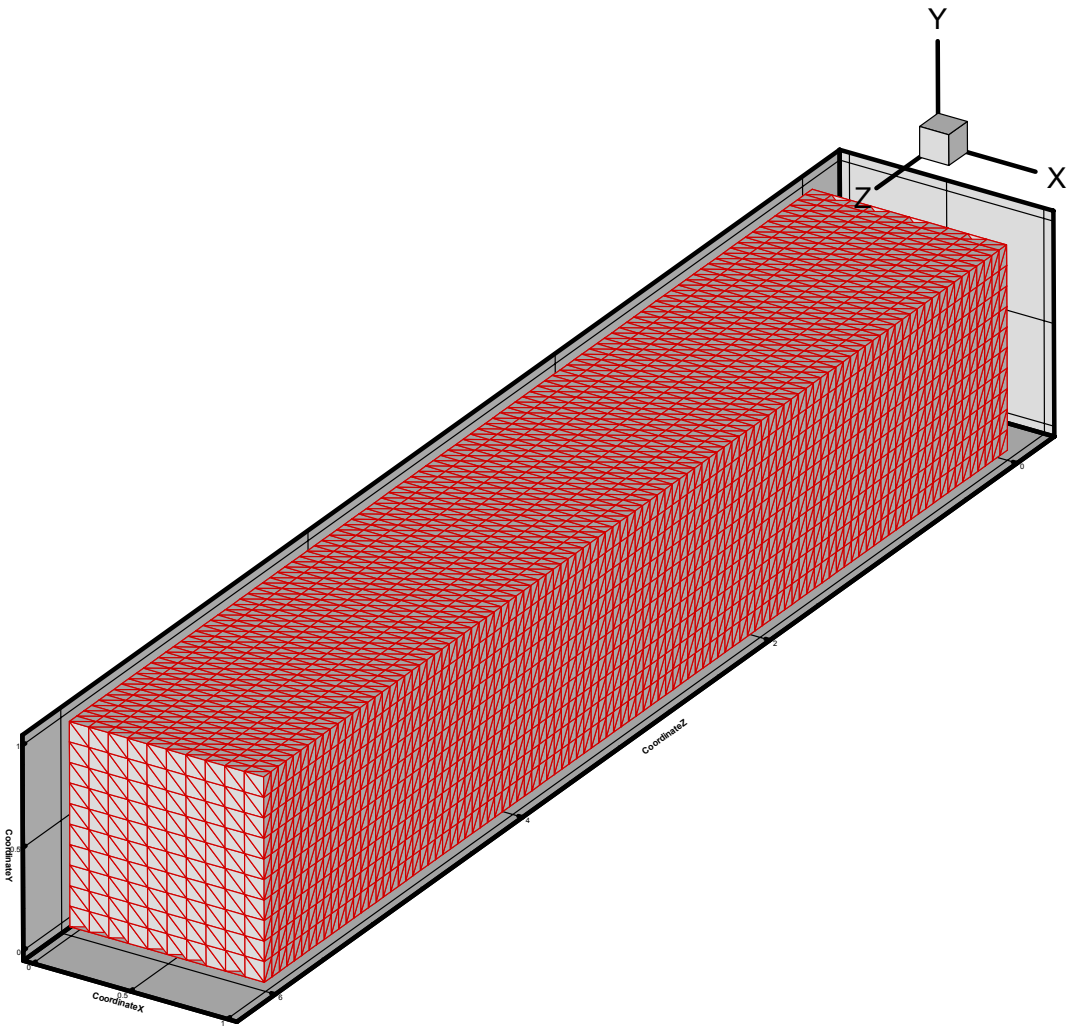

Figure 8.60: Unsteady 2-D flow in a lid-driven cavity. Initial mesh, in which the z axis corresponds to time.

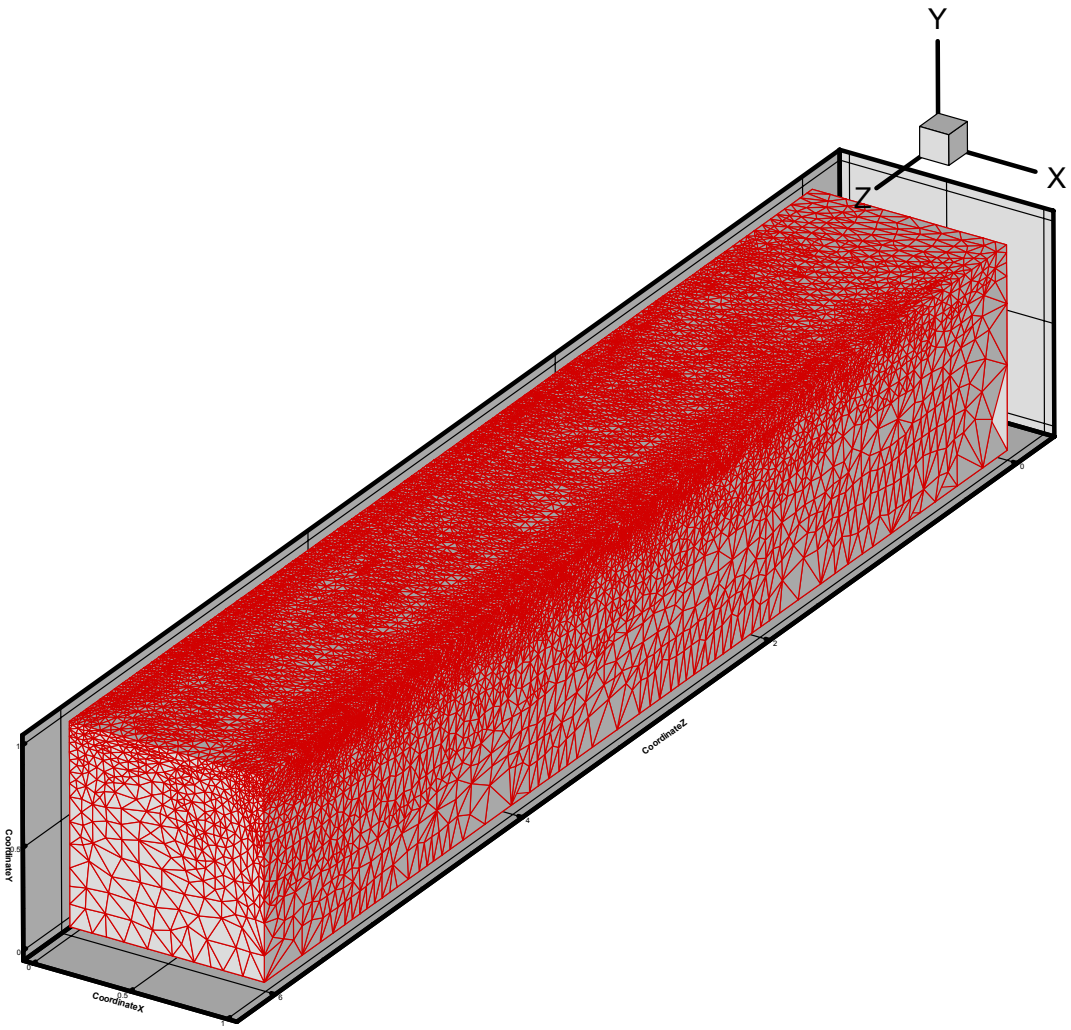

Figure 8.61: Unsteady 2-D flow in a lid-driven cavity. Adapted mesh, in which the z axis corresponds to time.

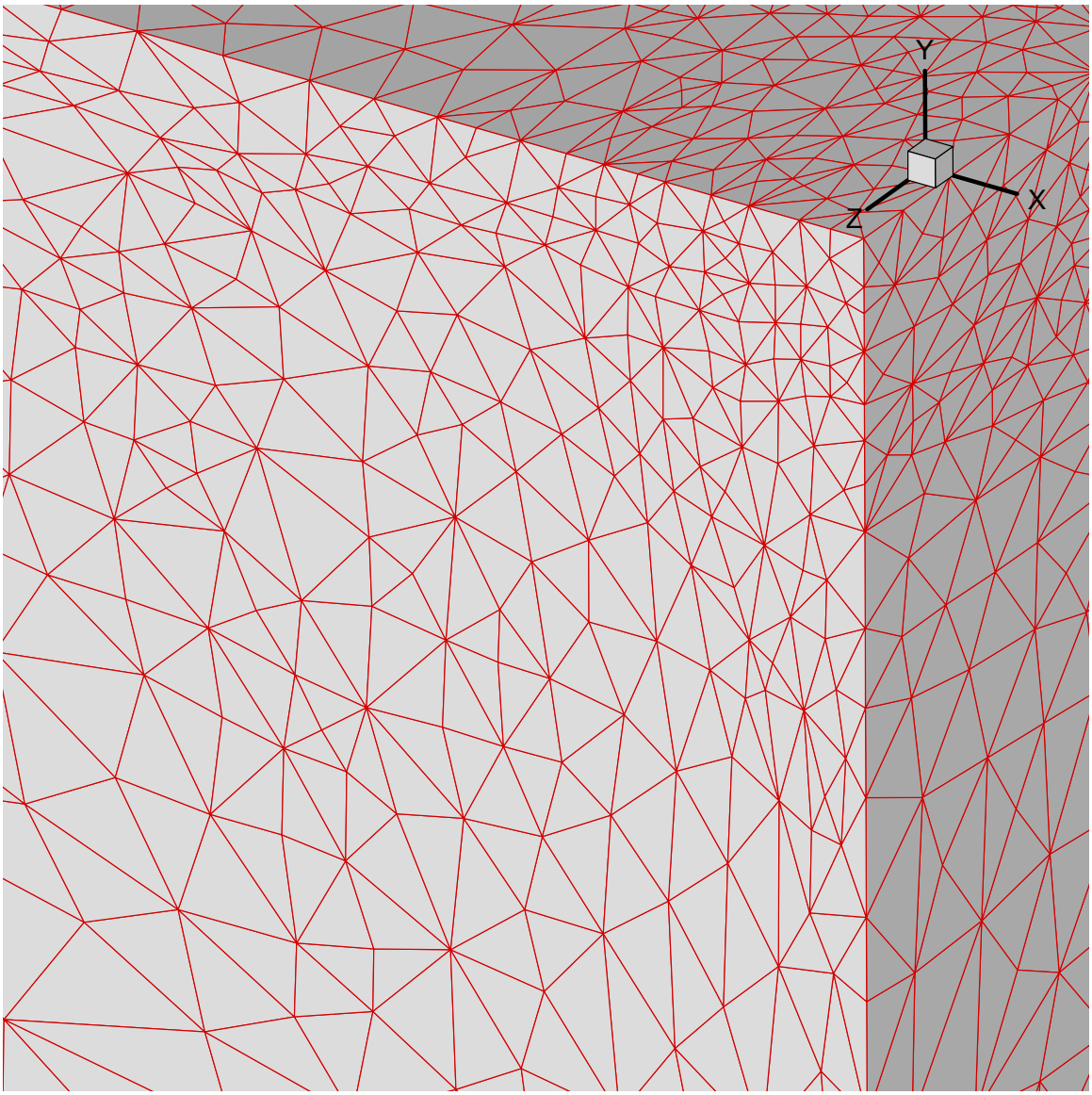

Figure 8.62: Unsteady 2-D flow in a lid-driven cavity. Adapted mesh near the point (1, 1, 6), in which the z axis corresponds to time.

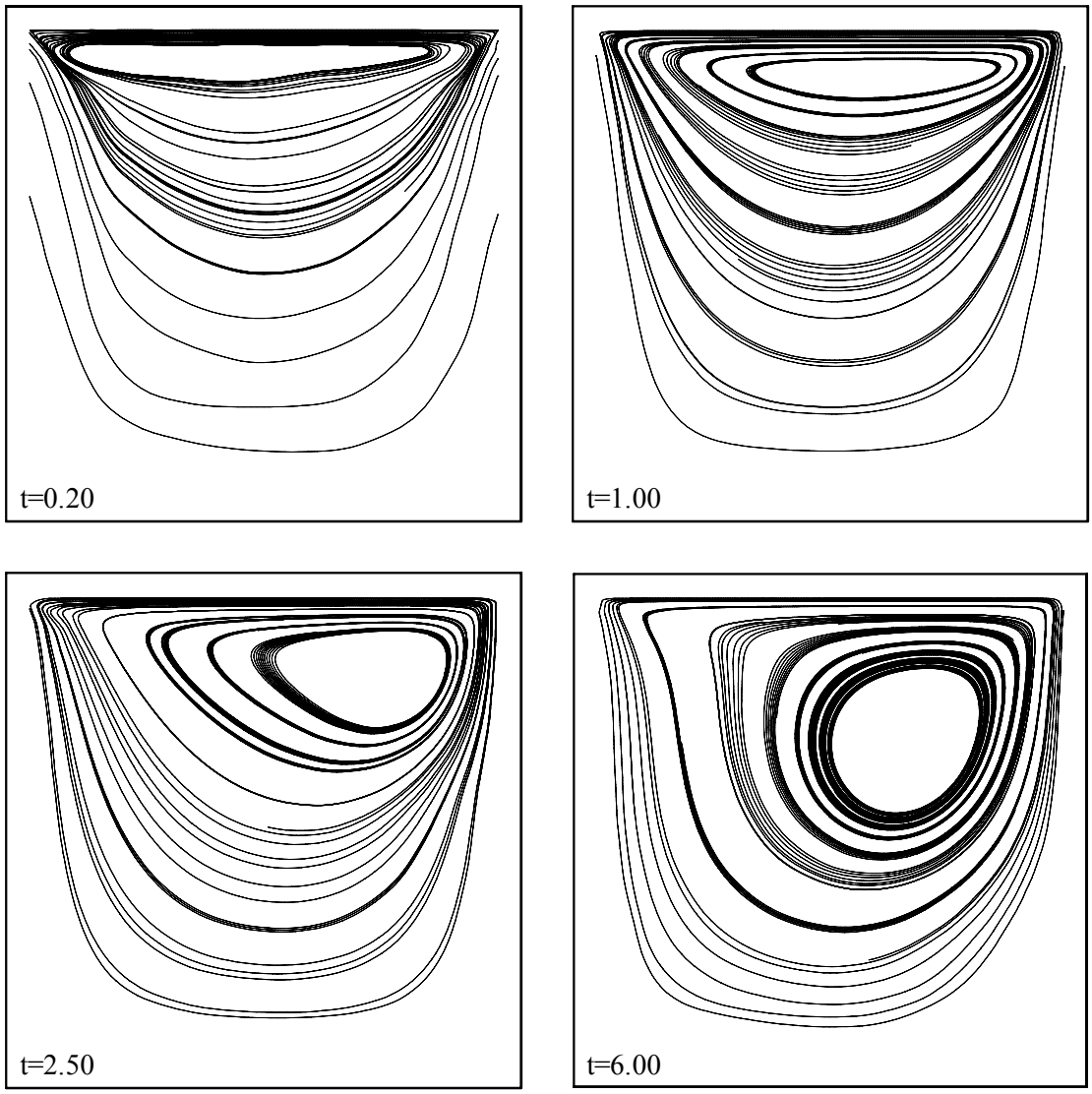

Figure 8.63: Unsteady 2-D flow in a lid-driven cavity. Streamlines at $t = 0.2, 1.0, 2.5$ and 6.0.

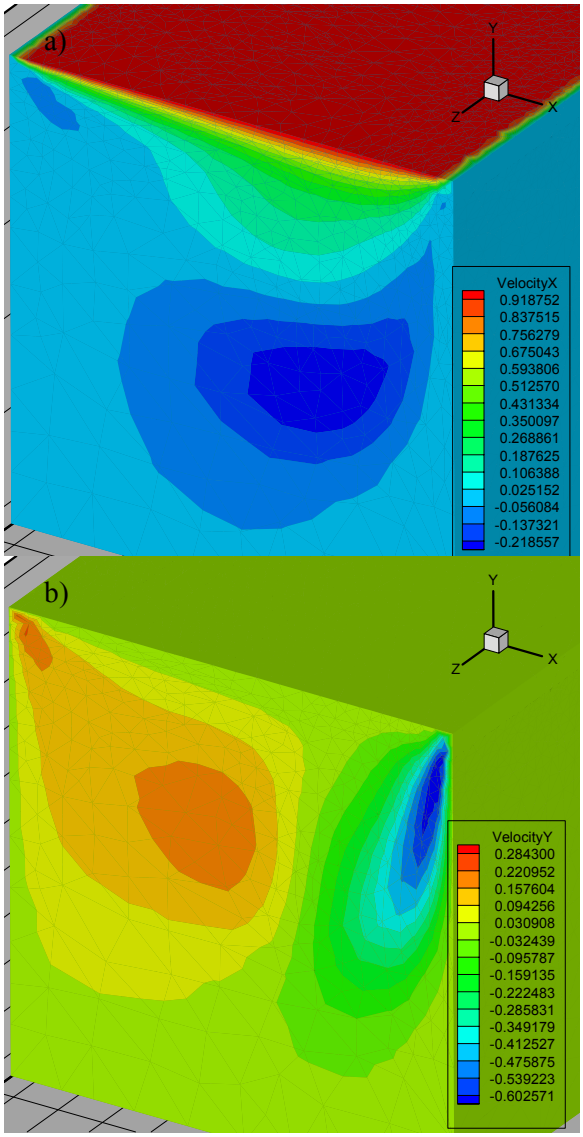

Figure 8.64: Unsteady 2-D flow in a lid-driven cavity. Solutions at $t = 6$ using the adapted mesh, in which the z axis corresponds to time. a) Isocontours of u; b) isocontours of v.

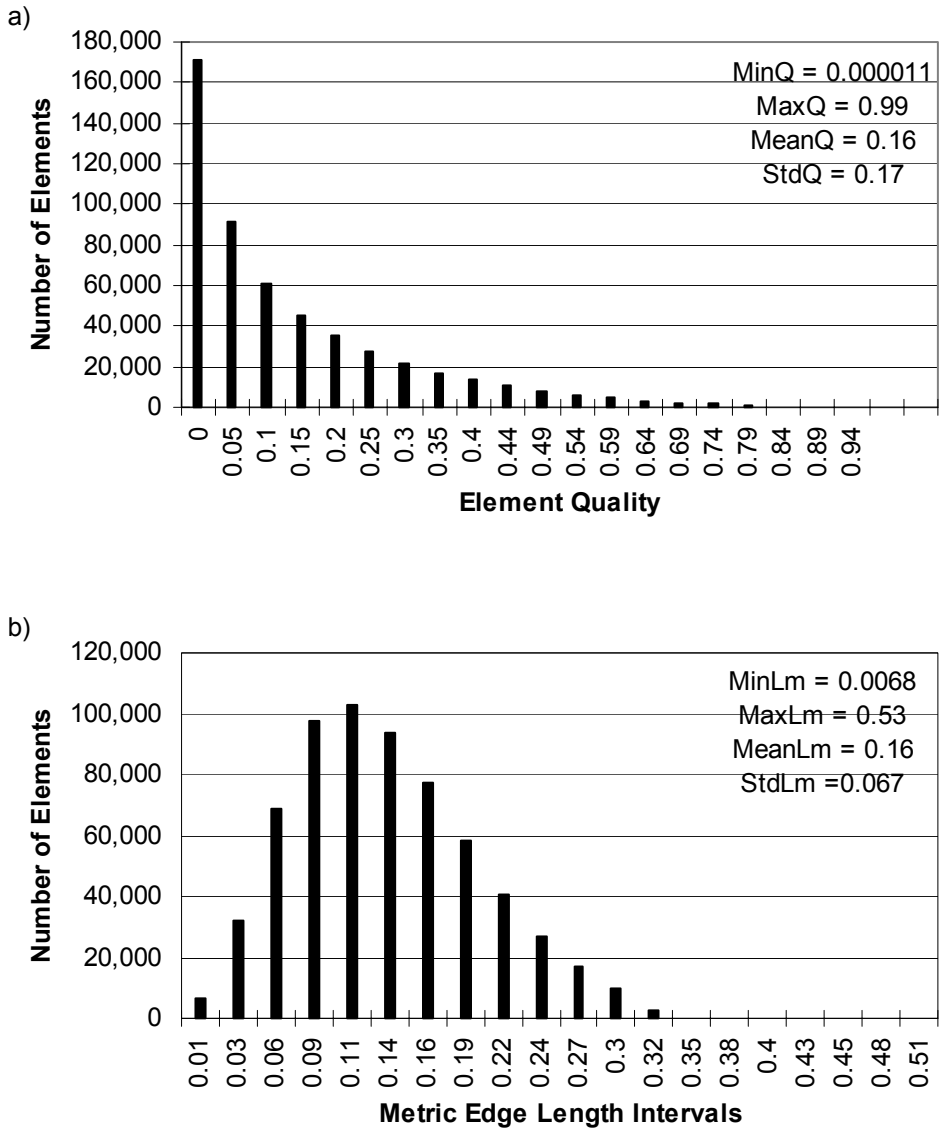

Figure 8.65: Unsteady 2-D flow in a lid-driven cavity. a) Histogram of element quality; b) histogram of metric edge length.

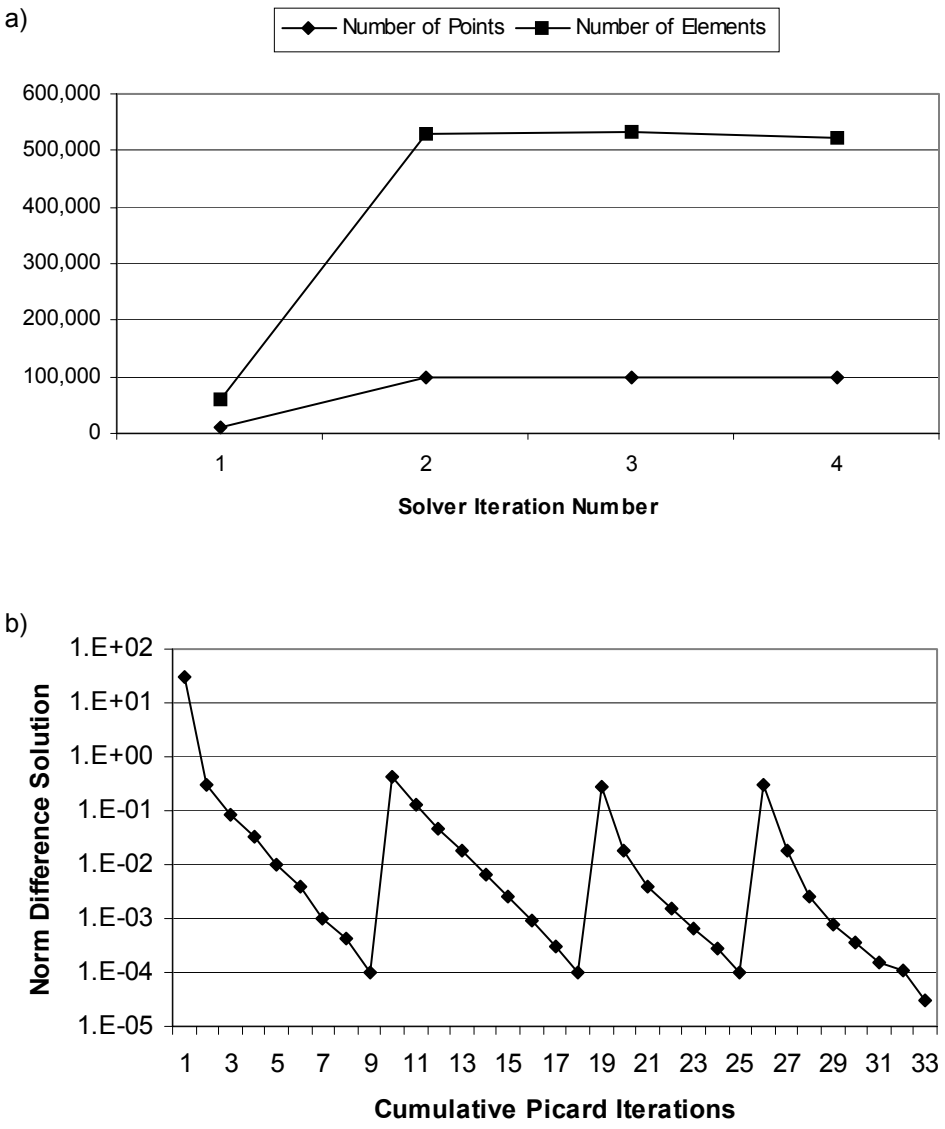

Figure 8.66: Unsteady 2-D flow in a lid-driven cavity. a) Numbers of mesh points and elements; b) relative norm of the difference between two consecutive solutions as a function of the cumulative number of Picard iterations.

Chapter 9

Conclusions and Recommendations for Future Research

9.1 Conclusions

The primary objective of this research was to develop a mesh adaptation methodology that is suitable for solving unsteady partial differential equations discretized using the finite element method. The approach chosen was to extend existing edge based anisotropic mesh adaptation strategies to operate on meshes in 2-D, 3-D and 4-D, in combination with a fully-coupled space-time finite element formulation and an interpolation-based error estimator that accounts for error estimates in both space and time in a unified fashion. The presented method can be viewed as a generalization of mesh adaptation strategies for steady problems to unsteady ones by extending fully unstructured simplicial meshes of spacial dimension d to simplicial meshes of dimension $d+1$ which cover the entire space-time

domain.

A data structure was written from the ground up, which is capable of implementing mesh modification algorithms in a dimension-independent manner. Mesh modification operators were developed, including edge splitting, edge collapsing and simulated edge swapping algorithms and were combined with an anisotropic mesh smoothing strategy based on the inscribed ellipsoids. The key in the successful extension of these algorithms to 4-D was the realization that an edge swapping algorithm can be expressed by a sequence of two simpler operators, namely an edge split algorithm followed by an edge collapse algorithm. The advantage of this approach is that the simulated edge swapping can operate in 2-D, 3-D and 4-D both inside the mesh and on its boundary without any special treatments that would be dimension-specific or that would be applied only on the boundary of the mesh. To the best of our knowledge, this is the first time that mesh modifications were shown to operate in a dimension higher than 3 with the ability to modify the boundary mesh. In contrast, previously existing methods that operate on higher dimensional meshes cannot keep track of the boundary of the domain.

An evaluation of the anisotropic meshing strategy on 2-D, 3-D and 4-D meshes using an analytical metric and the unsteady heat equation was presented. The resulting element quality was found to be very high for the 2-D cases, comparable to those produced by methods found in the literature for the 3-D cases, but unsatisfactory for the 4-D cases. For the analytical metric field, it appears that some low quality elements remain for the 4-D case, resulting in a minimum element quality of 0.0 and an average quality of 0.058, although the maximum quality was 0.95. For the heat equation with a manufactured

solution, the minimum element quality was 10^{-5} the average was 0.061, and the maximum was 0.89. Further improvements appear to be necessary for the highly anisotropic case, but, for the heat equation for which the anisotropic aspects of the metric field are much less pronounced, the element quality was fair. The element quality seems to be highly correlated to the degree of anisotropy required by the metric field that drives the mesh optimization procedure, irrespectively of whether this metric field is specified by an analytical function or constructed from an error estimator.

Another salient characteristic that appears when comparing results with the 2-D, 3-D and 4-D meshes is the ratio of the number of elements and the number of mesh points. This ratio for the test case with the analytical metric was found to be 1.79, 5.17 and 15.43, respectively, for the 2-D, 3-D and 4-D meshes, growing by a factor of approximately 3 when increasing the space dimension by 1. The computation cost per 1000 mesh points increases by a factor of 18, when moving from 2-D to 3-D, and by a factor of 7, when moving from 3-D to 4-D. The cost associated with simplicial meshes drastically increases with increasing space dimension, which indicates that generalizing conclusions based on studies with 2-D meshes to higher dimensions can be very misleading.

For the meshing algorithms to operate on geometries that are of engineering interest, a geometry reconstruction algorithm was presented and implemented using the same data structure as for the meshing algorithms. Because only linear interpolation functions were used for the finite element method, it was found sufficient to use quadratic finite element interpolation functions to represent the geometry for the simple cases of this thesis. The geometry algorithm shown is relatively simple and can be implemented with most ex-

isting finite element solvers, provided that they support quadratic shape functions. This permits the reconstruction of a geometry from existing finite element meshes while allowing for the mesh to be modified without depending on a CAD package.

To investigate the fully coupled adaptive space-time approach, a finite element flow solver for the incompressible Navier-Stokes equations was developed using a Galerkin/least-square formulation, linear interpolation functions and a Picard method to handle the non-linearity of the equations. The salient feature of this simple flow solver, compared to approaches commonly found in the literature, is that the space-time formulation is fully coupled, which allows the solution to the equations to be sought over the entire space-time domain (without any time-stepping strategies). Time dependent boundary conditions were also specified to simulate the acceleration of the flow from rest. This permits the use of a null velocity field as the initial condition for the problem in a way that satisfies the continuity equation. Verifications for the unified space-time flow solver were first done for a problem using a manufactured solution; for this case, the behaviour of the L^2 error norm shows a good agreement between the numerical and the analytical solutions. The same test was repeated for meshes that were adapted anisotropically. The agreement between the numerical solution and the analytical solution was good, but it is was more difficult to determine a characteristic mesh size on anisotropic meshes than on uniform meshes because the element size can vary considerably throughout the space-time domain.

The anisotropic space-time mesh adaptation scheme was also evaluated by numerical solutions of the flow past a circular cylinder at a Reynolds number of 100, the flow over a backward facing step at a Reynolds number of 800 and the flow in a lid-driven cavity at a

Reynolds number of 400. For these test cases, the Picard method with the combined mesh adaptation strategy and solution interpolation, introduced to provide a restart solution for the solver after mesh adaptation, exhibit excellent convergence behaviour. These tests were performed using 3-D space-time meshes only, so they correspond to 2-D space domains. The qualitative development of these solutions appear to be in agreement with expected trends, but these tests were limited because they extended only over a relatively short time span. Overall, however, it may be concluded that the adaptive space-time flow solver behaves in a robust manner for these simple problems at relatively low Reynolds numbers.

The research performed in this thesis has shown that extending meshing algorithms to a dimension higher than three is possible. Furthermore, when combined with a fully-coupled space-time finite element formulation and a suitable error estimator that accounts for both the discretization errors in space and in time, this approach allows for existing mesh adaptation strategies for steady problems to be extended to unsteady problems. The proposed framework paves the way for further research on space-time mesh adaptation for unsteady problems.

9.2 Recommendations for Future Research

Several recommendations can be made to pursue research on the proposed space-time mesh adaptation method to increase its capability and efficiency. These include the following:

- Adoption of mesh-free methods may alleviate problems associated with the increase of the ratio of the number of elements to the number of mesh points in 4-D dimensions

and relax the requirement on mesh quality. It is believed that most of the meshing algorithm presented here could be extended in such a context.

- It would be productive to implement higher order finite element methods on simplicial meshes, such as the method presented by Pontaza and Reddy (2004), and to assess the impact that the space-time mesh adaptation can have in this case compared to standard time-stepping approaches, including also methods using hexahedral meshes.

- The impact of space-time mesh adaptation on the discretization error for unsteady problems could be assessed for strongly time-dependent problems, such as moving shock waves or sound waves.

- Different error estimators, for example a residual based error estimator, could be evaluated and their relative impacts on the space-time mesh adaptation procedure could be compared, thus extending their application to unsteady problems. It would also be interesting to determine the impacts of different scaling methods used for the temporal part of the error estimator and how they can affect the refinement of the mesh in time versus in space.

- The stabilization parameters used in the Galerkin/least-square appear to be particularly sensitive in the case of anisotropic meshes. Therefore, it would be preferable to adopt methods that do not require such stabilization terms, for example discontinuous Galerkin methods (Remacle et al. (2005)).

- The fully coupled approach presented leads to the assembly of a global matrix for the solution on the entire space-time domain such that the size of the matrix grows as

the time span increases. To circumvent this limitation, iterative methods that do not require the assembly of a global matrix could be employed and the space-time mesh could be partitioned to compute the solution for only one partition in each step, while temporarily storing the rest of the space-time solution and mesh on disk. Algebraic or geometric multigrid methods could also be extended in a space-time context.

- For the pure advection equation, Perrochet and Azérad (1995) have shown that the convective term can be combined with the temporal derivative to solve the equation using a linear operator in a unified space-time domain. If this can be extended to the Navier-Stokes equations, then iterative methods for linear problems could be directly employed, which would result in significant computational savings.

- The impact of space-time mesh adaptation could be studied on a variety of problems that are of engineering interest, including compressible flows, turbulent flows, structural dynamics, the Schrödinger equation, Maxwell equations and possibly others.

References

Abdoulaev, B., S. Cadeddu, G. Delussu, M. Donizelli, L. Formaggia, and A. Giachetti (1998). Viva: The virtual vascular project. *IEEE Transactions on Information Technology in Biomedicine 2*(4), 268–274.

Ainsworth, M. and J. T. Oden (2000). *A Posteriori Error Estimation in Finite Element Analysis*. New York, U.S.A.: Wiley-Interscience.

Alam, J. M., N. K.-R. Kevlahan, and O. V. Vasilyeve (2006). Simultaneous space-time adaptive wavelet solution of nonlinear parabolic differential equations. *Journal of Computational Physics 214*, 829–857.

Alauzet, F., P. J. Frey, P. L. George, and B. Mohammadi (2007). 3d transient fixed point mesh adaptation for time-dependent problems: Application to CFD simulations. *Journal of Computational Physics 222*, 592–623.

Alauzet, F., P. L. George, B. Mohammadi, P. Frey, and H. Borouchaki (2003). Transient fixed point-based unstructured mesh adaptation. *International Journal for Numerical Methods in Fluids 43*, 729–745.

Alauzet, F., A. Loseille, A. Dervieux, and P. Frey (2006). Multi-dimensional continuous

metric for mesh adaptation. In *15th International Meshing Roundtable*, Birmingham, AL, USA, 2006.

Amenta, N. (1999). Optimal point placement for mesh smoothing. *Journal of Algorithms 30*, 302–322.

Babuska, I. (1973). The finite element method with lagragian multipliers. *Numer. Math 20*, 179–192.

Baker, T. J. (2002). Mesh movement and metamorphosis. *Engineering with Computers 18*, 188–198.

Baker, T. J. (2004). Identification and preservation of surface features. In *International Meshing Roundtable*. Sandia National Laboratories.

Balay, S., K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang (2004). PETSc users manual. Technical Report ANL-95/11 - Revision 2.1.5, Argonne National Laboratory.

Balay, S., K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang (2001). PETSc Web page. http://www.mcs.anl.gov/petsc.

Balay, S., V. Eijkhout, W. D. Gropp, L. C. McInnes, and B. F. Smith (1997). Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen (Eds.), *Modern Software Tools in Scientific Computing*, pp. 163–202. Birkhäuser Press.

Belhamadia, Y., A. Fortin, and É. Chamberland (2004a). Anisotropic mesh adaptation for the solution of the stefan problem. *Journal of Computational Physics 194*, 233–255.

Belhamadia, Y., A. Fortin, and É. Chamberland (2004b). Three-dimensional anisotropic mesh adaptation for phase change problems. *Journal of Computational Physics 201*, 753–770.

Belytschko, T., J. M. Kennedy, and D. F. Schoeberle (1978). Quasi-eulerian finite element formulation for fluid-structure interaction. In *Joint ASEM/CSME Pressure Vessels and Piping Conference*, New York, U.S.A., pp. 13. Americain Society of Mechanical Engineers: ASME.

Bonciu, C. L., C. Leger, G. Lamarque, and L. D. Nguyen (1998). 4d reconstruction of the left ventricule using two successive cardiac cycles. *Innovation et technologie en biologie et médecine 19*(4), 249–263.

Borouchaki, H., P. L. George, F. Hecht, P. Laug, and E. Saltel (1997). Delaunay mesh generation governed by metric specifications. PART i. algorithms. *Finite Elements in Analysis and Design 25*(1), 61–83.

Borouchaki, H., P. L. George, and B. Mohammadi (1997). Delaunay mesh generation governed by metric specifications. PART II. applications. *Finite Elements in Analysis and Design 25*(1), 85–109.

Borouchaki, H., F. Hecht, and P. J. Frey (1998). Mesh gradation control. *International Journal for Numerical Methods in Engineering 43*, 1143–1165.

Bottasso, C. L. (2004). Anisotropic mesh adaptation by metric-driven optimization. *International Journal for Numerical Methods in Engineering 60*, 597–639.

Brezzi, F. (1974). On the existence, uniqueness and approximation of saddle-point problems arising from lagrange multipliers. *RAIRO Anal. Numér. R-2*, 129–151.

Buscaglia, G. C. and E. A. Dari (1997). Anistropic mesh optimization and its application in adaptivity. *International Journal for Numerical Methods in Engineering 40*, 4119–4136.

Cascon, J. M., L. Ferragut, and M. I. Asensio (2006). Space-time adaptive algorithm for the mixed parabolic problem. *Numerische Mathematik 103*, 367–392.

Castro-Díaz, M. J., F. Hecht, and B. Mohammadi (1995, October). New progress in anisotropic grid adaptation for inviscid and viscous flows simulations. Technical Report 2671, Institut National de Recherche en Informatique et en Automatique.

Castro-Diaz, M. J., F. Hecht, B. Mohammadi, and O. Pironneau (1997). Anisotropic unstructured mesh adaptation for flow simulations. *International Journal for Numerical Methods in Fluids 25*, 475–491.

Cebral, J. R. and R. Lohner (1999, October). From medical images to CFD meshes. In *Proceedings, 8th International Meshing Roundtable*, South Lake Tahoe, CA, U.S.A., pp. 321–331.

CGAL (2007). CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.

Chen, Y.-P. and Y.-Q. Huang (2001). Improved error estimates for mixed finite element for nonlinear hyperbolic equations: The continous-time case. *Journal of Computational Mathematics 19*(4), 385–392.

Chew, L. P. (1989). Constrained delaunay triangulations. *Algorithmica 4*, 97–108.

Chippada, S., C. N. Dawson, M. L. Martinez, and M. F. Wheeler (1998). Finite element

approximations to the system of shallow water equations I: Continous-time a priori error estimates. *SIAM Journal of Numerical Analysis 35*(2), 692–711.

Cook, R. D., D. S. Malkus, M. E. Plesha, and R. J. Witt (2002). *Concepts and Applications of Finite Element Analysis* (Fourth ed.). Wiley.

D'Azevedo, E. and R. Simpson (1991). On optimal triangular-meshs for minimizing the gradient error. *Numerische Mathematik 59*(4), 321–348.

Delibasis, K. K., N. Mouravliansky, G. K. Matsopoulos, K. S. Nikita, and A. Marsh (1999). MR functional cardiac imaging: Segmentation, measurement and WWW based visualisation of 4d data. *Future Generations in Computer Systems 15*(2), 185–193.

Deplano, V. and M. Siouffi (1999). Experimental and numerical study of pulsatile flows through stenosis: Wall shear stress analysis. *Journal of Biomechanics 32*(10), 1081–1090.

Diachin, L. F. and P. Knupp (2006). A comparison of two optimization methods for mesh quality improvement. *Engineering with Computers 22*, 61–74.

Dompierre, J., P. Labbé, F. Guibault, and R. Camarero (1998). Proposal benchmarks for 3d unstructured tetrahedral mesh optimization. In *7th International Meshing Rountable*, pp. 459–478.

Dompierre, J., M.-G. Vallet, Y. Bourgault, M. Fortin, and W. G. Habashi (2002). Anisotropic mesh adaptation: Towards user-independent, mesh-independent and solver-independent CFD. part III. unstructured meshes. *International Journal for Numerical Methods in Fluids 39*, 675–702.

Dompierre, J., M.-G. Vallet, M. Fortin, Y. Bourgault, and W. Habashi (1997, January). Anisotropic mesh adaptation: Towards a solver and user independent CFD. In *AIAA 35th Aerospace Sciences Meeting & Exhibit*, Number AIAA–97–0861, Reno, NV.

Donea, J., P. Fasoli-Stella, and S. Giuliani (1977). Lagrangian and eulerian finite element techniques for transient fluid-structure interaction problems. *Transaction of the 4th SMIRT Conference B.*, paper B1/2.

Donea, J. D. and A. Huerta (2003). *Finite Element Methods for Flow Problems*. Wiley.

Du, Q., Z. Huang, and D. Wang (2005). Mesh and solver co-adaptation in finite element methods for anisotropic problems. *Numerical Methods for Partial Differential Equations 21*(4), 859–874.

Du, Q. and D. Wang (2004a). Boundary recovery for three dimensional conforming delaunay triangulation. *Computer Methods in Applied Mechanics and Engineering 193*, 2547–2563.

Du, Q. and D. Wang (2004b). Constrained boundary recovery for three dimensional delaunay triangulations. *International Journal for Numerical Methods in Engineering 61*, 1471–1500.

Du, Q. and D. Wang (2005). Anisotropic centroidal voronoi tessellations and their applications. *SIAM Journal of Scientific Computing 26*(3), 737–761.

Feng, Y. T. and D. Peric (2001). A time-adaptive space-time finite element method for incompressible lagrangian flows with free surfaces: Computational issues. *Computer Methods in Applied Mechanics and Engineering 190*(5-7), 499–518.

Feng, Y. T. and D. Peric (2003). A spatially adaptive linear space-time finite element solution procedure for incompressible flows with moving domains. *International Journal for Numerical Methods in Fluids 43*, 1099–1106.

Fortin, M., W. Habashi, M.-G. Vallet, J. Dompierre, Y. Bourgault, and D. Ait-Ali-Yahia (2000, March). Anisotropic mesh adaptation: Towards user-independent, mesh-independent and solver-independent CFD. Part I: General Principles. *International Journal for Numerical Methods in Fluids 32*, 725–744.

Franca, L. P. and S. L. Frey (1992). Stabilized finite element methods: II. the incompressible navier-stokes equations. *Computer Methods in Applied Mechanics and Engineering 99*, 209–233.

Freitag, L. A. (1997). On combining laplacian and optimization-based mesh smoothing techniques. In *Trends in Unstructured Mesh Generation*, Volume AMD-Vol 220. ASME Applied Mechanics Division.

French, D. A. (1999). Continuous Galerkin finite element methods for a forward-backward heat equation. *Numerical Methods for Partial Differential Equations 15*(2), 257–265.

French, D. A. and T. E. Peterson (1996). A continuous space-time finite element method for the wave equation. *Mathematics of Computations 65*(214), 491–506.

Frey, P. J. and F. Alauzet (2005). Anisotropic mesh adaptation for CFD computations. *Computer Methods in Applied Mechanics and Engineering 194*, 5068–5082.

Frey, P. J. and P.-L. George (1999). *Maillages*. Paris: Hermes Science Publications.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides (1995). *Design Patterns. Elements of*

Reusable Object-Oriented Software. Addison-Wesley.

George, P. L. (2001). *Maillage et Adaptation* (Lavoisier ed.). 11, rue Lavoisier, 75008 Paris: Germes Science.

George, P. L. (2003). Back to edge flips in 3 dimensions. In *12th International Meshing Roundtable*, Santa Fe, New Mexico, U.S.A. Sandia National Laboratories.

George, P.-L. and H. Borouchaki (1997). *Triangulation de Delaunay et Maillage*. Paris: Editions HERMES.

George, P. L., H. Borouchaki, and P. Laug (2002). An efficient algorithm for 3d adaptive meshing. *Advances in Engineering Software 33*, 377–387.

Gresho, P. M. and R. L. Sani (1998). *Incompressible Flow and the Finite Element Method. Volume Two Isothermal Laminar Flow*. Wiley.

Gruau, C. and T. Coupez (2005). 3d tetrahedral, unstructured and anisotropic mesh generation with adaptation to natural and multidomain metric. *Computer Methods in Applied Mechanics and Engineering 194*, 4951–4976.

Habashi, W., M. Fortin, D. Ait-Ali-Yahia, S. Boivin, Y. Bourgault, J. Dompierre, M. P. Robichaud, A. Tam, and M.-G. Vallet (1996, August). Anisotropic mesh optimization: Towards a solver-independent and mesh-independent CFD. In *Computational Fluid Dynamics*, VKI Lecture Series 1996–06, Montréal. von Karman Institute for Fluid Dynamics – Concordia University.

Habashi, W., M. Fortin, J. Dompierre, M.-G. Vallet, and Y. Bourgault (1998, May). Certifiable CFD through mesh optimization. *American Institute of Aeronautics and*

Astronautics Journal 36(5), 703–711.

Hand, R. P. and J. Lu (1999). A space-time finite element method for elasto-plastic shock dynamics. *Journal of Sound and Vibration 222*(1), 65–84.

Heckbert, P. S. (1994). *Graphics Gems IV*. The Graphics Gems Series. A Collection of Practical Techniques for the Computer Graphics Programmer. San Francisco, CA, U.S.A.: Morgan Kaufmann (Academic Press).

Hughes, T., W. Liu, and T. Zimmermann (1981). Lagrangian-eulerian finite element formulation for incompressible viscous flows. *Computer Methods in Applied Mechanics and Engineering 66*, 339–349.

Hyun, S. and L.-E. Lindgren (2001). Smoothing and adaptive remeshing schemes for graded element. *Communications in Numerical Methods in Engineering 17*, 1–17.

Jamet, P. (1978). Galerkin-type approximations which are discontinuous in time for the parabolic equations in a variable domain. *Journal of Computational Physics 15*(5), 912–928.

Joe, B. (1993). Construction of k-dimensional delaunay triangulations using local transformations. *SIAM Journal of Scientific Computing 14*(6), 1415–1436.

Johnson, C., U. Nävert, and J. Pitkäranta (1984). Finite element methods for linear hyperbolic problems. *Computer Methods in Applied Mechanics and Engineering 45*, 285–312.

Karakashian, O. and C. Makridakis (1999). A space-time finite element method for the nonlinear schrödinger equation: The continuous galerking method. *SIAM Journal of*

Numerical Analysis 36(6), 1779–1807.

Ladyshenskaya, O. (1969). *The Mathematical Theory of Viscous Incompressible Flows*. Gordon and Breach.

Langtangen, H. P. (1999). *Computational Partial Differential Equations, Numerical Methods and Diffpack Programming*. Lecture Notes in Computational Science and Engineering. Springer.

Legensky, S. M., D. E. Edwards, R. G. Bush, D. M. A. Poirier, C. L. Rumsey, R. R. Cosner, and C. E. Towne (2002). CFD general notation system (CGNS): Status and future directions. In *40th Aerospace Sciences Meeting and Exhibit*, Reno, Nevada, USA. AIAA. AIAA-2002-0752.

Leval, M. R. D., G. Dubini, F. Migliavacca, H. Jalali, G. Camporini, and A. Redington (1996). Use of computational fluid dynamics in the design of surgical procedures: Application to the study of competitive flows in cavopulmonary connections. *Journal of Thoracic Cardiovascular Surgery 111*(3), 502–512.

Li, X., J.-F. Remacle, N. Chevaugeon, and M. S. Shephard (2004). Anisotropic mesh gradation control. In *13th International Meshing Roundtable*, Williamsburg, VA, U.S.A.

Li, X., M. S. Shephard, and M. W. Beall (2005). 3d anisotropic mesh adaptation by mesh modification. *Computer Methods in Applied Mechanics and Engineering 194*, 4915–4950.

Li, X. and N.-E. Wiberg (1998). Implementation and adaptivity of a space-time finite element method for structural dynamics. *Computer Methods in Applied Mechanics and Engineering 156*(1), 211–229.

Li, X.-Y. (2000). *Sliver-Free Three Dimensional Delaunay Mesh Generation*. Ph. D. thesis, University of Illinois at Urbana-Champaign.

Liu, A. and B. Joe (1994). Relationship between tetrahedron shape measures. *Bit 34*, 268–287.

Lohner, R. (2001). *Applied CFD Techniques. An Introduction Based on Finite Element Methods*. 22 Worcester Road, Rexdale, Ontario M9W 1L1, Canada: Wiley.

Masud, A. and T. J. R. Hughes (1997). A space-time Galerkin/Least-squares finite element formulation of the navier-stokes equations for moving domain problems. *Computer Methods in Applied Mechanics and Engineering 146*(1), 91–126.

Mcinerney, T. and D. Terzopoulos (1995). A dynamic finite element surface model for segmentation and tracking in multidimensional medical images with application to cardiac 4d image analysis. *Computerized Medical Imaging and Graphics 19*(1), 69–83.

Meek, D. S. and D. J. Walton (2000). On surface normal and gaussian curvature approximations given data sampled from a smooth surface. *Computer Aided Geometric Design 17*(6), 521–543.

Migliavacca, F., M. R. D. Leval, G. Dubini, and R. Pietrabissa (1996). A computational pulsatile model of the bidirectional cavopulmonary anastomosis: The influence of pulmonary forward flow. *Journal of Biomechanics 118*(4), 520–529.

Miller, G. L., E. P. Steven, and J. W. Noel (2002). Fully incremental 3d delaunay refinement mesh generation. In *11th International Meshing Roundtable*, pp. 75–86. Sandia National Laboratories.

Mitchell, S. A. and S. A. Vavasis (2000). Quality mesh generation in higher dimensions. *SIAM Journal on Computing 29*(4), 1334–1370.

Mittal, S. (2000). On the performance of high aspect ratio elements for incompressible flows. *Computer Methods in Applied Mechanics and Engineering 188*, 269–287.

Mohammadi, B., P. L. George, F. Hecht, and E. Saltel (2000). 3d mesh adaptation by metric control for CFD. In *Revue Européene Des Éléments Finis*, pp. 439–449.

N'dri, D. (2001, Juillet). *Formulation éléments finis espace-temps pour les équations de Navier-Stokes*. Ph. D. thesis, Département de mathématiques et de génie industriel, École Polytechnique de Montréal.

N'dri, D., A. Garon, and A. Fortin (2002). Incompressible navier-stokes computations with stable and stabilized space-time formulations: A comparative study. *Communications in Numerical Methods in Engineering 18*, 495–512.

Nithiarasu, P. and O. C. Zienkiewicz (2000). Adaptive mesh generation for fluid mechanics problems. *International Journal for Numerical Methods in Engineering 47*(1-3), 629–662.

Onate, E. and M. Manzan (1999). A general procedure for deriving stabilized space-time finite element methods for advective-diffusive problems. *International Journal for Numerical Methods in Fluids 31*(1), 203–221.

O'Rourke, J. (1998). *Computational Geometry in C* (Second Edition ed.). New York: Cambridge University Press.

Owen, S. J. and D. R. White (2003). Mesh-based geometry. *International Journal for*

Numerical Methods in Engineering 58(2), 375–395.

Perktold, K., M. Hofer, G. Rappistsch, M. Loew, B. D. Kuban, and M. H. Friedman (1998). Validated computation of physiologic flow in a realistic coronary artery branch. *Journal of Biomechanics 31*(3), 217–229.

Perrochet, P. and P. Azérad (1995). Space-time integrated least-squares: Solving a pure advection equation with a pure diffusion operator. *Journal of Computational Physics 117*, 183–193.

Piper, B. (1987). Visually smooth interpolation with triangular bezier patches. In G. Farin (Ed.), *Geometric Modeling: Algorithms and New Trends*, Philadelphia, U.S.A., pp. 221–233. SIAM.

Plaza, A., J. P. Suarez, M. A. Padron, S. Falcon, and D. Amieiro (2004). Mesh quality improvement and other properties in the four-triangles longest-edge partition. *Computer Aided Geometric Design 21*, 353–369.

Poirier, D., S. R. Allmaras, D. R. McCarthy, M. F. Smith, and F. Y. Enomoto (1998). The CGNS system. In *36th Aerospace Sciences Meeting and Exhibit*, Reno, Nevada, USA. AIAA. http://www.cgns.org/.

Pontaza, J. P. and J. N. Reddy (2004). Space-time coupled spectral/hp least-squares finite element formulation for the incompressible navier-stokes equations. *Journal of Computational Physics 197*, 418–459.

Rajan, V. T. (1994). Optimality of the delaunay triangulation in RD. *Discrete and Computational Geoemetry 12*(2), 189–202.

Reddy, J. N. and D. K. Gartling (1994). *The Finite Element Method in Heat Transfer and Fluid Dynamics*. CRC Press.

Reinhardt, J. M., A. J. Wang, P. T. Weldon, and W. E. Higgins (2000). Note - cue - based segmentation of 4d cardiac image sequences. *Computer Vision and Image Understanding 77*(2), 251–262.

Remacle, J.-F., X. Li, M. S. Shephard, and J. E. Flaherty (2005). Anisotropic adaptive simulation of transient flows using discontinuous galerkin methods. *International Journal for Numerical Methods in Engineering 62*, 899–923.

Réthoré, J., A. Gravouil, and A. Combescure (2005). A combined space-time extended finite element method. *International Journal for Numerical Methods in Engineering 64*, 260–284.

Rivara, M. and N. Hitschfeld (1999). LEPP-delaunay algorithm: A robust tool for producing size-optimal quality triangulations. In *8th International Meshing Roundtable*, South Lake Tahoe, United States.

Roache, P. J. (2002). Code verification by the method of manufactured solutions. *Journal of Fluids Engineering 124*(1), 4–10.

Ruppert, J. (1995). A delaunay refinement algorithm for quality -dimensional mesh generation. *Journal of Algorithms 18*(3), 548–585.

Schroeder, W., K. Martin, and B. Lorensen (1998). *The Visualization Toolkit. An Object-Oriented Approach to 3D Graphics*. Upper Saddle River, NS 07458, U.S.A.: Prentice Hall.

Shakib, F. (1988, November). *Finite Element Analysis of the Compressible Euler and Navier-Stokes Equations*. Ph. D. thesis, Stanford University.

Shakib, F., T. J. R. Hughes, and Z. Johan (1991). A new finite element formulation for computational fluid dynamics: X. the compressible euler and navier-stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 141–219.

Shewchuk, J. R. (1997). *Delaunay Refinement Mesh Generation*. Ph. D. thesis, Carnegie Mellon University.

Shewchuk, J. R. (2000, June). Sweep algorithms for constructing higher-dimensional constrained delaunay triangulations. In *14th Annual Symposium on Computational Geometry*, Hong Kong, pp. 350–359.

Shewchuk, J. R. (2002). Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry Theory and Applications 22*, 21–74.

Tam, A. (1998, April). *An Anisotropic Adaptive Method for the Solution of 3-D Inviscid and Viscous Compressible Flows*. Ph. D. thesis, Concordia University, Montreal, Quebec, Canada.

Tam, A., D. Ait-Ali-Yahia, M. P. Robichaud, M. Moore, V. Kozel, and W. G. Habashi (2000). Anisotropic mesh adaptation for 3d flows on structured and unstructured grids. *Computer Methods in Applied Mechanics and Engineering 189*, 1205–1230.

Tam, A., M. Robichaud, P. Tremblay, W. Habashi, M. Hohmeyer, G. Guevremont, M. Peeters, and P. Germain (1998). A 3-d adaptive anisotropic method for external and internal flows. In *36th Aerospace Sciences Meeting and Exhibit*, Reno, Nevada, USA. AIAA. AIAA 98-0771.

Terdiman, P. (2000). Radix sort revisited. http://codercorner.com/RadixSortRevisited.htm.

Tezduyar, T. E. and M. Behr (1994). A new strategy for finite element computations involving moving boundaries and interfaces – the deforming-spatial-domain / space-time procedure: I. the concept and the preliminary numerical tests. *Computer Methods in Applied Mechanics and Engineering 94*, 339–351.

Tezduyar, T. E., M. Behr, and S. Mittal (1992). A new strategy for finite element computations involving moving boundaries and interfaces – the deforming-spatial-domain / space-time procedure: II. computation of free-surface flows, two-liquid flows, and flows with drifting cylinders. *Computer Methods in Applied Mechanics and Engineering 94*, 353–371.

Thompson, J. F., B. K. Soni, and N. P. Weatherill (Eds.) (1999). *Handbook of Grid Generation*. Washington, D.C.: CFC Press LLC.

Tremblay, P., Y. Bourgault, and S. Tavoularis (2003, June 17-20). Control of discretization error for time-continuous space-time FEM through mesh movement. In K. J. Bathe (Ed.), *Computational Fluid and Solid Mechanics*. Elsevier.

Turek, S. (1999). *Efficient Solvers for Incompressible Flow Problems. An Algorithmic and Computational Approach*, Volume Vol. 6 of *Lecture Notes in Computational Science and Engineering*. Springer.

Vallet, M.-G. (1992). *Génération de maillages éléments finis anisotropes et adaptatifs*. Ph. D. thesis, Université Pierre et Marie Curie, Paris VI, France.

Walton, D. J. and D. S. Meek (1995). Point normal interpolation for stereolithography modelling. *Computers and Graphics 19*(3), 345–353.

Walton, D. J. and D. S. Meek (1996). A triangular g1 patch from boundary curves. *Computer Aided Design 28*(2), 113–123.

Watanabe, Y., N. Yamamoto, and M. T. Nakao (1999). A numerical verification method of solutions for the navier-stokes equations. *Reliable Computing 5*(3), 347–357.

Wesseling, P. (2001). *Principles of Computational Fluid Dynamics*. Springer.

Xu, H. and T. S. Newman (2006). An angle-based optimization approach for 2d finite element mesh smoothing. *Finite Elements in Analysis and Design 42*, 1150–1164.

Xue, D., L. Demkowicz, and C. Baja (2004). Reconstruction of gź surfaces with bi-quadratic patches for h p FE simulations. In *13th International Meshing Roundtable*, Williamburg, VA, U.S.A., pp. 323–332. Sandia National Laboratories.

Yamakawa, S. and K. Shimada (2003). Anisotropic tetrahedral meshing via bubble packing and advancing front. *International Journal for Numerical Methods in Engineering 57*, 1923–1942.

Zienkiewicz, O. C. and J. Z. Zhu (1992, May). The superconvergent patch recovery and a posteriori error estimates. Part II: Error estimates and adaptivity. *33*(7), 1365–1382.

Zwart, P. J., G. D. Raithby, and M. J. Raw (1999). The integrated space-time finite volume method and its application to moving boundary problems. *Journal of Computational Physics 154*, 497–519.