Geometric Shape Preservation in Adaptive Refinement of Finite Element Meshes

ASHRAFUL KADIR

KTH Computer Science
and Communication

Master of Science Thesis
Stockholm, Sweden 2010

Geometric Shape Preservation in Adaptive Refinement of Finite Element Meshes

A S H R A F U L K A D I R

Master's Thesis in Numerical Analysis (30 ECTS credits)
at the Scientific Computing International Master Program
Royal Institute of Technology year 2010
Supervisor at CSC was Johan Hoffman
Examiner was Michael Hanke

TRITA-CSC-E 2010:001
ISRN-KTH/CSC/E--10/001--SE
ISSN-1653-5715

Royal Institute of Technology
School of Computer Science and Communication

KTH CSC
SE-100 44 Stockholm, Sweden

URL: www.csc.kth.se

Abstract

This thesis presents a vertex repositioning based boundary smoothing technique so that the finite element mesh, as refined, converges to the geometric shape of the domain. Adaptive mesh refinements based on a posteriori error estimates is already available within FEniCS, a free software for automated solution of differential equations. However, as the refinement process runs independently of the CAD (Computer-Aided Design) data, once the initial mesh is created the geometric shape is fixed and remains unchanged. The proposed technique removes this limitation enabling the mesh to be reshaped during the mesh refinement process according to the geometry. In the proposed technique, upon mesh refinements, the boundary vertices are repositioned to the orthogonal projection point on to the surface geometry. Newton iteration based iterative closest point algorithms have been used to search points projection on to the geometry represented as NURBS (Nonuniform Rational B-splines). For the new boundary vertices, old neighbor-nodes' projection data have been used for obtaining an initial guess for the Newton's iteration. Projection parameters for the boundary vertices of an initial coarse mesh is gathered using exhaustive search for points projection on the whole geometry. Once the mesh boundary vertices are repositioned, positions of the internal vertices are adjusted using mesh smoothing. Experimental results show that the method is sufficiently accurate and efficient for both 2D and 3D problems for some simple test cases. Inverted cells, which could invalidate the mesh, are not developed. Cell based mesh quality analysis and time performance analysis have been presented. Experimentally, it has been shown that using the proposed technique it is sufficient to start with a coarse mesh all over the domain and adaptively refine the mesh and the coarse mesh, as refined, captures the curved geometry more accurately. Since the proposed technique does not change the mesh topology and the finite element solver or mesh elements are not required to be changed, the developed tools can be plugged in to the finite element solver without much involvements. Although the tools are developed to become integrated parts of FEniCS, the parts representing geometric data manipulation can be used with other applications as well. Implementations in this thesis are limited for single NURBS patch geometry only and recommendations are given for future improvements to incorporate multiple patch NURBS geometry and trimmed NURBS.

Referat

Bevarande av geometrisk form i finita elementmetoder med adaptiv nätförfining

I den här avhandlingen presenteras en teknik för anpassning av randnoder, så att beräkningsnätet under förfining konvergerar till den exakta geometriska domänen. Adaptiv nätförfining baserat på a posteriori-feluppskattning finns redan i FEniCS: fri mjukvara för automatiserad lösning av differentialekvationer. Dock saknas kopplingen mellan CAD (geometri) och nätförfining så efter det ursprungliga nätet har skapats tas inte längre någon hänsyn till geometrin. Den presenterade tekniken tar bort denna begränsning och möjliggör en anpassning av nätet till geometrin under nätförfiningsprocessen. I den presenterade tekniken flyttas randnoder, efter nätförfining, till den ortogonalt projicerade punkten på ytan av randgeometrin. Iterativa Newton-algoritmer har använts för att söka efter projektionspunkter på geometri representerad som NURBS (Nonuniform Rational B-splines). För att behandla de nya randnoderna har projektionsdata från grannoder använts som en gissning till Newton-iterationen. Projektionsparametrar för randnoderna i ett ursprungligt grovt nät insamlas med en uttömmande sökning för projektionspunkter i hela geometrin. När anpassningen av randnoderna har gjorts flyttas de interna noderna enligt en utslätningsmetod. Experiment visar att metoden ger tillräcklig noggrannhet och effektivitet för enkla 2D och 3D-problem. Inverterade celler genereras ej. Tekniken ämnas integreras i FEniCS, men kan även användas självständigt. Implementationen som ges i avhandlingen är begränsad för en ensam NURBS-yta men rekommendationer ges för framtida utökning för multipla NURBS-ytor och trimmade ytor.

Acknowledgements

First, I would like to say thanks to my supervisor, Prof. Johan Hoffman, for giving me the opportunity to work in this interesting project and for his excellent guidance throughout the period. I would also like to say thanks to Prof. Michael Hanke for reading the manuscript and advising valuable improvements. I would like to say thanks to Johan Jansson for the interesting discussions, sharing his knowledge on FEniCS, and finally for translating the abstract of the thesis for me. I also thank Murtazo Nazarov for sharing his ideas and knowledge on FEniCS. I would also like to say thanks to the other members of the CTL (Computational Technology Laboratory) group for the interesting discussions and support.

I thank my loving wife for her understanding, encouragements and support. Finally, I dedicate my thesis to my wonderful parents for their love, encouragements and unconditional support throughout my life. I specially acknowledge my mother's great contributions to my life who dedicated all her efforts for her two sons.

Contents

List of Figures

Chapter 1

Introduction

Finite element analysis and computer-aided design (CAD) are very close to each other in computational science & engineering. Although there are differences between the geometry representation and manipulations in these two fields, they are often coupled to solve computational problems in numerous fields. Historically, major finite element programs were technically mature long before modern CAD was widely adopted as finite element analysis had its origins in the 1950s and 60s where CAD had its origins later in the 1970s and 1980s and perhaps, as Hughes et al. (2005) argued, this is the reason for the difference between the underlying geometry of finite element analysis and CAD.

Due to the differences in the geometry representations and operations in finite element analysis and CAD, difficulties occur in the finite element solvers, from both accuracy and performance points of view. One would require to choose the degree of accuracy and performance to get the optimal approximations of the solution for the given problem within the finite time. This turns out to be another optimization problem which, like many other optimization problems, is extremely difficult or even impossible to solve exactly. In this thesis we intend to solve the problem in a way which is practically effective to solve large computational problems efficiently.

Within finite element analysis, one would use an approximation of the original curved geometry often in the form of triangulated mesh (here it is to be noted that, in recent days, examples are available where exact curve geometry definition is used in the finite elements near the boundary (see Sevilla 2009)). However, very fine mesh elements are necessary to represent the curved geometry quite accurately. Although we may almost never get the same curved shape, a fine approximation specially in the curved regions are often considered appropriate for finite element solutions. Typically a fine mesh will give better accuracy of the solution than a coarse mesh. The tradeoff for using very fine mesh is, it will be computationally expensive to solve a problem where a coarse mesh will give the result much faster. Often it is of interest to start solving the problem with a coarse mesh and adaptively refine the mesh based on an a posteriori error estimation (see Eriksson et al. 1996). However, many mesh refinement algorithms will refine the mesh without changing

the overall shape according to the geometry and this is the area where we want to contribute so that the coarse mesh, as refined, represents the geometric shape more accurately.

Research works in the last few decades are going on to apply CAD concepts into analysis, which also can be referred as computer-aided engineering (CAE). Specially finite element mesh generation and refinement areas are closely related to CAD technologies. Most finite element application packages today are based on linear meshing and potential to take advantages of well practiced curve and surface techniques in CAD such as NURBS (Non-Uniform Rational B-spline) curves and surfaces, etc.

It is important that an adaptive mesh refinement module within finite element analysis communicates with the CAD data during each refinement iteration. However, this is often not the case, i.e. often the adaptive mesh refinement module has no communication with the CAD data (Hughes et al. 2005) and hence the shape of the computational domain is fixed by the initially generated mesh, shape of which remains unchanged in the adaptive mesh refinement process. In this thesis we propose a solution to address this limitation.

We propose techniques to improve the adaptive refinements of two dimensional (2D) and three dimensional (3D) finite element meshes so that the mesh, as refined, preserves the geometric structure of the domain specifically around the boundary curve (2D) or surface (3D). The solution is designed to be an integral part of the software for automated solution of differential equations, FEniCS (2009), which is a free and open source software. The implemented geometry interface stores the geometric information about the problem space and provides important information for mesh generation and refinement. There is also an implementation of an interface to enable FEniCS to use standard data exchange techniques. These features will improve the mesh generation and refinement capabilities of the FEniCS software.

1.1 Related Works

The importance of considerations of exact geometrical shape within finite element mesh refinements has drawn attention of the researchers within computational techniques. Accurate mesh generation according to the CAD geometry has always been within interest in this field. The initiatives can be roughly divided in two categories. First methods are to use the exact geometry information in finite element mesh generations and refinements. In this technique the finite element analysis is kept unchanged and some techniques are used so that the shape of the mesh, as refined, converges to the original geometry. Although the idea is simple and straightforward, there are a number of rather complex issues should be addressed with sophistication. Second category methods are to modify the finite element method itself or its formulation, for example, by changing the finite elements attached to the curved boundary. The solutions proposed in is thesis can be put in the first category. Although we do not use the other methods in this thesis, we discuss about the related

works in this direction to keep the reader informed and we also believe the work in this direction will improve the finite element analysis process. We emphasize that both the techniques have strengths and weaknesses and it can be a good idea to keep both the functionalities within a finite element solution package.

Zhou & Lu (2005) proposed a method for mechanical analysis for deformable bodies by combining NURBS geometric representation and the Galerkin method. They applied the so called NURBS-FEM to skeletal muscle modeling. They extended the NURBS surface bounding a 3D body to a trivariate NURBS solid by adding another parametric domain represented by additional control points. The so called NURBS solid defines the interior of the object along with the surface boundary.

Hughes et al. (2005) proposed to match the exact CAD geometry by NURBS surfaces, then construct a coarse mesh of NURBS elements. These are the solid elements in 3D that exactly represent the geometry. They introduced a new analysis framework, called isogeometric analysis, which is based on NURBS and has similarities and as well as dissimilarities with finite element analysis. Their refinement process can proceed without interaction with the CAD system, a distinct advantage over finite element procedures in which mesh refinement strategies require interaction with the CAD system at each stage. Isogeometric analysis has emerged from the idea that the act of modeling a geometry exactly at the coarsest levels of discretization significantly simplifies the refinement process (Cottrell 2007). Bazilevs et al. (2006) advocated the use of the NURBS based isogeometric analysis for vascular applications as a viable alternative to the standard finite element approach. The key motivating factors, in different of finite elements, is that NURBS are able to compactly and accurately represent smooth exact geometries and NURBS-based isogeometric analysis is inherently a higher-order technique in comparison to low-order finite elements.

Sevilla et al. (2008) proposed an improvement to the classical finite element method, named NURBS-enhanced finite element method (NEFEM), which is able to exactly represent the geometry by means of usual CAD description of the boundary with NURBS. They used a standard finite element interpolation and integration for the internal elements while elements intersecting the NURBS boundary needed a specifically designed piecewise polynomial interpolation and numerical integration (also see Sevilla 2009). The method is intended to similar goal as Hughes et al. (2005), but, as claimed, it is simpler since NURBS are restricted to the boundary of the computational domain and only the boundary of the computational domain is directly related with the CAD.

The recent research trends in this field are active and exciting. It is potential to find a CAD influenced mesh generation methodology, specially for the curved surface boundary cases, within the finite element software packages. However, this field is comparatively new and require more analysis and yet to be proved robust to be implemented within a regular finite element solution software. A rather straightforward solution can be to move the new discretization points on the surface boundary edges on to the geometry. Using this method surface boundary of the mesh will

converge to the geometry as the mesh is refined. We do not, however, argue that this method does not have robustness and efficiency related issues. For example, it is possible that this method produces inverted cells within the finite element mesh which makes the mesh inappropriate for a finite element solution. However, we show that state of the art techniques within this field can effectively minimize these problems and it is possible to use these techniques within adaptive mesh refinements of a finite element solution process.

Our proposed techniques are based on points projection on NURBS curves and surfaces. There are a good number of scientific literatures available on solving the point projection problem. There are examples of successful attempts to solve these problems in the 1980's and 90's. However, the field is still alive and there are research groups around the world who are actively contributing and examples within the scientific literatures are available in the recent time. The point projection techniques based on Newton iterations are efficient enough to find the point projection locally. However, a good initial guess about the projection is required to achieve the solution which is sufficiently accurate. There are numerous examples of efforts continuously made to improve the algorithms to find an efficient way to obtain the initial guess which will give the result accurately with high computational efficiency. We will give a survey on the literatures in this topic in the section 2.1.1.

1.2 Motivation

The motivation of solving the stated problem, that the mesh refinement process does not consider the exact geometry of the domain, is clear and settled. Clearly one will be solving a different problem rather than the problem in interest if the mesh is only a coarse linear approximation of the curved geometry and the mesh is not adapted according the underlying geometry of the domain. Either techniques like Isogeometric analysis or NEFEM or other similar techniques should be employed to consider the exact geometry of the domain at the coarsest level of the mesh or some technique should be used so that the shape of the mesh is changed with mesh refinements according to the underlying geometry under consideration. In the case we want to consider changing the elements near the boundary or over the whole domain, changes will be required in the numerical integration and other parts within the finite element solver. Adaptive mesh refinements based on a posteriori error estimates is also an area which is to be adapted with the changes. This will require comparatively much more changes within the existing process and also a significant test requirements will needed to be considered.

As stated already, in this thesis we consider changing the shape of the mesh as the mesh is refined the higher degrees of freedom at the boundary of the mesh allows the mesh to capture a better approximation of the geometry. The key advantage of this technique is simplicity in the sense that it does not require major changes in the existing finite element solver. We just plug in the newly developed tool, which may perform stand alone with some data from the adaptive mesh refinement,

in the process. Obtaining the data is not any considerable overhead on the mesh refinement process and demands only a little involvements. As the newly introduced tool in our design can perform stand alone it is easy to test the impact since the tool can be easily disabled in the pipeline and we are then back to what we had earlier to understand whether this newly introduced tool is responsible for any unexpected behavior of the finite element solver.

1.3 Outline

This thesis is organized as follows. Chapter 2 will describe the proposed techniques and algorithms and chapter 3 will give the details of the implementation. Chapter 4 shows the computational results and chapter 5 gives the conclusions and future directions. We will give a brief introduction to curves and surfaces in appendix A. Appendix B will give an overview of the techniques for point projection and inversion on a NURBS curve or surface. Appendix C gives an introduction to the Initial Graphics Exchange Specification (IGES) in the context of this thesis.

Chapter 2

Mesh Boundary Smoothing

The mesh boundary smoothing techniques presented in this thesis are based on vertex repositioning where the nodes in the boundary of the mesh, that are not on the geometry, are repositioned on to the geometry. Orthogonal point projection from the vertex to the curve segments (2D) or surface patches (3D) has been used to obtain the cartesian coordinate of the point where the vertex is to be repositioned. Although the idea is simple and straightforward, efficiency and accuracy of the orthogonal projection of a vertex are the key points to be addressed. We first start with giving an overview of the point projection on NURBS curves and surfaces. Then we outline our proposed algorithms for boundary smoothing.

2.1 Point Projection on Curves and Surfaces

A point projection problem is to find the nearest point on a curve or surface from a given point outside of the curve or surface. The nearest point can be thought as the point on the curve or surface which gives the minimum Euclidian distance from the given point. Often the point projection problem is defined in scientific literature (e.g. Chen et al. 2008) as

Definition 1 *The distance of a point $x_0 \in \mathbf{R}^n$ to a closed set $C \subseteq \mathbf{R}^n$, in some suitable norm $\|\cdot\|$, is defined as*

$$\mathbf{dist}(x_0, C) = \inf\{\|x_0 - x\| | x \in C\}$$

Then for any point $z \in C$ which is the closest to x_0 is referred as the projection of x_0 on C, were z closest to x_0 is analogous to the expression $\|z - x_0\| = \mathbf{dist}(x_0, C)$.

We note that, in general, there can be more than one projection of x_0 on C, i.e. several points in C closest to x_0. However, in some special cases, it is known a priori that the projection of a point on a set is unique. This happens, for example, if C is closed and convex and the norm is strictly convex (e.g. the Euclidian norm) (see Boyd & Vandenberghe 2004). Within this thesis, we assume that there exists always at least one such point.

Now we define the point projection problem for a NURBS curve (or surface) as

Definition 2 *Given a NURBS curve* $\mathbf{C}(u)$ *(or surface* $\mathbf{S}(u, v)$*) and a point* $\mathbf{P} = (x, y, z)$*, a point projection problem is to find the point* $\mathbf{P}'$ *lying on* $\mathbf{C}(u)$ *(or for the surface case,* $\mathbf{S}(u, v)$*) so that the Euclidian distance between* $\mathbf{P}$ *and* $\mathbf{P}'$ *is minimized.*

As it is often common, we are also interested to find the corresponding parameters which gives the closest point.

Often a situation may occur where the given point $\mathbf{P}$ actually lies on the curve $\mathbf{C}(u)$ (or surface $\mathbf{S}(u, v)$). In that case we solve the problem namely '*point inversion problem*', which is actually finding the parameters on the curve (or surface) related to the point, i.e. finding the corresponding parameter $\bar{u}$ for a curve $\mathbf{C}$, such that $\mathbf{C}(\bar{u}) = \mathbf{P}$ or analogously, for the surface case, finding the parameters $\bar{u}$ and $\bar{v}$ such that $\mathbf{S}(\bar{u}, \bar{v}) = \mathbf{P}$.

2.1.1 Point Projection Algorithms: a Literature Review

Zhou et al. (1993) proposed to first subdivide the rational B-spline curve segment or surface patch into a number of rational Bézier curves or patches and then reformulating the problem in terms of solution of n polynomial equations with n variables expressed in the tensor product Bernstein basis. According to Zhou et al., the local numerical techniques based on variations of Newton-Raphson iteration or numerical optimization had been in use within CAD applications requiring high accuracy because of the efficiency and ease of implementation. A details of point projection algorithms for NURBS curves and surfaces using Newton's iterations is available in Piegl & Tiller (1996). However, it is well known that the effectiveness of the Newton iterations based algorithms depends significantly on the initial guess. Piegl & Tiller (1996) suggested to evaluate the curve points at n equally spaced parameter values on each candidate span and compute the distance for each point from the test point $\mathbf{P}$. Then one would accept the projected point for the sequence of candidate solutions which gives the minimum Euclidian distance. For a NURBS surface, an equivalent algorithm starts with dividing the NURBS surface patch parameters in both u and v directions. However, this method is computationally expensive for complex geometries as one will need to perform many iterations over the whole geometry where only one parameter interval will give the nearest point. Most of the research works for finding point projection using Newton's iterations are done towards obtaining a good initial guess efficiently. The goal is to eliminate the parameter intervals which do not contain the projection point and then apply the Newton iterations on the remaining parameter intervals (Chen et al. 2008).

Dyllong & Luther (1999) presented a two-steps algorithm for distance calculation between a point and a NURBS curve or surface. In the first step, the NURBS curve is decomposed into an array of the rational Bézier curves and then some evaluations of suitable scalar products decide on further subdivision of the rational Bézier curve segments. Elber & Kim (2001) presented an approach for solving a set of geometric constraints represented in multivariate rational functions. Piegl &

Tiller (2001) suggested to turn the NURBS curve or surface into line-segments or quadrilaterals and then to project the point on to the nearest line or quadrilateral and finally estimate the parameters from the closest quadrilateral. However, Ma & Hewitt (2003) argued that decomposing the NURBS surface into quadrilaterals and finding the closest quadrilateral is computationally expensive. They proposed subdividing the NURBS curve segment or surface patch into a set of Bézier curve segments or surface patches and then based on the relationship between the test point and the control polygon of the Bézier curves or the control point net of the Bézier surfaces the candidate Bézier curve segment or surface patch is selected and also the point projection is approximated. Finally, they compare the distance between the test point and candidate points the closest point is evaluated. They used Newton-Raphson method to improve the accuracy of their algorithm. Although, the technique based on the control polygon for a Bézier curve may fail for the non-planar case (see Chen et al. 2007), it can be still in use for many cases, for example, most of the 2D geometry in finite element analysis are based on planar curves.

Johnson & Cohen (2005) gave another method which is based on the tangent cone method. Their technique is to perform a robust search for all distance extrema between a point and a spline model. They employed geometric operations rather than numerical methods to find all local extrema. However, the methods Zhou et al. (1993) and Johnson & Cohen (2005) used for root-finding have two disadvantages, i.e. finding multiple roots is numerically more sensitive than finding single root and needs more computations than finding a single root and secondly, one would not require to compute all roots to find the nearest point (see Chen et al. 2008). Chen et al. (2008) used an elimination circle, with the test point as its center point, to eliminate the curve parts outside a circle. The radius of the initial circle is taken based on some heuristics where the radius of the elimination circle becomes smaller and smaller during the subdivision process. Xu et al. (2008) proposed a technique for finding the closest point on a parametric curve which also gives a good initial value for the Newton-Raphson iteration. Their method combines the arithmetic for Bernstein-form polynomial with subdivision and minimization, and employs the convex hull property of the Bézier curve to eliminate the parts of the parameter domain excluding the roots. Chen et al. (2009) used a spherical clipping method to compute the minimum distance between a point and a clamped B-spline surface where Liu et al. (2009) proposed a second order geometric iteration algorithm, namely '*torus patch approximation approach*', for point projection and inversion on parametric surface.

From the above literature review, we see that numerous research works are done towards solving the point projection problems. Newton iteration based solutions often attracted interests. However, due to sensitivity to the good initial guess, often other methods appeared. Often Newton iteration based algorithms are used in combination with other subdivision techniques since for sufficiently good initial guess Newton iteration performs well.

Algorithm 1: Boundary Smoothing on Mesh Adaptation

```
Input: Refined mesh M
Output: M after boundary smoothing
Extract boundary mesh B from M;
foreach vertex v ∈ M do
    if v ∈ B then
        Find the projection point p of v on to the geometry;
        Move v to p;
    end
end
return M
```

2.2 Boundary Smoothing Algorithm

Algorithm 1 outlines the basic idea for the boundary smoothing of a refined mesh. We iterate over all the vertices of a given mesh. For each vertex on the mesh boundary, which is expected to be on the geometry, nearest point from the vertex on to the geometry is obtained. The vertex is then repositioned to the obtained nearest point on to the geometry. Clearly the nearest point projection is the orthogonal projection point from the vertex to the geometry.

In this thesis we have employed the point projection and inversion algorithms based on Newton iteration which are described in Piegl & Tiller (1996). An overview of the techniques are given in the appendix B and for a more details description we refer to Piegl & Tiller (1996). In section 2.3 below we describe our proposed technique for obtaining the initial guess to start the Newton iteration.

2.3 Neighbor Information based Searching

The Newton-Raphson method is well-known for its efficiency and accuracy provided that a good initial guess of the solution is available. Hence, we had a focus on obtaining the initial guess to start the Newton iteration. For this purpose, we exploit the neighborhood information among the mesh vertices to obtain a trimmed region on the curve segment or surface patch which are much smaller comparing to the whole curve segment or surface patch. We require to initialize a mesh before we can apply the boundary smoothing using this technique.

2.3.1 Initializing a Mesh

The proposed idea is based on the prerequisite that all the nodes on the curved boundary of the initial mesh must be already on the geometry and we must know the corresponding parameters of the curve or surface for these points. This information is ideally available from the mesh generation process or can be obtained by applying an initialization process for the coarse initial mesh which effectively per-

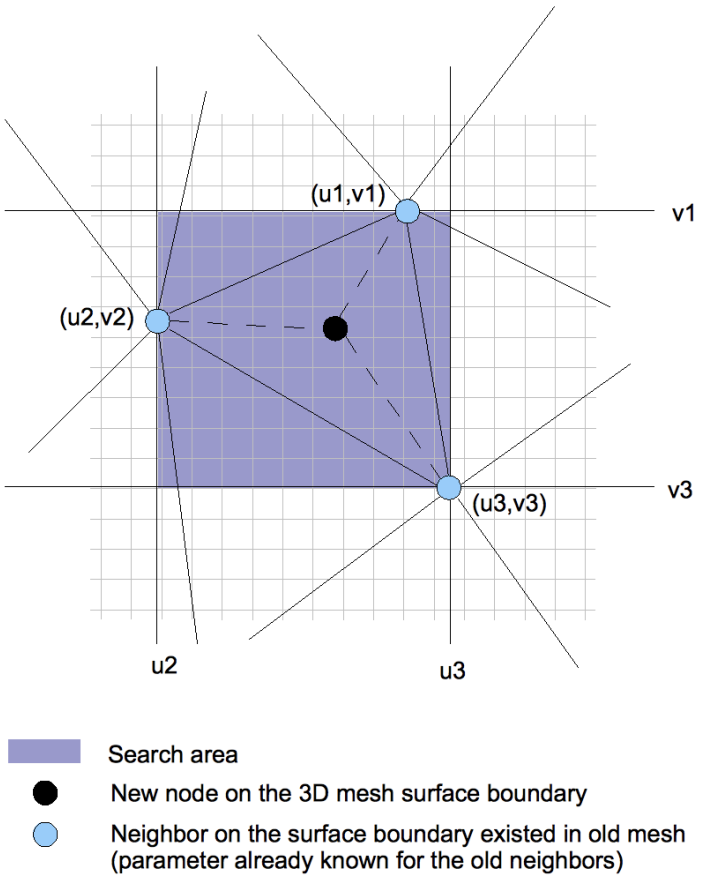


Figure 2.1. Neighbors' parameter based search area for new mesh nodes. Projection for the new node is searched based on the parameters of its directly connected neighbors.

forms an exhaustive search for point projection or inversion on the whole geometry by using a fine discretization of the parametric space of the curve or surface. For this initialization we followed the suggestion given by Piegl & Tiller (1996) to evaluate the curve or surface points at equally spaced parameter values on each candidate span and compute the distance for each point from the vertex point. However, one may want to use the other improved methods stated in the section 2.1.1.

2.3.2 Storing Information with Mesh Nodes

The point projection or inversion parameters for each boundary vertex of a finite element mesh is required to be stored for using the proposed technique. This can be easily done within FEniCS applications using the so called '*mesh functions*' as we will see in the implementation section. For a geometry defined by multiple NURBS curve segments or surface patches, we will also require to store the index of the

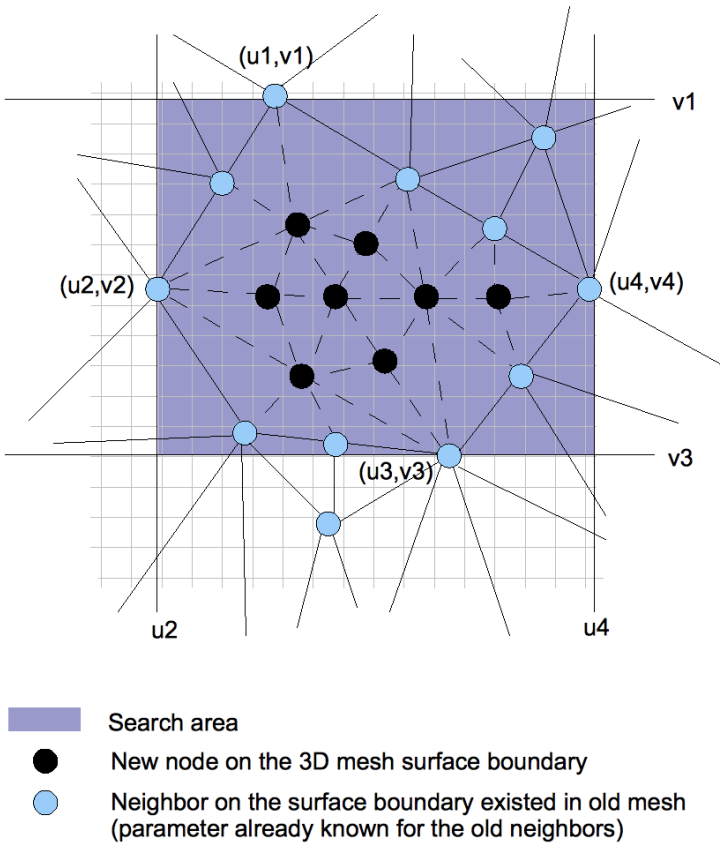


Figure 2.2. Neighbors' parameter based search area for new mesh nodes. If new nodes are connected to other new nodes, depth first search based nearest neighbors common in both old and new meshes are searched and that parameters are used.

related NURBS. Hence for the 2D case, each boundary vertex will require to store the 2-tuple (n, u) where n is the index of the curve segment the boundary vertex resides on and u is the corresponding parameter. Similarly in 3D, each boundary vertex will require to store the 3-tuple (n, u, v). We assume the connectivity among the NURBS curve segments or surface patches are available from the CAD data.

For the initial coarse mesh we either have this data available from the CAD data or fill this information during the initialization process. During mesh refinements we retrieve the stored data with the old mesh and for the new mesh if a vertex is common between the old and refined meshes we update data from the information retrieved from the old mesh data or, for the new nodes, we simply fill the related fields.

Algorithm 2: Find Neighbors' Parameters

Input: Vertex v
Input: Associativity information between the vertices of old and refined meshes, vertex_map
Input: Parameters assigned to boundary vertices of old mesh, old_bnd_params
Input: Number of vertices in old mesh, num_vertices_old
Input: Neighbors set parameters, N (initially empty)
Output: Neighbors set parameters, N
if *v is already visited* **then**
  return *false*;
end
if *v not on boundary or on other boundary* **then**
  return *false*;
end
Include v in the visited vertices list;
if *v was also available in old mesh* **then**
  Include old parameter value of the vertex in N (use vertex_map);
end
else
  foreach *Vertex v_1 on the boundary connected to v* **do**
    Recursively call "Find Neighbor Parameters" for v_1;
  end
end
return *true*;

2.3.3 Finding and using Neighbors' Information

For the simple case, a new mesh point on the mesh boundary may have all the connected nodes which existed in the previous mesh, i.e. the mesh before refinements (see figure 2.1). In that case we simply take the minimum and maximum parameter values (computed in earlier iteration) and search for the point projection in the trimmed region bounded by the minimum and maximum values of the neighbor parameters. Now, we also discretize this comparatively much smaller region which improves the efficiency and accuracy of point projection significantly.

However, practically it may happen that once a mesh is refined two or more new nodes are actually direct neighbors (see figure 2.2). In such cases all these new nodes do not have valid point projection or inversion parameters information. In such cases for a new node we will require to search for further for the nearest neighbors which had been available in the old mesh, i.e. the mesh before refinement took place. A depth first search has been used to find the nearest neighbors which are from the old mesh. Algorithm 2 illustrates the basic idea has been used for retrieving neighbors' information.

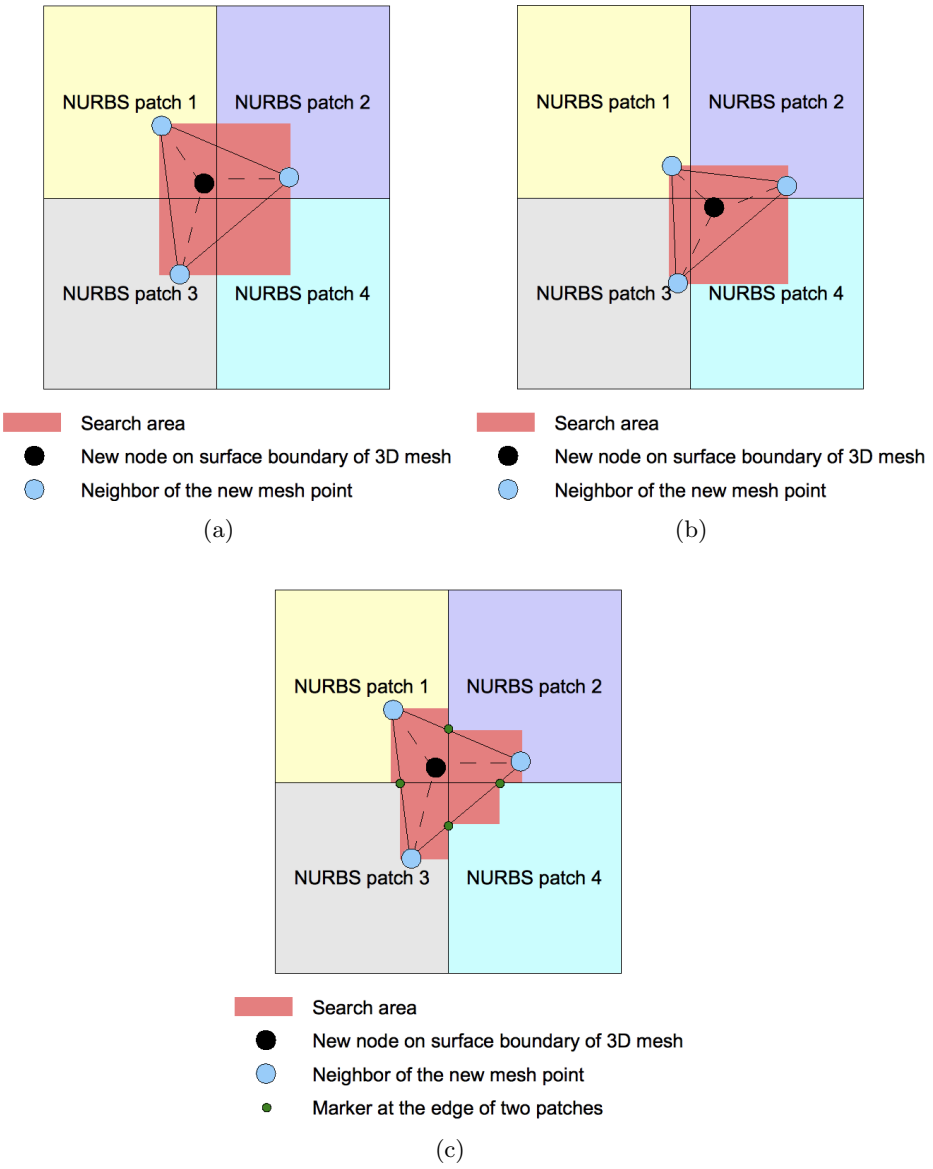


Figure 2.3. Neighbor information based searching on multiple patches, (a) A new node with neighbors in different patches, (b) It is required to search in the fourth patch although the new node doesn't have any neighbor there, (c) Shows how the search region can be slightly further narrowed down.

2.3.4 Searching in a NURBS Array

If we put a prerequisite on the initial coarse mesh that one element face cannot have connected vertices distributed in two or more different NURBS curve segments or surface patches except the case when a vertex is shared by two or more adjoining

NURBS at the edges or corners, we may simplify the neighbor based searching requirements for a geometry consisting of multiple patches. This is often assumed within the finite element mesh generation codes (Sevilla 2009, ch 2.1). However, it may happen that the mesh is actually created independently using a different geometry definition. For example, it is often common that the geometry is defined using Bézier, etc. curves or surfaces or using an implicit representation. In such cases, we will need to search for a point projection on two or more NURBS curve segments or surface patches. This can be done using some simple calculations to decide on the trimmed area on the each possible NURBS and then searching in all the possible trimmed regions which may be distributed in more than one NURBS curve or surface. Figure 2.3 shows the simple cases for point projection searching in several NURBS surface patches where the patches are simply connected at the edges.

2.3.5 Time Complexity

For an M-points discretization of the parametric space of the search region of the NURBS curves or surfaces, for each point projection, the time complexity is $\mathcal{O}(MN)$. Here, N is the maximum Newton iteration try allowed for a single point projection which is some predefined constant value or could be varied. Clearly, to search projection for n points the time complexity is $\mathcal{O}(nMN)$.

Here, for searching a point on multiple NURBS curves or surfaces, M represents the sum of the total number of initial guess points for all the NURBS. The same complexity may apply for both the brute-force searching during mesh initialization process or neighbors' information based searching in mesh refinements. However, as one may expect, the value of M during mesh initialization will be much larger as we search on the whole geometry and clearly a much smaller M will work for searching in the trimmed regions during mesh refinements.

For the ideal cases, we expect that a new mesh node will have direct edges with only those boundary nodes that were common in the old mesh before refinements. In such cases, we can directly get the neighbors' information. For the other cases, we may actually use a depth limiting search so that the search does not continue more than the depth l where l is a predefined small integer. By this way we can minimize the neighbor searching time as one can expect that sufficiently valid information can be retrieved from the neighbor nodes within a depth of a few levels.

Chapter 3

Implementation

This chapter gives the details of the implementation within this project. The related software packages are introduced first and then we introduce the implementation details of our solution.

3.1 FEniCS

FEniCS (2009) is a free software for automated solution of differential equations. This is a collection of software tools for working with computational meshes, finite element variational formulations of PDEs, ODE solvers and linear algebra. DOLFIN and UNICORN are the two software components within FEniCS that are within interest in the context of this thesis. The vertex repositioning based boundary smoothing tool is developed to become an integrated part of the FEniCS applications i.e. DOLFIN, UNICORN, etc. FEniCS applications, i.e. DOLFIN and UNICORN, etc. has the adaptive mesh refinement functionalities based on an a posterioiri error estimation from the dual solver. However, without the boundary smoothing during refinements, the coarse initial mesh with coarse elements around the geometric boundary remains the same shaped for the entire solution. Currently this problem is avoided by taking the initial mesh with fine mesh elements near the geometry so that the mesh captures the geometry well from the beginning or sometimes by moving the boundary mesh data using some ad-hoc techniques for some simple geometries. Initial fine mesh is often used to compute the solutions, however, if we can look into the mesh and geometry in the micro level, there may be still improvement opportunities in the mesh around the curved geometric surface since the mesh is only a linear approximation of the geometry and it cannot support the curved boundary of the geometry exactly. The vertex repositioning based boundary smoothing during mesh refinements improves the mesh around the geometry if the elements are refined to achieve finer elements.

3.1.1 DOLFIN

DOLFIN (Logg et al. 2009) is a C++/Python interface of FEniCS providing a problem solving environment for ordinary and partial differential equations. DOLFIN has the functionalities of simplex meshes in 1D, 2D (triangles) and 3D (tetrahedra) including adaptive mesh refinements using an a posteriori error estimation based on the solution of dual problem and the residuals. A details discussions on the computational meshing features within DOLFIN is given by Logg (2009) where the details on the adaptive algorithm for mesh refinements based on an a posteriori error estimation is given by Hoffman (2009).

3.1.2 UNICORN

UNICORN (Hoffman et al. 2009) is an adaptive finite element solver within FEniCS for fluid and structure mechanics, including fluid-structure interaction problems. UNICORN is dependent on DOLFIN and hence shares some features of DOLFIN, e.g. adaptive mesh refinements based on an a posteriori error estimates. In this project, we have integrated the developed tool mainly in the finite element solving process using UNICORN and also the experimented simulations are done using UNICORN. The simulations are mainly the same as Hoffman (2009) with changes in mesh refinements in the sense that the vertex repositioning based mesh boundary smoothing is applied after each adaptive mesh refinements which takes place at the end of each run of the finite element solver.

3.2 NURBS++

NURBS++ (Lavoie 2002) is an open source C++ library for manipulating NURBS curves and surfaces. The library hides the basic mathematics of NURBS. The NURBS++ package includes the following parts

- A *NURBS library* containing the NURBS related classes,
- A *matrix library* offering the basic mathematical operations used by all the other libraries,
- An *image manipulation library*, and
- A *numerical library* offering least squares fitting, SVD and statistical functions.

A list of features offered by the NURBS++ library is available in Lavoie (2002) where Lavoie (1999) gives an introduction on how to use the NURBS curve manipulation routines of the library. The features that are in particular attractive for our project are the implementation of the point projection and inversion algorithms given in Piegl & Tiller (1996).

3.3 Implementation of the Geometry Library

A simple object oriented geometry library is implemented in C++ which implements the basic geometric features required for DOLFIN or UNICORN to use the boundary smoothing during adaptive mesh refinements.

The geometry library contains the basic routines for input from and output to the IGES (US PRO 1996) format CAD files containing NURBS curve segments or surface patches. The IGES (Initial Graphics Exchange Specification) defines a data format that allows the digital representation and exchange of information among CAD systems. The IGES is one of the widely used file formate in the CAD industries. An introduction to the IGES in the context of the this project is given in appendix C and for the details we refer to US PRO (1996).

Output routines of the NURBS curves and surfaces to VTK (see Schroeder et al. 2006) format files are implemented in the geometry library. We have implemented the core VTK output routines in the NURBS++ library to maintain the uniformity since the NURBS++ library already had a few output routines to write the images into postscript, VRML, etc. formats. The XML syntax data format for the VTK files has been used. The newly implemented geometry library contains a wrapper of the VTK output routines implemented in NURBS++ and also gives some additional features to enable parallel VTK files having multiple NURBS curve segments or surface patches. Then we have used Paraview (Kitware 2009) to view the VTK files.

Point projection and inversion routines are implemented in the geometry library. These routines uses the functions available in the NURBS++ library. Also a new function for point projection on the NURBS surface has been implemented in the NURBS++ library based on the algorithm given in Piegl & Tiller (1996). The point projection functionalities within the geometry library considers the whole geometry for searching the nearest projection point where the NURBS++ routines are responsible for performing Newton iterations for a single curve segment or surface patch for a given single initial value.

3.4 Integration with FEniCS

3.4.1 Implementation of the Boundary Smoothing Tool

The boundary smoothing tool developed in this project is designed to perform stand alone on the refined mesh. However, the mapping between the vertices of the refined mesh and the old mesh (from which the refined mesh is generated) is required from the adaptive mesh refinement module of FEniCS. For the point projections the boundary smoothing tool calls the routines of the geometry library.

Once the mesh boundary vertices are repositioned, to avoid development of thin elements near the boundary, positions of the internal vertices of the mesh are adjusted using the smoothing algorithm already available within FEniCS. This smoothing function within FEniCS does not impact the boundary vertices of the

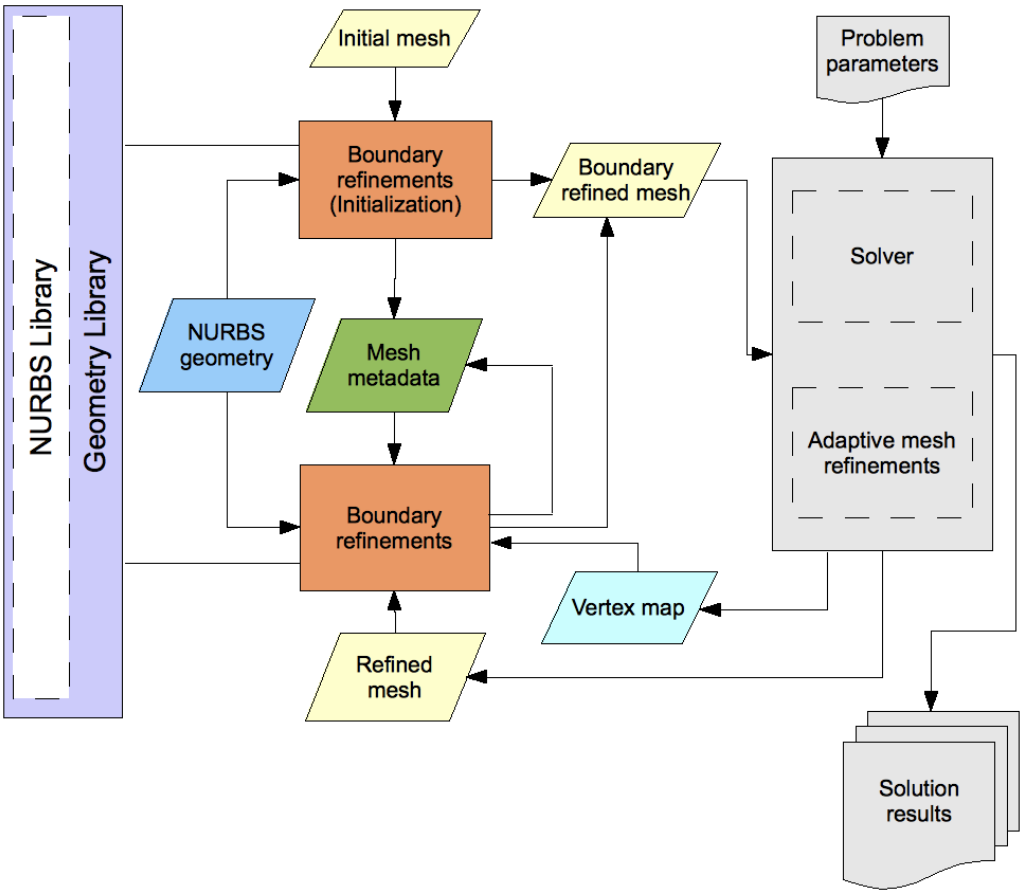


Figure 3.1. Flowchart showing how the boundary refinement tool is integrated with FEniCS finite element solver

mesh and hence the shape of the mesh is not changed due to this.

Minor modification of the mesh refinement function is done within UNICORN so that the vertex mapping is available with the refined mesh. We have used the '*mesh function*' feature of the mesh module within FEniCS for storing the vertex mapping from a refined mesh vertices to the old mesh vertices. Within FEniCS, a mesh function is defined as a discrete function that takes a value on the set of mesh entities of a given fixed dimension $0 \leq d \leq D$ (Logg 2009). A C++ class `MeshFunction` object stores a single array of N_d values on the mesh entities of a given fixed dimension d. This minor changes may not cost any considerable overhead on the finite element solver or adaptive mesh refinement module within FEniCS. The vertex map information is written into a text file.

The refined mesh and the vertex map are given to the boundary refinement tool as input along with the geometry and mesh metadata, i.e. projection parameters evaluated for the old mesh using the boundary projection tool. However, for a

newly created mesh a vertex map and mesh metadata are not available. In that case the initial mesh and the corresponding geometry are given to the boundary projection tool and the mesh metadata with the projection parameters is prepared for use in the next iteration. Mesh functions of FEniCS have been used to store the point projection parameters as well as the index of the NURBS curve segment or surface patch with each boundary vertex of a mesh. For a point projection on a NURBS curve, a single projection parameter is stored for each vertex where for a point projection on a surface, two parameters are used and two mesh function objects are used for that purpose. The respective index of the NURBS are stored using another mesh function object. The mesh functions are stored within text files which are later used to project the new node parameters for the refined mesh.

Explicit checking for the development of inverted cell is done at the end of mesh smoothing to ensure the mesh is valid for finite element analysis. Once an inverted cell is found, the whole mesh is rejected.

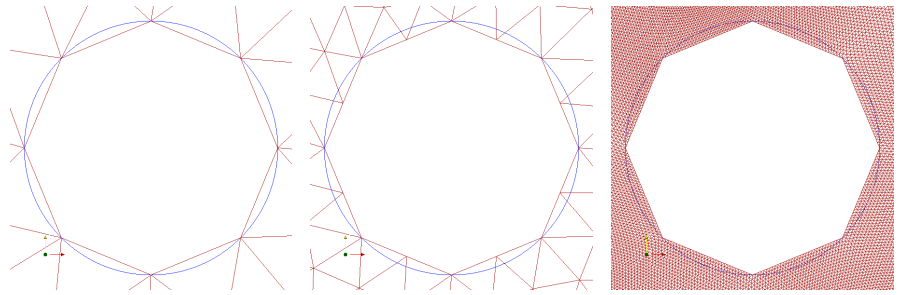
3.4.2 Running the Systems

In the current process flow for solving a problem using UNICORN, once a solver is configured, a mesh along with the required parameters are given as input. Based on the given parameters the dual problems are solved using finite element method and based on the dual solution and the residuals an a posteriori error estimate is measured. According to the a posteriori error estimate the mesh is refined and then the solver continues to perform the same cycle in the next iteration using the refined mesh. The boundary smoothing tool is plugged in the process to perform after the mesh is adaptively refined. Figure 3.1 shows the flow chart of the boundary refinement process (i.e. the boundary smoothing tool) integrated with the UNICORN finite element solver. Once the boundary smoothing is performed the refined mesh is fed back to the solver for the next iteration.

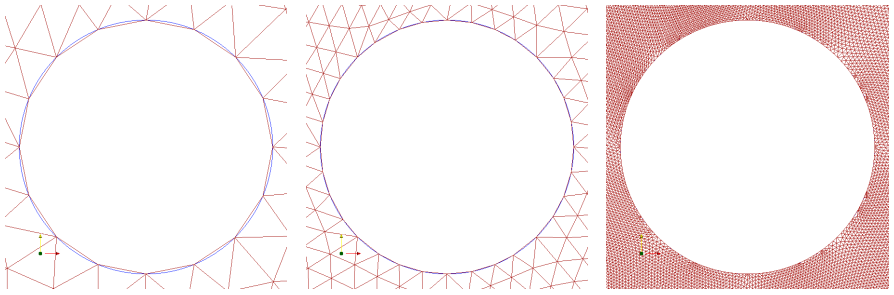
Chapter 4

Results

From the experimental results we see that the vertex repositioning based boundary smoothing works practically well for the simple cases of 2D and 3D geometries. We start with a very coarse mesh and apply mesh refinements along with the boundary smoothing to show the effectiveness of the proposed technique.



(a) Initial coarse mesh (b) After one refinement (c) After five refinements

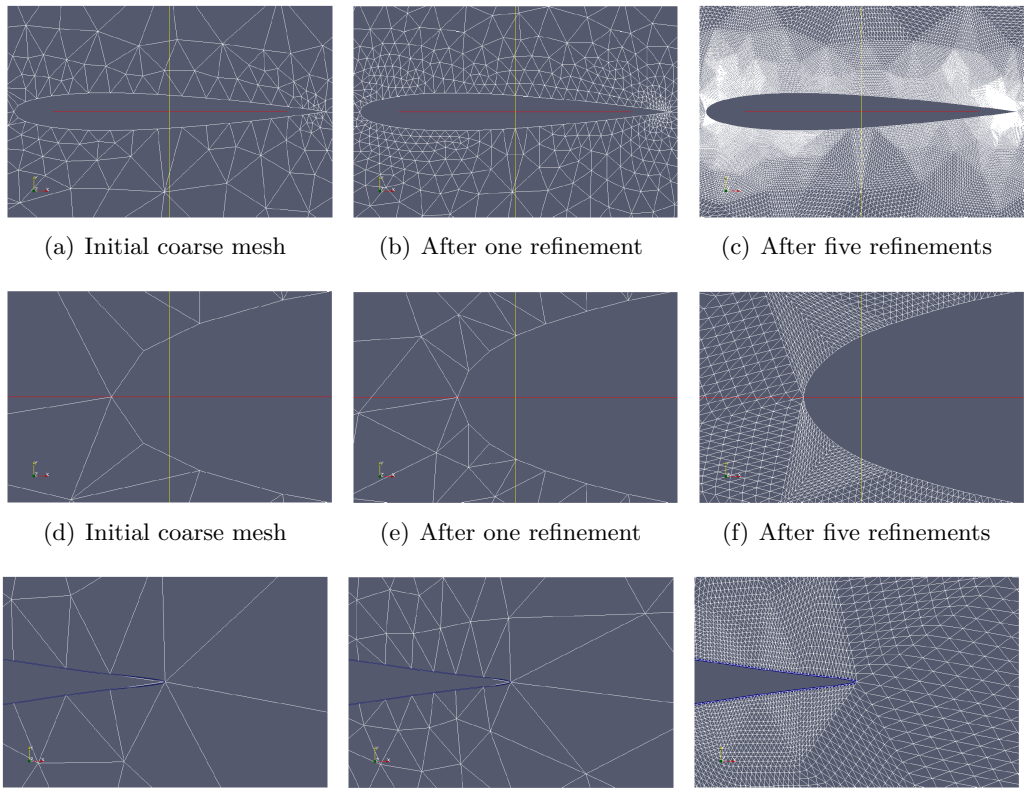


(d) After one refinement and boundary projections (e) After two refinements and boundary projections (f) After five refinements and boundary projections

Figure 4.1. Uniform mesh refinements for a circular cylinder in 2D using DOLFIN, (a) Shows the initial coarse mesh, the blue circle gives the exact geometry which is approximated by the mesh, (b)-(c) Show the result of mesh refinements around the body without boundary projections and (d)-(f) Show the mesh refinements with boundary projections

Figure 4.1 shows how the mesh is re-shaped near the geometric boundary as the mesh is uniformly refined for the simple 2D cylinder case. In the figure (see 4.1(b)-4.1(c)), we see that without the boundary smoothing during refinements, the coarse initial mesh with coarse elements around the geometric boundary remains the same shaped for the entire solution where the vertex repositioning based boundary smoothing captures the geometry well as we see in the figure (see 4.1(d)-4.1(f)). We see that with five mesh refinements we achieved a much better approximation of the 2D cylinder geometry.

Figure 4.2 shows the vertex repositioning based boundary smoothing of curved geometry for a case of 2D NACA airfoil in the case of uniform mesh refinements. As we observe in the figure the curved geometry is captured much more accurately as the mesh is refined. Clearly we have more smoothing done in the front as well as the rear parts of the airfoil where the the geometry is more curved than the upper and bottom parts of the airfoil. It is to be noted that, for the NACA airfoil geometry representation we used total 38 NURBS curve segments. It is possible



(a) Initial coarse mesh (b) After one refinement (c) After five refinements

(d) Initial coarse mesh (e) After one refinement (f) After five refinements

(g) Initial coarse mesh (h) After one refinement (i) After five refinements

Figure 4.2. Uniform mesh refinements using DOLFIN and boundary points projections for a NACA airfoil in 2D, (a)-(c) Whole airfoil, (d)-(f) Front part of the airfoil, (g)-(i) Rear part of the airfoil

that a smaller number of curve segments can effectively represent the NACA airfoil, however, since the original source of the geometry were represented in Bézier curves, a direct conversion from Bézier curves to NURBS gave the 38 NURBS curve segments. Please also note that, as neighbor information based point projection is not implemented within this project, this projection is done using a search in all curve segments, i.e. by discretizing parameter spaces of all the curve segments, for a single point projection using Newton iteration. However, for the 2D cases, we have computationally much less expensive problem comparing to the 3D cases, hence performance issues were not observed.

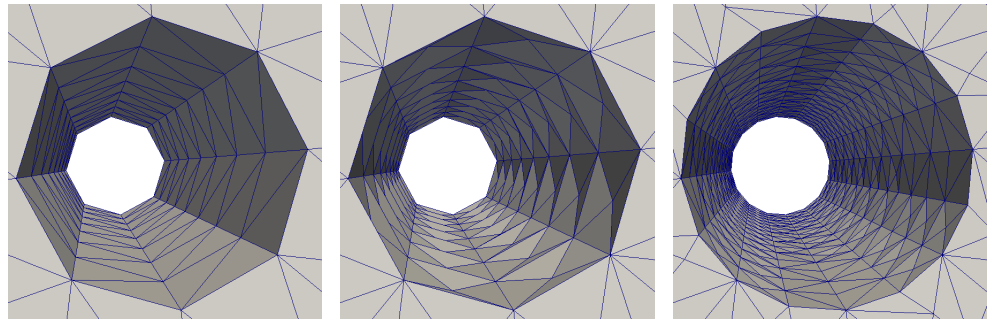
Figures 4.3 and 4.4 show the result of mesh refinements with boundary smoothing for the 3D cylinder case. Similarly for the 2D cylinder case, we see that the mesh captures the geometry quite well with the mesh refinements and boundary smoothing iterations. For the experiments for the 3D cylinder case, we marked the boundary elements near the cylinder for refinements before each refinement iteration. Hence, we have a uniform mesh refinement near the boundary.

Table 4.1. Projection time performance for uniform mesh refinements around the mesh boundary near to geometry of 3D cylinder (case A: projection without considering the neighbor information and searching on the whole geometry, case B: projection using neighbor information to narrow down the search area). Please note that the mesh was only refined around the boundary near the cylinder geometry.

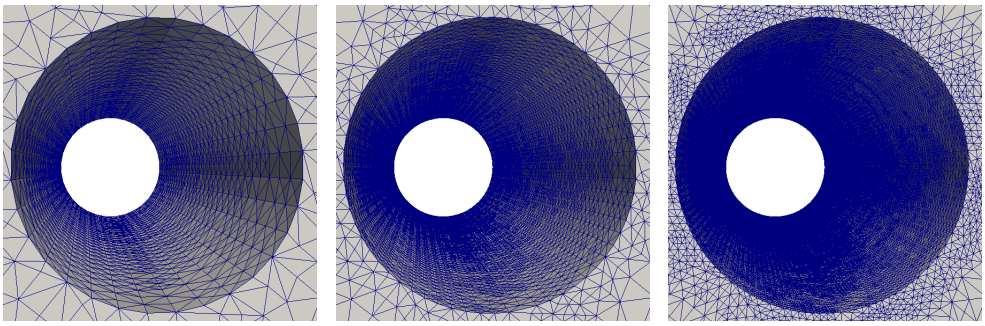
Old mesh points	**New mesh points**	**New points added**	**Boundary points projected**	**Time for case A (seconds)**	**Time for case B (seconds)**
9,159	9,427	268	64	1.13	0.43
9,427	10,011	584	88	0.59	0.59
10,011	10,903	892	196	2.27	1.29
10,903	15,952	5,049	504	7.25	3.38
15,952	21,983	6,031	1,162	7.61	7.67
21,983	35,614	13,631	2,142	19.35	14.07
35,614	69,978	34,364	5,820	120.97	39.49
69,978	146,517	76,539	1,3121	225.50	87.85
146,517	314,120	167,603	28,212	563.78	185.91
314,120	691,079	376,959	68,158	1,637.78	477.51

4.1 Performance

Table 4.1 shows the time performance for running the program in a single computing node for the brute-force and neighbor information based search techniques. The performance measurement is done in a server within the Computational Technol-

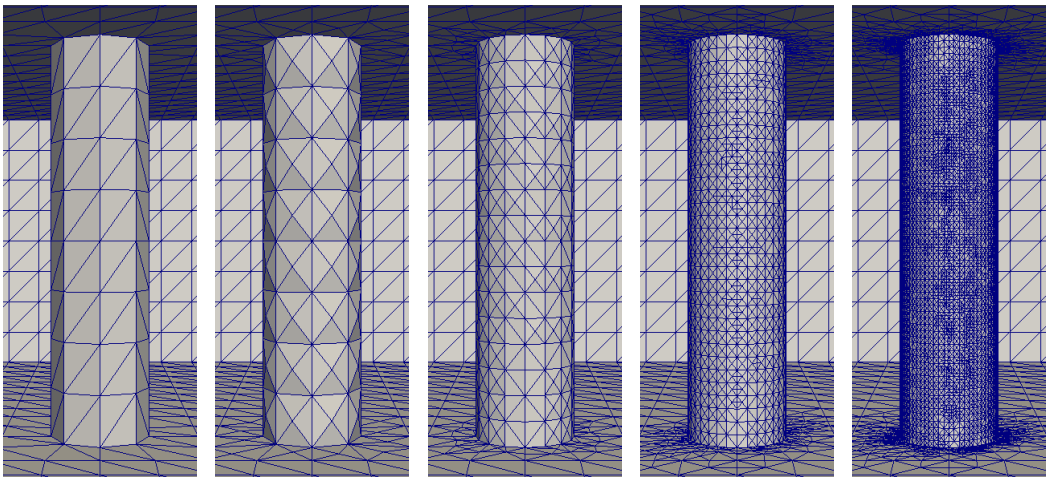


(a) Initial coarse mesh (b) 2 refinements (c) 4 refinements



(d) 6 refinements (e) 8 refinements (f) 10 refinements

Figure 4.3. Uniform mesh refinements around the surface boundary with boundary projections for a circular cylinder in 3D, (a) Initial coarse mesh, (b)-(f) Refined mesh with boundary projections



(a) Initial mesh (b) 2 refinements (c) 4 refinements (d) 6 refinements (e) 8 refinements

Figure 4.4. A different view on uniform mesh refinements around the surface boundary with boundary projections for a circular cylinder in 3D, (a) Initial coarse mesh, (b)-(e) Refined mesh with boundary projections

Table 4.2. Distribution of iterations required for convergence for points projection on a 3D cylinder geometry. Maximum 1000 iterations allowed, hence, the frequencies in the row of 1000 iterations apparently represents non-convergence (case A: projection without considering the neighbor information and searching on the whole geometry, case B: projection using neighbor information to narrow down the search area)

Iterations count	**Case A frequency**	**case B frequency**
1-5	11,802,941	12,066,155
6-10	47,270	12
11-15	1,606	0
16-20	296	0
21-25	100	0
26-30	10	0
31-35	30	0
36-999	0	0
1000	213,914	0
Total	12,066,167	12,066,167

ogy Laboratory (CTL)[1]. The server has four Dual-Core AMD Opteron(tm) 2 GHz processors and overall 4.12 GB system memory. The system runs on Ubuntu Linux. The GNU C++ Compiler version 4.3 is used.

From the experimental results we found that the time requirements per iteration is approximately constant. For the test cases more than 95% of the iterations are done within a range of 10-12 nano seconds. Given the iteration time is constant, one would be interested to play with the limit of maximum allowed iterations since limiting the maximum iterations within relatively small number would bound the running time for each point projection for a given initial guess smaller.

We argue that it can be possible to limit the maximum iterations within few tens of numbers (at least for simple cases such as 3D cylinder, etc.) as the table 4.2 shows that for the point projection without neighbor information based search maximum iterations required are within 35 (34, to be exact) for successful cases and otherwise there are a considerable number of cases where 1000 iterations are used, which is the maximum allowed iterations for a projection search. We observe that no projection required iterations from 35 to 999. We got support to believe that if a point projection is not found within a sufficiently small number it is less likely that we will reach the solution within finite number of iterations. This is because the Newton iterations are good for searching local optima and very sensitive to the initial guess.

From the other case where neighbor node information is used to narrow down the search area, we find the performance is much better as we see in table 4.2. For this case, actually a maximum of 7 iterations were required for 12 point projection cases.

[1]Computational Technology Laboratory WWW link: `http://ctl.csc.kth.se/`

The technique improved the point projection search since this makes the Newton iteration to search within sufficiently local region. In this case only a maximum seven iterations are used by all the 1.2066167×10^7 cases. Moreover, most of the projections are reached within only two iterations.

Once we set the maximum allowed iterations to a sufficiently low number, time per search with each initial guess will be bounded and we would be interested in minimizing discretization points of the narrowed down search area.

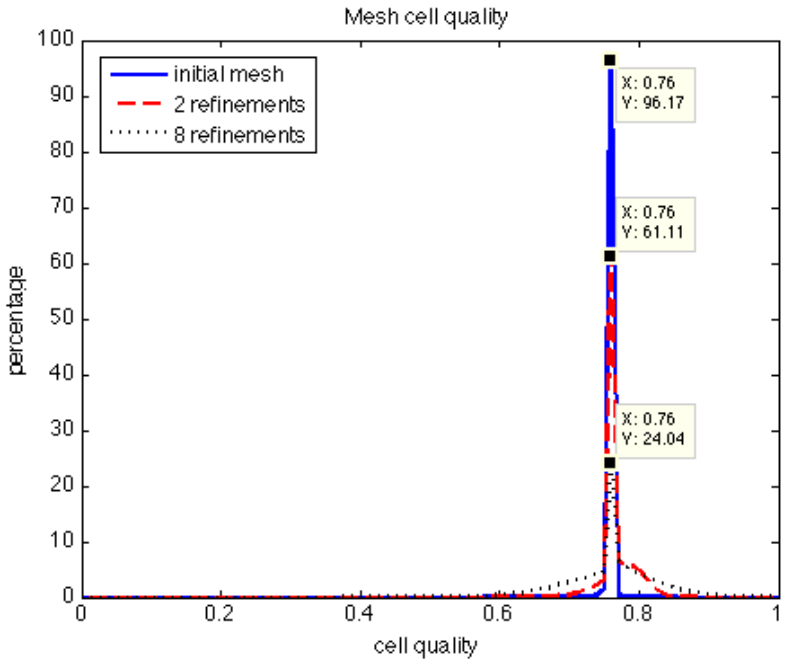


Figure 4.5. Cell quality based on affine cell mapping to a reference cell for the 3D cylinder after the mesh refinements and boundary projections. Adaptive mesh refinements in UNICORN based on the dual solution is used for the refinements.

4.2 Mesh Quality

Avoiding inverted cells is a must have requirement for a finite element mesh. There are other features a quality mesh should have. A quality mesh is very important in finite element analysis since a mesh without sufficient quality may negatively impact on the accuracy and the efficiency of the solution. Knupp (2007) discussed about mesh quality in the context of both finite difference and finite element methods. Although Knupp (2007) emphasized on underlying problem oriented quality measurements, in this thesis we focus on mesh quality measurement independent of the problem, rather based on the available mesh only.

Boundary smoothing using point projection based on the neighbor information successfully avoided creation of any inverted cell for the test cases discussed in this

thesis. We have used 10×10 discretization points in the parameter space of the trimmed region for the 3D case.

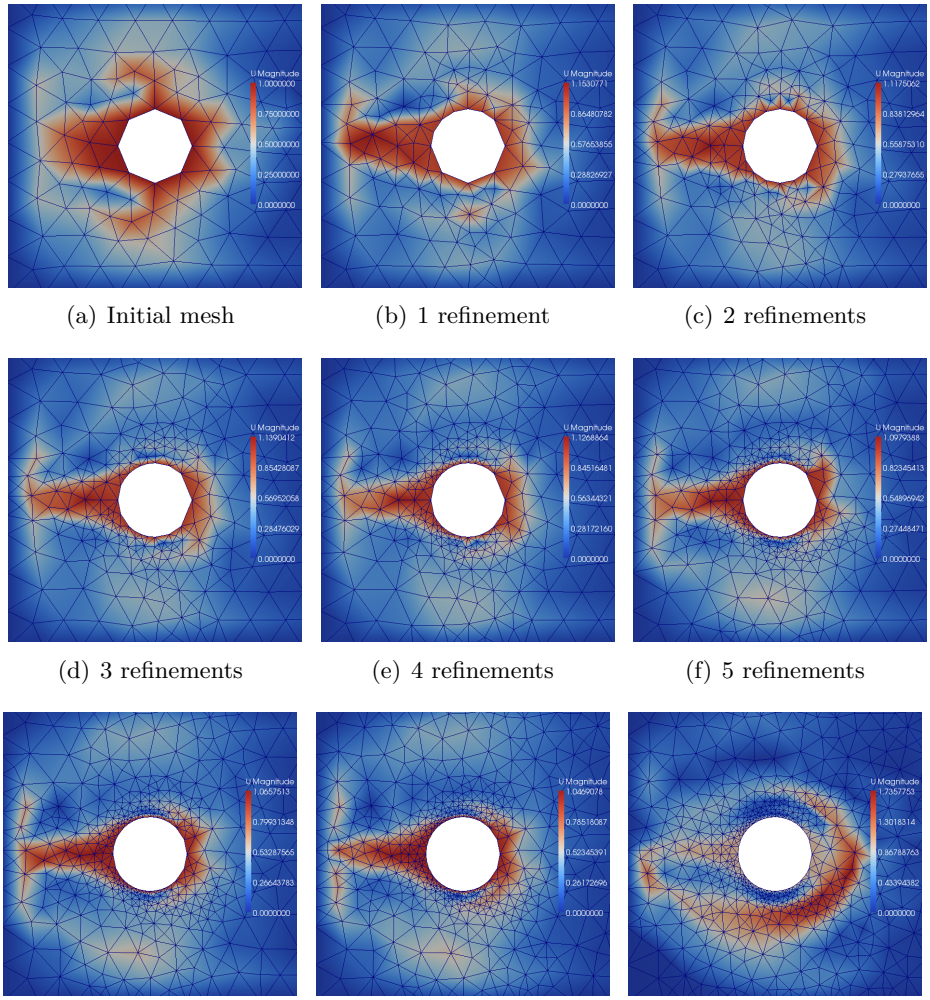
UNICORN has been used to measure the cell quality where the mean-ratio metric (see Diachin et al. 2006) is implemented. Here the cell quality q_e can have a range of values $0 < q_e \leq 1$ with $q_e = 1$ only when the element attains the ideal reference shape. Figure 4.5 shows the cell quality after boundary smoothing after the mesh is refined around the boundary for the 3D cylinder case. The diffusion of the cell quality distribution from the initial mesh was as expected and we do not see the algorithm to have any major impact on the mesh quality.

Radius ratio based cell quality measurement also shows that the boundary smoothing done well in maintaining the cell quality as starting the the coarse mesh and applying 20 adaptive mesh refinements iterations gives a maximum radius ratio 2.1437 when boundary smoothing is not done and on the other hand using boundary smoothing the radius ratio is 1.9246. Also after 25 adaptive iterations the values are 2.1436 and 1.9563 respectively. Surprisingly in these two cases using the vertex repositioning based boundary smoothing along with the refinements performed better than the refinements without boundary smoothing. However, please note that in the boundary smoothing tool the internal mesh is smoothed as well which helps the mesh quality. We also note that the mesh refinements for this geometry without boundary smoothing gives a more stable radius ratio than using boundary smoothing as for the case after 10 refinements the values are respectively 2.1437 and 2.5301 where the cell quality result is favored to the non-boundary smoothing case. However, the mesh here had a very rough approximation of the 2D cylinder that should have helped the mesh refinement without boundary smoothing to maintain a fixed cell quality as the cells near the curved geometry are often irregular than a coarse linear approximation of the geometry.

4.3 Boundary Smoothing with Adaptive Refinements

Figure 4.6 shows the mesh refinement iterations using adaptive mesh refinement features available in UNICORN based on an a posteriori error estimates. From the figure we see that the mesh refinements is not uniform around the geometry. This is what one should expect from adaptive mesh refinements. However, we see that due to this reason the boundary smoothing is not uniform either and it took quite a few adaptive refinements iterations to obtain a shape of the mesh which captures the geometry better. However, we once observe more closely, we may see that comparatively larger mesh elements are available on the rear end of the cylinder and that causes more linear shape of the mesh. We emphasize that such elements will eventually be refined if the mesh is adaptively more refined and as the mesh will be refined the vertex repositioning based boundary smoothing technique will use the increased degree of freedom to capture the geometry better.

Adaptive mesh refinements based on an a posteriori error estimate using dual solutions refines a percent of the mesh elements which is often non-uniformly dis-

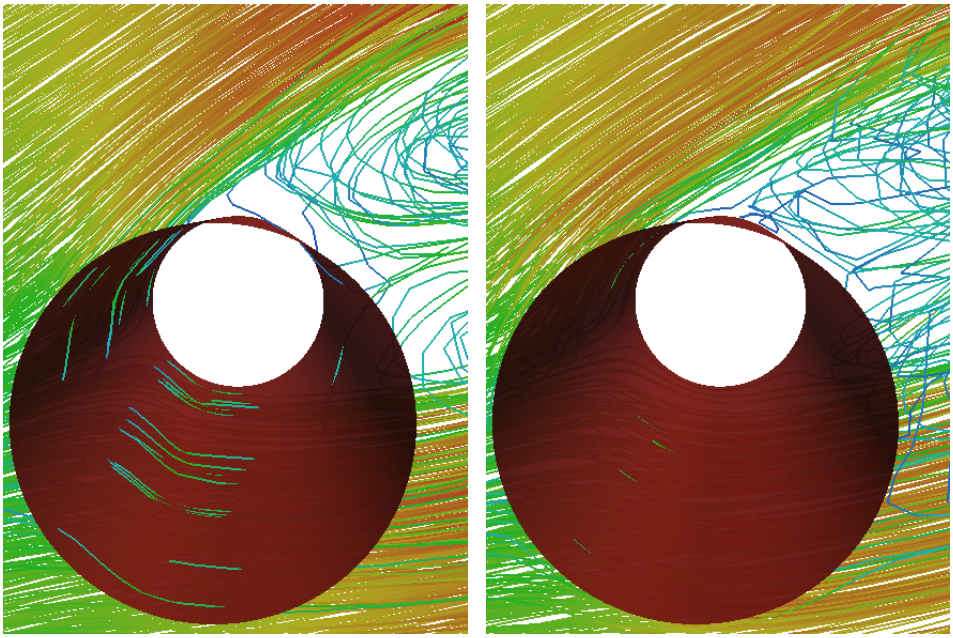


(a) Initial mesh (b) 1 refinement (c) 2 refinements

(d) 3 refinements (e) 4 refinements (f) 5 refinements

(g) 6 refinements (h) 7 refinements (i) 8 refinements

Figure 4.6. Adaptive mesh refinements based on the solution of the dual velocity problem using the dual solver provided by UNICORN. Boundary projections are done for boundary smoothing after each refinements by adaptive refinements. It took a few iterations step to the mesh to capture the geometry of the circular cylinder quite well due to non-uniform mesh refinements around the body of the geometry. Plots show the dual velocity solution and the corresponding mesh.



(a) Mesh with rough geometry (b) Mesh closer to the geometric shape

Figure 4.7. Differences in flow streamlines due to differences in geometric shape of the mesh, (a) Shows how the streamlines going through a region which is actually inside the cylinder which is outside of the domain, (b) Shows how a better mesh, that captures the geometry better, gives an improved solution.

tributed. In that case the cells refinements around the geometry is not uniform and which causes elements of different sizes around the geometry. This is often problem specific since for a flow past a cylinder problem one would expect that the mesh elements above and below of cylinder are finer than the elements in the front and rear parts of the cylinder (flow is from the left) as we observe in the figure 4.6 for the 2D cylinder case. In the figure we see that some elements near the cylinder body are much larger comparing to other elements. Such larger elements cause sharp edges since if there are not enough number of mesh points near the boundary the curved shape of the geometry may not be captured.

Figure 4.7 shows the changes in the streamlines near the curved boundary of the geometry in the case of 3D cylinder. As we expect the coarse mesh that captures the geometry roughly, results wrong path of the streamlines which apparently go through the cylinder, i.e. the flow goes through the outside of the domain. Once the mesh is refined with boundary smoothing we get a better result.

Chapter 5

Conclusion

In this thesis we have presented a vertex repositioning based boundary smoothing technique for the finite element mesh. Experimental results show that using the proposed technique it is sufficient to start with a coarse mesh all over the domain and adaptively refine the mesh and the coarse mesh, as refined, captures the curved geometry more accurately.

In the proposed technique, the boundary vertices are repositioned on the geometry and the shape of the mesh captures the geometric shape of the domain with a higher accuracy as the mesh is refined. Newton iteration based iterative closest point search algorithms have been used to search point projection of a vertex on to the CAD geometry represented in NURBS curves (2D) or surfaces (3D). A technique is proposed for obtaining sufficiently accurate initial guess region which is narrow enough comparing to the whole geometry. Once a mesh is refined the new boundary vertices are projected on to the geometry using neighbor nodes' projection data for obtaining more accurate initial guess for the Newton iteration. Projection parameters for the original coarse mesh are gathered for a single time using exhaustive search for points projection on the whole geometry.

Experimental results show that the method is sufficiently accurate and efficient for both 2D and 3D problems for some simple test cases. The process of boundary smoothing takes a minor fraction of time in comparison to the finite element solution and adaptive mesh refinements in the sense that the boundary smoothing is done within seconds to minutes where the solver runs for hours to days.

Since our techniques does not change the mesh topology and the finite element solver or mesh elements are not required to be changed, the developed tools can be plugged in to the finite element solver without much involvements. Although the tools are developed to be integrated parts of FEniCS applications, the parts of geometric data manipulation can be used with other applications as well.

5.1 Future Works

In this thesis we have considered orthogonal point projection for vertex repositioning. However, orthogonal point projection may sometimes result thin elements. This is within the scope of future improvements to consider other projections where an investigation on orthogonal and other projection techniques will be required.

One major missing feature is that we did not consider the trimmed NURBS. Trimmed NURBS are often necessary in CAD since it is a well-used tool when two or more NURBS patches are attached. In CAD using NURBS often one may encounter difficulties to represent a surface using NURBS due to is rectangular definition. Trimmed surfaces are commonly used to overcome such difficulties. A trimmed surface is an ordinary tensor product surface along with a restricted parameter domain which is given by a trimming curve (Casciola & Morigi 1999). Efremov (2004) described an algorithm for '*trimming test*' which effectively identifies whether a point on a NURBS surface patch lies within the trimmed region. Sederberg et al. (2008) discussed on the problems of having gaps when two NURBS surfaces are attached using trimmed NURBS method and proposed a resolution for that.

Within our application, if a projected point on a NURBS surface lies within the trimmed region, the point projection has to be rejected since the trimmed region of a NURBS surface does not represent the outer surface boundary of a finite element mesh. Also, our implementation does not support geometry with multiple NURBS curves or patches. A natural continuation of this project may be to remove the limitations by considering the cases where a surface geometry contains multiple NURBS patches connected at the edges or using trimmed NURBS.

The initial projection is done using brute-force technique in this thesis. Although, it is to be used once with the initial mesh to prepare the mesh metadata for the boundary projections, one would be interested to improve the algorithm by using some search heuristics.

Bibliography

Angel, E. (1999), *Interactive Computer Graphics: A Top-Down Approach with OpenGL*, 2nd edn, Addison-Wesley.

Bazilevs, Y., Zhang, Y., Calo, V. M., Goswami, S., Bajaj, C. L. & Hughes, T. J. (2006), 'Isogeometric analysis of blood flow: a NURBS-based approach', *CompIMAGE, Coimbra, Portugal* pp. 20–21.

Boyd, S. & Vandenberghe, L. (2004), *Convex Optimization*, Cambridge University Press.

Casciola, G. & Morigi, S. (1999), The trimmed NURBS age, *in* 'Advances in Computation: Theory and Practice; Recent Trends in Numerical Analysis'.

Chen, X.-D., Su, H., Yong, J.-H., Paul, J.-C. & Sun, J.-G. (2007), 'A counterexample on point inversion and projection for NURBS curve', *Computer Aided Geometric Design* 24(5), 302.

Chen, X.-D., Xu, G., Yong, J.-H., Yong, J.-H. & Paul, J.-C. (2009), 'Computing the minimum distance between a point and a clamped B-spline surface', *Graphical Models* 71(3), 107–112.

Chen, X.-D., Yong, J.-H., Wang, G., Paul, J.-C. & Xu, G. (2008), 'Computing the minimum distance between a point and a NURBS curve', *Computer-Aided Design* 40, 1051–1054.

Chou, J. J. & Logan, M. A. (1995), NASA-IGES translator and viewer, Technical Report 19950022354, NASA, USA.

Cottrell, J. (2007), Isogeometric Analysis and Numerical Modeling of the Fine Scales within the Variational Multiscale Method, PhD thesis, The University of Texas at Austin, USA.

de Boor, C. (1972), 'On calculating with B-splines', *Journal of Approximation Theory* 6, 50–62.

Diachin, L. F., Knupp, P., Munson, T. & Shontz, S. (2006), 'A comparison of two optimization methods for mesh quality improvement', *Engineering with Computers* 22(2), 61–74.

Dyllong, E. & Luther, W. (1999), Distance calculation between a point and a NURBS surface, *in* P.-J. Laurent, P. Sablonnitre & L. L. Schurnaker, eds, 'Curve and Surface Design', pp. 55–62.

Efremov, A. (2004), Efficient ray tracing of trimmed NURBS surfaces, Master's thesis, University of Saarland, Computer Science Department, Max-Planck-Institut fur Informatik, Computer Graphics Group, Saarbrucken, Germany.

Elber, G. & Kim, M.-S. (2001), Geometric constraint solver using multivariate rational spline functions, *in* 'SMA '01: Proceedings of the sixth ACM symposium on Solid modeling and applications', ACM, New York, NY, USA, pp. 1–10.

Eriksson, K., Estep, D., Hansbo, P. & Johnson, C. (1996), *Computational Differential Equations*, 2nd edn, Cambridge University Press.

FEniCS (2009), 'FEniCS project'.
URL: *http://www.fenics.org/*

Foley, J. D., van Dam, A., Feiner, S. K. & Hughes, J. F. (1995), *Computer Graphics Principles and Practice*, 2nd edn, Addison-Wesley Professional.

Hoffman, J. (2009), 'Efficient computation of mean drag for the subcritical flow past a circular cylinder using general Galerkin G2', *International Journal for Numerical Methods in Fluids* 59(11), 1241–1258.

Hoffman, J., Jansson, J., Jansson, N., Nazarov, M. & Stöckli, M. (2009), 'UNICORN'.
URL: *http://www.fenics.org/wiki/Unicorn*

Hughes, T. J. R., Cottrell, J. A. & Bazilevs, Y. (2005), 'Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement', *Computer Methods in Applied Mechanics and Engineering* 194(39–41), 4135–4195.

Johnson, D. E. & Cohen, E. (2005), Distance extrema for spline models using tangent cones, *in* 'Proceedings of Graphics Interface 2005', Vol. 112, Canadian Human-Computer Communications Society, Ontario, Canada, pp. 169–175.

Kitware (2009), 'Pareview'.
URL: *http://www.paraview.org/*

Knupp, P. M. (2007), Remarks on mesh quality, *in* '45th AIAA Aerospace Sciences Meeting and Exhibit', Sandia National Laboratories, New Mexico, USA.

Lavoie, P. (1999), *An introduction to NURBS++*.
URL: *http://libnurbs.sourceforge.net/user.pdf*

Lavoie, P. (2002), 'The NURBS++ library'.
URL: *http://libnurbs.sourceforge.net/index.shtml*

Liu, X.-M., Yang, L., Yong, J.-H., Gu, H.-J. & Sun, J.-G. (2009), 'A torus patch approximation approach for point projection on surfaces', *Computer Aided Geometric Design* 26(5), 593–598.

Logg, A. (2009), 'Efficient representation of computational meshes', *International Journal of Computational Science and Engineering* .

Logg, A., Wells, G. N. et al. (2009), 'DOLFIN'.
URL: *http://www.fenics.org/dolfin/*

Ma, Y. L. & Hewitt, W. T. (2003), 'Point inversion and projection for nurbs curve and surface: Control polygon approach', *Computer Aided Geometric Design* 20(2), 79–99.

Piegl, L. A. & Tiller, W. (1996), *The NURBS Book*, 2nd edn, Springer.

Piegl, L. A. & Tiller, W. (2001), 'Parametrization for surface fitting in reverse engineering', *Computer-Aided Design* 33(8), 593–603.

Schroeder, W., Martin, K. & Lorensen, B. (2006), *Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*, 4th edn, Kitware.

Sederberg, T. W., Li, X., Lin, H. & Ipson, H. (2008), 'Watertight trimmed NURBS', *ACM Transactions on Graphics* 27(3), 1–8.

Sevilla, R. (2009), NURBS-Enhanced Finite Element Method (NEFEM), PhD thesis, Polytechnic University of Catalonia, Barcelona, Spain.
URL: *http://www-lacan.upc.edu/sevilla/webNEFEM.html*

Sevilla, R., Fernández-Méndez, S. & Huerta, A. (2008), 'NURBS-enhanced finite element method (NEFEM)', *International Journal for Numerical Methods in Engineering* 76(1), 56–83.

Smith, B. M. (1983), 'IGES: A key to CAD/CAM systems integration', *IEEE Computer Graphics and Applications* 3(8), 78–83.

Thompson, J. F., Soni, B. K. & Weatherill, N. P., eds (1998), *Handbook of Grid Generation*, 1st edn, CRC-Press.

US PRO (1996), *The Initial Graphics Exchange Specification (IGES)*, 5.3 edn, IGES/PDES Organization.

Xu, J., Liu, W., Bian, H. & Li, L. (2008), 'Accurate and efficient algorithm for the closest point on a parametric curve', *International Conference on Computer Science and Software Engineering* .

Zhou, J., Sherbrooke, E. C. & Patrikalakis, N. M. (1993), 'Computation of stationary points of distance functions', *Engineering with Computers* 9(4), 231–246.

Zhou, X. & Lu, J. (2005), NURBS-based Galerkin method and application to skeletal muscle modeling, *in* 'Proceedings of the 2005 ACM symposium on Solid and physical modeling', pp. 71–78.

Appendix A

Introduction to Curves and Surfaces

There are three ways commonly used to model curves and surfaces, i.e. the explicit equations, implicit equations and parametric functions. Implicit equations and parametric functions are the two most commonly used methods of representing curves and surfaces in geometric modeling. NURBS (Non-Uniform Rational B-Spline) are categorized within the parametric functions. We consider NURBS as the geometric representation within the thesis. However, we will give brief introduction to other representations to motivate the choice of representation. Other than the reason that NURBS are widely used in the CAD industries, NURBS also offers enhanced flexibility and precision for handling both analytic and freeform shapes. We will give a brief introduction to the three forms below. We refer to Angel (1999), Foley et al. (1995), Piegl & Tiller (1996) for the details on curves and surfaces.

A.1 Explicit equations

The explicit form of a curve in two dimensions gives the value of one variable the dependent variable, in terms of the other, the independent variable. In the two dimensional x, y-space we have the explicit equations of the following form

$$y = f(x)$$

where x is the dependent variable and y is the independent variable and they are connected using the function f. Sometimes it is possible to find the invert relations of the form, $x = g(y)$.

In three dimensions, a surface is represented using one equation of the form

$$z = f(x, y)$$

where we have two independent variables x, y and one dependent variable z, while two equations are required to represent a curve

$$\begin{cases} y = f(x) \\ z = g(x) \end{cases}$$

where we have one independent variable x and two dependent variables y and z.

A major disadvantage of the explicit form of geometry representation is that we don't have guarantee that either form exists for a given curve. For example, for a line represented in the form $y = mx + c$ where m is the slope of the line and c is the y-intercept, we cannot represent a vertical line. Also to represent a simple circle we require two explicit equations for different parts of it. Another example is for a circle where we require two equations for different parts of it, i.e. using explicit representation we can write the equation for half of it, $y = \sqrt{r^2 - x^2}$ and write another equation for the second half. Here, r is the radius of the circle with center at the origin. Similarly, for a line in the three dimension defined using two equations ($y = ax + b$ and $z = cx + d$) would fail to represent a line with constant x and a surface represented by an equation of the form $z = f(x, y)$ cannot represent a sphere since we would require two z points for a given x and y pair which is not defined in such form of equations.

A.2 Implicit equations

Implicit equation of a curve in two dimensional xy plane has the form

$$f(x, y) = 0$$

describing an implicit relationship between the x and y coordinates of the points lying on the curve and a surface in the three dimension is defined by an implicit equation has the form

$$f(x, y, z) = 0$$

For example, a straight line in implicit representation has the form, $ax + by + c = 0$ where a circle, with the center in the origin and radius r, can be represented as $x^2 + y^2 - r^2 = 0$ and a plane in 3D can be represented as $ax + by + cz + d = 0$. Here a, b, c and d are constants. A sphere with the center at the origin can be written as, $x^2 + y^2 + z^2 - r^2 = 0$. Such an implicit equation describes the implicit relationship between the x, y and z coordinates of the points lying on the curve or surface.

Using the implicit representation we test the points (x_i, y_i) to know whether the point is on the curve or not. We can take a point (x, y) and evaluate f to determine whether the point lies on the curve.

Although, most of the curves and surfaces with which we work have implicit representations (Angel 1999), in general, however, it gives us no analytic way to find a value y on the curve the corresponds to a given x, or vice versa. However, the implicit form is less dependent on the coordinate systems than the explicit form, hence, it is possible to represent all lines and circles using the implicit representations.

Another disadvantage is that curves in three dimensions are not as easily represented in implicit form. We require to represent a curve as the intersection, if it

exists, of the two surfaces:

$$\begin{cases} f(x,y,z) = 0 \\ g(x,y,z) = 0 \end{cases}$$

Then we test if a point (x, y, z) lies on both the surfaces and in that case we know that it must lie on their intersection curve.

A.3 Parametric functions

In the parametric form of a curve the spatial variables for the points on the curve is defined in terms of an independent variable u where u is called the parameter. In 2D the we have two explicit functions

$$x = x(u), \quad y = y(u)$$

where in 3D we have

$$x = x(u), \quad y = y(u), \quad z = z(u)$$

One of the advantages of the parametric form is that we simply drop the equation for z from the three dimensional representation to obtain the two dimensional representation. In this sense, the parametric form is the same in two and three dimensions.

Using parametric functions we have the definition of the curve in two dimensional plane as

$$\mathbf{C}(u) = (x(u), y(u)), \quad a \leq u \leq b$$

where a, b are arbitrary constants (usually normalized to $[0, 1]$). Here, $\mathbf{C}$ is a vector-valued function of the independent variable u. It is common to think of $\mathbf{C}(u) = (x(u), y(u))$ as the path traced out by a particle as a function of time where u is the time variable and $[a, b]$ is the time interval.

A surface in the parametric representation is defined as

$$\mathbf{S}(u,v) = (x(u,v), y(u,v), z(u,v)), \quad a \leq u \leq b, c \leq v \leq d$$

As we see that we require two independent variables u and v to define a surface in the parametric representation.

For example, the first quadrant of a unit circle can be defined using the parametric functions

$$\mathbf{C}(u) = (x(u), y(u)) = (\cos(u), \sin(u))^T, \quad 0 \leq u \leq \pi/2$$

or alternatively using $t = \tan(u/2)$, we can represent the same curve using

$$\mathbf{C}(t) = (x(t), y(t)) = \left(\frac{1-t^2}{1+t^2}, \frac{2t}{1+t^2}\right)^T, \quad 0 \leq t \leq 1$$

A sphere in 3D can be defined in parametric representation as

$$\mathbf{S}(u,v) = (x(u,v), y(u,v), z(u,v))^T = (\sin u \cos u, \sin u \sin v, \cos u)^T \qquad (A.1)$$

where $0 \leq u \leq \pi$ and $0 \leq v \leq 2\pi$. From the above representation of the sphere we see that two independent variables u and v are required to define a surface. Holding u fixed and varying v generates the latitudinal lines of the sphere and on the other hand holding v fixed and varying u generates the longitudinal lines.

As we have observed for the circle's representation above, parametric representation of the sphere is not unique. Parametric forms of curves and surfaces representations are not unique. Similarly, any given curve or surface can be represented in different ways. Actually, the representation can be different and different representations have different benefits and disadvantages and one should carefully select the appropriate representation of the CAD model based on the requirements of the applications.

For the coordinate system related difficulties and non-existence of the representation (although the curve or surface exists, the representation is not available), explicit equations are not generally suitable for using in computer graphics and computer aided designing. Implicit and parametric functions are the two most common methods of representing curves and surfaces in geometric modeling. As we will see in the following discussions, there are different representations available within parametric functions.

The parametric form of curves and surfaces is the most flexible and robust in computer graphics, although one could still argue that we have not fully removed all dependencies on a particular coordinate system or frame as we are still using the x, y and z for a particular representation (Angel 1999).

In the paremetric forms in which the functions are polynomials in u for curves, and polynomials in u and v for surfaces, are of most use in computer graphics (Angel 1999). Such forms are called as the parametric polynomial forms of curves and surfaces representation.

NURBS for external data storage and more sophisticated geometry definition. Other representations are mainly for internal use to take advantages of different features that would be important for computations.

A.3.1 Parametric Polynomial Curves

We consider a curve of the form

$$\mathbf{p}(u) = (x(u), y(u), z(u))^T$$

Then a polynomial parametric curve of degree n is of the form

$$\mathbf{p}(u) = \sum_{k=0}^{n} u^k \mathbf{c}_k$$

where each $\mathbf{c}_k = (c_{xk}, c_{yk}, c_{zk})^T$ has independent x, y and z components. Here the $n+1$ column matrices $\{\mathbf{c}_k\}$ are the coefficients of $\mathbf{p}$, which clearly gives $3(n+1)$ degrees of freedom in how we choose the coefficients of a particular $\mathbf{p}$.

An interesting properties of polynomial curves is that there is no coupling, among the x, y, and z components and this allows us to work with three independent equations, each of the form

$$p(u) = \sum_{k=0}^{n} u^k c_k$$

and each with $n+1$ degrees of freedom. Although u can be defined for any range $u_{\min} \leq u \leq u_{\max}$, it is a common practice to consider $0 \leq u \leq 1$.

Polynomials are classified within a widely used class of functions. In this section we will introduce a few methods of expressing polynomial functions. Although they are mathematically equivalent, some of the representations are better than the others in the terms of computer representations and manipulations.

Degree of a parametric polynomial curve must be chosen carefully which is dependent on the applications. Choosing a high degree will give us many parameters to control, i.e. increased flexibility and control over the shape of the curve, but evaluation of points on the curve will be costly. Also there is more danger that the curve will become rougher as the degree of a polynomial curve becomes higher. On the other hand, choosing too low a degree, we may not have enough parameters and we may require to design each curve segment only over a short interval. Cubic polynomials are most often used due to insufficient flexibility in controlling the shape of the curve using lower-degree polynomials and higher-degree polynomials can introduce unwanted wiggles and can be computationally expensive. However, it is noted that, although higher-degrees curves require more conditions to determine the coefficients and can wiggle back and forth in ways that are difficult to control, they are used in applications– such as the design of cars and planes– in which higher-degree derivatives must be controlled to create surface that are aerodynamically efficient.

An nth degree power basis curve is given by

$$\mathbf{p}(u) = (x(u), y(u), z(u))^T = \sum_{i=0}^{n} \mathbf{a}_i u^i, \quad 0 \leq u \leq 1 \tag{A.2}$$

where $\mathbf{a}_i = (x_i, y_i, z_i)$ are vectors, hence

$$x(u) = \sum_{i=0}^{n} x_i u^i, \quad y(u) = \sum_{i=0}^{n} y_i u^i, \quad z(u) = \sum_{i=0}^{n} z_i u^i$$

Equation (A.2) can be written in matrix form as

$$\mathbf{p}(u) = \begin{bmatrix} \mathbf{a}_0 & \mathbf{a}_1 & \cdots & \mathbf{a}_n \end{bmatrix} \begin{bmatrix} 1 & u & u^2 & \cdots & u^n \end{bmatrix}^T = [\mathbf{a}_i]^T [u_i]$$

Here, the $n+1$ functions $\{u_i\}$ are called the *basis functions* (or *blending functions*).

A parametric cubic polynomial curve is a special case of the power basis form of a curve where we have $n = 3$. A cubic parametric polynomial can be given as

$$\mathbf{p}(u) = \mathbf{a}_0 + \mathbf{a}_1 u + \mathbf{a}_2 u^2 + \mathbf{a}_3 u^3 = \sum_{i=0}^{3} \mathbf{a}_i u^i$$

Two prominent examples of cubic polynomial curve representations are Hermite and Bézier representations. The Hermite form of the cubic polynomial curve segment is determined by constraints on the endpoints and tangent vectors at the endpoints, where the Bézier form of the cubic polynomial curve segment is determined by constraints on the endpoints and two intermediate points that, along with the endpoints, specify the tangent vectors at the endpoints.

A.3.2 Parametric Polynomial Surfaces

A parametric polynomial surface can be defined as

$$\mathbf{p}(u, v) = (x(u, v), y(u, v), z(u, v))^T = \sum_{i=0}^{n} \sum_{j=0}^{m} \mathbf{c}_{ij} u^i v^j$$

Here we require to specify $3(n + 1)(m + 1)$ coefficients to determine a particular surface $\mathbf{p}(u, v)$. This is common practice to take $n = m$, and let u and v vary over the rectangle $0 \leq u, v \leq 1$, defining a surface patch. Please note that any surface patch can be viewed as the limit of collection of curves that are generated by holding either u or v constant and varying the other.

A.4 B-Spline Basis Functions

Let $U = \{u_0, \ldots, u_m\}$ be a nondecreasing sequence of real numbers (i.e. $u_i \leq u_{i+1}, i = 0, \ldots, m-1$). Then the ith B-spline basis function of p-degree, denoted by $N_{i,p}(u)$, is recursively defined as

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \tag{A.3}$$

where the base case is defined as

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

Here, u_i is called '*knots*' and U is the '*knot vector*'. The recurrence formula to define the *B-spline basis functions* is the most useful for computer implementation and also widely used within computational geometry.

Although $N_{i,p}(u)$, which are piecewise polynomials, are defined on the entire real line, generally only the interval $[u_0, u_m]$ is of interest. The B-spline basis functions has *local support*, i.e. each $N_{i,p}(u)$ is nonzero only on a limited number of

subintervals, not the entire domain, $[u_0, u_m]$. $N_{i,p}(u) = 0$ if u is outside the interval $[u_i, u_{i+p+1})$. Clearly, in any given knot span, $[u_j, u_{j+1})$, at most $p+1$ of the $N_{i,p}$ are nonzero, namely the functions $N_{j-p,p}, \ldots, N_{j,p}$. The B-spline basis functions are nonnegative, i.e. $N_{i,p}(u) \geq 0, \forall i, p, u$.

A.4.1 B-Spline Curves

A pth-degree B-spline curve is defined by

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

where $\{\mathbf{P}_i\}$ are the '*control points*', and the $\{N_{i,p}(u)\}$ are the pth-degree B-spline basis functions defined on non-periodic (and non-uniform) knot vector,

$$U = \{\underbrace{a, \ldots, a}_{p+1}, u_{p+1}, \ldots, u_{m-p-1}, \underbrace{b, \ldots, b}_{p+1}\},$$

with a total $m+1$ knots. The polygon formed by the control points $\{\mathbf{P}_i\}$ is called the '*control polygon*'. The steps to compute a point on a B-spline curve is fairly straightforward. At a fixed u value, first we require to find the *knot span* in which u lies and compute the nonzero basis functions and finally we multiply the values of the nonzero basis functions with the corresponding control points. de Boor (1972) gave an efficient way of evaluating B-spline basis functions. Based on de Boor's (1972) discussions Piegl & Tiller (1996) formulated the algorithms for finding the knot span and computing the nonzero basis functions and also the properties of the B-spline curves.

$\mathbf{C}(u)$ is a piecewise polynomial curve (since the $N_{i,p}(u)$ are piecewise polynomials) where the degree, p, the number of control points, $n+1$, and the number of knots, $m+1$, related by the relation: $m = n + p + 1$. The control polygon of the B-spline curve can be taken as a piecewise linear approximation to the curve and the approximation is improved by knot insertion or degree elevation (see Piegl & Tiller 1996). As a general rule, the lower the degree, the closer a B-spline curve follows its control polygon. This is because, the lower the degree, the fewer the control points that are contributing to the computation of $\mathbf{C}(u_k)$ for any given u_k. In the extreme case where $p = 1$ we find the curve $\mathbf{C}(u)$ just by taking a linear interpolation between two control points. In that case, the curve is actually the control polygon. Another special case is if $n = p$ and $U = \{0, \ldots, 0, 1 \ldots, 1\}$, then $\mathbf{C}(u)$ is a Bézier curve.

A.4.2 B-Spline Surfaces

A B-spline surface is obtained by taking a bidirectional net of control points, two knot vectors, and the products of the univariate B-spline basis functions. A B-spline

surface, constructed as tensor product of two B-spline curves, is defined by

$$\mathbf{S}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)\mathbf{P}_{i,j}, \quad a \le u \le b, c \le v \le d$$

where $\{\mathbf{P}_{i,j}\}$ are the control points forming the control net, and the $\{N_{i,p}(u)\}$ are the pth-degree B-spline basis functions in the direction of U knot vector and the $\{N_{j,q}(v)\}$ are the qth-degree B-spline basis functions in the direction of V knot vector defined on non-periodic (and non-uniform) knot vectors,

$$U = \{\underbrace{a,\ldots,a}_{p+1}, u_{p+1},\ldots,u_{r-p-1},\underbrace{b,\ldots,b}_{p+1}\}, V = \{\underbrace{c,\ldots,c}_{q+1}, v_{q+1},\ldots,v_{s-q-1},\underbrace{d,\ldots,d}_{q+1}\}.$$

Here U has $r+1$ knots and V has $s+1$ with the relations: $r = n+p+1$ and $s = m+q+1$.

A.5 NURBS

Non-Uniform Rational B-Spline (NURBS) curves and surfaces are widely used for computer-aided design, manufacturing and engineering industries for its advantages over other representations.

A.5.1 NURBS Curves

A pth-degree NURBS curve is defined as

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

where $\{\mathbf{P}_i\}$ are the control points, $\{w_i\}$ are the weights and $\{N_{i,p}(u)\}$ are the pth-degree B-spline basis functions (equation (A.3)) defined on the non-periodic (and non-uniform) knot vector

$$U = \{\underbrace{a,\ldots,a}_{p+1}, u_{p+1},\ldots,,u_{m-p-1}\underbrace{b,\ldots,b}_{p+1}\}$$

Like the B-spline curves, the control points, $\{\mathbf{P}_i\}$, forms the control polygon.

The NURBS curves are often written in the following form

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

where

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

The $\{R_{i,p}(u)\}$ are called the '*rational basis functions*'. They are piecewise rational functions on $u \in [0,1]$.

A.5.2 NURBS Surfaces

A NURBS surface of degree p in the u direction and degree q in the v direction is defined as

$$\mathbf{S}(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)w_{i,j}\mathbf{P}_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)w_{i,j}}, \quad a \le u \le b, c \le v \le d$$

Here $\mathbf{S}(u,v)$ is a bivariate vector-valued piecewise rational function. Here, $\{\mathbf{P}_{i,j}\}$ form a bidirectional control net, $\{w_{i,j}\}$ are weights and $\{N_{i,p}(u)\}$ and $\{N_{j,q}(v)\}$ are the non-rational B-spline basis functions defined on the knot vectors,

$$U = \{\underbrace{a,\ldots,a}_{p+1}, u_{p+1},\ldots,u_{r-p-1},\underbrace{b,\ldots,b}_{p+1}\}, V = \{\underbrace{c,\ldots,c}_{q+1}, v_{q+1},\ldots,v_{s-q-1},\underbrace{d,\ldots,d}_{q+1}\}.$$

Here U has $r+1$ knots and V has $s+1$ with the relations: $r = n+p+1$ and $s = m+q+1$.

The NURBS surfaces are often written in the following form

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

where

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

are the piecewise rational basis functions.

A.6 Derivatives of parametric curves and surfaces

For the point projection and inversion on NURBS curves and surfaces algorithms under considerations in this thesis, we must evaluate the derivatives of NURBS curve. The first and second derivatives of parametric curves $\mathbf{C}(u)$ can be thought of the velocity and acceleration of the particle, respectively (Piegl & Tiller 1996). We refer to Piegl & Tiller (1996) for the details and evaluation of the derivatives of NURBS curves and surfaces.

The first derivative of the parametric curve can be written in the form:

$$\frac{d\mathbf{C}(u)}{du} = \left(\frac{dx(u)}{du}, \frac{dy(u)}{du}, \frac{dz(u)}{du}\right)^T$$

The derivatives of parametric surface $\mathbf{S}(u,v)$, $\partial\mathbf{S}/\partial u$ and $\partial\mathbf{S}/\partial v$

$$\frac{\partial\mathbf{S}(u,v)}{\partial u} = \left(\frac{\partial x(u,v)}{\partial u}, \frac{\partial y(u,v)}{\partial u}, \frac{\partial z(u,v)}{\partial u}\right)^T$$

and

$$\frac{\partial \mathbf{S}(u,v)}{\partial v} = \left(\frac{\partial x(u,v)}{\partial v}, \frac{\partial y(u,v)}{\partial v}, \frac{\partial z(u,v)}{\partial v} \right)^T$$

determine the tangent plane at each point of the surface. Here, $\partial \mathbf{S}/\partial u$ and $\partial \mathbf{S}/\partial v$ respectively gives the velocities along latitudinal and longitudinal lines for the case of the sphere.

The unit normal vector, $\mathbf{N}$ at any point on the surface is defined as below (Piegl & Tiller 1996)

$$\mathbf{N} = \frac{\mathbf{S}_u \times \mathbf{S}_v}{|\mathbf{S}_u \times \mathbf{S}_v|} \tag{A.4}$$

provided that the vector cross product $\mathbf{S}_u \times \mathbf{S}_v$ does not vanish at the point. As we understand that there can be different parameterizations for the same surface, different parameterizations give different partial derivatives. However, the existence of a normal vector at the point, and the corresponding tangent plane, is a geometric property of the surface independent of the parameterization. We will get the same $\mathbf{N}$ in (A.4) independently of the parameterization provided that the denominator does not vanish.

However, it might be impossible to find the normal vector at some points of the surface in some parameterization even though the unit normal vector exists at the point and can be obtained using a different parameterization. We illustrate this with an example taken from (Piegl & Tiller 1996). From equation (A.1) (a non-unique parameterization of a sphere), it can be seen that $\mathbf{S}_v(0,v) = \mathbf{S}_v(\pi,v) = 0, \forall v, 0 \leq v \leq 2\pi$, i.e., $\mathbf{S}_v$ vanishes at the north and south poles of the of the sphere. Although, normal vectors do exist at the two poles, we are unable to compute them using equation (A.4) under this parameterization.

We note that, of the implicit and parametric forms, it is difficult to maintain that one is always more appropriate than other. Both the representations have their advantages and disadvantages and a successful geometric modeling is done using both techniques (Piegl & Tiller 1996).

Appendix B

Point Projection on Curves and Surfaces

Piegl & Tiller (1996) discussed Newton's iterations based straightforward algorithms to solve point projection problem for NURBS curves and surfaces. The algorithms can be categorized as the so called '*iterative closest points*' algorithms or simply ICP. However, as it is common for any Newton's iterations based algorithms, a good initial guess is required for the success of the algorithms which is the point to be addressed for effective applications of these algorithms. In this chapter we give an overview of the algorithms and for the details discussions we refer to Piegl & Tiller (1996). Our techniques for obtaining sufficiently accurate initial guess for the Newton's iterations are discussed in chapter 2.

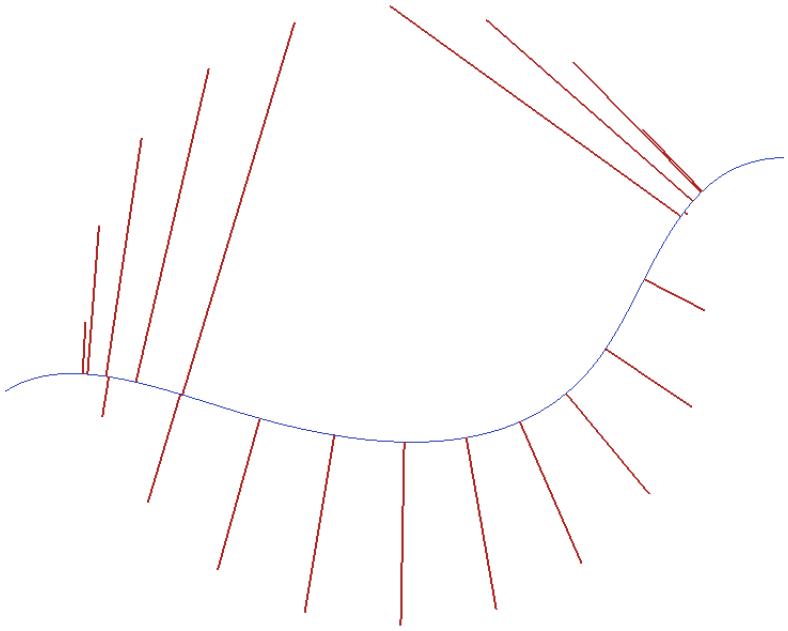


Figure B.1. Points projection on a NURBS curve

B.1 Point Projection on NURBS Curve

Let's assume that we have a initial guess $u_0 \in [u_{\min}, u_{\max}]$ and the minimum distance from a point $\mathbf{P}$ to the curve $\mathbf{C}(u)$ is found for u^*, i.e. $\|\mathbf{C}(u) - \mathbf{P}\| = \min\{\|\mathbf{C}(u) - \mathbf{P}\| | u \in [u_{\min}, u_{\max}]\}$. Then the parameter u^* that gives the minimum distance is found by solving for $f(u) = 0$ where $f(u)$ is defined as the form of the dot product function

$$f(u) = \mathbf{C}'(u) \cdot (\mathbf{C}(u) - \mathbf{P})$$

Now, assuming the start value u_0 and denoting by u_i the parameter obtained at the ith Newton iteration. Then

$$u_{i+1} = u_i - \frac{f(u_i)}{f'(u_i)} = u_i - \frac{\mathbf{C}'(u_i) \cdot (\mathbf{C}(u_i) - \mathbf{P})}{\mathbf{C}'' \cdot (\mathbf{C}(u_i) - \mathbf{P}) + |\mathbf{C}'(u_i)|^2} \tag{B.1}$$

Then the convergence criteria is given by the following inequalities where the convergence is reached if any of the following holds

1. If two points coincides, i.e. $|\mathbf{C}(u_i) - \mathbf{P}| \leq \epsilon_1$ or
2. The parameter does not change significantly, i.e.$|(u_{i+1} - u_i)\mathbf{C}'(u_i)| \leq \epsilon_1$ (this may happen, for example, if the point is off the end of the curve) or
3. If zero cosine reached, i.e. $\dfrac{|\mathbf{C}'(u_i) \cdot (\mathbf{C}(u_i) - \mathbf{P})|}{|\mathbf{C}'(u_i)||\mathbf{C}(u_i) - \mathbf{P}|} \leq \epsilon_2$.

If any of the above conditions is satisfied, the iterations are stopped and we reached to the desired projection point. Otherwise we continue to iterations up to a large finite number of times which we decide based on the problem using some heuristics. Here, to indicate convergence, two zero tolerances ϵ_1 and ϵ_2 are used as measure for Euclidian distance and zero cosine, respectively.

From the iteration equation (B.1) we see that there is no condition that will restrict u_{i+1} from going outside the range $u_{\min} \leq u_{i+1} \leq u_{\max}$. To enforce the parameter to stay within the range $u_{i+1} \in [u_{\min}, u_{\max}]$ we require additional checking on the new parameter (Piegl & Tiller 1996). If the curve is not closed, we simply set $u_{i+1} = u_{\min}$ if $u_{i+1} < u_{\min}$ and $u_{i+1} = u_{\max}$ if $u_{i+1} > u_{\max}$. If the curve is closed we set $u_{i+1} = u_{\max} - (u_{\min} - u_{i+1})$ and $u_{i+1} = u_{\min} + (u_{i+1} - u_{\max})$ respectively for $u_{i+1} < u_{\min}$ and $u_{i+1} > u_{\max}$.

To obtain a start value, u_0, for the Newton's iterations, Piegl & Tiller (1996) suggested to evaluate the curve points at n equally spaced parameters values on each candidate span, and compute the distance of each point from $\mathbf{P}$. We require to choose u_0 to be the value yielding the point closest to $\mathbf{P}$ while the number n is generally chosen by some heuristic method.

Figure B.1 shows examples of point projection on a NURBS curve where the external given points stays on a circular path.

B.2 Point Projection on NURBS Surface

Point projection and inversion for NURBS surface are solved in similar way as for the NURBS curve. Here we are given a NURBS surface $\mathbf{S}(u, v)$ and a point $\mathbf{P}$ and we are interested to find a point on $\mathbf{S}(u, v)$ that minimizes the distance from $\mathbf{P}$. We start with defining the vector function $\mathbf{r}(u, v)$ as

$$\mathbf{r}(u, v) = (S)(u, v) - \mathbf{P}$$

Now we require $\mathbf{r}(u, v)$ to be orthogonal to the partial derivatives $\partial\mathbf{S}/\partial u$ and $\partial\mathbf{S}/\partial v$ and we define two scalar functions $f(u, v)$ and $g(u, v)$ as

$$\begin{cases} f(u, v) = \mathbf{r}(u, v) \cdot \mathbf{S}_u(u, v) = 0 \\ g(u, v) = \mathbf{r}(u, v) \cdot \mathbf{S}_v(u, v) = 0 \end{cases} \tag{B.2}$$

Solving the system (B.2) using an iterative method to obtain the parameters (u_i, v_i) gives the point on $\mathbf{S}(u, v)$ having minimum Euclidian distance from $\mathbf{P}$.

Now, let

$$\delta_i = \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix} = \begin{bmatrix} u_{i+1} - u_i \\ v_{i+1} - v_i \end{bmatrix}$$

$$J_i = \begin{bmatrix} f_u & f_v \\ g_u & g_v \end{bmatrix} = \begin{bmatrix} |\mathbf{S}_u|^2 + \mathbf{r} \cdot \mathbf{S}_{uu} & \mathbf{S}_u \cdot \mathbf{S}_v + \mathbf{r} \cdot \mathbf{S}_{uv} \\ \mathbf{S}_u \cdot \mathbf{S}_v + \mathbf{r} \cdot \mathbf{S}_{vu} & |\mathbf{S}_v|^2 + \mathbf{r} \cdot \mathbf{S}_{vv} \end{bmatrix}$$

$$\kappa_i = - \begin{bmatrix} f(u_i, v_i) \\ g(u_i, v_i) \end{bmatrix}$$

Here, all the functions in the Jacobian matrix J_i are evaluated at (u_i, v_i). At the ith iteration we require to solve the simple 2×2 system of linear equations, $J_i\delta_i = \kappa_i$, to obtain the unknown δ_i which we use to obtain the parameter at next iteration taking $u_{i+1} = u_i + \Delta u$ and $v_{i+1} = v_i + \Delta v$.

Similar to the curve case, we have two zero tolerance parameters ϵ_1 and ϵ_2 that are used respectively for point coincidence and zero cosine checking. The convergence criteria is given by the following inequalities where the convergence is reached if any of the following holds

1. If two points coincides, i.e. $|\mathbf{S}(u_i, v_i) - \mathbf{P}| \leq \epsilon_1$ or
2. The parameter does not change significantly,
   i.e. $|(u_{i+1} - u_i)\mathbf{S}_u(u_i, v_i) + (v_{i+1} - v_i)\mathbf{S}_v(u_i, v_i)| \leq \epsilon_1$ or
3. If zero cosine reached,
   i.e. $\dfrac{|\mathbf{S}_u(u_i, v_i) \cdot (\mathbf{S}(u_i, v_i) - \mathbf{P})|}{|\mathbf{S}_u(u_i, v_i)||\mathbf{S}(u_i, v_i) - \mathbf{P}|} \leq \epsilon_2$ and $\dfrac{|\mathbf{S}_v(u_i, v_i) \cdot (\mathbf{S}(u_i, v_i) - \mathbf{P})|}{|\mathbf{S}_v(u_i, v_i)||\mathbf{S}(u_i, v_i) - \mathbf{P}|} \leq \epsilon_2$.

If any of the convergence criteria is satisfied, the iterations are halted and we successfully reached to the desired projection point. Otherwise we continue to iterations up to a maximum number of iterations.

We enforce the parameters to stay within the range $u_{\min} \leq u_{i+1} \leq u_{\max}$ and $v_{\min} \leq v_{i+1} \leq v_{\max}$. If the surface is not closed in the u direction, we simply set $u_{i+1} = u_{\min}$ if $u_{i+1} < u_{\min}$ and $u_{i+1} = u_{\max}$ if $u_{i+1} > u_{\max}$, otherwise we set $u_{i+1} = u_{\max} - (u_{\min} - u_{i+1})$ and $u_{i+1} = u_{\min} + (u_{i+1} - u_{\max})$ respectively for $u_{i+1} < u_{\min}$ and $u_{i+1} > u_{\max}$. Similarly, if the surface is not closed in the v direction, we set $v_{i+1} = v_{\min}$ if $v_{i+1} < v_{\min}$ and $v_{i+1} = v_{\max}$ if $v_{i+1} > v_{\max}$, otherwise we set $v_{i+1} = v_{\max} - (v_{\min} - v_{i+1})$ and $v_{i+1} = v_{\min} + (v_{i+1} - v_{\max})$ respectively for $v_{i+1} < v_{\min}$ and $v_{i+1} > v_{\max}$.

Appendix C

IGES

Clearly the geometric CAD model is created using some external software and we will require some import procedures to read the geometric data. There are several different available file formats; most CAD systems have their own file formats. However, there are a number of standard specifications that are developed and widely adopted in the CAD industry, e.g. IGES, STEP, and PHIGS (see Piegl & Tiller 1996, ch 12). All of these specifications have their own advantages and disadvantages and all are widely used. Many professional CAD software support import from and export to IGES, STEP, etc. formats.

The Initial Graphics Exchange Specification (IGES) (US PRO 1996) is one of the industry standard data format specification for product design and manufacturing information. The information is generally created in a CAD/CAM system and stored for further use. IGES enables digital data exchange among CAD systems by defining a neutral data format. This specification defines file structure and language formats to represent geometric, topological, and non-geometric product definition data and these formats are independent of the modeling method used. Smith (1983) listed two benefits of this common format. First, a CAD/CAM system does not require to have import and export systems for all the available data formats available in the industry, which can be widely varied, rather one will only require to develop systems for import from the IGES format CAD data file and export to such format. Second, ease in data storage in magnetic tape or disk and transmission through telecommunications media. In today's computing world, data storage and transmission is far advanced than it was more than two decades ago, hence, the second reason for using IGES format can be ignored. However, the first reason is still valid and even more important since today we have even more CAD systems in the industry where most of them have their own data formats. In this project we are interested to the geometric data exchange in particular.

IGES File 1 2D Cylinder

```
Description:  Circle / Cylinder cross section (2D), NURBS                S      1
              Center = (0.2, 0.2, 0.0), Radius = 0.05                    S      2
1H,,1H;,,,,,,,,,,,,1,4HINCH,,,,,,,,,,,;                                 G      1
     126       1       0       0       0       0       0       000000000D      1
     126       0       0       0       0                               0D      2
126,                                                                     1P      1
8,2,                                                                     1P      2
1,1,0,0,                                                                 1P      3
0.0,0.0,0.0,0.25,0.25,0.5,0.5,0.75,0.75,1.0,1.0,1.0,                     1P      4
1.0,0.707107,1.0,0.707107,1.0,0.707107,1.0,0.707107,1.0,                 1P      5
0.25,0.2,0.0,                                                            1P      6
0.25,0.25,0.0,                                                           1P      7
0.2,0.25,0.0,                                                            1P      8
0.15,0.25,0.0,                                                           1P      9
0.15,0.2,0.0,                                                            1P     10
0.15,0.15,0.0,                                                           1P     11
0.2,0.15,0.0,                                                            1P     12
0.25,0.15,0.0,                                                           1P     13
0.25,0.2,0.0,                                                            1P     14
0.0,1.0,                                                                 1P     15
0.0,0.0,1.0;                                                             1P     16
S      2G      1D      2P     16                                         T      1
```

C.1 File Format and Structure

A IGES specification file can be either an ASCII file or a binary file. ASCII format files are more popular due to the flexibility that it can be easily modified using any text editor. Two types of ASCII format files are used in IGES specifications; either in fixed or in compressed format. The compressed format file has not been widely implemented since commercial file compression software can reduce the size of fixed format files. In this thesis, we consider the ASCII fixed format files and the other IGES files formats along with other standard data exchange specifications (e.g. STEP, etc.) are left for future considerations.

The fixed format files contain 80-character lines, grouped into sections. A section is identified by a letter code in column 73 of each line. The section specific data fields are available in the first 72 columns of each line and each section has ascending sequence numbers, starting from 1, right justified (with leading space or zero filled) in columns 74-80. For a fixed format IGES specification file, the flag section is not required.

The IGES files have six sections that appear contiguously in the file (no blank line is allowed). The order of the sections are start section (S), global section (G), directory entry section (D), parameter data section (P) and terminate section (T). The directory entry and terminate sections contain data in fixed-length fields where the global and parameter data sections contain delimited, variable-length fields and the start section is free from. IGES files 1 and 2 illustrates the structure of IGES

for 2D and 3D cylinders respectively. Details of the file structure is available in US PRO (1996).

C.2 Entities

Within the IGES format file the fundamental unit of data is the '*entity*'; geometry entities define the physical shape of a product and include points, curves, surfaces, solids and relations that are collections of similarly structured entities and non-geometry entities specify associations among entities, etc. Since geometry entities generally are defined independently of one another, property and associativity entities are used to augment and define their relationships. Each entity format includes its unique entity type and form numbers. In this thesis we are interested in geometry entities, and to be more specific, in NURBS curves and surfaces.

From the definition of the NURBS curve we see that we will require a knot vector, control points with corresponding weight information to describe a NURBS curve. The IGES specification of rational B-spline curves (entity type 126) gives these information along with other required information (i.e. the degree of the curve and information about the number of control points, etc.) and additional information (e.g. curve is planner or not, if the curve is open or closed, etc.). A NURBS surface is also specified using similar data format. Within IGES specifications a B-spline surface is identified by the entity type number 128.

C.2.1 Rational B-Spline Curve Entity (Type 126)

Rational B-Spline curve entity may represent analytic curves of general interest and this is used to represent NURBS curves within this project. The form numbers from 0 through 5 are valid form numbers where we have opted the form number 0 for our use where the form of the curve is determined from the rational B-spline parameters. We have not used the other form numbers which are used to represent curves form of lines and arcs of circular, elliptical, parabolic and hyperbolic shapes, respectively. Please note that using the form number preference 0 it is entirely possible to represent these curve forms, however, the system is not aware of the special form of a curve in that case. For our applications, this does not make any difficulty since our algorithms consider all the curves as generic NURBS curve.

As the NURBS curves are defined with knot sequences, control points and related weight parameters, these information are available from the IGES specification of rational B-spline curve entity. There are other parameters available from the definition of a B-spline curve entity, some of which, depending on the system, are important parameters for the applications where some are for informational purpose only. For our applications the parameter with the information that whether a curve is closed or open is used within our projection algorithm. A curve is considered closed if the beginning and ending points of the curve are coincident. The starting and ending points on the curve are defined by evaluating the curve at the starting and ending parameter values. Since the algorithms we are using are same for

planner/non-planner, rational/polynomial, periodic/non-periodic curves, we store these information for informational purposes only.

IGES File 2 3D Cylinder

```
Description:  Cylinder (3D), NURBS                                          S      1
              Center(x,y)=(0.55, 0.5), zmin=0, zmax=0.4, Radius=0.05        S      2
1H,,1H;,,,,,,,,,,,,1,4HINCH,,,,,,,,,,,;                                    G      1
     128       1       0       0       0       0       0       000000000D      1
     128       0       0       0       0                               0D      2
128,                                                                        1P      1
8,1,2,1,                                                                    1P      2
1,0,0,0,0,                                                                  1P      3
0.0,0.0,0.0,0.25,0.25,0.5,0.5,0.75,0.75,1.0,1.0,1.0,                        1P      4
0,0,1,1,                                                                    1P      5
1.0,0.707107,1.0,0.707107,1.0,0.707107,1.0,0.707107,1.0,                    1P      6
1.0,0.707107,1.0,0.707107,1.0,0.707107,1.0,0.707107,1.0,                    1P      7
0.60,0.5,0.0,                                                               1P      8
0.6,0.55,0.0,                                                               1P      9
0.55,0.55,0.0,                                                              1P     10
0.50,0.55,0.0,                                                              1P     11
0.50,0.5,0.0,                                                               1P     12
0.50,0.45,0.0,                                                              1P     13
0.55,0.45,0.0,                                                              1P     14
0.6,0.45,0.0,                                                               1P     15
0.60,0.5,0.0,                                                               1P     16
0.60,0.5,0.4,                                                               1P     17
0.6,0.55,0.4,                                                               1P     18
0.55,0.55,0.4,                                                              1P     19
0.50,0.55,0.4,                                                              1P     20
0.50,0.5,0.4,                                                               1P     21
0.50,0.45,0.4,                                                              1P     22
0.55,0.45,0.4,                                                              1P     23
0.6,0.45,0.4,                                                               1P     24
0.60,0.5,0.4,                                                               1P     25
0.0,0.0,0.0,0.0;                                                            1P     26
S      2G      1D      2P     26                                            T      1
```

C.2.2 Rational B-Spline Surface Entity (Type 128)

Analytical surfaces of general interest are represented by rational B-spline surfaces. As for the case of rational B-spline curve entity, rational B-spline surface entity's form number corresponds to the most preferred surface type. The preference order is from 0 to 9. Form number 0 is used for the case where the form of the surface is determined from the rational B-spline parameters. For our application, we only consider the form number 0. The other form numbers through 1 to 9 are used for the surfaces of the form of plane, right circular cylinder, cone, sphere, torus, surface of revolution, tabulated cylinder, ruled surface and generic quadric surface, respectively.

The NURBS surfaces are defined by the knot vectors, control net containing the control points and the related weight parameters; these information is available from the specification of rational B-spline surface entity. Two important parameters, describing whether the keeping one parameter fixed and other parameter variable the resulting curves are closed or not, are considered within the projection algorithm. Special considerations are required for the closed cases. In the same way for the rational B-spline curves, the other parameters informing us whether the surface is rational or polynomial, periodic or not, are used for informational purpose only.

C.3 Circular cylinder

Cross section of a circular cylinder in 2D, which is simply a circle, can be exactly represented using NURBS. IGES file 1 shows the IGES specification for 2D cylindrical cross-section. Here the circle is represented using a NURBS curve of degree two with the knot vector

$$\mathbf{U} = \{0.0, 0.0, 0.0, 0.25, 0.25, 0.5, 0.5, 0.75, 0.75, 1.0, 1.0, 1\},$$

and control points

$$\begin{aligned}\mathbf{P} = \{&(0.25, 0.2, 0.0), (0.25, 0.25, 0.0), (0.2, 0.25, 0.0),\\ &(0.15, 0.25, 0.0), (0.15, 0.2, 0.0), (0.15, 0.15, 0.0),\\ &(0.2, 0.15, 0.0), (0.25, 0.15, 0.0), (0.25, 0.2, 0.0)\}\end{aligned}$$

with the corresponding weight values,

$$\mathbf{w} = \{1.0, 0.707107, 1.0, 0.707107, 1.0, 0.707107, 1.0, 0.707107, 1.0\}.$$

Please note that the representation of a circle using NURBS is not unique. A details discussions on the construction of circle using NURBS is available in (Piegl & Tiller 1996, pp. 298).

The 2D cylinder can easily be extended to 3D cylinder by extruding it to the normal direction of the plane it resides in. IGES file 2 gives the IGES specification for a cylinder in 3D. The related visualization of the cylinder along with the corresponding control net is given in figure C.1.

C.4 IGES for NURBS

Although the full IGES specifications with numerous entity types are often considered difficult to adopt, often it is possible to use a much smaller subset of entities and still represent complex geometries accurately. One will also require to represent the geometry entirely in NURBS to be able to use the mesh boundary smoothing techniques presented in this thesis. It is possible and fairly easy in most cases to model the entire CAD geometry using only NURBS representation. A prominent

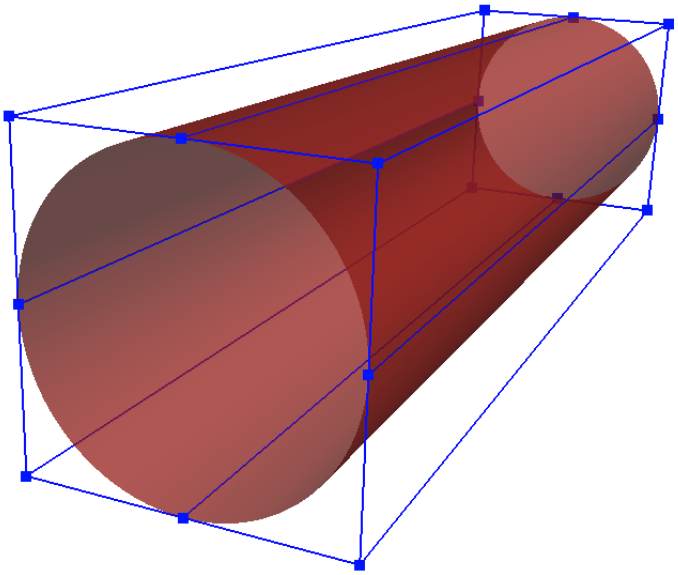


Figure C.1. 3D cylinder using NURBS and the control net

example of simplification of IGES is NASA-IGES-NURBS-Only (NINO) format (see Thompson et al. 1998) which only uses a handful of geometric entities (transformation matrix, rational B-spline curve, rational B-spline surface, boundary, curve on a parametric surface and bounded surface) along with a few non-geometric entities (null entity, general note, color definition, associativity instance and name form only property entity). However, in this project, we considered even less number of entity types for our purpose, which is certainly not adequate for industrial level complex geometries. Future improvements will be required to consider at least all the entity types in NINO format. Also, converter (e.g. Chou & Logan 1995) from other standard IGES format entity types to NINO format will be required as the industry standard CAD systems use more entity types than supported in NINO format. Since, our projection algorithms consider geometries defined by only NURBS, NINO format specifications along with robust converter from standard IGES format data to NINO format data should be adequate for our purposes and to make use of the industry standard complex geometries represented in IGES format.

TRITA-CSC-E 2010:001
ISRN-KTH/CSC/E--10/001--SE
ISSN-1653-5715

www.kth.se