HIERARCHICAL HEXAHEDRAL ELEMENTS FOR FLUID DYNAMIC SIMULATIONS USING STABILIZED FINITE ELEMENT METHODS

By

Anil Kumar Karanam

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE, MECHANICAL ENGINEERING

Approved:

Prof. Kenneth. E. Jansen
Thesis Adviser

Rensselaer Polytechnic Institute
Troy, New York

December 2000
(For Graduation December 2000)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

I would like to express my gratitude to my advisor, Prof. Kenneth Jansen who introduced me to the challenging field of stabilized finite element methods. He was of great help both as a thesis advisor and as a friend. I am also grateful to Dr. Chris Whiting who was always ready to help me with my numerous questions and concerns.

I would like to acknowledge the Scientific Computation Research Center, which has been an incredibly fruitful environment to work in. Not only the computer resources, but also my fellow graduate students and research staff have greatly aided this work. These interactions have enhanced this research, by helping me solve many technical difficulties that have arisen during the course of the work. Special thanks are due to Dr. Farzin Shakib of Acusim software, for providing access to his linear algebra package which was used in all the simulations presented in this thesis.

Finally, I would like to thank my parents for all that they have provided me throughout my educational experience, without which none of this work would have been possible.

Anil Kumar Karanam

ABSTRACT

Stabilized finite element methods have been shown to yield robust, accurate numerical solutions to both the compressible and incompressible Navier-Stokes equations for laminar and turbulent flows. This work presents the development and application of a mesh entity based, hierarchical basis functions for hexahedral elements to a new stabilized finite element formulation, which is shown to yield high accuracy and more cost effective simulations when compared with the traditional, linear basis methods. Some examples of steady incompressible flow are provided that demonstrate this point.

CHAPTER 1
INTRODUCTION AND HISTORICAL REVIEW

Computational fluid dynamics (CFD) has been rapidly gaining popularity over the past several years for technological as well as scientific interests. For many problems of industrial interest, experimental techniques are extremely expensive or even impossible due to the complex nature of the flow configuration. Analytical methods are often useful in studying the basic physics involved in a certain flow problem, however, in many interesting problems, these methods have limited direct applicability. The dramatic increase in computational power over the past several years has led to a heightened interest in numerical simulations as a cost effective method of providing additional flow information, not readily available from experiments, for industrial applications, as well as a complementary tool in the investigation of the fundamental physics of turbulent flows, where analytical solutions have so far been unattainable. It is not expected (or advocated), however, that numerical simulations replace theory or experiment, but that they be used in conjunction with these other methods to provide a more complete understanding of the physical problem at hand. Turbulence researchers are now able to use direct numerical simulation (DNS) to study the basic physics of turbulent flows. Kim *et al.* [36] present an application of DNS to channel flow, and Le *et al.* [38] present a DNS application to flow over a backward-facing step. Both of these studies were conducted to gain new insight into the physical mechanisms involved in turbulent flow.

As computational power grows, the need for more advanced numerical algorithms also increases. There are many different techniques for constructing numerical solutions of fluid flow problems, e.g. finite difference methods, finite volume methods, and finite element methods, to name a few, and all have their strengths and weaknesses. Since the goal of the present research lies in the development of methods which may ultimately be used for large-scale applications of industrial interest, finite element methods have been chosen, given their accuracy as well as their ability to approximate arbitrarily complex geometric configurations. The finite element method applied to fluid dynamics has reached a level of maturity over the past two decades such that it is now being successfully applied to industrial strength problems including turbulent flows (for example, see Haworth and Jansen [20] for an application to reciprocating IC engines). Due to its robustness and proven accuracy, this numerical technique has been chosen for the foundation of the present research.

Hierarchical basis functions for tetrahedral elements have been shown to be effective and accurate for complex flows in Whiting [58] The goal of the present work is to extend the hierarchical basis functions for hexahedral elements, as a means of attaining more accurate and cost-effective finite element simulations of complex flows. It is hoped that this will enable simulations of fluid dynamical problems that are not presently feasible due to current cost restrictions. With these goals in mind, we have chosen a stabilized finite element formulation based on the formulation of Taylor *et al.* [54] for incompressible flows, that has been generalized to accommodate higher-order basis functions. This formulation has been demonstrated to be robust and accurate for laminar as well as turbulent flow simulations using linear basis functions. The new stabilized formulation builds global conservation into the weak formulation that is lacking in many previous formulations due to the presence of the stabilization of the continuity equation. This, combined with the higher-order accuracy that stabilized methods have been shown to attain, has influenced our selection of this formulation for constructing higher-order simulations.

Over the last two decades, stabilized finite element methods have grown in popularity, especially for fluid dynamics applications. Starting with the SUPG method of Brooks and Hughes [7] through the work of Hughes *et al.* [24] on the Galerkin/least squares (GLS) method, and the streamline diffusion method (related to the SUPG method) of Hansbo and Szepessy [19], a number of stabilized formulations have been proposed. Recent work on variational multiscale methods of Hughes [22] and related work on residual-free bubbles by Ruśso [40] and Brezzi *et al.* [6] have not only proposed new directions for these methods, but have also begun to uncover the theoretical basis for their design. Recent application of the variational multiscale method to large eddy simulation of turbulence by Hughes *et al.* [26] has also proven extremely fruitful. A key feature of stabilized methods is that they have been proven (for relevant model problems) to be stable and to attain optimal convergence rates with respect to the interpolation error (see Franca *et al.* [14], and Hughes *et al.* [24]). Johnson and Szepessy [33] have also carried out a nonlinear analysis of the related streamline diffusion method for the Burgers equation. This implies that as the polynomial order of the underlying finite element space is increased, the error in the numerical solution is of the same order as the interpolation error. This property is crucial to the effective use of higher-order basis functions.

Over the past several years, many research groups have applied higher-order discretization methods to fluid dynamics simulations in an effort to achieve highly accurate simulations on unstructured grids. Sherwin and Karniadakis [50] developed a C^0 continuous hierarchical basis based on a generalized tensor product using mixed-weight Jacobi polynomials and applied it to a higher-order splitting scheme for the incompressible Navier-Stokes equations in [52]. They presented numerical results to verify the convergence properties of their method. For Euler flows, the discontinuous Galerkin method provides a straightforward way of constructing higher-order solutions (see Biswas *et al.* [5] and also Devine [12]). Oden *et al.* [39] have recently successfully applied the discontinuous Galerkin method to diffusion type problems using arbitrary polynomial order in each element. Others have generalized spectral methods to unstructured grids to achieve spectral accuracy without being restricted to regular domains (see Carpenter and Gottlieb [9] and Sherwin and Karniadakis [51]). All these methods, however, use the standard Galerkin method for the spatial discretization. Most of these shortcomings have been addressed in [58]. This work attempts to replicate some of the results in that, to demonstrate the effectiveness of higher order basis functions on hexahedral elements.

In the present work, the spatial discretization of the stabilized formulation for the Navier-Stokes equations is carried out using a higher-order, hierarchical basis which is C^0 continuous between finite elements. The hierarchical basis used here is based on the abstract mesh data structure of Beall and Shephard [2], where basis functions are associated with the individual topological entities of the mesh. This type of basis construction was first introduced by Shephard *et al.* [49] (using the basis functions of Carnevali *et al.* [8]) who considered the basis functions to be associated with the mesh entities in a special way. Their mesh entity based hierarchical basis functions support non-uniform k-refinement of meshes of arbitrary element type, e.g. tetrahedral, hexahedral, and pyramidal, by employing an explicit decomposition of shape functions into element blends (discussed in greater detail in chapter 2), ensuring the correct element support and entity level functions, giving the desired polynomial order on an entity. To gain this generality, we have dispensed with the traditional finite element mesh data structures consisting of only element nodal connectivity (see Hughes [21]) in favor of this more general and complete topological adjacency mesh representation. To maintain efficiency on large-scale problems, however, the abstract data structure is only currently used in the pre- and post-processing stages of the simulation, and is therefore not read by the analysis code. A compact data structure will be described that is simple to implement within existing finite element codes, as it represents a relatively straightforward generalization of the traditional data structures. Finally, note that we are using k to refer to the polynomial order of the finite element basis. This is in place of the more standard notation, p, which we reserve for the pressure variable.

Another key aspect of the present research is the use of parallel computers to effectively speed up computations. Finite element calculations are extremely well suited to parallel computing environments since much of the work is in computing element level integrals, and performing sparse matrix-vector products which both parallelize well. Several methods have been proposed which use parallel computers for finite element implementations, see, for example, Bastin [1], Johan and Hughes [30], Kennedy *et al.* [35], Bey *et al.* [4], and Biswas *et al.* [5]. Many of these implementations rely on some high level language, such as CM-Fortran (used by Johan and Hughes [30] as well as Kennedy *et al.* [35]), where interprocessor communication patterns are actually constructed by the compiler, requiring minimal coding effort, however performance is far from optimal (Bastin [1] showed these methods can take up to 15% of total CPU time for communication as opposed to 3% using pre-processed data structures). The current implementation is closely related to that used by Bastin [1], taking advantage of the MPI library for interprocessor communication using “message passing”. The use of message passing for these communications also enables the use of distributed computing environments which are quickly gaining popularity. To enable rapid communication of all information lying on partition boundaries during the analysis, the data structures necessary for parallel communication are pre-processed. This pre-processed data structure contains all the information necessary to carry out the interprocessor communication, which includes hierarchical degree-of-freedom information associated with mesh entities (edges and faces) that lie on the interprocessor boundary, as well as the linear vertex modes. Care has been taken to reduce communication cost by requiring that any pair of processors communicate no more than once.

Numerical simulations of the Navier-Stokes equations (through cubic polynomial order basis) will be presented that verify that nearly optimal convergence rates are observed for problems where analytical results are available. The method will then be applied to more complex (though still laminar) flow simulations which demonstrate a clear advantage of higher-order methods over the traditional, linear basis methods for the incompressible and compressible Navier-Stokes equations. For several of the numerical simulations, a careful cost vs. accuracy study will be conducted to determine the cost-effectiveness of the hierarchical basis. This study will consider the cost with respect to various measures which will quantify where improvements can be made to make the higher-order methods even more cost effective. The results presented will show that for steady problems, cubic basis simulations can be much more cost effective than the standard linear-basis methods.

CHAPTER 2
MESH ENTITY BASED HIERARCHICAL BASIS

The hierarchical basis functions used in the present work are based on the constructions of Shephard *et al.* [49] for specifying variable k-order meshes. These constructions are based on the topological hierarchy of mesh entities (vertices, edges, faces, and regions) which define the finite element mesh. Due to the restrictions of standard finite element data structures consisting only of nodal coordinates and element connectivity, variable k-order finite element meshes must rely on richer structures that allow the independent assignment of polynomial order over the elements as noted by Demkowicz *et al.* [11].This chapter presents a detailed discussion of the finite element basis used in the present work. The description of a new compact mesh data structure that is used to maintain efficiency for large-scale problems will be presented in Chapter 4.

2.1 Abstract mesh data structure

In order to define the finite element basis, we will first introduce the abstract mesh data structure on which the element level basis will be defined (more detail on the mesh data structure used in the present work may be found in the work of Beall and Shephard [2]). The abstract mesh is represented by a data structure (mesh database) that maintains a complete set of adjacency relationships between the various entities in the finite element mesh, known as “mesh entities”. A mesh entity is defined as an individual topological object that is used to define the domain and boundary of a traditional finite element. These entities are of type: region, face, edge, and vertex. The first-order adjacencies between these mesh entities are as follows: a region is bounded by faces, a face is bounded by edges, and an edge is bounded by vertices.

We will refer to the abstract mesh data structure, including the adjacency relationships, as T_M. This mesh database is complemented by a set of functions which support general query operations such as first-order adjacencies (e.g. a function that returns the eight vertices attached to a given hexahedral mesh region), allows arbitrary data to be attached to mesh entities (or geometric model entities), and provides additional functionality. The mesh database is also a powerful tool for many other tasks relating to pre- and post-processing higher-order simulations (e.g. boundary conditions and parallel processing data structures rely heavily on the abstract mesh adjacency representation).

In addition to the mesh entity adjacencies (and their auxiliary functions), the mesh database also maintains a unique relationship between the finite element mesh and the geometric model of the underlying physical domain. This geometric model is represented in terms of “geometric model entities”, in analogy with mesh entities, and similar topological adjacency information is stored. The relationship between mesh and model is known as “classification” and defines the unique model entity that each mesh entity is classified on (more details of mesh-model classification may be found in Beall and Shephard [2]). Mesh-model classification is critical for the assignment of boundary conditions in a mesh-independent manner and greatly simplifies the application of boundary conditions (see Shephard [48]). As part of this work, a graphical user interface (GUI) was developed to enable the assignment of boundary conditions directly to the geometric model entities, which are subsequently inherited by the mesh entities based on their classification (there are generally *many* fewer model entities than mesh entities). Boundary conditions for a simulation are thus assigned without reference to a mesh, therefore, different meshes of the same physical model may be used without re-assigning boundary condition attributes.

In practice, the mesh database is a library of C++ classes that define the various mesh and model objects and have member functions that return the desired adjacency information. The concepts introduced here can be illustrated by the simple example C++ code fragment given in Program 2.1.1. First, the geometric model, `model.dmg`, and the mesh, `mesh.sms`, are loaded (it is presumed that the mesh is classified on this model). Then all regions associated with this mesh are visited and the list of vertices attached to the current region is retrieved. This vertex list may then be processed in any way, for example, coordinates or ID numbers could be collected into an array.

2.2 Finite element basis functions

To proceed with the definition of the element level basis, we first precisely define the finite element. The definition given here is similar to the standard finite element, although additional information is also included. Given the topological description of the

Program 2.1.1 Mesh database example

```
DiscreteModel *model = new DiscreteModel("model.dmg",0);

Mesh *mesh = MM_new(1,model);
M_load(mesh,"mesh.sms");

MRegion *region;
SimpleMeshRegionIter rIter = mesh->firstRegion();
while ( rIter(region) ) {
  SPList<MVertex*>  *vertices  = region->vertices();
  process list of vertices...
}
```

mesh along with its adjacency relationships, T_M, we define:

Definition 2.1 *The closure of a finite element, denoted $\bar{\Omega}_e$, of dimension d_e, is defined as*

$$\bar{\Omega}_e = \{M_e^{d_e}, M_e^{d_e}\{M_j^{d_e-1}\}, \ldots, M_e^{d_e}\{M_j^0\}\}, \tag{2.1}$$

where $M_e^{d_e}$ represents mesh entity e of dimension d_e.

We have followed the notation of Beall and Shephard [2] for the mesh entity adjacencies as

$$M_e^{d_e}\{M_j^{d_j}\} \tag{2.2}$$

which is the j^{th} mesh entity of dimension d_j, bounding mesh entity e of dimension d_e. In other words, a finite element is a mesh region along with its lower order bounding mesh entities. For example, a hexahedral finite element has eight bounding vertices, twelve bounding edges, six bounding faces, and one region. Additional information, such as the direction a face (or edge) is used by a region (or face), is also maintained in the mesh database, and there are functions that return this information.

To construct the discrete, finite element solution, we expand the continuous quantities appearing in the weak form (given in the following chapter) in terms of a C^0 continuous, piecewise polynomial basis defined on each element (as described below). We define this element level basis with the aid of the piecewise polynomial space defined as:

Definition 2.2 *Let $P_k(\bar{\Omega}_e)$ be the piecewise polynomial space, complete to order k, defined on the finite element $\bar{\Omega}_e$.*

The basis for $P_k(\bar{\Omega}_e)$ consists of functions, $N_a(\xi_i),\ a = 1 \ldots n_{es}$, contributed by the mesh entities in $\bar{\Omega}_e$. Here, n_{es} is the number of basis functions contributing to a given element's basis and equals the sum of the number of functions associated with each bounding entity. The polynomial order assigned to each entity is used to compute the number of basis functions it will contribute. The local coordinate system, ξ_i, will be described below. Although the polynomial order may be assigned independently to each mesh entity, it should be noted that the order of complete polynomial representable by a given element's basis will be constrained by the minimum complete order assigned to any of the entities in $\bar{\Omega}_e$, with the exception of vertex modes, which are linear, regardless of the basis order. The direct assignment of the polynomial order to each mesh entity, however, enables a straightforward extension to non-uniform k meshes and meshes of mixed-topology elements, and may also be useful to resolve strong gradients in a pre-determined spatial direction such as boundary layers where strong gradients occur in predictable directions.

2.2.1 Parametric coordinate systems

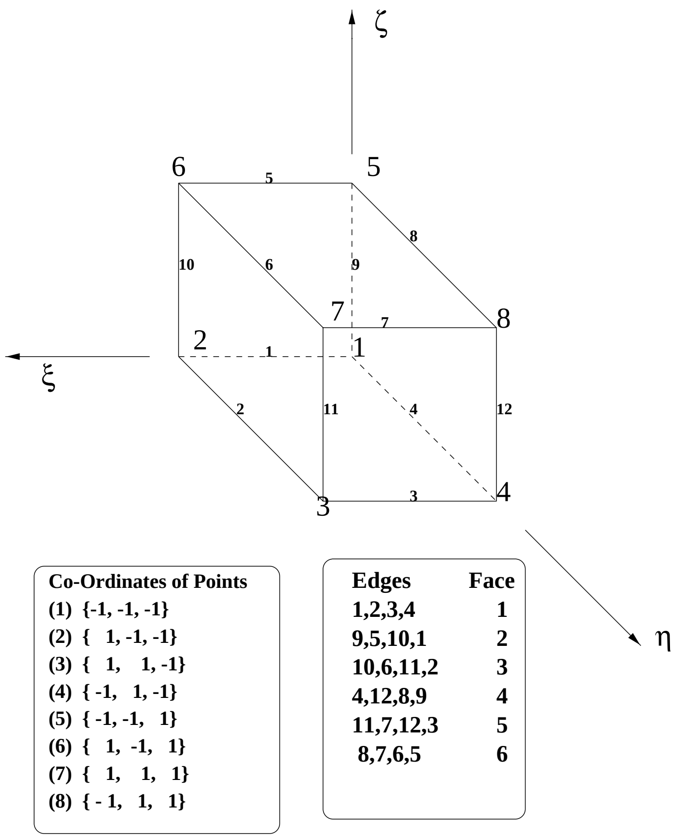
The basis functions are defined in terms of parametric coordinate systems, $\hat{\xi}_i$, associated with the individual mesh entities, as well as, ξ_i, the local coordinate of the element that is using the function. Each edge, face, and region in the finite element mesh has its own local coordinates. These coordinate systems need not be the same for all entities (of a given type) in the mesh, particularly when elements of different topologies are present, which is in contrast to Lagrange basis functions, that are defined solely in terms of a single element coordinate system. Since we will be dealing with meshes composed entirely of hexahedral elements, we will concentrate the discussion on Q-type element coordinate systems (e.g. quadrilaterals in 2D and hexahedra in 3D), more details on coordinate systems useful for different types of element topologies can be found in Dey [13] and also Shephard *et al.* [49]. Figure 2.2.1 shows the co-ordinate system of the template hexahedral element.

Topology	Parametric coordinates
Edge	$-1 \leq \hat{\xi}_1 \leq 1$
Face	$-1 \leq \hat{\xi}_1, \hat{\xi}_2 \leq 1$
Region	$-1 \leq \hat{\xi}_1, \hat{\xi}_2, \hat{\xi}_3 \leq 1$

Table 2.1: Local simplex-type coordinate systems

2.2.2 Description of Shape Functions

Based on the local co-ordinate system described in section 2.2.1 we can define hierarchic shape functions for the template element. Two independent methods exist for generating identical shape functions for this type of element Both of these are based on the Legendre polynomials. The first approach which we refer to as *Classical* [53] generates these functions for a complete template element and each function completely defines the shape function for a particular sub entity on the template element. The shape functions



Co-Ordinates of Points

(1)	{-1, -1, -1}
(2)	{ 1, -1, -1}
(3)	{ 1, 1, -1}
(4)	{ -1, 1, -1}
(5)	{ -1, -1, 1}
(6)	{ 1, -1, 1}
(7)	{ 1, 1, 1}
(8)	{ - 1, 1, 1}

Edges	Face
1,2,3,4	1
9,5,10,1	2
10,6,11,2	3
4,12,8,9	4
11,7,12,3	5
8,7,6,5	6

Figure 2.1: Figure Illustrating the numbering convention and the co-ordinate system used for the hexahedra

obtained by this method are described in detail in section 2.2.3. The other approach is the topology based approach [49] which splits the shape functions into two parts an *Entity level function* which is associated only with the entity irrespective of its location in the element and the *element blend* which multiplies the previous function to generate the complete function for the entity on the element. This strategy is completely discussed in section 2.2.4.

Both these methods are introduced in some detail but the primary focus will be on the second form, which has been implemented here.

2.2.3 Classical Approach

- Shape Functions for the Nodes these are eight in number are identical to the shape functions for a 8 noded tri-linear hexahedron.

$$N_1 = \frac{1}{8}(1-\xi)(1-\eta)(1-\zeta) \tag{2.3}$$
$$N_2 = \frac{1}{8}(1+\xi)(1-\eta)(1-\zeta) \tag{2.4}$$
$$N_3 = \frac{1}{8}(1+\xi)(1+\eta)(1-\zeta) \tag{2.5}$$
$$N_4 = \frac{1}{8}(1-\xi)(1+\eta)(1-\zeta) \tag{2.6}$$
$$N_5 = \frac{1}{8}(1-\xi)(1-\eta)(1+\zeta) \tag{2.7}$$
$$N_6 = \frac{1}{8}(1+\xi)(1-\eta)(1+\zeta) \tag{2.8}$$
$$N_7 = \frac{1}{8}(1+\xi)(1+\eta)(1+\zeta) \tag{2.9}$$
$$N_8 = \frac{1}{8}(1-\xi)(1+\eta)(1+\zeta) \tag{2.10}$$

- Edge modes , there are 12(k-1) edge modes. The edge modes are all listed below The superscript denotes the edge and the subscript give the mode number

$$N_{i-1}^{1,2} = 1/4\phi_i(\xi)(1-\eta)(1-\zeta) \tag{2.11}$$
$$N_{i-1}^{2,3} = 1/4\phi_i(\eta)(1+\xi)(1-\zeta) \tag{2.12}$$

and so on . . .

The subscript i varies from 2 to k. The function $\phi_i()$ is defined as

$$\phi_j(x) = \frac{1}{\sqrt{2(2j-1)}}(P_j(x) - P_{j-2}(x)) \tag{2.13}$$

where $P_j(x)$ are the *Legendre Polynomials* and can be generated using *Bonnet's* recursion formula.

$$\begin{aligned} P_0(x) &= 1 && (2.14)\\ P_1(x) &= x && (2.15)\\ P_2(x) &= \frac{1}{2}(3x^2-1) && (2.16)\\ & && (2.17)\\ (n+1)P_{n+1}(x) &= (2n+1)xP_n(x) - nP_{n-1}(x) && (2.18) \end{aligned}$$

The Legendre Polynomials also satisfy another relationship,

$$(2n+1)P_n(x) = P'_{n+1}(x) - P'_{n-1}(x) \tag{2.19}$$

This allows us to write

$$\phi'_j(x) = \frac{\sqrt{2j-1}}{\sqrt{2}}P_{j-1}(x) \tag{2.20}$$

- Face Modes, there are 3(k-2)(k-3) face modes the shape functions associated with the face modes are as follows. The superscript denotes the face and the subscript spans over the number of shapefunctions contributed by the face.

$$\begin{aligned} N_m^{(1,2,3,4)} &= \frac{1}{2}(1-\zeta)\phi_i(\eta)\phi_j(\xi) && (2.21)\\ i,j &= 2,3,4,5,\ldots,p-2 && (2.22)\\ i+j &= 4,5,\ldots,p && (2.23) \end{aligned}$$

The other face modes are similar and can be written down in the same pattern.

- Interior modes, there are (k-3)(k-4)(k-5)/6 for $k \geq 6$, these modes have not been implemented in the present code as p greater than 5 was not used.

2.2.4 Topology Based Approach

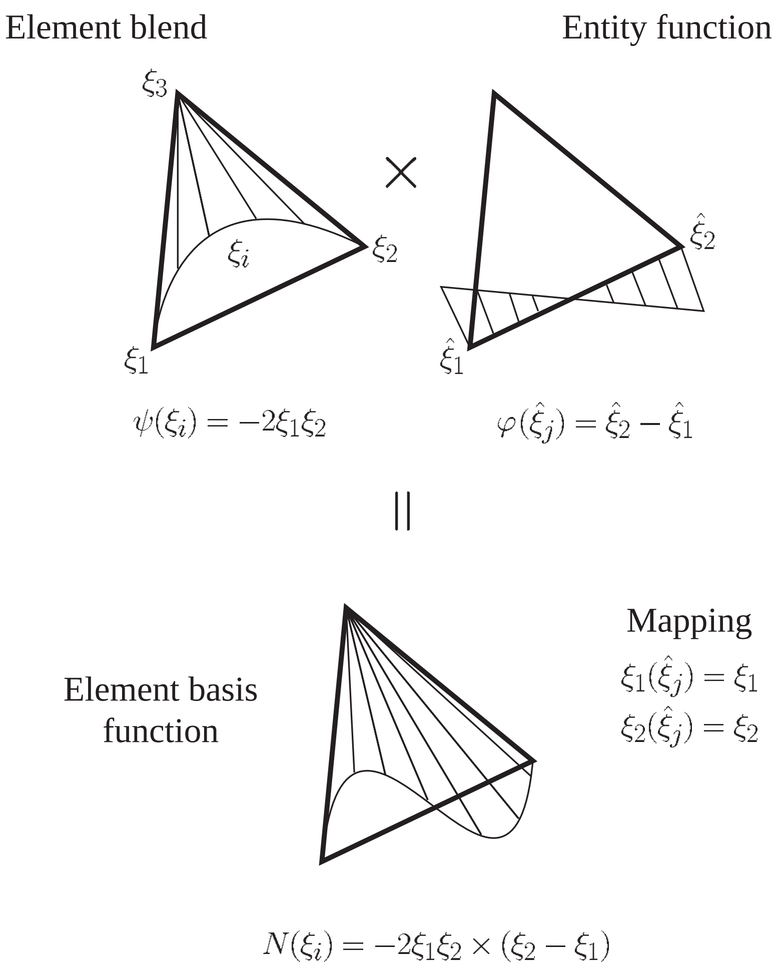
A general methodology has been developed by Shephard *et al.* [49] for the construction of k-version finite element meshes, which is used in the present work. Each basis function (for $k > 1$) is decomposed as

$$N(\xi_i) = \varphi(\xi_i(\hat{\xi}_j)) \times \psi(\xi_i) \tag{2.24}$$

where $\psi(\xi_i)$ is a blending function of fixed polynomial order ensuring that $N(\xi_i)$ has the correct global support, $\varphi(\xi_i(\hat{\xi}_j))$ is an entity level function giving the desired polynomial order on the entity, and $\xi_i(\hat{\xi}_j)$ represents the mapping from entity, $\hat{\xi}_j$, to element, ξ_i, coordinates. This decomposition allows for the efficient implementation of non-uniform k-order meshes as well as the use of meshes with mixed-topology elements. Since the blending function depends only on the element coordinate, it may differ for topologically different elements sharing the same mesh entity (which provides the correct polynomial order behavior, regardless of the topology of the bounding element). The decomposition of a shape function in terms of an entity level function and an element blend is illustrated in Figure 2.2 for a cubic basis function on a triangular element. In Figure 2.2, the element blend is shown in the upper left, and the entity level function (specific to the mesh edge) is shown on the upper right, their product is the resulting (cubic) shape function for the triangular element and is shown on the bottom.

2.2.5 Blending functions

The blending function appearing in Equation 2.24, $\psi(\xi_i)$, depends only on the element, and ensures that each basis function has the correct global support (i.e. it must be zero on lower order mesh entities it does not bound). For hexahedral regions, the blends



Element blend
Entity function
ξ_3
ξ_i
ξ_2
ξ_1
$\hat{\xi}_2$
$\hat{\xi}_1$
$\times$
$\psi(\xi_i) = -2\xi_1\xi_2$
$\varphi(\hat{\xi}_j) = \hat{\xi}_2 - \hat{\xi}_1$
$||$
Mapping
$\xi_1(\hat{\xi}_j) = \xi_1$
$\xi_2(\hat{\xi}_j) = \xi_2$
Element basis function
$N(\xi_i) = -2\xi_1\xi_2 \times (\xi_2 - \xi_1)$

Figure 2.2: Shape function decomposition

will be defined as (see Dey [13]):

$$\psi = \frac{1}{8}(\xi_k^2 - 1)(1 \pm \xi_l)(1 \pm \xi_m) \tag{2.25}$$

$$\psi = \frac{1}{8}(\xi_l^2 - 1)(\xi_m^2 - 1)(1 \pm \xi_k) \tag{2.26}$$

for the edges and faces, respectively. The subscripts are defined by the local vertex ordering in Figure 2.3, for example $\frac{1}{2}(\xi_1^2 - 1)$ is the element blend for the edge between vertices 1 and 2. And $\frac{1}{4}(\xi_1^2 - 1)(\xi_2^2 - 1)$ is the element face blend for the face comprising of vertices $1, 2, 3, 4$. These choices of element blend are not the only possibility and additional ones are explored in Dey [13].

2.2.6 Entity level functions

The entity level function (in Equation 2.24), $\varphi(\hat{\xi}_i)$ provides the desired polynomial order for a given entity's basis function. These functions can be comprised of any set of hierarchical basis functions, and in general are of order ($k - q$), where q is the order of the blend, and k is the desired order. The hierarchical basis functions used here are based on standard Legendre polynomials. The entity level functions are given by the following expressions (see also Dey [13]):

Edge ($k \geq 2$): There is only one shape function of a given polynomial order p. Using the edge parameterization $\zeta \in [-1, 1]$ and using the *Legendre* polynomials, we write

$$(\frac{\zeta^2 - 1}{2})\phi = \sqrt{\frac{2k-1}{2}} \int_{-1}^{\zeta} P_{k-1}(t)dt, \quad k \geq 2 \tag{2.27}$$

where

P_k is the *Legendre* polynomial of order k.

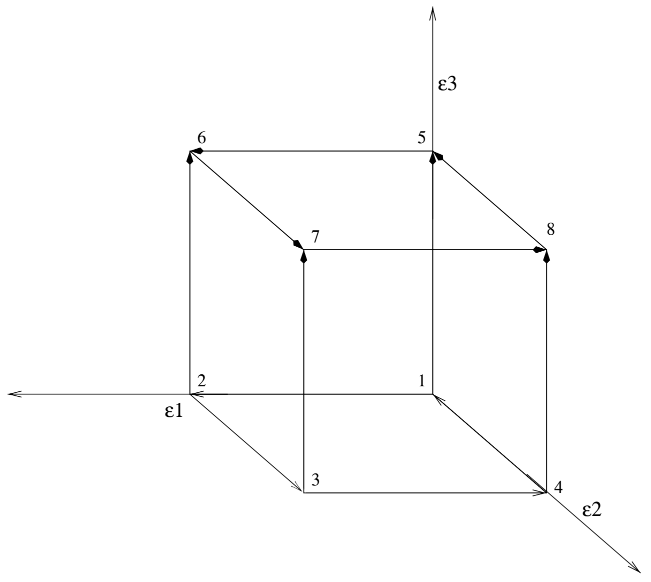


Figure 2.3: Local Q-type vertex ordering and edge direction

The term $(\frac{\zeta^2-1}{2})$ is the edge blend and can be factored from the right hand side of the previous equation.

Face ($k \geq 4$):

For a quadrilateral mesh face, there are a total of $(k-3)$ face functions that contribute to shape functions of polynomial order k. The face functions are obtained as tensor product polynomials of one-dimensional edge functions. They are written as follows

$$\frac{1}{4}[\xi_1^2-1][\xi_2^2-1]\phi = \sqrt{\frac{2\alpha_1-1}{2}}\int_{-1}^{\xi_1} P_{\alpha_1-1}(t)dt\sqrt{\frac{2\alpha_2-1}{2}}\int_{-1}^{\xi_2} P_{\alpha_2-1}(t)dt \tag{2.28}$$

with

$$\alpha_1, \alpha_2 \geq 2 \tag{2.29}$$

$$\alpha_1 + \alpha_2 = k \tag{2.30}$$

and $k \geq 4$. The term $\frac{1}{4}[\xi_1^2-1][\xi_2^2-1]$ is the quadrilateral face blend and can be factored from RHS of the above equation.

Region ($k \geq 6$):

For a hexahedral mesh region, there are a total of $\frac{(k-4)(k-5)}{2}$ shape functions of a polynomial order k, for $k \geq 6$. The entity functions for the region are derived based on the tensor product polynomial basis using the *Legendre* polynomials and are given by

$$\phi = A(\xi_1) \times B(\xi_2) \times C(\xi_3) \tag{2.31}$$

$$A(\xi_1) = \sqrt{\frac{2\alpha_1-1}{2}}\int_{-1}^{\xi_1} P_{\alpha_1-1}(t)dt \tag{2.32}$$

$$B(\xi_2) = \sqrt{\frac{2\alpha_2-1}{2}}\int_{-1}^{\xi_2} P_{\alpha_2-1}(t)dt \tag{2.33}$$

$$C(\xi_3) = \sqrt{\frac{2\alpha_3-1}{2}}\int_{-1}^{\xi_3} P_{\alpha_3-1}(t)dt \tag{2.34}$$

with

$$\alpha_1, \alpha_2, \alpha_3 \geq 2 \tag{2.35}$$

$$\alpha_1 + \alpha_2 + \alpha_3 = k \tag{2.36}$$

To construct element matrices and residual vectors the discrete solution is expanded in terms of these basis functions as

$$\phi^e(\xi_i, t) = \sum_{a=1}^{n_{es}} \phi_a(t) N_a(\xi_i) \tag{2.37}$$

where $\phi^e(\xi_i, t)$ is the finite element approximation of any variable (e.g. pressure or velocity) on element e and $\phi_a(t)$ are the desired coefficients with respect to the basis (since we are using a semi-discrete formulation, the coefficients depend on time).

At this point we would like to point out some important differences between hierarchical and Lagrange basis functions. The main difference is that the hierarchical basis of order k is a subset of the basis of order $k + 1$, i.e. $P_k(\bar{\Omega}_e) \subset P_{k+1}(\bar{\Omega}_e)$. This property greatly simplifies the generation of basis functions, and the varying of polynomial order. Hierarchical and Lagrange basis functions also differ as follows: for a given polynomial order, say N, all functions for the Lagrange basis are of order N, in contrast, the individual hierarchical basis functions will be of different order, however the complete polynomial order is still N. The polynomial order of each of the basis functions for each entity type is discussed in detail in Shephard *et al.* [49]. To get the total (global) number of basis functions, n_s (related to the total number of equations to be solved), we sum over the number of shape functions contributed by each mesh entity for all entities in the mesh. (Note that for a Lagrange basis n_s simply equals the number of "nodal points" in the mesh.) Another key difference is that the hierarchical basis function coefficients do not correspond to solution values at specific spatial locations (as they do for Lagrange elements), they are actually related to higher-order moments of the solution (and its derivatives) on the associated entity. This property makes many routine operations on finite element data more difficult to carry out. For example, post-processing must rely on more advanced techniques when dealing with the higher-order basis functions.

2.3 Chapter summary

This chapter, the hierarchical basis that will be used for the fluid dynamics simulations in this thesis, was presented. The shape function decomposition in terms of entity level function and element blend was discussed in detail, as was the use of the abstract mesh data structure, on which the constructions are based. In the following chapters, the hierarchical basis will be used for numerical simulations of incompressible flow, and its accuracy and cost effectiveness will be demonstrated.

CHAPTER 3
NUMERICAL SOLUTION OF THE NAVIER-STOKES EQUATIONS

This chapter presents the finite element formulation for the Navier-Stokes equations. A semi-discrete finite element formulation that has restored conservation properties is presented which uses the hierarchical basis functions for the spatial discretization. Stabilized finite element formulations have been used by several researchers and have been shown to be robust, accurate, and stable on a variety of examples from steady and unsteady laminar flows to large eddy simulations (LES) and Reynolds averaged simulations of complex turbulent flows (see, for example, Jansen [28], Jansen [27], Tezduyar *et al.* [55], Hughes and Jansen [25], Bastin [1], and Taylor *et al.* [54]). The temporal discretization is based on the generalized-α method of Chung and Hulbert [10], generalized to first-order systems in Jansen *et al.* [29]. This new implicit time integrator is proven to be second-order accurate (on linear model problems) and contains a user specified amount of numerical dissipation.

Stabilized finite element methods have been proven to be stable and higher-order accurate for a linear advective-diffusive system (the closest model problem to the Navier-Stokes equations) in Hughes *et al.*[24], for the linearized incompressible Navier-Stokes equations in Franca and Frey [14], and for a representative nonlinear problem (the Burgers equation) in Johnson and Szepessy [33]. The higher-order accuracy properties, as well as the robustness on complex flows has motivated our choice of finite element formulation. We first present the strong form of the incompressible Navier-Stokes equations, followed by a description of the semi-discrete, stabilized finite element formulation used to discretize the spatial portion of the associated weak form. The generalized-α method time integrator is then introduced to integrate the system of ordinary differential equations resulting from the spatial integration.

Finally, the chapter concludes with a discussion of the numerical evaluation of the diffusive flux terms that appear in the formulation, and methods will be presented for use with both linear and higher-order basis functions.

3.1 Incompressible strong form

Consider the application of the mesh entity based hierarchical basis functions (described in Chapter 2) to the time-dependent, incompressible Navier-Stokes equations. First, consider the strong (or differential) form of the continuity and momentum equations written in the advective form (see Gresho and Sani [18])

$$u_{i,i} = 0 \tag{3.1}$$

$$\dot{u}_i + u_j u_{i,j} = -p_{,i} + \tau_{ij,j} + f_i \tag{3.2}$$

where u_i is the i^{th} component of velocity, p the pressure divided by the density ρ (assumed constant), f_i the prescribed body force, and τ_{ij} the viscous stress tensor given by:

$$\tau_{ij} = \nu(u_{i,j} + u_{j,i}) \tag{3.3}$$

where $\nu = \frac{\mu}{\rho}$ is the kinematic viscosity, and the summation convention is used throughout (sum on repeated indices). We have chosen to write the diffusive terms in the stress-divergence form, which gives rise to a more meaningful set of natural boundary conditions. This system of equations is supplemented with an appropriate set of prescribed boundary conditions on $\Gamma = \partial\Omega$. The incompressible Navier-Stokes equations can be written in many equivalent forms (for the continuous system) which are not necessarily equivalent when discretized. A complete description of the various forms of the equations and the strengths and weaknesses of each, as well as a complete discussion of boundary conditions, are described in the book by Gresho and Sani [18].

3.2 Weak form – Finite element discretization

Finite element methods are based on the weak form (or integral form) of the Navier-Stokes equations (3.1) and (3.2) which is obtained by dotting the entire system from the left by a vector of weight functions and integrating over the spatial domain. The diffusive term, pressure term, and continuity equation are all integrated by parts. The diffusive term is integrated by parts to reduce continuity requirements, otherwise we would have second derivatives on our solution space. The pressure term is integrated by parts to provide symmetry with the continuity equation which is integrated by parts to provide discrete conservation of mass. The consequences of not integrating the pressure term by parts are discussed in detail in Gresho and Sani [18] pages 449–450.

The finite element formulation is based on finite dimensional subspaces of the continuous weight and solution spaces. Recall that $\bar{\Omega} \subset \boldsymbol{R}^N$ represents the closure of the physical spatial domain, $\Omega \cup \Gamma$, in N dimensions; only $N = 3$ is considered. The boundary is decomposed into portions with natural boundary conditions, Γ_h, and essential boundary conditions, Γ_g, i.e., $\Gamma = \Gamma_g \cup \Gamma_h$. In addition, $H^1(\Omega)$ represents the usual Sobolev space of functions with square-integrable values and derivatives on Ω. Subsequently Ω is discretized into n_{el} finite elements, $\bar{\Omega}_e$, as defined above. With this, we can define the discrete trial solution and weight spaces for the semi-discrete formulation as

$$\boldsymbol{\mathcal{S}}_h^k = \{\boldsymbol{v} | \boldsymbol{v}(\cdot, t) \in H^1(\Omega)^N, t \in [0, T], \boldsymbol{v}|_{x \in \bar{\Omega}_e} \in P_k(\bar{\Omega}_e)^N, \boldsymbol{v}(\cdot, t) = \hat{\boldsymbol{g}} \text{ on } \Gamma_g\}, \tag{3.4}$$

$$\boldsymbol{\mathcal{W}}_h^k = \{\boldsymbol{w} | \boldsymbol{w}(\cdot, t) \in H^1(\Omega)^N, t \in [0, T], \boldsymbol{w}|_{x \in \bar{\Omega}_e} \in P_k(\bar{\Omega}_e)^N, \boldsymbol{w}(\cdot, t) = \boldsymbol{0} \text{ on } \Gamma_g\}, \tag{3.5}$$

$$\mathcal{P}_h^k = \{p | p(\cdot, t) \in H^1(\Omega), t \in [0, T], p|_{x \in \bar{\Omega}_e} \in P_k(\bar{\Omega}_e)\} \tag{3.6}$$

where $P_k(\bar{\Omega}_e)$ is as defined in Definition 2.2. Here, $\hat{\boldsymbol{g}}$ represents an approximation to the prescribed boundary condition in the finite element basis. Let us emphasize that the local approximation space, $P_k(\bar{\Omega}_e)$, is the same for both the velocity and pressure variables (although this is not necessary, it is computationally convenient, especially when working with higher-order discretizations). This equal-order interpolation is possible due to the stabilized nature of the formulation to be introduced below, without which, attention must be paid to the Babuška-Brezzi condition. Note that here and throughout, we have omitted the superscript h that is normally included in the discrete representation of the continuous variables, as in $u_i^{(h,k)}$, for notational simplicity. Where there is any chance of confusion, the full notation is retained.

The stabilized formulation used in the present work is based on that described by Taylor *et al.* [54] generalized to include the higher-order basis functions. Given the spaces defined above, we first present the semi-discrete Galerkin finite element formulation applied to the weak form of (3.1) as:

Find $\boldsymbol{u} \in \boldsymbol{\mathcal{S}}_h^k$ and $p \in \mathcal{P}_h^k$ such that

$$B_G(w_i, q; u_i, p) = 0$$

$$\begin{aligned} B_G(w_i, q; u_i, p) = &\int_\Omega \{w_i\left(\dot{u}_i + u_j u_{i,j} - f_i\right) + w_{i,j}\left(-p\delta_{ij} + \tau_{ij}\right) - q_{,i}u_i\}dx \\ &+ \int_{\Gamma_h} \{w_i\left(p\delta_{in} - \tau_{in}\right) + qu_n\}\,ds \end{aligned} \tag{3.7}$$

for all $\boldsymbol{w} \in \boldsymbol{\mathcal{W}}_h^k$ and $q \in \mathcal{P}_h^k$. The boundary integral term arises from the integration by parts and is only carried out over the portion of the domain without essential boundary conditions. Since all the weight coefficients are arbitrary, this gives us a separate equation for each of the i components (and for each of the basis functions). The standard Galerkin method is well known to be unstable for advection-dominated flows (see Brooks and Hughes [7]) and in the diffusion dominated limit for equal-order interpolation of the velocity and pressure, i.e. the Babuška-Brezzi condition. Stabilized methods are well-known to address both of these issues (see Brooks and Hughes [7] and Hughes *et al.* [23], respectively). To remedy both of these situations we add additional stabilization terms as follows:

Find $\boldsymbol{u} \in \boldsymbol{\mathcal{S}}_h^k$ and $p \in \mathcal{P}_h^k$ such that

$$B(w_i, q; u_i, p) = 0$$

$$\begin{aligned} B(w_i, q; u_i, p) = &\, B_G(w_i, q; u_i, p) \\ &+ \sum_{e=1}^{n_{el}} \int_{\bar{\Omega}_e} \{\tau_M(u_j w_{i,j} + q_{,i})\mathcal{L}_i + \tau_C w_{i,i} u_{j,j}\}\,dx \\ &+ \sum_{e=1}^{n_{el}} \int_{\bar{\Omega}_e} \{w_i \overset{\Delta}{u}_j u_{i,j} + \bar{\tau}\overset{\Delta}{u}_j w_{i,j} \overset{\Delta}{u}_k u_{i,k}\}\,dx \end{aligned} \tag{3.8}$$

for all $\boldsymbol{w} \in \boldsymbol{\mathcal{W}}_h^k$ and $q \in \mathcal{P}_h^k$. We have used $\mathcal{L}_i$ to represent the residual of the i^{th}

momentum equation,

$$\mathcal{L}_i = \dot{u}_i + u_j u_{i,j} + p_{,i} - \tau_{ij,j} - f_i \tag{3.9}$$

The second line in the stabilized formulation, (3.8), represents the typical SUPG stabilization added to the Galerkin formulation for the incompressible set of equations (see Franca and Frey [14]). The first term in the third line of (3.8) was introduced by Taylor *et al.* [54] to overcome the lack of momentum conservation introduced as a consequence of the momentum stabilization in the continuity equation. The second term on this line was introduced to stabilize this new advective term. To see that this formulation conserves momentum, set $\boldsymbol{w} = \{1, 0, 0\}$ and $q = u_1$ in (3.8) which leaves only boundary terms if we choose

$$\overset{\Delta}{u}_i = -\tau_M \mathcal{L}_i \tag{3.10}$$

which may be identified with a modified, conservation-restoring, advective velocity. This term must itself be stabilized since it is an advective type term which will lead to advective instabilities. The stabilization parameters for continuity and momentum are defined as

$$\tau_M = \frac{\rho}{\sqrt{c_1/{\Delta_t}^2 + u_i g_{ij} u_j + c_2 \nu^2 g_{ij} g_{ij}}} \tag{3.11}$$

$$\tau_C = \frac{1}{8\tau_M \mathrm{tr}(g_{ij})} \tag{3.12}$$

and the stabilization of the new advective term is defined in direct analogy with the advective portion of τ_M as

$$\bar{\tau} = \frac{\rho}{\sqrt{\overset{\Delta}{u}_i g_{ij} \overset{\Delta}{u}_j}} \tag{3.13}$$

where c_1 and c_2 are defined based on considerations of the one-dimensional, linear advection-diffusion equation using a linear finite element basis and $g_{ij} = \xi_{k,i}\xi_{k,j}$ is the covariant metric tensor related to the mapping from global to element coordinates. This term may be identified with the element length-scale, and is hence a mesh dependent parameter. These stabilization parameters are related to those proposed by Shakib [44] and were also used (in a slightly different form) by Taylor *et al.* [54]. The constant c_2 is a modification for higher-order elements to obtain the correct order of convergence in the diffusive limit as required by the use of the inverse estimates in the accuracy analysis of Franca and Frey [14]. There is some guidance as to how to select this parameter, however, experience has shown the method to be relatively insensitive to its choice. Currently we use $c_2 = 36, 60, 128$ for linear, quadratic, and cubic basis, respectively, for the modification, which has provided good results in all cases presented. The parameter c_1 is related to the temporal portion of the stabilization, and we have selected it to be 4 for most problems.

To derive a discrete system of algebraic equations, the weight functions w_i and q, the solution variables u_i and p, and their time derivatives are expanded in terms of the finite element basis functions (c.f. Equation 2.37). Gauss quadrature of the spatial integrals results in a system of first-order, nonlinear differential-algebraic equations which can be written as

$$\boldsymbol{R}_A(\boldsymbol{u}_i, \dot{\boldsymbol{u}}_i, \boldsymbol{p}) = 0, \quad A = 1 \ldots n_s \tag{3.14}$$

where we have assumed the coefficients of the weight functions to be arbitrary, indicated by the index A, and $\boldsymbol{u}_i, \dot{\boldsymbol{u}}_i$, and $\boldsymbol{p}$ are vectors of the basis coefficients for the discrete representations of these flow variables. The generalized-α method described below is used to solve this nonlinear system in a predictor corrector format utilizing Newton's method.

3.3 Generalized-α time integrator

While several methods have been proposed for the integration of the Navier-Stokes system (both semi-discrete as well as space-time), there has yet to emerge a clear favorite. For example, space-time finite element methods where proposed and analyzed by Shakib *et al.* [45, 46] and expanded and used extensively by Tezduyar *et al.* [56, 57, 31, 32]. Here, as the name implies, the weight and solution space are both given a temporal dependence in addition to the usual spatial dependence. While these methods have yielded very accurate results, the cost has only been justifiable on problems with a moving domain such as free surface flows and/or deforming spatial domains that account for moving solid boundaries. In these cases, the additional cost of space-time methods is put to good use by providing a consistent tracking of the moving boundary.

In cases where the boundary is not moving, semi-discrete methods remain in favor (see Behr *et al.*[3]). Part of the attraction to semi-discrete methods is their long history of use in computational solid dynamics. Many algorithms have been proposed, analyzed and even designed to provide particular behaviors needed in particular conditions. Of particular interest is the behavior of these algorithms in situations where a broad range of temporal scales are present, such as the case of turbulent flows. In this case, the time step is often chosen (out of necessity) such that certain frequencies are only marginally resolved or perhaps even completely unresolved. Given the nonlinearities present in most interesting engineering systems, it is of great importance to ensure that there is temporal damping for frequencies beyond the chosen resolution level. However, it is equally important that this damping not effect the frequencies within the chosen resolution level, leading to a degradation of accuracy (see Jansen *et al.* [29] for a complete analysis of the Generalized-α method).

3.3.1 Generalized-α Method for the Navier-Stokes Equations

In addition to the application of the time integrator to a nonlinear system, the application of the generalized-α method to the Navier-Stokes equations introduces the difficulty of integrating the pressure in time, which has no explicit temporal dependence. This type of a system is technically referred to as a differential-algebraic equation (or DAE), and the theory for integrating such systems is quite involved (see Gresho and Sani [18]). The pressure here is not really integrated in time, it is just iterated to remain consistent with the velocity which is integrated with the generalized-α method. A more complete discussion of such topics may be found in the work of Gresho and Sani [18]. The other primary difficulty in extending the work from the previous section to the full Navier-Stokes equations is the nonlinearity that is introduced. We first recall from Section 3.2 that, once spatially discretized, the momentum and continuity equations may be written in the form:

$$\boldsymbol{R}_A(\boldsymbol{u}_i, \dot{\boldsymbol{u}}_i, \boldsymbol{p}) = 0, \quad A = 1 \dots n_s \tag{3.15}$$

which will be the starting point of the application. Note that this system can also be written as:

$$\begin{pmatrix} \boldsymbol{R}_m \\ \boldsymbol{R}_c \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \tag{3.16}$$

where $\boldsymbol{R}_m$ and $\boldsymbol{R}_c$ represent the residuals of the momentum and continuity equations, respectively.

With these considerations in mind, application of the method discussed in Jansen *et al.* [29] yields the following set of equations describing the time integration algorithm. The first equation is the nonlinear residual with the velocity and acceleration evaluated at the intermediate time steps $t_{n+\alpha_f}$ and $t_{n+\alpha_m}$, respectively

$$\boldsymbol{R}(\boldsymbol{u}_{n+\alpha_f}, \dot{\boldsymbol{u}}_{n+\alpha_m}, p_{n+1}) = 0 \tag{3.17}$$

followed by the update equations relating the velocity to its time derivative,

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + \Delta_t \dot{\boldsymbol{u}}_n + \gamma \Delta_t (\dot{\boldsymbol{u}}_{n+1} - \dot{\boldsymbol{u}}_n) \tag{3.18}$$

and finally the equations that relate the temporal locations n and $n + 1$ to $n + \alpha_m$ and $n + \alpha_f$

$$\dot{\boldsymbol{u}}_{n+\alpha_m} = \dot{\boldsymbol{u}}_n + \alpha_m (\dot{\boldsymbol{u}}_{n+1} - \dot{\boldsymbol{u}}_n) \tag{3.19}$$

$$\boldsymbol{u}_{n+\alpha_f} = \boldsymbol{u}_n + \alpha_f (\boldsymbol{u}_{n+1} - \boldsymbol{u}_n) \tag{3.20}$$

The nonlinearities are best handled by introducing a predictor-multicorrector algorithm similar to those proposed by Brooks and Hughes [7]. By making a prediction of the solution and its time derivative at time t_{n+1}, we start the algorithm. Since we will be making multiple corrections, we introduce a superscript (inside parentheses) to represent the corrector iteration number. In this notation our predictor is initialized with an iteration count of zero and is given by

$$p_{n+1}^{(0)} = p_n \tag{3.21}$$

$$\boldsymbol{u}_{n+1}^{(0)} = \boldsymbol{u}_n \tag{3.22}$$

$$\dot{\boldsymbol{u}}_{n+1}^{(0)} = \frac{\gamma - 1}{\gamma}\dot{\boldsymbol{u}}_n \tag{3.23}$$

where (3.21) and (3.22) predict that the solution will be the same as it was at the previous time step and (3.23) is the time derivative at t_{n+1} that is consistent with (3.18) (i.e. the predictor that preserves second order accuracy). Other choices of predictors are also possible.

After making the prediction, the algorithm enters a loop of multi-corrector passes with i initialized to zero. The first operation within the loop is the calculation of velocity at $t_{n+\alpha_f}$ and the acceleration at $t_{n+\alpha_m}$

$$\boldsymbol{u}_{n+\alpha_f}^{(i)} = \boldsymbol{u}_n + \alpha_f(\boldsymbol{u}_{n+1}^{(i-1)} - \boldsymbol{u}_n) \tag{3.24}$$

$$\dot{\boldsymbol{u}}_{n+\alpha_m}^{(i)} = \dot{\boldsymbol{u}}_n + \alpha_m(\dot{\boldsymbol{u}}_{n+1}^{(i-1)} - \dot{\boldsymbol{u}}_n) \tag{3.25}$$

These quantities enable the evaluation of $\boldsymbol{R}^{(i)}(\boldsymbol{u}_{n+\alpha_f}^{(i)}, \dot{\boldsymbol{u}}_{n+\alpha_m}^{(i)}, p_{n+1}^{(i)})$ which, for small i, can be expected to be far from its desired value of $\mathbf{0}$. To find an improvement to the current values of (3.24) and (3.25) we use a Newton type linearization of $\boldsymbol{R}^{(i)}$ with respect to the acceleration, $\dot{u}_i$, for both the momentum and continuity residuals which yields a matrix problem to solve for the acceleration and pressure increments, given by

$$\begin{pmatrix} \boldsymbol{K}^{(i)} & \boldsymbol{G}^{(i)} \\ \boldsymbol{D}^{(i)} & \boldsymbol{C}^{(i)} \end{pmatrix} \begin{pmatrix} \Delta\dot{\boldsymbol{u}}_{n+1}^{(i)} \\ \Delta\boldsymbol{p}_{n+1}^{(i)} \end{pmatrix} = - \begin{pmatrix} \boldsymbol{R}_m^{(i)} \\ \boldsymbol{R}_c^{(i)} \end{pmatrix} \tag{3.26}$$

which is solved for each corrector pass and the solution is updated according to

$$\dot{\boldsymbol{u}}_{n+1}^{(i+1)} = \dot{\boldsymbol{u}}_{n+1}^{(i)} + \Delta\dot{\boldsymbol{u}}_{n+1}^{(i)} \tag{3.27}$$

$$\boldsymbol{u}_{n+1}^{(i+1)} = \boldsymbol{u}_{n+1}^{(i)} + \gamma\Delta_t\Delta\dot{\boldsymbol{u}}_{n+1}^{(i)} \tag{3.28}$$

$$p_{n+1}^{(i+1)} = p_{n+1}^{(i)} + \Delta p^{(i)} \tag{3.29}$$

and i is incremented. The definition and numerical evaluation of the sub-matrices $\boldsymbol{K}^{(i)}$, $\boldsymbol{G}^{(i)}$, $\boldsymbol{D}^{(i)}$, and $\boldsymbol{C}^{(i)}$ is discussed below. If $i < i_{\max}$ the algorithm returns to solve (3.26) thus initiating the next corrector pass. Otherwise, the solution at time step t_{n+1} is updated, and the algorithm proceeds to the next time step. This completes the step from $t_n \rightarrow t_{n+1}$. If more time steps are required, n is incremented and the algorithm returns to the prediction phase for the next step (i.e. (3.24) and (3.25)). The entire algorithm is summarized in Algorithm 3.3.1.

The linear system of equations, (3.26), is difficult, and special care should be taken in setting up and solving it. The linear algebra solver of Shakib [47] (a highly optimized linear algebra package for the incompressible Navier-Stokes equations) is used to solve this linear system, after it is set up as described below. This linear solver is based on a Generalized Minimum Residual (GMRES, see Shakib [44]) type solution method for the velocity and a conjugate gradient projection method for the pressure. The matrices appearing in (3.26) are the tangent matrices of the residual vectors with respect to the acceleration and pressure at time t_{n+1}, and are defined as follows:

$$\boldsymbol{K}^{(i)} \approx \frac{\partial \boldsymbol{R}_m^{(i)}(\boldsymbol{u}_{n+\alpha_f}^{(i)}, \dot{\boldsymbol{u}}_{n+\alpha_m}^{(i)}, p_{n+1}^{(i)})}{\partial \dot{\boldsymbol{u}}_{n+1}^{(i)}} \tag{3.30}$$

$$\boldsymbol{G}^{(i)} \approx \frac{\partial \boldsymbol{R}_m^{(i)}(\boldsymbol{u}_{n+\alpha_f}^{(i)}, \dot{\boldsymbol{u}}_{n+\alpha_m}^{(i)}, p_{n+1}^{(i)})}{\partial p_{n+1}^{(i)}} \tag{3.31}$$

$$\boldsymbol{D}^{(i)} \approx \frac{\partial \boldsymbol{R}_c^{(i)}(\boldsymbol{u}_{n+\alpha_f}^{(i)}, \dot{\boldsymbol{u}}_{n+\alpha_m}^{(i)}, p_{n+1}^{(i)})}{\partial \dot{\boldsymbol{u}}_{n+1}^{(i)}} \tag{3.32}$$

$$\boldsymbol{C}^{(i)} \approx \frac{\partial \boldsymbol{R}_c^{(i)}(\boldsymbol{u}_{n+\alpha_f}^{(i)}, \dot{\boldsymbol{u}}_{n+\alpha_m}^{(i)}, p_{n+1}^{(i)})}{\partial p_{n+1}^{(i)}} \tag{3.33}$$

The approximation symbols are used here to indicate that these matrices are only approximations to the consistent tangent matrices (given on the right-hand-sides of Equations (3.30)-(3.33)) which have been shown to yield better convergence, and are given in detail below. Care should be taken in computing (3.30) through (3.33), e.g. the differenti-

```
Given solution at time $t_n$: $\boldsymbol{u}_n$, $\dot{\boldsymbol{u}}_n$, and $p_n$

  predict:

    $\boldsymbol{u}_{n+1}^{(0)} = \boldsymbol{u}_n$
    $\dot{\boldsymbol{u}}_{n+1}^{(0)} = \frac{\gamma - 1}{\gamma}\dot{\boldsymbol{u}}_n$
    $p_{n+1}^{(0)} = p_n$

  correct:
    for $i = 1$ to $i_{max}$
        (compute intermediate solution values)

        $\boldsymbol{u}_{n+\alpha_f}^{(i)} = \boldsymbol{u}_n + \alpha_f(\boldsymbol{u}_{n+1}^{(i-1)} - \boldsymbol{u}_n)$
        $\dot{\boldsymbol{u}}_{n+\alpha_m}^{(i)} = \dot{\boldsymbol{u}}_n + \alpha_m(\dot{\boldsymbol{u}}_{n+1}^{(i-1)} - \dot{\boldsymbol{u}}_n)$

        (solve linear system)

        $\begin{pmatrix} \boldsymbol{K}^{(i)} & \boldsymbol{G}^{(i)} \\ \boldsymbol{D}^{(i)} & \boldsymbol{C}^{(i)} \end{pmatrix} \begin{pmatrix} \Delta\dot{\boldsymbol{u}}_{n+1}^{(i)} \\ \Delta\boldsymbol{p}_{n+1}^{(i)} \end{pmatrix} = - \begin{pmatrix} \boldsymbol{R}_m^{(i)} \\ \boldsymbol{R}_c^{(i)} \end{pmatrix}$

        (update solution values)

        $\dot{\boldsymbol{u}}_{n+1}^{(i+1)} = \dot{\boldsymbol{u}}_{n+1}^{(i)} + \Delta\dot{\boldsymbol{u}}_{n+1}^{(i)}$
        $\boldsymbol{u}_{n+1}^{(i+1)} = \boldsymbol{u}_{n+1}^{(i)} + \gamma\Delta_t\Delta\dot{\boldsymbol{u}}_{n+1}^{(i)}$
        $p_{n+1}^{(i+1)} = p_{n+1}^{(i)} + \Delta p^{(i)}$

    end
```

Algorithm 3.3.1: Predictor-multi-corrector algorithm

ation in equation (3.30) gives rise to a mass term since $\dot{\boldsymbol{u}}_{n+\alpha_m}$ is related to $\boldsymbol{u}_{n+\alpha_f}$ through equation (3.18). Since we are differentiating with respect to $\dot{\boldsymbol{u}}_{n+1}^{(i)}$, we also need to use the chain rule, i.e.

$$\begin{aligned}\frac{\partial \boldsymbol{R}_m^{(i)}}{\partial \dot{\boldsymbol{u}}_{n+1}^{(i)}} &= \frac{\partial \boldsymbol{R}_m^{(i)}}{\partial \boldsymbol{u}_{n+\alpha_f}^{(i)}} \frac{\partial \boldsymbol{u}_{n+\alpha_f}^{(i)}}{\partial \dot{\boldsymbol{u}}_{n+1}^{(i)}} + \frac{\partial \boldsymbol{R}_m^{(i)}}{\partial \dot{\boldsymbol{u}}_{n+\alpha_m}^{(i)}} \frac{\partial \dot{\boldsymbol{u}}_{n+\alpha_m}^{(i)}}{\partial \dot{\boldsymbol{u}}_{n+1}^{(i)}} \\ &= \alpha_f \gamma \Delta_t \, \frac{\partial \boldsymbol{R}_m^{(i)}}{\partial \boldsymbol{u}_{n+\alpha_f}^{(i)}} + \alpha_m \, \frac{\partial \boldsymbol{R}_m^{(i)}}{\partial \dot{\boldsymbol{u}}_{n+\alpha_m}^{(i)}}\end{aligned} \tag{3.34}$$

which enables us to directly update our solution to t_{n+1} after the linear solve.

Following the standard finite element assembly process described in Hughes [21], the matrices are formed by evaluating element level integrals (using numerical quadrature). The matrices are given by

$$\begin{aligned}K_{ij}^{ab} = \int_{\bar{\Omega}_e} &\{\alpha_m N_i^a N_j^b + \alpha_f \gamma \Delta_t [\bar{u}_k N_i^a N_{j,k}^b \\ &+ N_{i,k}^a (\mu N_{j,k}^b + \tau_M u_k u_m N_{j,m}^b + \bar{\tau} \mathcal{L}_k \mathcal{L}_m N_{j,m}^b) \\ &+ \mu N_{(i),(j)}^a N_{(j),(i)}^b + \tau_C N_{(i),(i)}^a N_{(j),(j)}^b]\} \; dx\end{aligned} \tag{3.35}$$

$$G_p^{ab} = -\int_{\bar{\Omega}_e} N_{i,i}^a N_p^b \tag{3.36}$$

$$D_i^{ab} = \int_{\bar{\Omega}_e} N_{p,i}^a N_p^b \tag{3.37}$$

$$C^{ab} = -\tau_M \int_{\bar{\Omega}_e} N_{p,i}^a N_{p,i}^b \tag{3.38}$$

where, here, $a, b = 1 \ldots n_{es}$ refer to the individual basis function contributions, the subscripts $i, j = 1 \ldots 3$ are included to indicate the basis functions related to the momentum equations (velocity degrees of freedom) and the subscript p indicates continuity equation (pressure degrees of freedom). Here, indices enclosed in parentheses imply that no sum should be carried out. Since, as mentioned above, we are interpolating velocity and pressure with the same basis functions, the subscripts i and j are only used to indicate the place of these terms in the resulting matrices. The terms we have chosen to include in the tangent matrices given above are essentially formed from the frozen coefficient assumption while differentiating the equations. Assumptions on these matrices also enable the relationship, $\boldsymbol{D} = -\boldsymbol{G}^T$ which is a desirable symmetry property between the discrete divergence and gradient operators ($\boldsymbol{G}$ and $\boldsymbol{D}$, respectively). Gresho and Sani [18] provide additional details pertaining to this symmetry.

3.4 Diffusive flux computation

We would like to conclude the chapter with details of the computation of the diffusive flux terms appearing in the stabilized finite element formulation presented above. Careful inspection of the weak form, (3.8), and in particular the momentum residual equation, (3.9), reveals that it is necessary to calculate the second derivative of the solution variable when evaluating the residual of the diffusive flux stabilization terms (for the incompressible equations)

$$q_i \equiv \tau_{ij,j} = \nu(u_{i,j} + u_{j,i})_{,j} \tag{3.39}$$

While these terms are often neglected for linear basis calculations (with some justification), their inclusion is vital to the accuracy of higher-order simulations (examples run without these terms show a significant degradation of solution quality). It is possible to evaluate these terms directly from the second derivatives of the basis functions, however this involves the evaluation of the second derivative of the mapping if curved elements are used, which is a costly operation. We opt instead for a more efficient method using a local reconstruction of the diffusive flux terms based on an L^2 projection followed by a re-interpolation.

3.4.1 Local, element-level reconstruction

The local reconstruction technique provides a relatively straightforward method to compute an approximation to the diffusive flux involving only element level data. This technique is more cost effective than directly evaluating the second derivatives of the basis functions which involves the second derivative of the geometric mapping for non-straight-sided elements, since the inverse of the element-level projection matrix needs to be computed only one time and stored, and may be used for all subsequent evaluations.

The general idea is to project the viscous stress field, τ_{ij}, (which may be computed with the first derivatives of the basis) onto the element basis, then re-interpolate it with the first derivative of the basis to form the diffusive flux field, i.e. $q_i \equiv \tau_{ij,j}$. The projection is constructed such that the L^2 error is minimized over each element independently, i.e., find $\hat{\tau}_{ij} \in \boldsymbol{\mathcal{S}}_h^k$ such that

$$\int_{\bar{\Omega}_e} w \left(\hat{\tau}_{ij} - \tau_{ij}^{(h,k)} \right) \, dx = 0 \tag{3.40}$$

for all $w \in \boldsymbol{\mathcal{W}}_h^k$, where $\tau_{ij}^{(h,k)}$ represents the current finite element approximation of the stress field. It should be noted that each of the components in $\tau_{ij}^{(h,k)}$ is projected independently. Expanding the weight function in terms of the basis functions yields a system of linear equations to be solved for the basis coefficients of $\hat{\tau}_{ij}$, of the form

$$\boldsymbol{M} \hat{\boldsymbol{\tau}}_{ij} = \boldsymbol{R}_{ij} \tag{3.41}$$

where,

$$\boldsymbol{M} = [\boldsymbol{M}_{ab}] = \int_{\bar{\Omega}_e} N_a N_b \, dx, \quad \boldsymbol{R} = \{\boldsymbol{R}_a\} = \int_{\bar{\Omega}_e} N_a \tau_{ij}^{(h,k)} \, dx \tag{3.42}$$

This system is solved for the stress projection coefficients, $\hat{\boldsymbol{\tau}}_{ij} = \{\hat{\tau}_{ij}^a\}$, for each element, which are then re-interpolated with the gradients of the basis functions to form an approximation to q_i as

$$q_i = \sum_{a=1}^{n_{es}} N_{a,j} \hat{\tau}_{ij}^a . \tag{3.43}$$

The system of equations, 3.41, is inverted one time and stored, therefore the evaluation of the projection coefficients involves a single integral evaluation, which is computed using the same Gauss quadrature rule as the integration of the residual.

The inclusion of this term is vital to the performance of higher-order methods, since without it, the formulation no longer maintains its weighted residual character, i.e., consistency is violated.

3.5 Chapter summary

This chapter introduced the stabilized finite element formulation for the incompressible Navier-Stokes equations using mesh entity based hierarchical basis functions for the spatial discretization. The implementation involves relatively few modifications to a highly efficient linear basis solver, allowing us to maintain efficiency for large-scale problems. To achieve this goal, much of the computational effort has been transferred to the pre-processing stage of the analysis, where the data structures are created and written to disk for high efficiency when used by the flow solver. One key difference between linear and higher-order basis functions is the treatment of the diffusive portion of the residual in the stabilization terms, a problem unique to stabilized methods. A method for dealing with this term was presented for higher-order computations (local reconstruction). The next chapter will describe many implementational details that are encountered when using hierarchical basis methods for fluid dynamics.

CHAPTER 4
PRE-PROCESSING HIGHER ORDER SIMULATIONS

4.1 Introduction

Large scale finite element simulations require boundary condition information, parallel communication data structures, and higher order degree-of-freedom information to be preprocessed for rapid retrieval during the computation phase of the analysis. This is critical for achieving near perfect scalability and short turnaround times. Although this is not always possible, (e.g. dynamic mesh adaptivity during computation will require data structure modification during simulations), when it is, it represents a substantial decrease in computation time. Many large scale computations of laminar and turbulent flows, do meet the requirements for preprocessed data structures, and the additional speed gained is often a necessity to acquire a solution in a reasonable time-frame.

The preprocessing method described here is based on recent developments in abstract mesh representation and classification against a geometric model. Mesh-model classification (see Beall and Shephard [2]) provides a connection between the finite element mesh, which may be varied over the course of several analysis runs, and the geometric model of the physical domain, which remains fixed. Given this relationship, boundary condition attributes are associated with the geometric model the topological description of the problem domain, rather than the finite element mesh, thus enabling a more intuitive approach to their application [48]. The finite element mesh then inherits boundary condition attributes from the geometric model . The preprocessor interacts with the geometric model, as well as the finite element mesh, to pre-compute the element degree-of-freedom (*dof*) connectivity information (including information necessary for higher order computations), boundary condition arrays, and parallel communication data structures.

Essential and natural boundary condition attributes may be quite complex for fluid dynamics simulations and require substantial preprocessing for accurate specification. In addition, periodic boundary conditions are handled differently than other essential boundary conditions and pose additional difficulties. A method is presented for application of general sets of boundary conditions to the geometric model, which are conferred to the finite element mesh during preprocessing. Classical finite element procedures traditionally associate boundary conditions directly with the nodal degrees of freedom which requires that the boundary conditions be set up for each mesh, even when the geometric model is unchanged. In addition, higher-order computations rely heavily on the additional topology of the mesh (e.g. mesh edges and faces), information which is not readily accessible from the classical data structures. To address these issues, the boundary conditions are instead applied directly to the geometric model (no mesh even needs to exist at this stage) and the preprocessor reads the model and the mesh and associates the model information with the mesh. By preprocessing boundary condition information, the analysis code has no need to make expensive geometric model queries and therefore, can rapidly constrain the required *dofs*.

The increasing complexity of fluid dynamic simulations has lead to the heavy use of parallel computers. While the solvers themselves can be trivially parallelized and have been shown to yield near perfect scaling on large problems , a non trivial, extensive effort goes into the preprocessing of these parallel data structures and communication traces which make the parallelism almost transparent to the solver.

In this chapter we attempt to address the most critical aspects required for preprocessing higher order simulations. First we introduce the idea of topological mesh model hierarchy. Then we discuss the concepts used in applying boundary and initial conditions efficiently. At this point we also present the compact data structure which holds all the higher order information and is provided as input for the solver code. Following that, we briefly describe the additions and modifications required for parallel processing.

4.2 Topological mesh-model hierarchy

Finite element meshs have traditionally been described in terms of nodal coordinates and element connectivity. This representation has been dispensed in favor of a richer topological data structure [2]. This data structure or database, described in section 2.1 maintains information related to all mesh entities, vertices, edges, faces, and regions as well as adjacency relationships between them. The geometric model may be similarly defined in terms of model entities of the same type. The i^{th} mesh entity of dimension d_i will be denoted by $M_i^{d_i}$ similarly $G_i^{d_i}$ for a model entity. For linear basis computations this information is clearly more than necessary, however, preprocessing higher-order computations requires this . We have therefore confined the use of the entire mesh data structure to the preprocessing stage, and only the traditional finite element structures (co-ordinates and connectivity generalized to support higher order *dof* information) are used within the analysis code. This means that the traditional structures are created during preprocessing, and written to disk to be read by the analysis code. This rich topological data structure is identical to the traditional format, with the exception that information on higher order degress of freedom is included.

Central to the method of preprocessing described here is the idea of the classification of the finite element mesh on the geometric model. Not only is the preprocessor designed around this concept but every stage in the preprocessor makes extensive use of the cross referencing capability provided by this idea.

Mesh-model classification defines the relationship between the finite element mesh and the physical domain (geometric model) on which the problem is to be solved and is key to the development of geometry based boundary condition specification. Beall and Shephard [2] define *mesh classification against the geometric domain* as the unique association of a mesh entity of dimension d_i, $M_i^{d_i}$ to a geometric model entity of dimension d_j, $G_j^{d_j}$ where $d_i \leq d_j$, denoted $M_i^{d_i} \sqsubset G_j^{d_j}$. The classification symbol $\sqsubset$ indicates that the left-hand entity is classified on the right-hand entity.

For example a mesh vertex can be classified on a model vertex, model edge, model face or model region. Where as a mesh edge cannot be classified on model vertex (an entity of lower order than itself), similarly mesh faces cannot be classified on model edges or vertices and mesh regions can be classified only on model regions. This unique classification can be used to transfer the desired boundary condition information from the model entities to the mesh entities that are classified on them. In the remainder of the document geometric model entities are referred as model entities and the finite element mesh entities are referred to as mesh entities.

4.3 Boundary and Initial conditions

A central part of the finite element formulation is the accurate specification of boundary conditions, which help define the physical problem being solved. Furthermore initial conditions affect the time it takes to reach a steady solution, when that is the goal or strongly influence the solution when solving unsteady problems.

When solving differential equations, generally we have 3 kinds of boundary conditions. First there are essential or Dirichlet boundary conditions, which constrain the solution variables, on specified sections of the boundary. Secondly there are natural or Neumann boundary conditions, which constrain the fluxes (which are dependent on the derivatives) of the variables. The third type of boundary condition involves periodicity in which case the solution repeates itself after a certain spatial interval, in a given direction. We can take advantage of this fact by equating the solution variables at the ends of the periodic spatial interval. This periodic interval can be either a translation, rotation or both.

For Navier-Stokes equations essential boundary conditions can be applied to all of the solution variables, pressure, three components of velocity and temperature. There are two specific ways of specifying velocity essential boundary conditions. The first one constrains one component of velocity while leaving the other components free. This will be referred to as *one_component* boundary conditions. The velocity components in the plane perpendicular to this vector are free. The second way completely specifies the complete velocity vector, thus all components are constrained. This will be referred to as *three_component* boundary conditions. When only two components are prescribed, this can be done with the application of two one component conditions. Finally natural boundary conditions are applied in the form of mass or heat fluxes, traction on surfaces and pressure specified over a surface in a weak or integral sense.

4.3.1 Inheritance of boundary conditions

The ability to transfer boundary condition data attached to a topological model, to the finite element mesh is one of the most important attributes of this preprocessing method. This capability is made possible by the mesh-model hierarchy discussed in section 4.2.

While the mesh-model classification saves us from having to set boundary conditions on individual mesh entities, for complicated models even the model entities could run into the hundreds. To address this problem we introduce the concept of *boundary condition inheritance* . This allows for the boundary conditions to be set on the highest possible model entities (usually faces). The preprocessor uses a predefined set of rules to inherit the boundary condition attributes from the faces to the other lower order entities (edges and vertices). Setting boundary conditions on model faces is usually all that is required , but occasionally the user might need to specify boundary conditions on a small number of edges and vertices to correct for any known conditions of conflict of the inheritance.

The task of transferring the boundary condition values from model to mesh entities is performed by using the classification information provided by the mesh data structure. For instance if a particular value of pressure is set on a model face, the preprocessor iterates over all the mesh entities classified on this model face and applies the pressure boundary condition to them. There is no scope for any sort of conflict since each mesh entity is uniquely classified on a single model entity. On the other hand the transfer of information from the model faces (on the boundary conditions were actually specified) to the lower model entities (edges and vertices) is usually more involved. In the topological geometric model, edges are formed by the intersection of 2 model faces and vertices are defined at locations where at least two model edges meet. When deriving boundary conditions from multiple neighbors there could both resolvable and unresolvable conflicts. For instance when two different directions are specified for the *one_component* velocity on 2 neighboring faces of a model edge, both directions are preserved and 2 components are constrained instead of one. Where as, if the conflict occurs in the magnitudes only, then the result is unpredictable and it is suggested that velocity boundary condition be explicitly set on the edge overriding any possible inheritance.

The following general methodology is used to implement the boundary condition inheritance from higher order to lower order model entities. The preprocessor iterates over all the model entities, faces followed by edges and finally vertices. For each entity, if any boundary condition attribute is explicitly specified, that value is given precedence and used as it is (usually the case for model faces). When nothing is explicitly specified on an entity, the boundary condition attributes are derived from the higher order entities intersecting to create it. All the immediate higher order entities in contact with the current one are visited and the value of the attribute in the current entity is derived as a combination of all the surrounding higher entities. For instance a model edge will derive its boundary condition data from the model faces intersecting to create the edge. Similarly a vertex will derive its information from the model edges co-incident on it.

The above described method of **boundary condition inheritance** can be applied to both essential and periodic class of boundary conditions. It should be noted that the natural boundary conditions are defined only for model faces and are not derived for edges and vertices since they do not make physical sense for edges and vertices on which the formulation does not evaluate boundary integrals.

4.3.2 Boundary and initial conditions for hierarchic basis

The boundary condition co-efficients are calculated at the preprocessing stage where the rich mesh model classification structure is available and then the evaluated values are written out in the traditional form and can be efficiently applied at the solver stage. This also enables us to set boundary conditions on the higher order modes attached to mesh edges and faces.

Higher order basis functions using Lagrange basis functions enforce essential boundary conditions in a relatively straight forward manner, as the basis coefficients correspond to solution values at nodes , *vis.* the Lagrange interpolation equation $N_a(\xi_b) = \delta_{ab}$. Since the solution coefficients with respect to the hierarchical basis do not correspond to solution values at spatial locations, work must be done to determine the coefficients on the boundaries. To accomplish this, we interpolate the known Dirichlet boundary function with the hierarchical basis by solving a linear system of equations for the unknown basis coefficients. Additionally, a unique set of interpolation points must be chosen since there are no particular spatial locations associated with the higher-order coefficients.

The element level interpolation may be constructed by solving a linear system of equations for the coefficients on each element in the domain. Suppose we wish to specify that $\phi(x_i) = g(x_i)$ over some portion of the boundary, where $\phi(x_i)$ could be any of our solution variables. (note that this process may be extended to include initial conditions, in which case we seek an approximation over the entire domain, not just the boundary face.) We can find the coefficients of an approximation to $g(x_i)$ (for each element, e) as

$$g(x_i) \approx \hat{g}^e(x_i) = \sum_{a=1}^{n_{ip}} g_a^e N_a^e(x_i) \tag{4.1}$$

where N_a^e are the basis functions for element e, g_a^e are the unknown coefficients, and n_{ip} is the number of interpolation points, which must equal n_{es}, the number of element shape functions.Equation (4.1) can be expressed in the following matrix system.

$$\boldsymbol{M}\boldsymbol{g} = \boldsymbol{R} \tag{4.2}$$

$$\boldsymbol{M} = [M_{ab}] = N_a^e(\boldsymbol{\xi}_b^{int}), \text{ and } \boldsymbol{R} = [\boldsymbol{R}_b] = g(\boldsymbol{x}(\boldsymbol{\xi}_b^{int})) \tag{4.3}$$

where $\boldsymbol{\xi}_b^{int}$ is the b^{th} interpolation point (in element e's coordinates). This system of linear equations is solved for the basis coefficients, g_a^e, which are used when needed by the analysis code to evaluate $\phi(x_i)$ (which is expanded in the same basis as $\hat{g}(x_i)$). Since we are using an element level interpolation (which only couples local degrees of freedom) the resulting interpolation is not guaranteed to be continuous between elements. One solution to this problem is to average the coefficients, which is done in the present work. Another approach is to assemble the data to global arrays and solve a global problem, however the additional cost is not deemed worth the effort, as averaging has proven to work well for all cases we have considered. For computations using the Lagrange basis, the interpolation points are simply the nodal coordinates, and the matrix in (4.2) is the identity matrix. Furthermore, the system of equations described above may be simplified somewhat by statically condensing the coefficients since not all functions are coupled. In practice, however, the interpolation is only computed during pre-processing, making the time savings less significant.

As was noted earlier, the procedure described above for essential boundary conditions may also used to set an initial condition in cases where the exact initial condition is relevant to the simulation. However, experience has shown that in cases where such accuracy is not necessary (as is usually the case), using the linear interpolation of the initial conditions is sufficient to ensure convergence. The linear interpolation is obtained by simply setting all higher-order coefficients equal to zero.

4.3.3 Periodic boundary conditions

The application of periodic boundary conditions poses several difficulties in the context of hierarchical basis functions since all mesh entities must be identical on periodic planes (including edge and face directions). A general methodology has been developed for the application of periodic boundary conditions. The data necessary to enforce periodic boundary conditions can be contained in a single array which specifies the "periodic master" of each mesh entity. When essential boundary conditions are set, periodic boundary conditions are also set by copying the solution coefficients of the periodic masters to their periodic slaves. This operation is simply an indirect address of the solution array using the periodic boundary conditions array. The equations corresponding to the periodic entities are eliminated from the system by using this array to zero the corresponding residual components.

4.4 Compact data structure

To start with the traditional finite element mesh structure providing coordinates and connectivity is no longer sufficient. All the edges in the mesh pick up modes for $p \geq 2$, triangular faces pick up modes for $p \geq 3$ and quadrilateral faces for $p \geq 4$, in addition, different element topologies start having region modes associated with them starting at different polynomial orders. Here we exploit the rich data structure provided by mesh data base as described in section(4.2) to generate all the higher order mode and entity information and condense it into the compact data structure used by the solver code. Also all the information pertaining to any entity (such as a vertex, an edge, a face or a region) are kept local. This information includes the number of modes on the entity, its processor adjacency information, global equation number and its local polynomial order. This increases the memory requirements somewhat but simplifies the process of applying boundary conditions and generating parallel communication traces.

The first step in setting up the data structures is the assignment of global equation numbers to all mesh entities. This is done by visiting each entity, determining the number of shape functions it contributes based on its polynomial order and assigning a unique equation number for each of these functions. Next all the elements in the mesh are visited and the equation numbers associated with the lower order entities bound by it are collected. For example a tetrahedral region may collect equation numbers from 4 vertices, 6 edges and 4 faces and the region itself if the polynomial order is greater than 3. This procedure is similar to that described by Hughes [21] for meshes of Lagrange elements where the global node numbers associated with each finite element are stored in the data structure. For hierarchical basis functions, additional information needs to be maintained for $p > 2$ which emanates from the mapping from entity to element.

This connectivity information provides a complete description of the mapping between the element level computations and the global degrees of freedom (where the linear equations are formed and solved). For hierarchical basis functions of degree 3 and higher some of the basis functions need to have their sign reversed since the mapping from the entity to the element coordinate system introduces a sign change for some of the bounding elements. The situation is illustrated by a simple example shown in Figure 4.1

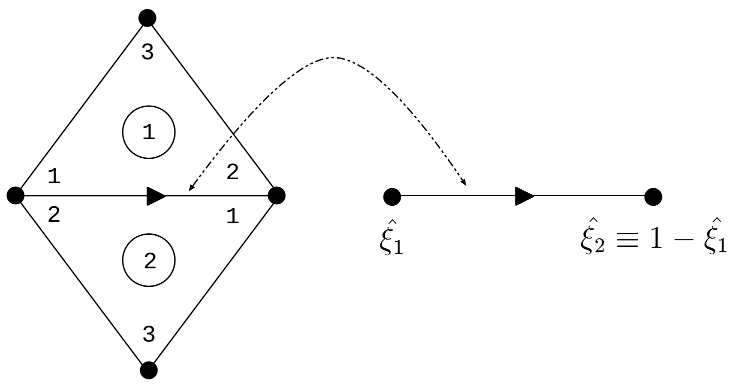


Figure 4.1: Mesh elements illustrating the reversal of shape functions

In figure 4.1 a two dimensional case is shown in which two triangular mesh faces share a common edge; we will consider each face as an element (this is also applicable to quadrilateral mesh faces, except for the actual expressions for the shape functions and generalizes to regions in three dimensions). This figure shows the element numbers in circles, as well as each local degree of freedom number with respect to each of the elements. The edge is also shown along with its local coordinate system, $\hat{\xi}_1$, which is directed as indicated by the arrow, note that the global direction of the edge is determined by the vertex ordering stored in the mesh database. The local coordinates of the edge must be mapped to the coordinate system of each bounding element in order to evaluate the function. With the data structure described here, it is possible to evaluate a single set of element shape functions to be used for all elements in the mesh. This enables the basis functions to be pre-computed and tabulated for each quadrature point, as commonly done for Lagrange-type elements.

Returning to the example, suppose $k = 3$ has been set on the edge depicted in figure 4.1. It will therefore contribute two functions, one quadratic and one cubic, to the local basis of each of the two bounding triangular elements (see [13] for a description of the shape functions) given by

$$N_2(\hat{\xi}_i) = -2\hat{\xi}_1\hat{\xi}_2 \tag{4.4}$$

$$N_3(\hat{\xi}_i) = -2\hat{\xi}_1\hat{\xi}_2(\hat{\xi}_2 - \hat{\xi}_1) \tag{4.5}$$

where the parametric coordinates for this edge are

$$\hat{\xi}_1 \quad \text{and} \quad \hat{\xi}_2 \equiv 1 - \hat{\xi}_1 \tag{4.6}$$

and the subscripts on the basis functions refer to their respective polynomial orders. When the coordinates are mapped from the edge to the element coordinates, the problem becomes apparent, i.e.

Element 1:

$$\xi_1 = \hat{\xi}_1, \qquad \xi_2 = \hat{\xi}_2 \tag{4.7}$$

Element 2:

$$\xi_1 = \hat{\xi}_2, \qquad \xi_2 = \hat{\xi}_1 \tag{4.8}$$

and the basis functions become:

Quadratic:

$$N_2^{(1)} = -2\xi_1\xi_2 \tag{4.9}$$

$$\begin{aligned} N_2^{(2)} &= -2\xi_2\xi_1 \\ &= N_2^{(1)} \end{aligned} \tag{4.10}$$

Cubic:

$$N_3^{(1)} = -2\xi_1\xi_2(\xi_2 - \xi_1) \tag{4.11}$$

$$\begin{aligned} N_3^{(2)} &= -2\xi_2\xi_1(\xi_1 - \xi_2) \\ &= -N_3^{(1)} \end{aligned} \tag{4.12}$$

where the superscript indicates the element that the function is associated with. The cubic function on element 2 is the negative of that on element 1, while the quadratic function is the same, regardless of the edge direction. This case generally occurs when an element uses an edge in the opposite direction than that edge was defined. Since the cubic edge function is different for each of the bounding elements, a single set of basis functions will clearly not suffice to completely describe the basis. To overcome this difficulty, during pre-processing the local degree of freedom numbers that correspond to shape functions that must be negated are flagged (e.g. there equation numbers are negated). This information is then used in the flow solver to create the correct element basis functions from the pre-computed table of element functions. For quadratic or linear basis, no functions need to be negated, and the data structures may be used as they are. This also implies that when using the hierarchical code with linear elements, no significant penalty is paid for having the generality of higher-order basis functions in the same code.

4.5 Parallel communications

Finite element methods are extremely well suited to use on parallel computers as much of the computational effort is in the calculation of element level integrals. To reduce the computational effort during the analysis phase, the structures specifying the interprocessor communications are pre-processed. Each processor then executes a copy of the analysis code, reading the pre-processed input data relating to its portion of the domain, as well as information relating to other processors it must communicate with. This section describes the information that processors must communicate to each other as well as the construction of these data structures.

4.5.1 Mesh partitioning

For a parallel simulation to be efficient, the load distribution on all the processors should be balanced at all times. This helps avoid some processors lying idle and some working too hard. For computations which depend on adaptive refinement, various strategies of dynamic load balancing and migration have been developed and have to be deployed at the solver stage. But for computations not depending on dynamic adaptivity all the load balancing and partitioning can be done at the pre-processing stage. This involves partitioning the mesh and assigning local equation numbers for all the entities on every partition.

Mesh partitioning can be done in many ways to suit specific needs. We have chosen 2 popular strategies suitable for our needs.

Inertial Partitioning: This is the method where partition boundaries are chosen based on geometrical considerations. The users specify criterion used to divide the mesh into partitions. While this method is not ideal for achieving optimal interface length or perfect load balancing in a generic case, this gives the user a greater control over the description of the partitions. This can prove very helpful in some special cases where specific sections of the mesh have to be on pre-determined partitions. An example for inertial partitioning can be seen in figure 4.2.

Dual Based Partitioning: This involves partitioning the dual graph of the mesh using various graph partitioning methods. We have used the generic graph partitioning package, *METIS* [42], which provides extensive opportunities for customization and is known to produce partitions with good load balancing and optimal interface length. An example of a dual based partition can be seen in figure 4.3

Each processor maintains a complete collection of data representing its portion of the finite element mesh and analysis information. This includes dof numbers and connectivity information as well as boundary condition data for all nodes that are physically on

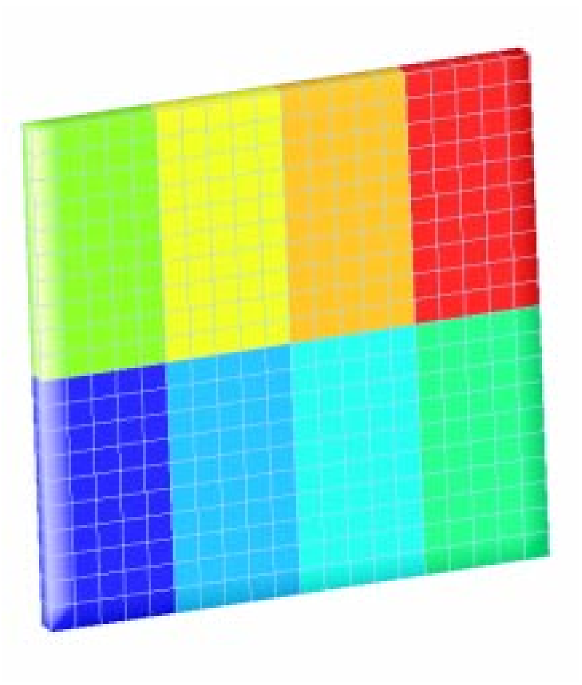


Figure 4.2: Inertial Partitioning

the processor or its interprocessor boundary. The finite elements are uniquely partitioned among the processors, so each element will be found on only one processor. The mesh entities that contribute to element level integrals, however, appear in multiple partitions if they are on an interprocessor boundary. Element level computations are performed completely local to each processor and must communicate only when the element level contributions,such as element integrals, are assembled to the global equations. This global assembly procedure involves the sending and receiving of *dof* information between processors. Another case in which processors must exchange information is when periodic boundary conditions are present with periodic partners residing on different processors. These two types of communications will be described below.

4.5.2 Communication

The basic idea behind the parallel communication of finite element information can be described with the aid of figure 4.4. This figure illustrates three processors and vertices on the interprocessor boundaries. For simplicity, only vertices are shown since edges and faces are handled identically. The element residuals associated with each vertex are first

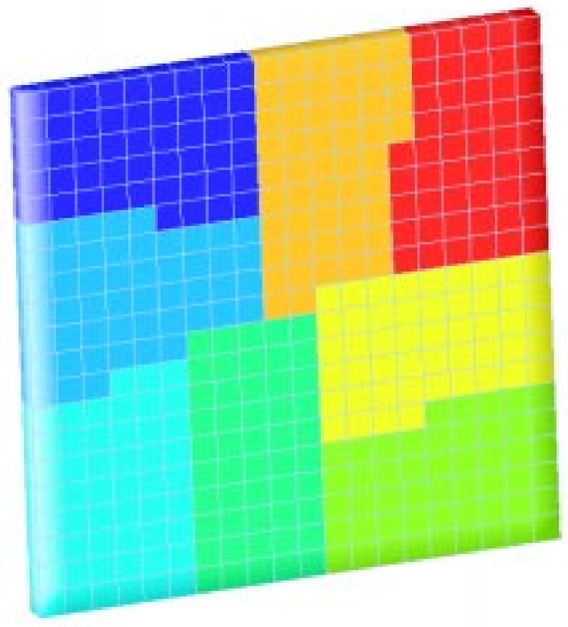


Figure 4.3: Dual Based Partitioning

assembled from elements on each of the bounding processors. After local assembly, these values are sent in the direction of the arrows to the values on the master processor and added. The sending processor (also known as the slave processor) then zeroes its residual values, essentially removing this vertex from it's system. In this manner, all residuals associated with entities on the interprocessor boundary are only solved for by a single processor, known as the entity's master image. After the equations are solved, the solution values are copied from the entities image on master processor to all of its slave images. The creation of the necessary data structures and the execution of these tasks is described in the next section.

For the processors to exchange information during the computation, each must maintain a data structure describing its communications in addition to the other finite element data. The MPI routines MPI_send and MPI_receive use this information to exchange data as described below. The procedure is as follows:

1. After each Newton iteration or time step, the solution values related to mesh entities on the interprocessor boundary are copied to all of its images on each of the adjacent

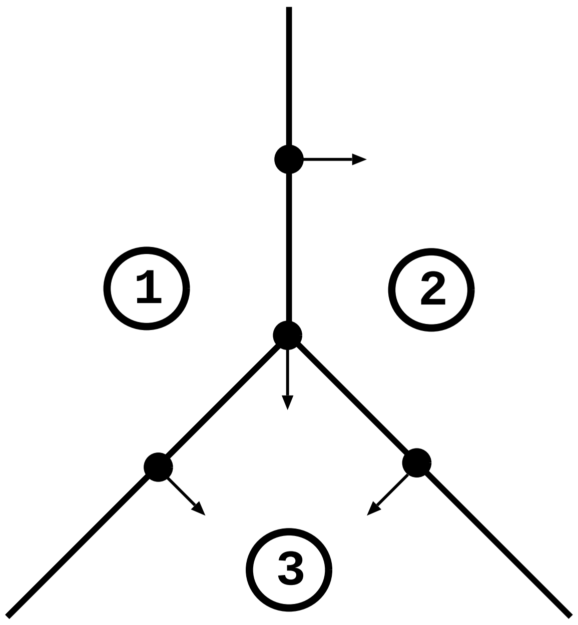


Figure 4.4: Overview of communication

processors. This includes periodic boundaries.

2. Using this data, each processor computes its element level residual values without any need for communication.

3. These contributions are then assembled to the global arrays (on processor) using the traditional finite element assembly procedures (Hughes [21]).

4. An additional interprocessor assembly (described above) is then performed between processors to account for *dof*'s on the interprocessor boundary as shown in Figure 4.4.

Let us define a *communication stage* as that which involves all processors making all their necessary communications. In other words, each communication stage consists of each processor sending to and receiving from each processor with which it must communicate. We will denote by N_P^i the number of processors with which processor i must communicate. There are two types of communication stage: one in which the residuals are added to masters and zeroed on slaves (type 1), and another in which the solution values are copied from masters to slaves (type 2). Both require the same information and differ only slightly. A single type 1 communication stage is necessary each time the element level residual formation and local assembly is completed and a type 2 communication stage is necessary each time the boundary conditions are set on the solution vector.

From the perspective of a single processor, i, a communication stage may be described as a sequence of *tasks*, denoted $T^i_j,\ j = 1 \ldots N^i_T$. To minimize the communication overhead, we require that $N^i_T = N^i_P$, in other words, two processors may communicate only one time during each communication stage. This forces us to designate one of the processors in the task as master, and one as slave and will dictate which processor will be master to each of the mesh entities on the interprocessor boundary. Since only one communication can occur between any two processors, the set of tasks, T^i_j, can be represented as a directed graph, with vertices and edges of the graph representing processors and communications, respectively. This directed graph indicates which processor each mesh entity will be solved on, it therefor must yield a unique master for each mesh entity. For example consider the communication between processors 1 and 3 in Figure 4.4. If the direction between these two processors was reversed, the vertex at the intersection of all three processors would have no unique master. This requirement poses the additional constrain that the graph be *acyclic*, i.e. contain no closed loops. A graph of this type is commonly referred to as a Directed Acyclic Graph, or DAG (see Sedgewick [43]). The procedure to create and set up the directed graph is as follows:

1. Mesh partitioning software such as METIS is used to assign each mesh region to a partition.

2. Each mesh entity is visited and the processor ID number for each of its adjacent regions is associated with the mesh entity (only once if multiple bounding regions are in the same partition).

3. These processor adjacency sets (for each mesh entity) are used to create the partition graph (no direction on graph edges yet).

Once the graph is set up, a simple method is used to create a directed graph from the undirected one. The direction associated with T^i_j is set to point to the greater of i and j, which is guaranteed to produce a DAG. Once a consistent graph is created, it is a straight forward matter to visit each mesh entity and associate the unique processor which is to be its master.

For MPI to carry out the communications described above, each task, T^i_j, must provide details of the exchange of data between processors i and j. To this end, T^i_j has associated with it, the following integer data:

tag: A unique tag associated with T^i_j which distinguishes this send and receive for the MPI functions.

type: Denotes whether this processor is master ($= 1$) or slave ($= 0$) in the current communication. For a type 1 communication, a master calls MPI_receive(...) to receive and add the data, while a slave calls MPI_send(...) to send and zero the data. In a type 2 communication, the slaves receive and copy and the masters send and thus there is no need to zero anything.

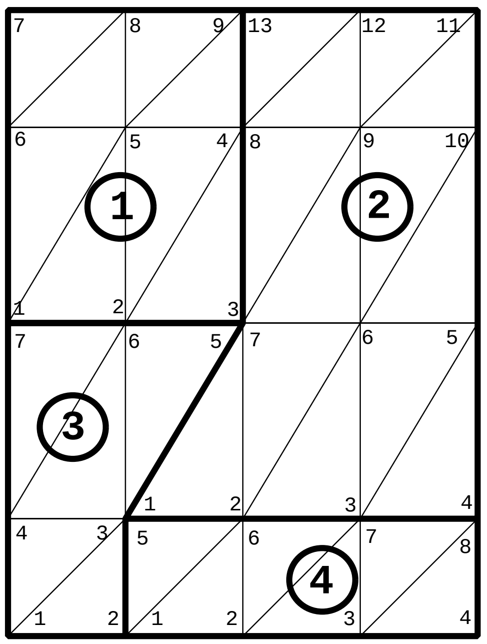
partner: PID number of the partner processor involved in this task.

numSeg: Number of data segments to be sent or received (see below).

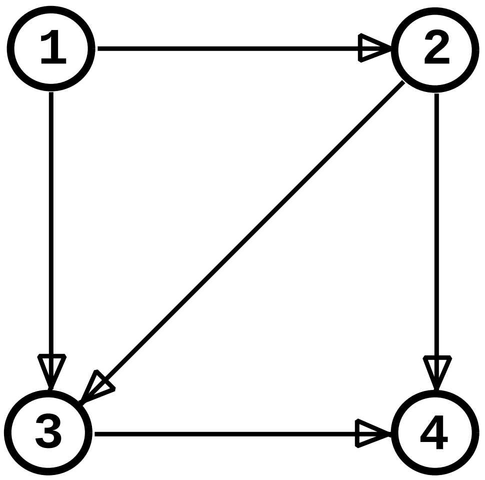
segData: local *dof* numbers on the other processor to send to or receive data from for each of the numSeg data segments.

Here a data segment is defined as a continuously numbered group of *dof* numbers. Each segment also contains its length and starting index. In the beginning of execution of the analysis code, the segment data is used in conjunction with the MPI_TYPE_HVECTOR(...) function to create new MPI data types which are used during the communication stages. These data types are simply masks that describe where information can be found on the various processors for each of the segments in T^i_j.

The above concepts can be clarified through a simple example. Consider the two dimensional mesh shown in Figure 4.5.2 which is decomposed into four partitions. This mesh can be considered as a single geometric face of a three dimensional model. The bold encircled numbers indicate processor ID numbers and the small numbers indicate local *dof* numbers on each partition. For simplicity, only vertex numbers are shown, however, edges (if $p \geq 2$) and faces (if $p \geq 3$) also get *dof* numbers associated with them and are handled identically to vertices. We also assume, for simplicity of discussion, that



(a) Mesh



(b) Graph

there are now periodic boundary conditions applied to the geometric model. A consistent graph corresponding to this mesh, created using the algorithm described above, is shown in Figure 4.5.2. Let us consider only the tasks associated with processor 2, T_j^2, where $j = 1 \dots 3$, since $N_P^2 = 3$. T_1^2 involves a communication where processor 2 is master, and *dof*'s 7, 8, and 13 on processor 2 will be added to the contributions from *dof*'s 3, 4, and 9, respectively, on processor 1. The other two tasks associated with processor 2, T_3^2 and T_4^2, are both slave communications. Here, *dof*'s 7 and 1 are sent to processor 3 and 1 through 4 are sent to processor 4, then these values are zeroed on processor 2.

The Fortran90 code fragment given in Program 4.5.1 illustrates how these data structures can be used within the analysis code to carry out a type 1 communication stage. In this program listing, `global` is a double precision vector to be operated on and `ilwork` is the local work array which contains the information described above.

Program 4.5.1 Type 1 communication stage

```
       dimension global(nshg,n), temp(max), ilwork(nlwork)
       ...
       numtask   = ilwork(1)
       do itask  = 1, numtask
         itag    = ilwork (itkbeg + 1)
         type    = ilwork (itkbeg + 2)
         partner = ilwork (itkbeg + 3)
         numseg  = ilwork (itkbeg + 4)
         isgbeg  = ilwork (itkbeg + 5)
         if (type .EQ. 0) then
           call MPI_SEND(global(isgbeg,1), 1,
       sevsegtype(itask,kdof),
     &                  partner,         itag, MPI_COMM_WORLD,
       ierr)
         else
           lfront = 0
           do is = 1,numseg
             lenseg = ilwork (itkbeg + 4 + 2*is)
             lfront = lfront + lenseg
           enddo
           call MPI_RECV(temp(1), lfront*n, MPI_DOUBLE_PRECISION,
     &                  partner, itag,    MPI_COMM_WORLD,
     &                  status,  ierr)
           itemp = 1
           do idof = 1,n
             do is = 1,numseg
               isgbeg = ilwork (itkbeg + 3 + 2*is)
               lenseg = ilwork (itkbeg + 4 + 2*is)
               isgend = isgbeg + lenseg - 1
               global(isgbeg:isgend,idof) = global(isgbeg:isgend,idof)
    &                                    + temp
       (itemp:itemp+lenseg-1)
               itemp = itemp + lenseg
             enddo
           enddo
         endif
         itkbeg = itkbeg + 4 + 2 * numseg
       enddo
```

4.6 Chapter Summary

In this chapter we have presented the basic concepts of pre-processing higher order finite element simulations. We have discussed the boundary conditions, the data structure used and the parallel communication model. In the next chapter we present some hierarchic simulation examples which have been pre-processed and solved using methods discussed so far.

CHAPTER 5
NUMERICAL EXAMPLES: STEADY LAMINAR FLOW & CONCLUSIONS

This chapter presents numerical simulations which are used to explore the methods introduced in the previous chapters. The problems presented here will verify the convergence rate of the finite element formulation and quantify the cost associated with various polynomial order basis functions. This chapter will demonstrate the cost effectiveness of the higher-order basis functions when compared against the linear basis. First we introduce the cost measures and their description.

5.1 Cost Comparison

Solution time, memory use and disk storage are clearly the most important measures when considering the cost of a simulation. However, by looking at time alone, we would fail to assess the scalability of the method to large-scale problems. In this section, we introduce three additional measures that will be used to quantify the benefit of using higher polynomial order. These measures take into account the costs of computing the distinct components of a simulation, i.e. tangent matrix, residual vector, and linear system solution.

The examples we present are two-dimensional, since we wish to discuss the cost for problems with well-understood benchmark results. It is somewhat difficult to get a fair cost comparison on 2D problems when using the 3D code, since the cost of the higher polynomial order simulations is penalized for adding many additional degrees of freedom in the third (inactive) dimension. Still, we would like to make some estimates of the relative simulation cost, so we consider three cost indices, C_1, C_2, and C_3, defined as:

$$C_1 = n_f \times n_{shp}^2 \times n_{int} \tag{5.1}$$

$$C_2 = n_f \times n_{shp} \times n_{int} \tag{5.2}$$

$$C_3 = n_k \times (n_v \times nnz_v + n_e \times nnz_e + n_f \times nnz_f) \tag{5.3}$$

where n_f is the number of equivalent 2D triangular face elements, n_{shp} the number of 2D degrees of freedom, and n_{int} is the number of triangular integration points required to accurately integrate a 2D element. For the linear solver cost, we have used n_k to represent the number of Krylov vectors needed, and nnz_v, nnz_e, and nnz_f for the non-zero fill pattern associated with vertices, edges, and faces (predictable only for uniform meshes). The first of these measures, C_1, represents the computational cost associated with the formation of the left hand side (LHS) or residual tangent matrix. C_2 is associated with the cost of forming the residual vector (RHS). C_3 relates to the cost of solving the linear system, which is dominated by the non-zero fill pattern. This cost is relevant to our linear solve since we are using a sparse, iterative solver which, for each Krylov vector, performs a matrix-vector product only with the non-zero matrix entries (see Saad [41]). The impact of each of these cost measures is somewhat problem dependent, and more details will be discussed with respect to the individual simulations presented in the following section.

5.2 Kovasznay flow

The first simulation is used to verify the convergence rate of the finite element formulation. It is well known that the interpolation error should converge at a rate of h^{k+1}, where h is a suitable measure of the element size (see Johnson [34]). Since we are simulating this flow on a structured, uniform grid, h is simply taken as the length of the element in the x_1 or x_2 direction, (i.e. Δx_1).

The Kovasznay flow may be associated with the incompressible flow some distance downstream from a rectangular grid (see Kovasznay [37]) and has a closed form analytical solution given by:

$$u_1 = 1 - e^{\lambda x_1} \cos(2\pi x_2) \tag{5.4}$$

$$u_2 = \frac{\lambda}{2\pi} e^{\lambda x_1} \sin(2\pi x_2) \tag{5.5}$$

with

$$\lambda = \frac{\mathrm{Re}}{2} - \sqrt{\frac{\mathrm{Re}^2}{4} + 4\pi^2} \tag{5.6}$$

and we have taken $\mathrm{Re} = 40$ for the present study. The flow is considered on a rectangular domain of $0 \leq x_1 \leq 1$ and $-\frac{1}{2} \leq x_2 \leq \frac{1}{2}$ with the exact solution imposed as an essential boundary condition at the inflow and upper and lower walls, while the pressure was set to zero at the outflow. Since this is a non-trivial essential boundary condition, the interpolation technique described in Section 4.3.2 is employed to determine the basis coefficients on the boundary. Failure to correctly interpolate the boundary condition results in sub-optimal convergence rates (particularly for the higher-order simulations). The qualitative behavior of the solution is depicted in Figure 5.1 which shows contours of fluid speed for the cubic simulation on the 21×21 mesh.

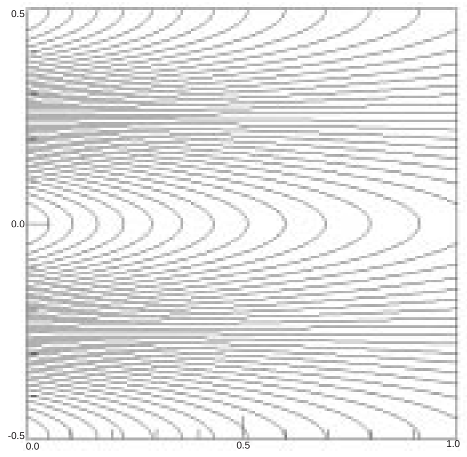
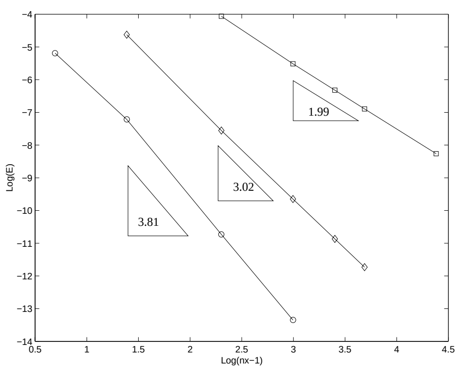
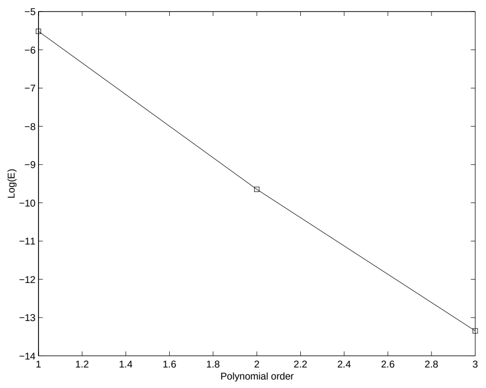


Figure 5.1: Kovasznay flow. Contours of fluid speed for cubic simulation on 21×21 mesh

A convergence study was performed for this flow to determine the accuracy of the different polynomial order simulations as functions of the mesh size, Δx_1, and the polynomial order of the basis. Figures 5.2(a) and 5.2(b) show the log of the normalized error in the L^2 norm of the velocity field versus $\log(\Delta x_1)$ and polynomial order, respectively. Here, the error in the L^2 norm is computed numerically from the formula

$$E^2 = \frac{\int_\Omega e_i e_i \; dx}{\int_\Omega u_i u_i \; dx} \tag{5.7}$$





(a) log of error in L^2 norm vs. log(Δx_1). $\cdots$:$k = 1$, $--$:$k = 2$, and $-$ $k = 3$.

(b) log of L^2 error vs. k

Figure 5.2: Kovasznay flow convergence study

where

$$e_i = u_i - u_i^{(h,k)} \tag{5.8}$$

represents the difference between the exact, u_i, and finite element, $u_i^{(h,k)}$, solutions for the velocity.

Figure 5.2(a) enables us to determine the rates of convergence of the different simulations to be 1.99, 3.02, and 3.81 for the linear, quadratic, and cubic simulations, respectively. These values compare well with the theoretical predictions for the interpolation error (i.e. 2, 3, and 4 for linear, quadratic and cubic). It is clear from Figure 5.2(a) that the constant in the error estimate also greatly improves for the higher-order simulations, making the higher-order basis most attractive even on the coarsest meshes. Figure 5.2(b) demonstrates the exponential convergence of the method when Δx_1 is fixed and the polynomial order is increased.

A cost analysis of the Kovasznay flow simulations was performed based on the cost measures discussed in the previous section. Of particular relevance is the value of each of these cost measures when trying to obtain a solution of a specified accuracy. Consider the case where the quadratic and the linear basis attempt to achieve the same accuracy as the cubic basis on the coarsest hexahedral mesh (5×5 points). The results of these cost studies are illustrated in table 5.1. From this Table it is clear that, for this level of accuracy, the cubic simulation is the most cost effective for almost all cost measures considered. For matrix formation costs, C_1, the higher order approaches enjoy only a slight advantage. This is to be expected since element matrices grow nonlinearly (i.e. n^2_{shp}) so that despite the reduction in the number of elements required to attain a given accuracy, this cost remains relatively constant. This issue can be mitigated by noting that it is not necessary to form the element stiffness matrix at every time step. We have observed that, once past the first few steps, where the tangent to the residual changes rapidly, the LHS can be stored and reused for 10 or more steps/iterations. This effectively takes this cost out of the problem. This is important since otherwise, as k rises, LHS matrix formation can become the dominant cost. Even unsteady problems can typically follow this strategy though greater care must be taken to ensure that convergence within a given step is not compromised. The second cost index, C_2, shows a significant cost reduction for increasing k (factor of 12 more for linears than cubic hexes).

Though we have a rather complicated RHS to form (viz. (3.8)) the total cost of our calculations are often dominated by the iterative solver. Therefore, C_3 is often an important cost to consider. Here, we see an even more dramatic increase in computational effort as k decreases. While one expects the equations to get more stiff as the polynomial order increases we observe that the accuracy attained more than offsets the cost. It is important to note that decreasing element size also results in element stiffness and lower order methods must undergo many more subdivisions to obtain the same accuracy of the higher polynomial orders, increasing the number of iterations required and increasing the number of degrees of freedom in the system, both direct contributors to the cost of solution.

Topology	k	C_1	C_2	C_3
Hex(5x5)	3	1.00	1.00	1.00
Hex(10x10)	2	1.27	1.90	1.84
Hex(49x49)	1	4.00	12.0	30.1

Table 5.1: Kovasznay relative cost comparison, extrapolating data to fictional meshes which would achieve the same accuracy as cubic hexahedra (5×5 points).

5.3 Flow over a backward-facing step

Consider a two-dimensional, incompressible flow over a backward-facing step at $Re = 800$, based on the step height and the average inflow velocity. The geometry and boundary conditions are similar to those used by Gartling [15]. The problem is specified by a fully developed flow entering a confined channel, and, at $Re = 800$, has been demonstrated by numerous researchers to be steady and stable (see Gresho *et al.* [17]). A complete description of the physical problem requires modeling the region upstream of the step, and careful attention to the singularity that may develop at the step corner. However, since the objective of this study was a comparison of various polynomial order bases rather than a complete description of the physics, the standard step flow geometry was simplified by excluding the region upstream of the step as described in Gartling [15]. This allows for a direct comparison with the benchmark results.

The geometry and boundary conditions are shown in Figure 5.3. The initial condition consists of a parabolic (Poiseuille) velocity profile with the same mass flow-rate as the inlet profile, imposed upon the entire channel. This initial condition is marched in time using the backward Euler technique until the steady solution is reached, confirmed in all cases by monitoring the changes in various flow quantities. The steady state is also verified by steady shear stress on the channel walls.

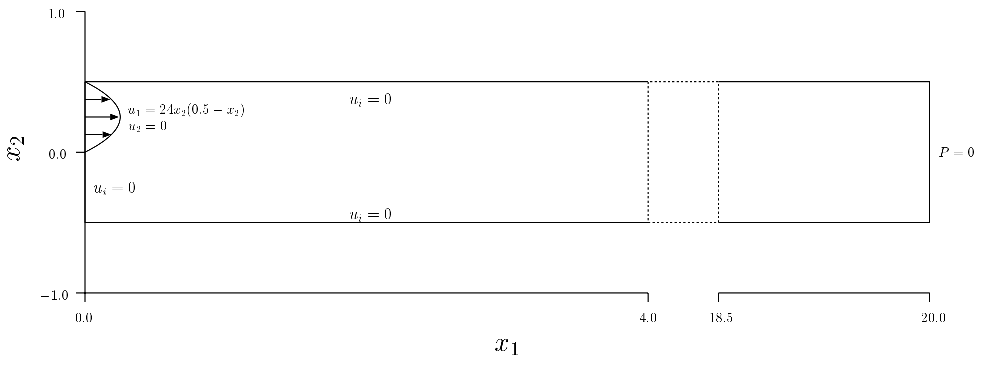


Figure 5.3: Step flow geometry and problem description

Numerical solutions were obtained on a variety of uniform tetrahedral meshes for several different polynomial orders. The mesh statistics are shown in Table 5.2 where each successive mesh represents a uniform refinement of the previous mesh, with the exception of mesh B. Here, Δx_1 and Δx_2 represent the element size in the x_1 and x_2

directions, respectively.

Mesh	Vertices	Edges	Faces	Δx_1	Δx_2
A	405	724	320	0.250	0.250
B	847	1566	7200	0.167	0.167
C	2,211	4210	2,000	0.100	0.100
D	8,421	16420	8,000	0.050	0.050
E	32,841	64840	16,000	0.025	0.025

Table 5.2: Step flow mesh statistics

There are three major factors that contribute to the cost of the finite element simulations discussed here: right hand side (or residual) evaluation, left-hand-side (or tangent and mass) formation, and linear algebra (involving matrix-vector products for iterative solution techniques). The first two of these measures rely on the numerical integration of element level quantities, and the dominant terms are in the left-hand-side calculation (proportional to the number of integration points times the number of element basis functions squared). The matrix-vector products are dominated by the fill pattern of the matrix. The relative size of these different measures depends on the order of the basis being used (as well as the problem). For example, the linear basis is dominated by the linear algebra, since we are using relatively cheap integration rules. Also, the fine meshes needed for linear basis computations lead to greater cost in solving the linear system (more Krylov vectors are needed).

In Table 5.3, we show the first cost index (C_1), and a “-” in the Cost column indicates that this simulation is not included in the study. The cost measured in this way reflects the tangent matrix formulation cost. The symbols in the far-right column indicate the meshes used in the comparison study discussed below. These calculations are in terms of the face data and face quadrature rules in an attempt to level the playing field for the 2-D comparison with the 3-D code.

The basic character of this flow is well known. At $Re = 800$, there are two separation regions, one starting at the step corner and continuing downstream approximately 12 step heights, and another on the upper wall of the channel occupying a region from approximately 10 to 20 step heights downstream. These features are shown in Figures 5.4(a)- 5.4(b) which represent the fluid speed, and pressure for the cubic simulation

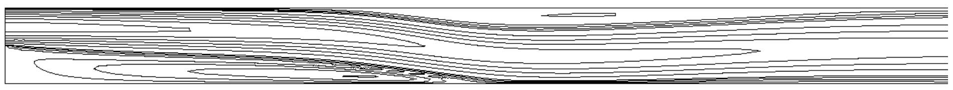
Mesh	k	n_s	n_{int}	C_1	
	1	405	4	-	
A	2	1,129	9	-	
	3	1,853	16	0.02372	$\circ$
	1	847	4	-	
B	2	3,133	9	0.05081	
	3	6,859	16	0.24528	
	1	2,211	4	0.0527	
C	2	8,421	9	1.0000	$+$
	3	18,631	16	4.87871	
	1	8,421	4	3.0593	
D	2	32,841	9	59.8383	
	3	73,261	16	-	
	1	32,841	4	185.9838	$*$
E	2	129,681	9	-	
	3	290,521	16	-	

Table 5.3: Step flow simulation cost comparison

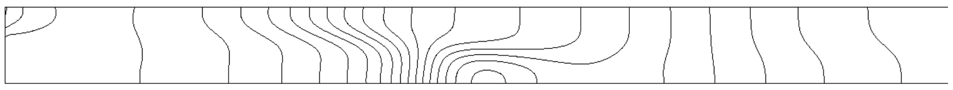
on mesh C. These figures are shown in the correct scale, however, only the first twenty heights of the step are shown. Qualitatively, these figures compare well with those presented in Gartling [15].

The contour plots look similar for all simulations making it difficult to quantify the benefit of the higher-order methods. We will therefore compare line plots of various flow quantities at different spatial locations. The first of these plots demonstrates that all of the methods are converging to the benchmark result (with linear being the slight exception). Figure 5.5(a) shows the (most refined) cubic, quadratic, and linear simulations on meshes C, D, and E, respectively, as well as the benchmark result of Gartling [15]. For each of these, the x_1- and x_2- velocities and pressure are shown at two locations along the channel, $x_1 = 7.0$ and $x_1 = 15.0$, the same locations presented in Gartling [15], which we have included on the velocity plots as a benchmark result. The cubic and quadratic are able to exactly reproduce the benchmark simulation, while the linear, even on the most refined grid, is still slightly off in the x_2-velocity at the $x_1 = 7.0$ location. This isn't surprising, since the benchmark result is from a quadratic simulation with 41 vertices across the channel, which has the same number of points as our most refined linear simulation.

Figure 5.5(b) presents a comparison between the cubic simulation on mesh A, the

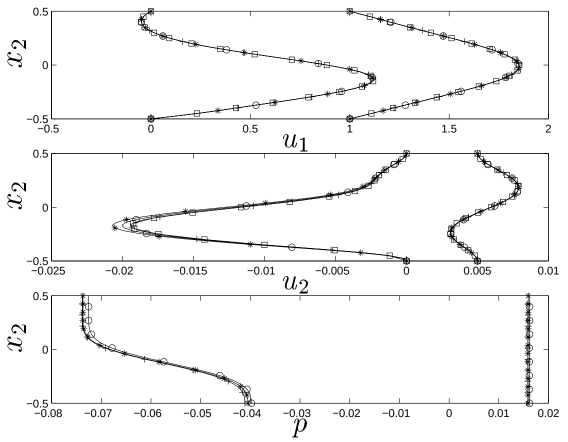


(a) contours of fluid speed

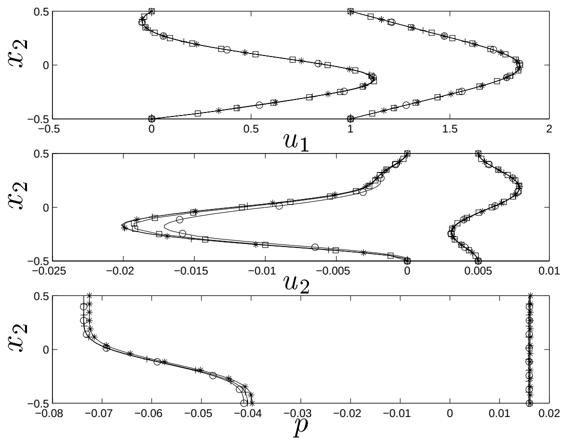


(b) contours of pressure

Figure 5.4: Step flow simulation characteristics: Mesh C, $k = 3$



(a) Comparison on finest meshes. $\circ$: $k = 1$, $+$: $k = 2$, $*$: $k = 3$, and $\square$: Gartling.



(b) Comparison of qualitatively similar solutions. $\circ$: $k = 3$, $+$: $k = 2$, $*$: $k = 1$, and $\square$: Gartling

Figure 5.5: Backward-facing step. Velocity and pressure plotted versus x_2 at $x_1 = 7$ and $x_1 = 15$. Velocities at $x_1 = 15$ were shifted for plotting.

quadratic simulation on mesh C, and the linear on mesh E. These three simulations were shown to produce qualitatively similar results for tetrahedral meshes by Whiting [58]. Clearly the cubic solution for the x_2 velocity on mesh A, seems to deviate most, and this is under further investigation. Table 5.3 shows the cost index for these three simulations as 0.02372, 1.000, and 185.9838 for the cubic, quadratic, and linear simulations, respectively.

5.4 Lid-driven cavity flow

The next problem considered is the steady, two-dimensional and incompressible flow inside a closed container driven by its lid. The lid slides to the right at unit velocity across the top of the cavity, shearing the fluid and setting up a recirculation region. There is a primary vortex in the center of the cavity and secondary eddies in the lower corners (the number of these secondary eddies depends on the Reynolds number). For the present study, we have chosen to consider $Re = 400$ (based on the lid velocity), for which there exist well-established benchmark results to compare with (see Ghia *et al.* [16]). Since the velocity is discontinuous at both upper corners, singularities will develop in the pressure and stress fields, which must be controlled by the method. In addition, there are singularities also in the lower corners, however, they are well resolved by the uniform meshes.

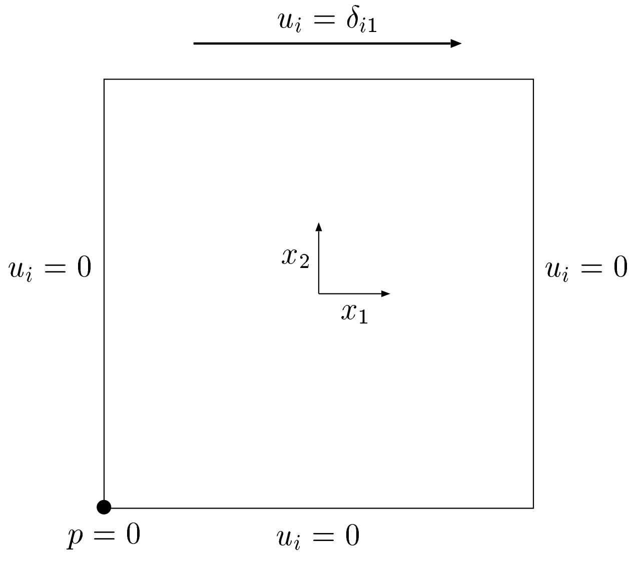


Figure 5.6: Lid-driven cavity geometry and boundary conditions

The geometry and boundary conditions are illustrated in Figure 5.6. In addition to the velocity constraints, the pressure field is constrained by setting its value at the single vertex in the lower left corner of the cavity. Uniform meshes were used with equal spacing in the x_1- and x_2- directions. To isolate the singularities in the upper corners, nested local mesh adaptivity was used by subdividing the original corner elements. The number of new corner elements was chosen such that the first point is 3.90625×10^{-4} units from the corner for each mesh. This distance dictates the extent to which the discontinuity in the velocity field is resolved (i.e. how much fluid is "leaked" from the cavity) and is fixed at the given value for all simulations by changing the number of corner elements. This procedure is crucial to obtaining identical solutions for the different polynomial orders, since the actual mesh size varies dramatically between the least refined cubic and most refined linear simulations. Figure 5.7 shows an schematic illustration of the corner refinement in the upper right corner. As mentioned earlier, the actual refinement is much finer so that the first point is 3.90625×10^{-4} units away from the corner vertex (the upper left corner is adapted the same way).

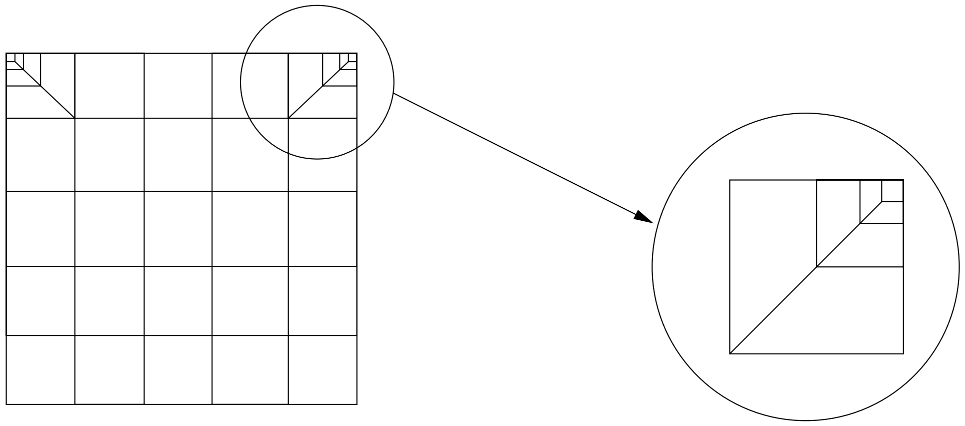


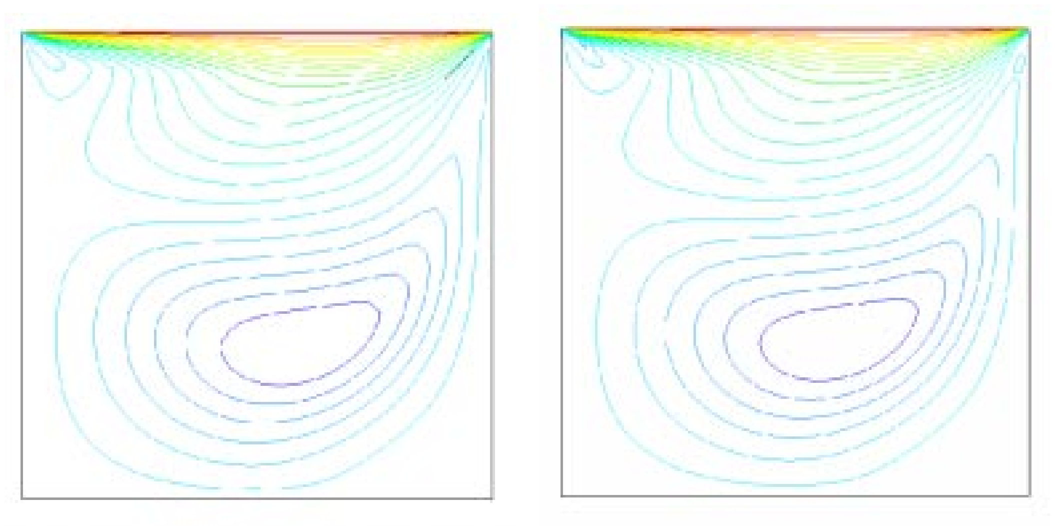
Figure 5.7: Mesh of lid showing corner adaptivity

The statistics for these meshes along with the polynomial orders used are shown in Table 5.4. This data does not include the refinement in the upper corners, as this represents a small percentage of the total mesh.

The basic solution characteristics of this flow are shown in Figures 5.8(a) - 5.8(b) which display contours of fluid speed for a cubic simulation on a 11 $\times$11 mesh and a quadratic simulation for a 21$\times$21 mesh respectively. As can be observed with these

Mesh	Vertices	k
A	11×11	3
B	21×21	2,3
C	41×41	1,2,3
D	81×81	1,2
E	161×161	1

Table 5.4: Lid-driven cavity mesh statistics

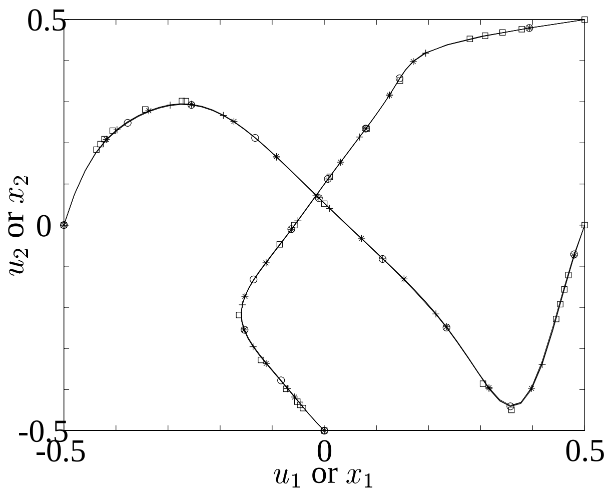


(a) 11×11, cubic simulation (b) 21×21, Quadratic Simulation

Figure 5.8: Lid-driven cavity fluid speed contours: Mesh A, $k = 3$ and Mesh B, $k = 2$

two figures, all converged simulations look identical. Figure 5.9(a) shows profiles of $u_2(x_1, x_2 = 0)$ and $u_1(x_1 = 0, x_2)$ for the the most refined mesh for each polynomial order. Note that u_1 was scaled by 0.5 to facilitate plotting. Also shown is the benchmark result of Ghia, *et al.*[16] (one stray point was removed from their tabular data). The three plots are virtually indistinguishable.

A cost comparison study similar to the Kovasznay flow study was carried out for the lid-driven cavity flow (see the velocity profiles in Figure 5.9(a)). The simulations, along with the second and third cost indices are shown in Table 5.5. Again, the cubic simulations show a dramatic advantage relative to all other choices on all cost indices. The “Matrix” column of Table 5.5 indicates the number of nonzero blocks for the sparse



(a) Qualitatively similar solutions for Lid-Driven Cavity flow.

Figure 5.9: Lid-driven Cavity flow. plots of $u_1(x_1 = 0, x_2)$ and $u_2(x_1, x_2 = 0)$, ($*$) linear on 161×161 mesh, ($\circ$) quadratic on 41×41 mesh and ($+$) cubic on 11×11 mesh, (□) Ghia *et al.*.

storage of the tangent matrix (the dominant memory requirement), indicating that the memory requirements for the cubic simulation are about 28 times less than that of the linear simulation , while the quadratic requires about a factor of 9 times the memory . The “Mesh” column compares the size of the file that stores the coordinates and connectivity of the mesh.

Table 5.5: Lid-driven cavity normalized cost comparison

Mesh	k	C_2	C_3	Mesh	Matrix
11×11 hex	3	1.00	1.00	1.00	1.00
41×41 hex	2	6.00	29.7	6.96	9.29
161×161 hex	1	21.3	480	52.3	27.8

5.5 Conclusions

A stabilized finite element method using a hierarchical basis has been applied to the incompressible Navier-Stokes equations. The implementation is general, allowing three-dimensional simulations on arbitrary, unstructured meshes to be carried out on parallel computers. This new formulation has been shown to yield accurate and robust simulations on a variety of problems using both linear and hierarchical basis function alike.

The hierarchical basis described in this thesis has been shown to attain near optimal rates of convergence with respect to the interpolation error for incompressible (Kovasznay) flows. These studies have been motivated by the desire to simulate more complicated physics problems, where traditional linear elements require too many grid points for current computers. It has been shown here for the first time, through several, relatively simple examples, that for steady problems the cubic and quadratic basis functions can be many times more cost effective than linears. This indicates that the higher-order basis functions may provide a means to attain simulations of physical problems unattainable with linear elements due to computational limitations.

There are, however, limitations of the present research. It is hoped that they may be addressed in future work, to further enhance the benefit of the hierarchical basis. The present formulation is limited to uniform polynomial order. The current implementation is also limited to straight-sided elements using a linear mapping to element coordinates. Non-uniform polynomial order will allow meshes for relatively simple configurations such as boundary layers to be more effectively simulated by using a lower polynomial order in the streamwise and span-wise directions while maintaining higher-order for the wall normal direction, where the flow gradients are highest. This type of polynomial order grading is easily made possible for meshes comprised of hexahedral elements due to the fact that they are well aligned with surfaces.

REFERENCES

[1] F. Bastin. Jet noise using large eddy simulation. In *Annual Research Briefs*, pages 115–132, NASA Ames / Stanford University, 1996. Center for Turbulence Research.

[2] M. W. Beall and M. S. Shephard. A general topology-based mesh data structure. *Int. J. Numer. Meth. Engng.*, 40(9):1573–1596, 1997.

[3] M. Behr, D. Hastreiter, S. Mittal, and T. E. Tezduyar. Incompressible flow past a circular cylinder: dependence of the computed flow field on the location of the lateral boundaries. *Comp. Meth. Appl. Mech. Engng.*, 123:309–316, 1996.

[4] K. S. Bey, A. Patra, and J. T. Oden. hp-version discontinuous galerkin methods for hyperbolic conservation laws: A parallel adaptive strategy. *Int. J. Numer. Meth. Engng.*, 1995.

[5] R. Biswas, K. D. Devine, and J. E. Flaherty. Parallel adaptive finite element methods for conservation laws. *Appl. Numer. Maths.*, 14:255–284, 1994.

[6] F. Brezzi, L. P. Franca, T. J. R. Hughes, and A. Russo. $b = \int g$. *Comp. Meth. Appl. Mech. Engng.*, 145:329–339, 1997.

[7] A. N. Brooks and T. J. R. Hughes. Streamline upwind / Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations. *Comp. Meth. Appl. Mech. Engng.*, 32:199–259, 1982.

[8] P. Carnevali, R. B. Morris, Y. Tsuji, and G. Taylor. New basis functions and computational procedures for p-version finite element analysis. *Int. J. Numer. Meth. Engng.*, 36:3759–3779, 1993.

[9] M. H. Carpenter and David Gottlieb. Spectral methods on arbitrary grids. *Journal of Computational Physics*, 1996.

[10] J. Chung and G. M. Hulbert. A time integration algorithm for structural dynamics with improved numerical dissipation: The generalized-α method. *Journal of Applied Mechanics*, 60:371–75, 1993.

[11] L. Demkowicz, J. T. Oden, W. Rachowicz, and O. Hardy. Toward a universal h-p adaptive finite element strategy, part 1: Constrained approximation and data structure. *Comp. Meth. Appl. Mech. Engng.*, 77:79–112, 1989.

[12] Karen M. D. Devine. *An adaptive hp-finite element method with dynamic load balancing for the solution of hyperbolic conservation laws on massively parallel computers*. Ph.D. Thesis, RPI, 1994.

[13] S. Dey. *Geometry-based three-dimensional hp-finite element modelling and computations*. PhD thesis, Rensselaer Polytechnic Institute, 1997.

[14] L. P. Franca and S. Frey. Stabilized finite element methods: II. The incompressible Navier-Stokes equations. *Comp. Meth. Appl. Mech. Engng.*, 99:209–233, 1992.

[15] D. K. Gartling. A test problem for outflow boundary conditions – flow over a backward-facing step. *International Journal of Numerical Methods in Fluids*, 11:953–967, 1990.

[16] U. Ghia, K. N. Ghia, and C. T. Shin. High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method. *Journal of Computational Physics*, 48:387–441, 1982.

[17] P. M. Gresho. Some current CFD issues relevant to the incompressible Navier-Stokes equations. *Comp. Meth. Appl. Mech. Engng.*, 87:201–252, 1991.

[18] P. M. Gresho and R. L. Sani. *Incompressible Flow and the Finite Element Method*. Wiley, New York, NY, 1998.

[19] P. Hansbo and A. Szepessy. Velocity-pressure streamline diffusion finite element method for the incompressible Navier-Stokes equations. *Comp. Meth. Appl. Mech. Engng.*, 1990.

[20] D. Haworth and K. E. Jansen. LES on unstructured deforming meshes: towards reciprocating IC engines. In *Proceedings of the 1996 Summer Program*, pages 329–346, NASA Ames / Stanford University, 1996. Center for Turbulence Research. also accepted for publication in Computers in Fluids.

[21] T. J. R. Hughes. *The finite element method: Linear static and dynamic finite element analysis*. Prentice Hall, Englewood Cliffs, NJ, 1987.

[22] T. J. R. Hughes. Multiscale phenomena: Green's functions, the Dirichlet-to-Neumann formulation, subgrid scale models, bubbles and the origins of stabilized methods. *Comp. Meth. Appl. Mech. Engng.*, 127:387–401, 1995.

[23] T. J. R. Hughes, L. P. Franca, and M. Balestra. A new finite element formulation for fluid dynamics: V. A stable petrov-Galerkin formulation of the Stokes problem accommodating equal-order interpolations. *Comp. Meth. Appl. Mech. Engng.*, 59:85–99, 1986.

[24] T. J. R. Hughes, L. P. Franca, and G. M. Hulbert. A new finite element formulation for fluid dynamics: VIII. The Galerkin / least–squares method for advective–diffusive equations. *Comp. Meth. Appl. Mech. Engng.*, 73:173–189, 1989.

[25] T. J. R. Hughes and K. E. Jansen. A stabilized finite element method for the Reynolds-averaged Navier-Stokes equations. *Surveys on Mathematics for Industry*, 4:279–317, 1995.

[26] T. J. R. Hughes, L. Mazzei, and K. E. Jansen. Large-eddy simulation and the variational multiscale method. *Computing and Visualization in Science*, to appear, 1999.

[27] K. E. Jansen. Large-eddy simulation of flow around a NACA 4412 airfoil using unstructured grids. In *Annual Research Briefs*, pages 225–232, NASA Ames / Stanford University, 1996. Center for Turbulence Research.

[28] K. E. Jansen. A stabilized finite element method for computing turbulence. *Comp. Meth. Appl. Mech. Engng.*, 174:299–317, 1999.

[29] K. E. Jansen, C. H. Whiting, and G. M. Hulbert. A generalized-α method for integrating the filtered Navier-Stokes equations with a stabilized finite element method. *Comp. Meth. Appl. Mech. Engng.*, 1999. accepted, SCOREC Report 10-1999.

[30] Z. Johan, T. J. R. Hughes, K. K. Mathur, and S. L. Johnsson. A data parallel finite element method for computational fluid dynamics on the Connection Machine system. *Comp. Meth. Appl. Mech. Engng.*, 99:113, 1992.

[31] A. A. Johnson and T. E. Tezduyar. Mesh update strategies in parallel finite element computations of flow problems with moving boundaries and interfaces. *Comp. Meth. Appl. Mech. Engng.*, 119:73–94, 1994.

[32] A. A. Johnson and T. E. Tezduyar. 3D simulation of fluid-particle interactions with the number of particles reaching 100. *Comp. Meth. Appl. Mech. Engng.*, 145:301–321, 1997.

[33] C. Johnson and A. Szepessy. On the convergence of a finite element method for a nonlinear hyperbolic conservation law. *Mathematics of Computation*, 1987.

[34] Claes Johnson. *Numerical solution of partial differential equations by the finite element method*. Cambridge University Press, Sweden, 1987.

[35] J. Kennedy, M. Behr, V. Kalro, and T. Tezduyar. Implementation of implicit finite element methods for incompressible flows on the CM-5. *Comp. Meth. Appl. Mech. Engng.*, 1994.

[36] J. Kim, P. Moin, and R. Moser. Turbulence statistics in fully developed channel flow at low Reynolds number. *Journal of Fluid Mechanics*, 177:133, 1987.

[37] L. I. G. Kovasznay. Laminar flow behind a two-dimensional grid. *Proc. Cambridge Philos. Soc.*, 44, 1948.

[38] H. Le, P. Moin, and J. Kim. Direct numerical simulation of turbulent flow over a backward-facing step. *Journal of Fluid Mechanics*, 330:349–374, 1997.

[39] J. T. Oden, I. Babuška, and C. E. Baumann. A discontinuous hp finite element method for diffusion problems. *Journal of Computational Physics*, 146:491–519, 1998.

[40] A. Rússo. Bubble stabilization of the finite element methods for the linearized incompressible Navier-Stokes equations. *Comp. Meth. Appl. Mech. Engng.*, 132:335–343, 1996.

[41] Y. Saad. *Iterative methods for sparse linear systems*. PWS Pub. Co., Albany, NY, 1996.

[42] K. Schloegel, G. Karypis, and V. Kumar. Multilevel diffusion algorithms for repartitioning of adaptive meshes. Technical Report #97-013, U. Minnesota, Dept. of Comp. Sci. and Army HPC Center, 1997.

[43] Robert Sedgewick. *Algorithms in C*. Addison–Wesley, Reading, Massachusetts, 1990.

[44] F. Shakib. *Finite element analysis of the compressible Euler and Navier-Stokes equations*. PhD thesis, Stanford University, 1989.

[45] F. Shakib and T. J. R. Hughes. A new finite element formulation for computational fluid dynamics: IX. Fourier analysis of space-time Galerkin/least-squares algorithms. *Comp. Meth. Appl. Mech. Engng.*, 87:35–58, 1991.

[46] F. Shakib, T. J. R. Hughes, and Z. Johan. A new finite element formulation for computational fluid dynamics: X. The compressible Euler and Navier-Stokes equations. *Comp. Meth. Appl. Mech. Engng.*, 89:141–219, 1991.

[47] Farzin Shakib. http://www.acusim.com.

[48] M. S. Shephard. The specification of physical attribute information for engineering analysis. *Eng. Comput.*, 4:145–155, 1988.

[49] M. S. Shephard, S. Dey, and J. E. Flaherty. A straight forward structure to construct shape functions for variable p-order meshes. *Comp. Meth. Appl. Mech. Engng.*, 147:209–233, 1997.

[50] S. J. Sherwin and G. E. Karniadakis. A new triangular and tetrahedral basis for high-order (hp) finite element methods. *Int. J. Numer. Meth. Engng.*, 38:3775–3802, 1995.

[51] S. J. Sherwin and G. E. Karniadakis. A triangular spectral element method; applications to the incompressible Navier-Stokes equations. *Comp. Meth. Appl. Mech. Engng.*, 123:189–229, 1995.

[52] S. J. Sherwin and G. E. Karniadakis. Tetrahedral hp finite elements: algorithms and flow simulations. *Journal of Computational Physics*, 124(1):14, 1996.

[53] B. A. Szabo and I. Babuška. *Finite Element Analysis*. Wiley Interscience, New York, 1991.

[54] C. A. Taylor, T. J. R. Hughes, and C. K. Zarins. Finite element modeling of blood flow in arteries. *Comp. Meth. Appl. Mech. Engng.*, 158:155–196, 1998.

[55] T. E. Tezduyar, S. Aliabadi, M. Behr, A. Johnson, V. Kalro, and M. Litke. Flow simulation and high performance computing. *Computational Mechanics*, 18:397–412, 1996.

[56] T. E. Tezduyar, M. Behr, and J. Liou. 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. *Comp. Meth. Appl. Mech. Engng.*, 94:339–351, 1992.

[57] T. E. Tezduyar, M. Behr, and J. Liou. 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. *Comp. Meth. Appl. Mech. Engng.*, 94:339–351, 1992.

[58] C. H. Whiting. *Stabilized finite element methods for fluid dynamics using a hierarchical basis*. PhD thesis, Rensselaer Polytechnic Institute, Sep. 1999.