Alma Mater Studiorum · Università di Bologna

DOTTORATO DI RICERCA IN INFORMATICA
Ciclo XXVIII

Settore concorsuale di afferenza: 01/B1
Settore scientifico disciplinare: INF01

CAD Aspects on Isogeometric Analysis and Hybrid Domains

Presentata da GIULIA MARTINI

Coordinatore Dottorato
PAOLO CIACCIA

Relatore
SERENA MORIGI

Correlatore
MAURIZIO GABBRIELLI

Esame finale anno 2016

Abstract

This thesis is the result of a Ph.D. program in Alto Apprendistato carried out at the Dipartimento di Informatica - Scienza e Ingegneria (DISI) of the University of Bologna and at the company devDept Software.
With regard to the professional side of my Individual Training Project, I developed technical and scientific skills in 3D geometry of curves and surfaces, CAD, and Finite Element Analysis (FEA). Regarding the academic side, I investigated CAD aspects in the field of Isogeometric Analysis (IGA) on both single and hybrid multipatch physical domains.
Simulations are performed in classical FEA systems, which require the conversion of designs, made by CAD systems, into finite element meshes. IGA is a new approach that aims to unify the worlds of CAD and FEA by using the same geometry for analysis as what is used for modeling. That is, the same set of basis functions are adopted both to describe the computational geometry in the CAD tool, and to span the solution space for FEA.
The traditional FEA pipeline works on meshes and the most advanced IGA systems work on NURBS or T-spline geometries. Hybrid geometric models (i.e., models in which mesh and NURBS entities coexist), are an emergent way to represent a solid object, but in most CAD systems mesh and NURBS geometries cannot interact with each other, and conversions to a common representation are often needed.
In this thesis, we investigate how IGA can be applied on 2D and 3D hybrid models made by both mesh and NURBS entities without requiring laborious and time consuming conversion processes.

Acknowledgments

I would never have finished this thesis without the guidance of my commissione members and the support of my family and friends.
In particular, I would like to thank my advisor Serena Morigi for her great knowledge, expertise, and availability.
My deepest gratitude goes to my tutor Maurizio Gabbrielli for understanding the difficulties of pursuing a Ph.D. while working full time and for never failing to believe that I could finish this.
I extend sincere thanks to the referees of this thesis, Mariantonia Cotronei, Alexandre Miller, and Rafał Nowak, for their detailed reviews and useful comments.
I am also extremely grateful to my family for their love and to sweet little Nora for making me smile even on the toughest days.
I would like to thank all my friends for always being there, especially Josh and Rachel for helping me out on such short notice.
Finally, a big thank you to Cristiano. Your support, honest opinions, decision making charts, and cappuccinos made this thesis possible.

Contents

Introduction

This work reports my experience within the Ph.D. program in Alto Apprendistato carried out at the Dipartimento di Informatica - Scienza e Ingegneria (DISI) of the University of Bologna. The Alto Apprendistato work contracts were recently started by the Regione Emilia-Romagna to strengthen the relationship between university and industry. They offer to Ph.D students the possibility to attend high level formation activities provided by the university, while working full time with high level technologies provided by a company. An Individual Training Project (ITP) is approved by both the educational institution and the workplace.

In particular, the aim of my ITP was twofold: from the professional side, it required me to develop technical and scientific skills in 3D geometry of curves and surfaces, CAD, and Finite Element Analysis (FEA), and from the academic side, the ITP required me to investigate CAD aspects in the field of isogeometric analysis.

The company where I have been working is devDept Software, which produces and distributes Eyeshot, a 3D graphics CAD/CAM/CAE/FEM component for the Microsoft development platforms. These acronyms respectively stand for Computer Aided Design, Manufacturing, Engineering and Finite Element Method. Most of the clients are software houses specialized in various sectors and they use Eyeshot for 3D modeling, analysis and visualization.

CAD is the use of computer technology for design: it allows for the creation, modification, analysis and optimization of precision drawings and geometric modeling. CAD softwares replace manual drafting with an automated process. Just like the most important CAD applications, Eyeshot can be used both for two-dimensional and three-dimensional modeling, and it is based on Non-Uniform Rational B-splines (NURBS). NURBS are a mathematical representation that is very convenient for free-form surface modeling, they can exactly represent all conic sections and thanks to their flexibility and precision, they are widely used in CAD systems to describe curves and surfaces.

Eyeshot also integrates a FEA component, based on the finite element method, that solves problems in structural and solid mechanics. It can perform linear elastic analysis to determine the behavior of structures and solids subjected to forces.

The FEM technique is a very successful computational method applied for the solution of Boundary Value Problems (BVP). It has its origins in the later 1950s through researches in the field of aerospace engineering, and since then it has been studied extensively and applied in engineering analysis in the CAE industry. Many CAE systems based on FEM were already mature when, about twenty years later, the first CAD programs started to appear. But now CAD is a much bigger industry than CAE, and many CAD systems are capable of executing calculations and analysis typical of CAE systems.

FEM subdivides a whole problem domain into simpler parts, called finite elements, and thus requires the generation of a mesh that approximates the original geometry.

Finite element analyses are often conducted on models with very complex geometries, which require that the system first generates a mesh from the given CAD geometry to approximate the physical domain and then applies the analysis to the mesh. It is estimated that about 80% of overall analysis time is devoted to mesh generation.

Isogeometric analysis (IGA) is a new approach, introduced by Hughes

Cottrell and Bazilevs in 2005, see [34], that aims to unify the worlds of CAD and FEA. The key idea is to use for analysis the same geometry used for modeling. That is, the same set of basis functions adopted to describe the computational geometry in the CAD tool, e.g., NURBS, are used also to span the solution space for FEA.

In the IGA context, the conversion of the CAD design into a mesh suitable for FEA analysis is thus no more necessary. Skipping the mesh generation process provides a significant reduction of time, and the results are more accurate due to the use of the exact geometry instead of its approximation. Since its introduction, IGA has become a focus of research within both the fields of FEA and CAD and it has attracted a lot of interest from the scientific community.

The problem of conversion between precise and approximate geometries is present also in the CAD context. In fact, most CAD systems allow both mesh and NURBS geometry representations, but only entities having the same representation can interact with each other. Therefore, to have interaction between entities of different types, a conversion to a common representation is needed. The conversion from mesh to NURBS is still a challenging task which is based on reverse engineering, while the conversion from NURBS to mesh is called tessellation, these two processes are time consuming and often cause loss of information. Both representations are fundamental and in many cases they are not interchangeable: NURBS can describe curved geometries exactly, and meshes are very simple objects, easy to manipulate and CAD applications can usually manage very large quantities of them without problems. Recently, the use of meshes for the representation of objects has become very popular thanks to the wide use of 3D scanner devices.

In several practical applications, while modeling in a CAD system, there are geometries that cannot be entirely represented with NURBS entities. An example of this comes from the wrinkled patterns present on the handles of many tools or on some shoes bottom: these geometries present severe difficulties when described with precise surfaces, while they can be easily represented by mesh entities. They are often integrated in more complex models containing other geometries that can be described exactly, and it would be counterproductive to approximate the exact geometries converting them into meshes.

We will denote by hybrid geometry models those models in which mesh and NURBS representations coexist. In addition to models with wrinkled pattern meshes and NURBS surfaces, hybrid geometries arise also when part of the model is a digital representation of a physical object: 3D scanners generate mesh geometries, and applying reverse engineering to convert them is time consuming and not always necessary. These are just some examples of a wide range of situations in which it is important for CAD systems to support hybrid geometries.

It would be very beneficial to have true interoperability between mesh and precise geometry in a CAD system, in which mesh and NURBS geometries can interact with each other. In such a system, it must be possible to create a solid object where some faces are NURBS and some others are meshes, to trim a surface with a mesh and vice versa, and to create a fillet between a surface and a mesh. Also, given an object in mesh representation, the system should be able to create the mold geometry and perform other CAD operations without converting the mesh to a precise representation.

On the other hand, being able to manage without any conversion, the representation of hybrid models would be of great benefit in the FEA systems. The traditional FEM pipeline works on meshes and the most advanced IGA systems work on NURBS or T-spline geometries.

One of the contributions of this thesis is to apply isogeometric analysis to solve BVPs on multi-patch hybrid 2D and 3D physical domains, constituted by mesh and NURBS patches. For the numerical examples, we used

GeoPDEs [20], an open source suite of tools for applications in IGA that has been suitably extended to support multipatch and hybrid geometries.

The thesis is organized in three main parts: the first (from Chapter 1 to 3) introduces the geometry representations and the basics of FEM and IGA, the second (Chapters 4 and 5) presents how we applied IGA for problems defined on different kinds of geometry, and the last part (Chapter 6) describes how I accomplished the professional requirements of my ITP working at DevDept Software.

In particular, Chapter 1 introduces the discrete and parametric geometries used to represent the surfaces on which we applied IGA. These include: NURBS geometries, analytic geometries and polygonal meshes of rectangular elements.

Chapter 2 provides an overview on the BVPs used for the analysis. In particular, we present the classical Poisson problem, the Laplace-Beltrami problem, for surfaces embedded in $\mathbb{R}^3$, and the linear elasticity problem, since the latter is the class of problems solved by the FEM component of Eyeshot.

Chapter 3 presents the isogeometric analysis: NURBS geometries are treated as basis for the analysis and the Galerkin method for IGA is introduced. IGA on multi-patch and hybrid geometries is introduced, and some methods added to GeoPDEs to handle multi-patch physical domains are presented.

Chapter 4 collects the examples of IGA applied to geometries made of one patch. Exploiting the potentialities of IGA to treat independently the basis functions for the geometry and the ones for the approximating subspace, we investigated the combinations between three different representations of the geometry (mesh, NURBS and analytic) and three choices of basis functions for the approximating space (nurbs, spline and piecewise polynomial functions represented in Bézier form). Then we compared the behavior of each geometry/solution pair under refinements of the domain.

Chapter 5 provides IGA examples on multi-patch geometries. It contains preliminary examples applied on hybrid geometries and explains how to use the method we developed to make two adjacent surfaces compatible, a necessary condition for the analysis.

Chapter 6 concludes the thesis describing how I fulfilled the professional aims of my ITP at DevDept Software. It provides an overview on some of the topics and functionalities I studied and implemented at work both on the CAD and on the FEM libraries of Eyeshot. Having worked full-time at DevDept Software for four years, this overview is far from being exhaustive. I selected and reported only a few algorithms developed in Eyeshot that are very useful when modeling a 3D complex NURBS geometry, before a possible application of FEM analysis. Future works will investigate the possibility to integrate IGA into Eyeshot.

Chapter 1

Geometry description: NURBS and polygonal meshes

To digitally represent a shape, it is necessary to know how to represent a geometry with a mathematical model, in terms of curves, surfaces and solids. NURBS are mathematical representations that can accurately describe any kind of shape: from a simple 2D line, conic section, or curve, to a complex 3D free-form solid or surface. Thanks to their flexibility and precision, NURBS have been widely used in engineering design, and became industry standard in CAD systems.

In classical FEM, geometries are described using polygonal meshes (mainly triangular and quadrilateral meshes) which approximate the exact geometry on which the analysis is carried out. Typically, the exact geometry of the physical domain is designed in a CAD system using NURBS geometries and then it is converted into a polygonal mesh model through a laborious mesh generating algorithm.

IGA is based on the isoparametric concept, which uses the same basis functions both to represent the geometry and to approximate the solution. In particular, NURBS-based IGA adopts NURBS basis functions for the description of the geometry and the approximation of the solution space. Thus, IGA allows to solve PDE-based problems on exact geometric domains, avoiding to approximate them with polygonal meshes like in classical FEM. This can be done using NURBS geometries for a wide range of domains.

However, in practical applications the geometry is often defined by multiple patches joined together: a geometric model represented by a collection of connected surface elements is called B-Rep (Boundary Representation). Moreover, the patches in a B-rep can be hybrid, which means that adjacent patches can be NURBS, modeled with a CAD system, and polygonal meshes, obtained for instance by a 3D scanner. We will refer to hybrid NURBS and mesh geometries as Extended B-rep.

This chapter introduces the types of geometric descriptions that we will use to model the physical domains in our applications of IGA: NURBS, meshes and how they can coexist in a hybrid geometry. A more detailed overview on NURBS can be found in [26, 47].

1.1 Parametric representation

In a parametric representation of a compact, connected and oriented Rimannian manifold $\Omega \subset \mathbb{R}^d$, each coordinate of a point on Ω is represented separately as an explicit function of a vector-valued independent variable $\hat{\xi} = (\hat{\xi}_1, \ldots, \hat{\xi}_k) \in \mathbb{R}^k$, where $d \geqslant k \geqslant 1$. The set of all such functions defines a geometrical mapping or parameterization, from a parameter domain $\hat{\Omega} \subset \mathbb{R}^k$ to the manifold Ω:

$$\begin{aligned} F : \hat{\Omega} &\longrightarrow \Omega \\ \hat{\xi} &\longmapsto \xi := F(\hat{\xi}). \end{aligned} \tag{1.1}$$

The mapping F is assumed to be smooth with piecewise smooth inverse $F^{-1} : \Omega \longrightarrow \hat{\Omega}$. Fig. 1.1 represents a geometrical mapping F for $k = 2$ and

$d = 3$. The parameter space is $\hat{\Omega} \subset \mathbb{R}^2$, the physical space is $\Omega \subset \mathbb{R}^3$, and the inverse of F is F^{-1}.

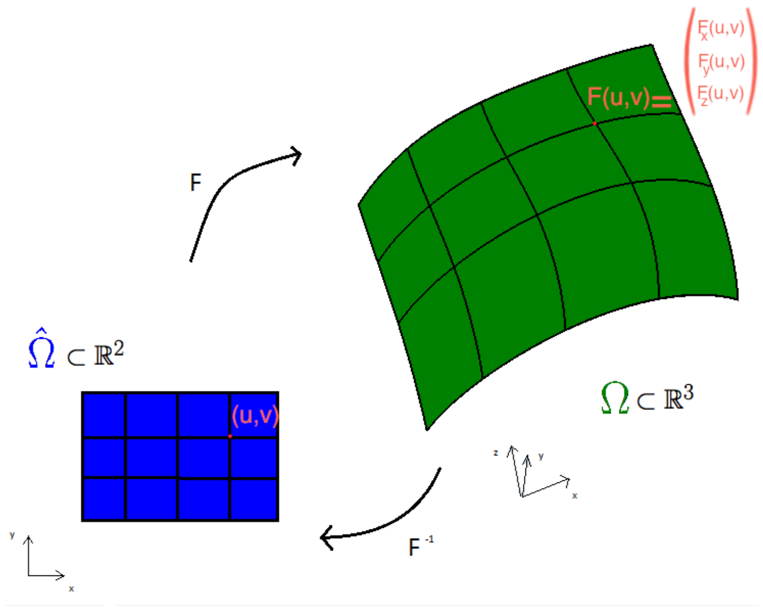


Figure 1.1: Example of geometrical mapping $F : \hat{\Omega} \longrightarrow \Omega$ and its inverse F^{-1}, with the parameter space $\hat{\Omega} \subset \mathbb{R}^2$ and the physical space $\Omega \subset \mathbb{R}^3$. In red, a point of the parameter space $\hat{\xi} = (u, v) \in \hat{\Omega}$ and its image under F $\xi := F(u, v) \in \Omega$.

The Jacobian of the parameterization F, denoted with $\hat{J}$, it is a $d \times k$ matrix, defined as

$$\hat{J}_{i,j}(\hat{\xi}) := \frac{\partial F_i}{\partial \hat{\xi}_j}(\hat{\xi}), \quad i = 1, \dots, d, \;\; j = 1, \dots, k, \tag{1.2}$$

that is, for a surface with $k = 2$, $d = 3$ and $\hat{\xi} = (u, v)$:

$$\hat{J}(u, v) = \begin{bmatrix} \frac{\partial F_x}{\partial u} & \frac{\partial F_x}{\partial v} \\ \frac{\partial F_y}{\partial u} & \frac{\partial F_y}{\partial v} \\ \frac{\partial F_z}{\partial u} & \frac{\partial F_z}{\partial v} \end{bmatrix}. \tag{1.3}$$

We introduce the first fundamental form of the mapping, which represents the induced metric on the manifold:

$$\hat{G} : \hat{\Omega} \longrightarrow \mathbb{R}^{k\times k}, \quad \hat{G}(\hat{\xi}) := (\hat{J}(\hat{\xi}))^T \hat{J}(\hat{\xi}), \tag{1.4}$$

where $(\hat{J}(\hat{\xi}))^T$ is the transpose of $\hat{J}(\hat{\xi})$, and the square root of its determinant:

$$\hat{g} : \hat{\Omega} \longrightarrow \mathbb{R}, \quad \hat{g}(\hat{\xi}) := \sqrt{\det(G(\hat{\xi}))}. \tag{1.5}$$

We observe that if $k = d$, then $\hat{J}(\hat{\xi}) \in \mathbb{R}^{d\times d}$, and $\hat{g}(\hat{\xi}) := \det(J(\hat{\xi}))$.

The same functions can be defined on the manifold as follows:

$$J : \Omega \longrightarrow \mathbb{R}^{d\times k}, \quad J(\xi) := \hat{J}(\hat{\xi}) \circ F^{-1}(\hat{\xi}), \tag{1.6}$$

$$G : \Omega \longrightarrow \mathbb{R}^{k\times k}, \quad G(\xi) := \hat{G}(\hat{\xi}) \circ F^{-1}(\hat{\xi}), \tag{1.7}$$

$$g : \Omega \longrightarrow \mathbb{R}, \quad g(\xi) := \hat{g}(\hat{\xi}) \circ F^{-1}(\hat{\xi}). \tag{1.8}$$

Let ϕ be a real-valued C^0 function on the manifold Ω, in virtue of the invertibility of the geometrical mapping, we can write:

$$\phi(\xi) = \hat{\phi}(\hat{\xi}) \circ F^{-1}(\hat{\xi}), \tag{1.9}$$

where $\hat{\phi}(\hat{\xi}) := \phi(F(\hat{\xi}))$. This makes it possible to avoid the distinction between the function ϕ, defined on the physical domain Ω, and the function $\hat{\phi}$, defined on the parameter domain $\hat{\Omega}$.

As shown in Fig. 1.2, k gives the dimension of the manifold: for $k = 1$ the manifold is a curve, for $k = 2$ a surface and for for $k = 3$ a volume. When $d > k$, Ω is a lower dimensional manifold embedded in the physical space $\mathbb{R}^d$.

In this work on BVP problems, we will focus on physical domains Ω that are planar surfaces in $\mathbb{R}^2$ or surfaces embedded in $\mathbb{R}^3$.

In the following, we will define B-spline and NURBS basis functions as suitable bases to represent the parametric geometry introduced in Eq. (1.1). Then we will briefly review the representations of NURBS curves and surfaces.

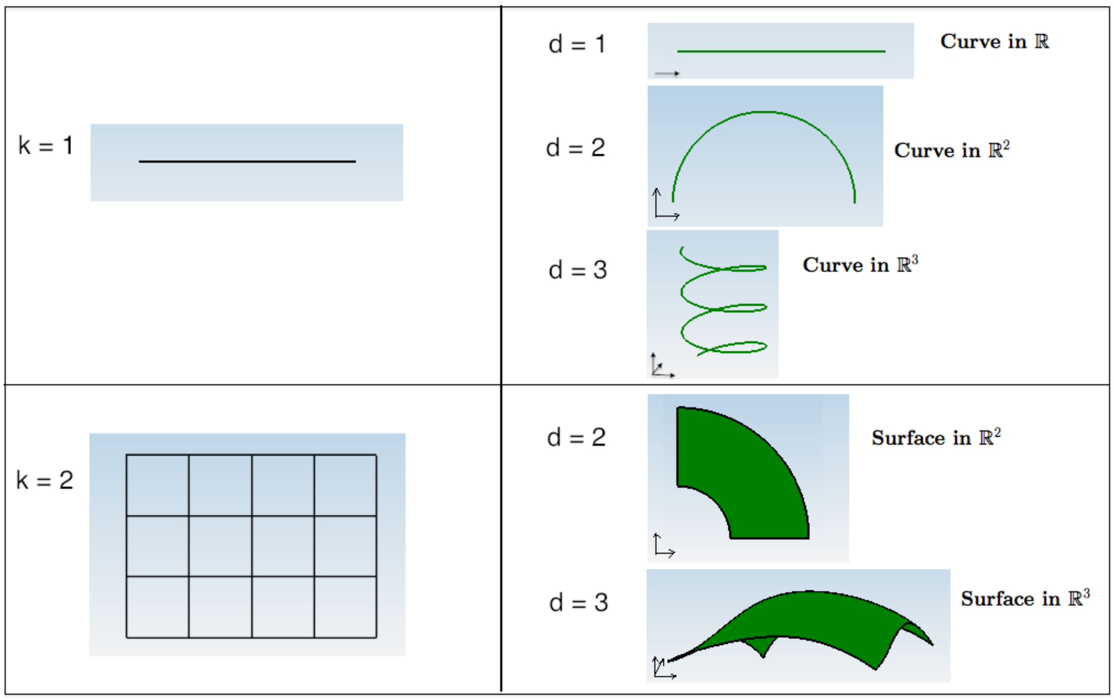


Figure 1.2: Examples of parametric geometries with different values of k and d. The parameter spaces $\hat{\Omega}$ are on the left and the physical spaces Ω are on the right.

1.2 B-spline basis functions

Let's consider a partition Ξ of a bounded and closed interval $[a,b] \subset \mathbb{R}$:

$$\Xi := \{x_i\}_1^k, \ a \equiv x_0 < x_1 < \cdots < x_k < x_{k+1} \equiv b. \tag{1.10}$$

This partition generates the $k+1$ subintervals:

$$I_i = [x_i, x_{i+1}[, \text{ for } i = 0, \dots, k-1 \text{ and } I_k = [x_k, x_{k+1}]. \tag{1.11}$$

The nodes are often referred to as knots, and the partition is called knot vector. Each subinterval defined in (1.11) is called knot span or element of the partition. So the number of elements of the partition Ξ is $N_e = k+1$.

Let m be a positive integer and let $M = (m_1, \dots, m_k)$ be a vector with integer components such that $1 \leq m_i \leq m$ for all $i = 1, \dots, k$.

The space of polynomial splines of order m and degree $p = m-1$, with knot vector Ξ and multiplicity vector M, is defined as:

$$\begin{aligned} \mathcal{S}(\mathbb{P}_{m-1}, M, \Xi) := \{ s : [a,b] \to \mathbb{R} : \\ &\exists\, s_0, \dots, s_k \in \mathbb{P}_{m-1} : s|_{I_i} = s_i, \ i = 0, \dots, k \\ &D^j s_{i-1}(x_i) = D^j s_i(x_i), \ j = 0, \dots, m-1-m_i, \ i = 0, \dots, k \}, \end{aligned} \tag{1.12}$$

where $\mathbb{P}_{m-1}$ is the space of polynomials of degree $\leq m-1$.

The multiplicity vector M determines the degree of smoothness of a spline function at the knots. If all the multiplicities $m_i = 1$, the spline is of class C^{m-2} at x_i, that is the maximum order of continuity. If all $m_i = m$, the spline can have a discontinuity at x_i. Thus, in the special case of $M = (m, \dots, m)$, the space $\mathcal{S}(\mathbb{P}_{m-1}, M, \Xi)$ coincides with the space of piecewise polynomials $P\mathbb{P}_{m-1}(\Xi)$, while if $M = (1, \dots, 1)$ the spline space reduces to the space of spline functions with simple nodes $x_1, \dots, x_k$, denoted by $\mathcal{S}_m(\Xi)$. It holds $\mathbb{P}_{m-1} \subset \mathcal{S}_m(\Xi) \subset P\mathbb{P}_{m-1}$.

Let K be the sum of the multiplicities, $K = \sum_{i=1}^k m_i$, then $\mathcal{S}(\mathbb{P}_{m-1}, M, \Xi)$ is a space of dimension $m+K$, see [40].

To define a basis for this space, we introduce the notion of extended partition Ξ^* associated with Ξ, which is defined as the extended knot vector $\Xi^* = \{\xi_i\}_{i=1}^{2m+K}$ such that:

- $\xi_1 \leq \xi_2 \leq \cdots \leq \xi_{2m+K}$
- $\xi_m \equiv a,\ \xi_{m+K+1} \equiv b$
- $(\xi_{m+1}, \xi_{m+2}, \ldots, \xi_{m+K}) = (\underbrace{x_1, \ldots, x_1}_{m_1}, \underbrace{x_2, \ldots, x_2}_{m_2}, \ldots, \underbrace{x_k, \ldots, x_k}_{m_k})$

An extended knot vector is said to be open if the extra nodes are coincident, i.e. the first and last knots have multiplicity m. In the following, we will use geometries with open knot vectors. When the knots are equally spaced in the parameter space, the knot vector is called uniform, otherwise it is non-uniform.

Given an extended partition Ξ^* associated with the space $\mathcal{S}(\mathbb{P}_{m-1}, M, \Xi)$, the set of B-spline $\{N_{i,m}(\hat{x})\}_{i=1}^{m+K}$ is defined by the following recursive formula, starting from piecewise constants for $m = 1$:

$$N_{i,1}(\hat{x}) = \begin{cases} 1 & \text{if } \xi_i \leq \hat{x} < \xi_{i+1}, \\ 0 & \text{otherwise,} \end{cases} \tag{1.13}$$

and for $l = 2, \ldots, m$:

$$N_{i,l}(\hat{x}) = \begin{cases} \frac{\hat{x}-\xi_i}{\xi_{i+l-1}-\xi_i} N_{i,l-1}(\hat{x}) + \frac{\xi_{i+l}-\hat{x}}{\xi_{i+l}-\xi_{i+1}} N_{i+1,l-1}(\hat{x}) & \text{if } \xi_i \leq \xi_{i+l}, \\ 0 & \text{otherwise.} \end{cases} \tag{1.14}$$

By convention, $0/0$ is considered equal to zero.

The $m+K$ B-spline functions defined by Eq. (1.13) and (1.14) constitute a basis for $\mathcal{S}(\mathbb{P}_{m-1}, M, \Xi)$. Then we can define a spline function of the space $\mathcal{S}(\mathbb{P}_{m-1}, M, \Xi)$ as:

$$s(\hat{x}) = \sum_{i=1}^{m+K} N_{i,m}(\hat{x}) c_i, \tag{1.15}$$

where $c_i \in \mathbb{R}$ are scalar coefficients.

B-Splines basis functions have the following important properties:

1. They are non-negative over the entire domain.

2. They are locally supported: the support of a basis function of order m is made of m knot spans. More precisely:

$$N_{i,m}(\hat{x}) = 0 \quad \forall\, \hat{x} \notin [\xi_i, \xi_{i+m}).$$

   At any given knot span $[\xi_i, \xi_{i+1})$, there are only m non-zero functions: $N_{i-m+1,m}, \dots, N_{i,m}$.

3. They form a partition of unity, i.e.

$$\sum_{i=1}^{m+K} N_{i,m}(\hat{x}) = 1 \qquad \forall\, \hat{x} \in [a, b].$$

As mentioned, a B-spline of order m has degree $p = m-1$. Fig. 1.3 shows an example of B-spline basis functions of order 4 associated to the open, non-uniform knot vector $\Xi^* = (0, 0, 0, 0, 0.25, 0.5, 0.5, 0.7, 0.7, 0.7, 1, 1, 1, 1)$ and their class of continuity at knots with different multiplicities. The multiplicity vector of Ξ^* is $M = (1, 2, 3)$.

1.3 Refinement tools

Refinement processes allow to enrich the basis of functions maintaining the geometry intact. There are three kinds of refinement techniques for spline used in IGA: h, p and k-refinement. They are based on the techniques of knot insertion and degree elevation, that will be introduced here.

- <u>Knot Insertion</u>

  Knot insertion consists of adding one or more knots to a knot vector.

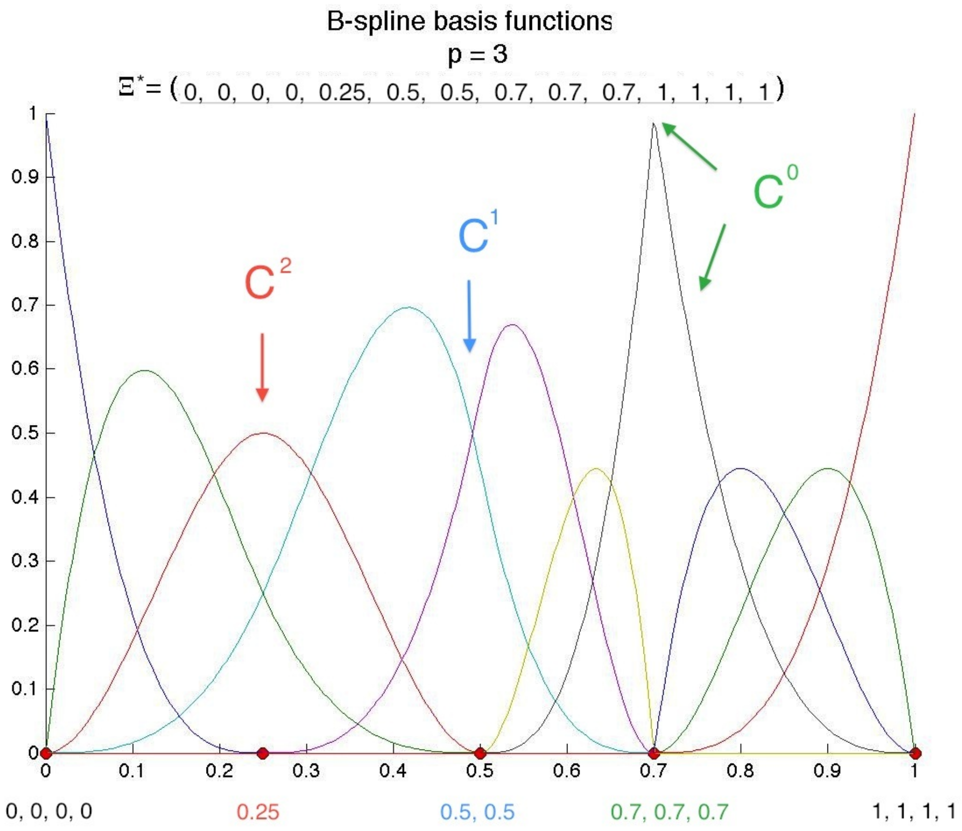


Figure 1.3: Univariate B-spline basis functions of order 4 (and degree 3) for the knot vector Ξ^*. They are of class C^2 at the knot 0.25 that has multiplicity 1, of class C^1 at 0.5 that has multiplicity 2, and of class C^0 at 0.7 that has multiplicity equal to the degree $p = 3$.

Let Ξ^* be a knot vector and let $\hat{\xi} \in [a, b]$ be a knot to be inserted, such that $\xi_l \leq \hat{\xi} \leq \xi_{l+1}$. Then the new knot vector after knot insertion will be:

$$\hat{\Xi}^* = \{\hat{\xi}_i\}_{i=1}^{2m+K+1}, \text{ with } \hat{\xi}_i = \begin{cases} \xi_i & i \leq l \\ \hat{\xi} & i = l+1 \\ \xi_{i-1} & i \geq l+2 \end{cases} . \tag{1.16}$$

The new space $\mathcal{S}(\mathbb{P}_{m-1}, \hat{M}, \hat{\Xi}^*)$ has dimension $K + m + 1$, and if the function $f \in \mathcal{S}(\mathbb{P}_{m-1}, M, \Xi)$, then $f \in \mathcal{S}(\mathbb{P}_{m-1}, \hat{M}, \hat{\Xi}^*)$. According to the Boehm's formula, the new basis of B-spline for $\mathcal{S}(\mathbb{P}_{m-1}, \hat{M}, \hat{\Xi}^*)$ is given by:

$$N_{i,m}(\hat{x}) = \begin{cases} \hat{N}_{i,m}(\hat{x}) & i \leq l-m, \\ \dfrac{\hat{\xi} - \hat{\xi}_i}{\hat{\xi}_{i+m} - \hat{\xi}_i} \hat{N}_{i,m}(\hat{x}) + \dfrac{\hat{\xi}_{i+m-1} - \hat{\xi}}{\hat{\xi}_{i+m-1} - \hat{\xi}_{i+1}} \hat{N}_{i+1,m}(\hat{x}) & l-m+1 \leq i \leq l \\ \hat{N}_{i+1,m}(\hat{x}) & i \geq l+1. \end{cases} , \tag{1.17}$$

and the new coefficients are:

$$\hat{c}_i = \begin{cases} c_i & i \leq l-m+1, \\ \lambda_i c_i + (1-\lambda_i) c_{i-1} & l-m+2 \leq i \leq l, \\ c_{i-1} & i \geq l+1, \end{cases} \tag{1.18}$$

where $\lambda_i = \frac{\hat{\xi} - \hat{\xi}_i}{\hat{\xi}_{i+m-1} - \hat{\xi}_i}$.

Inserting new knots in an existing knot vector is called knot refinement. In particular, h-refinement is the process of dividing every knot span into two halves by inserting a new knot between two existing knots. Fig. 1.4 shows the effect of the application of h-refinement to the knot vector given in Fig. 1.3.

As new knots are inserted without changing the geometry, the result is a richer basis for the solution space.

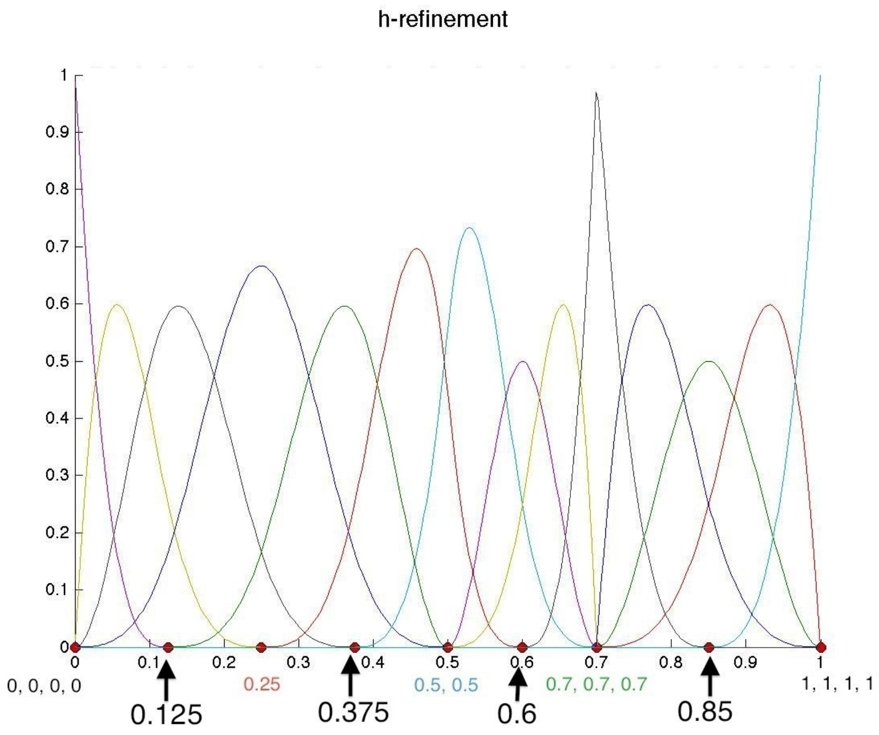


Figure 1.4: Applying the h-refinement process to the knot vector $\Xi^* = (0, 0, 0, 0, 0.25, 0.5, 0.5, 0.7, 0.7, 0.7, 1, 1, 1, 1)$, we get the knot vector $\hat{\Xi}^* = (0, 0, 0, 0, 0.125, 0.25, 0.375, 0.5, 0.5, 0.6, 0.7, 0.7, 0.7, 0.85, 1, 1, 1, 1)$. The new knots are indicated in black.

- Degree Elevation

  Degree elevation is achieved by elevating the degree of the basis functions used to represent the geometry. No new knots are added to the knot vector, but the multiplicity of each knot is increased by one, in order to preserve the same continuity at the knots in the original geometry.

If the original partition is:

$$\Xi^* = (\underbrace{a, \dots, a}_{m}, \underbrace{\xi_1, \dots, \xi_1}_{m_1}, \dots, \underbrace{\xi_k, \dots, \xi_k}_{m_k}, \underbrace{b, \dots, b}_{m}), \text{ with } K = \sum_{i=1}^{k} m_i,$$

the new partition after degree elevation is:

$$\hat{\Xi}^* = (\underbrace{a, \dots, a}_{m+1}, \underbrace{\xi_1, \dots, \xi_1}_{m_1+1}, \dots, \underbrace{\xi_k, \dots, \xi_k}_{m_k+1}, \underbrace{b, \dots, b}_{m+1}),$$

in which the total number of knots is $2(m+1)+K+k$. The associated vector space $\mathcal{S}(\mathbb{P}_m, \hat{M}, \hat{\Xi}^*)$ has dimension $m+1+K+k$.

The algorithm of degree elevation for splines consists of three steps:

1. Subdivision of the spline into piecewise Bézier polynomials. This can be achieved by inserting the internal knots until they all have multiplicity equal to the degree.
2. Degree elevation of each polynomial, which modifies only the coefficients.
3. Knot removal of the internal knots in excess, until the original continuity is reached.

The process of performing degree elevation from p to $p+1$ is also called p-refinement. The parameterization is unchanged and, like h-refinement, p-refinement does not change the geometry, but only the dimension of the space. Fig. 1.5 shows the effect of the application of p-refinement to the spline given in Fig. 1.3.

- k-Refinement

  k-refinement is carried out as a combination of p-refinement and h-refinement, that is, first elevating the degree of the basis functions and then inserting new knots. It results in curves with high continuity, with a smaller increase in the number of knots with respect to pure

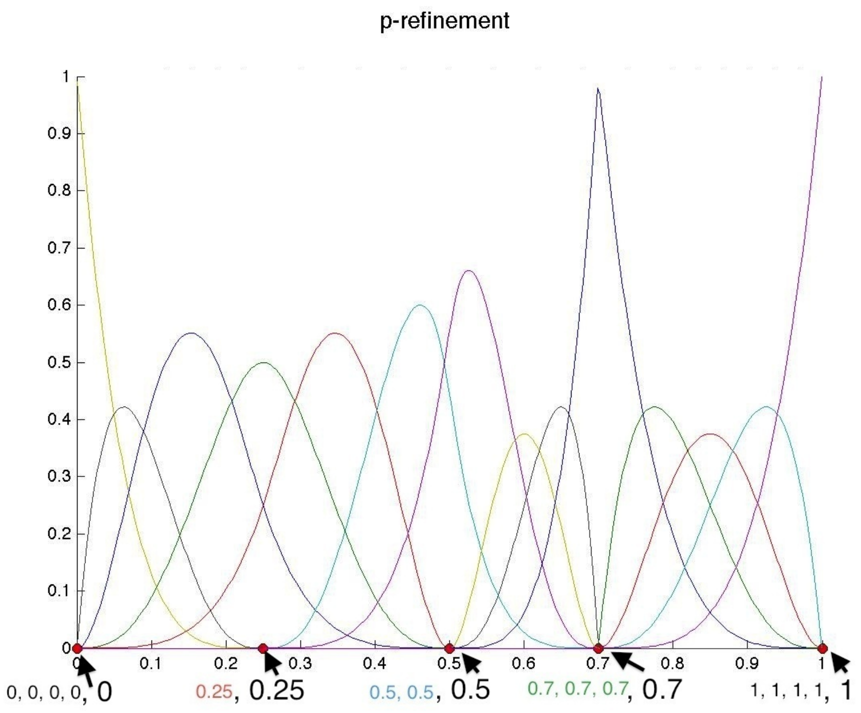


Figure 1.5: Applying the p-refinement process to the knot vector $\Xi^* = (0, 0, 0, 0, 0.25, 0.5, 0.5, 0.7, 0.7, 0.7, 1, 1, 1, 1)$, we get the knot vector $\hat{\Xi}^* = (0, 0, 0, 0, 0, 0.25, 0.25, 0.5, 0.5, 0.5, 0.7, 0.7, 0.7, 0.7, 1, 1, 1, 1, 1)$. The new knots are indicated in black.

p-refinement. The name k-refinement has been introduced in [34], and unlike p-refinement and h-refinement, this process has no analogue in classical FEM.

Given the degree $p = 2$ basis functions with knot vector $(0, 0, 0, 1, 1, 1)$, Fig. 1.6 shows the difference between the application of knot insertion first and degree elevation afterwards, and the application of k-refinement, which consists of performing degree elevation first and then knot insertion: the two processes in fact do not commute. The first process results in many functions that are C^1 across the element boundaries, while k-refinement gives a smaller number of functions, each of which is C^{p-1} across the element boundaries. This is even more evident in Fig. 1.7, where we applied three times the k-refinement process to get functions of degree 5.

All the functions preserve maximal C^{p-1} continuity across element boundaries only if the initial knot vector does not have internal knots.

Refinement methods can be applied to NURBS geometries of dimension k simply by refining all the k knot vectors of the parametric domain. While it is possible to refine a mesh locally, due to the tensor product nature of B-spline basis functions, refinement is a global procedure for B-spline geometries. This sometimes leads to superfluous knots and control points, which means, as we will see, superfluous elements in isogeometric analysis. In order to overcome this limitation, alternative technologies such as T-splines, hierarchical B-splines and isogeometric spline forests have been proposed [4, 50, 51].

1.4 Spline curves, surfaces and solids

Let $\Xi = (x_1, \dots, x_k)$ be a knot vector, and let $M = (m_1, \dots, m_k)$ be the vector of multiplicities associated to Ξ, with $K = \sum_{i=1}^{k} m_i$.

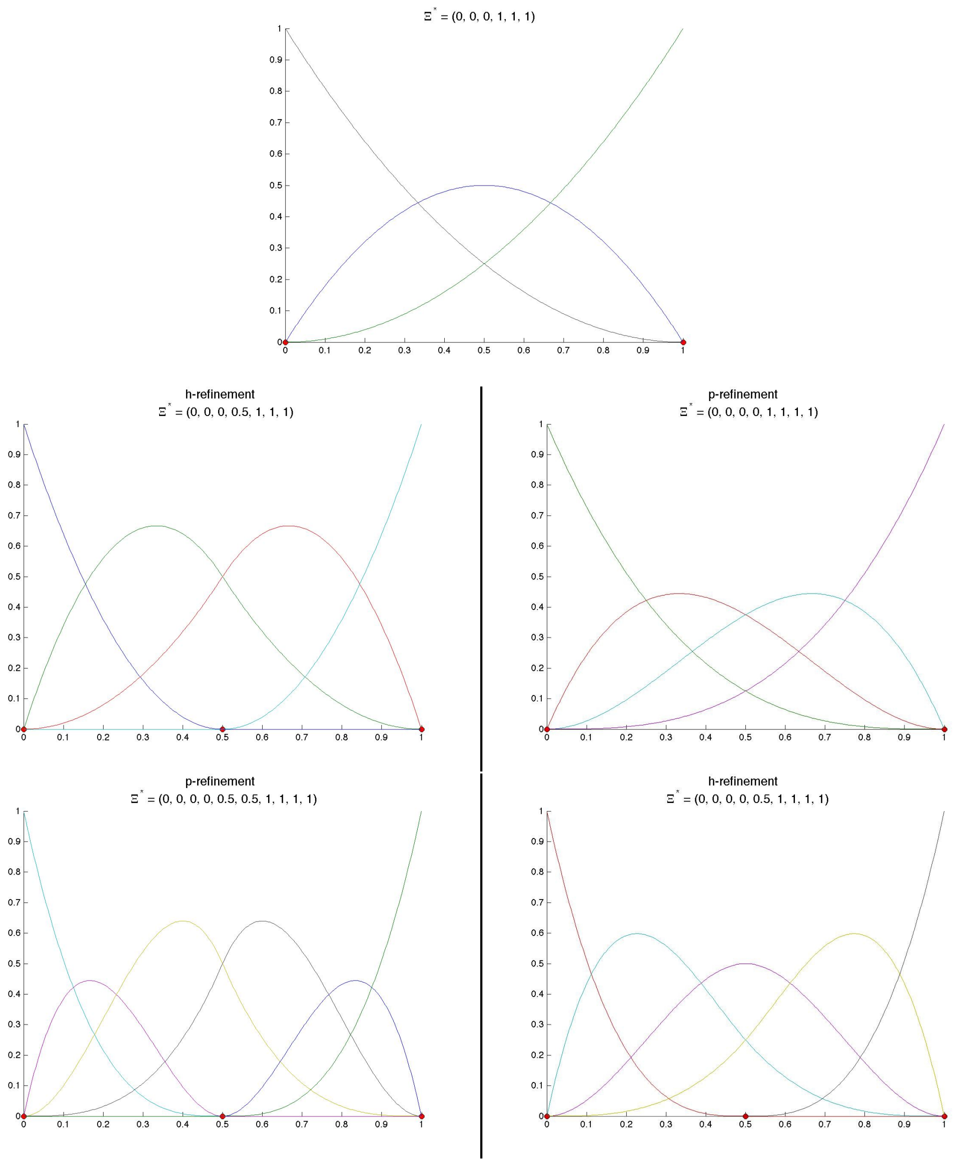


Figure 1.6: On the left: applying applying knot insertion and then order elevation to the knot vector $\Xi^* = (0, 0, 0, 1, 1, 1)$ we get the knot vector $\Xi^* = (0, 0, 0, 0, \frac{1}{2}, \frac{1}{2}, 1, 1, 1, 1)$. On the right: applying k-refinement to the same knot vector, we get the knot vector $\Xi^* = (0, 0, 0, 0, \frac{1}{2}, 1, 1, 1, 1)$ gives a smaller number of functions, each of which is C^{p-1} across the element boundaries..

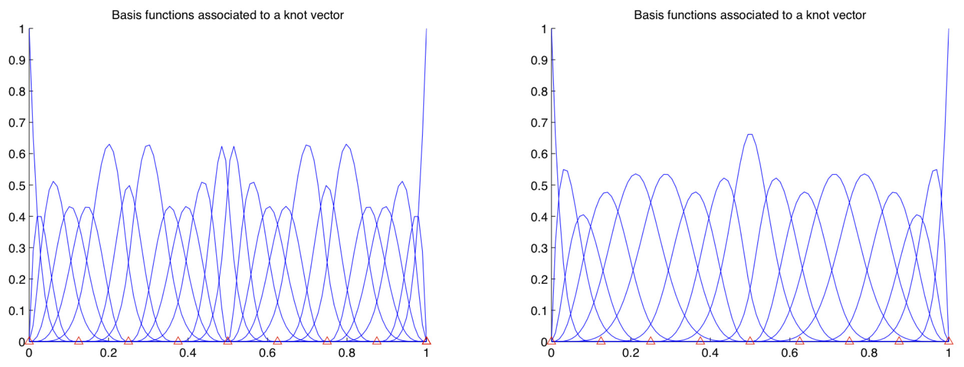


Figure 1.7: Same comparison as in Fig. 1.6, applying k-refinement three times to get functions of degree 5.

A parametric spline curve of order m in $\mathbb{R}^d$ is a vector function with d components

$$C(\hat{x}) = \begin{pmatrix} x_1 = c_1(\hat{x}) \\ \vdots \\ x_d = c_d(\hat{x}) \end{pmatrix} = \sum_{i=1}^{m+K} N_{i,m}(\hat{x}) P_i, \tag{1.19}$$

where $c_i(\hat{x})$ for $i = 1, \ldots, d$ are spline functions defined in Eq. (1.15), $N_{i,m}(\hat{x})$ are B-splines defined in Eq. (1.14), and $P_i \in \mathbb{R}^d$ are called control points.

Due to the properties of basis functions, spline curves are of class C^{m-1} at knots, unless there are repeated knots: repeating a knot m_i times decreases the number of continuous derivatives by m_i.

Spline curves don't interpolate the control points in general. A curve is interpolatory at the first and last control points when the knot vector is open, and at an internal control point if the knot vector has a knot with multiplicity $m - 1$.

The polyline connecting the control points of a curve, given by a piecewise linear interpolation of the control points, is referred to as control polygon. Spline curves have the property of affine invariance: an affine transformation of a spline curve is obtained by applying the same transformation to the control points.

Fig. 1.8 and 1.9 show the application of h and p-refinement to spline curves.

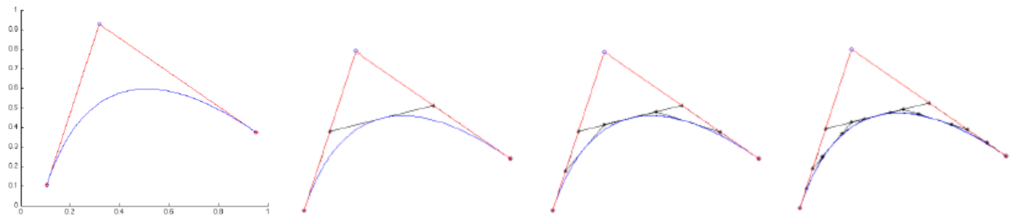


Figure 1.8: Three steps of h-refinement applied to the simple B-spline curve of $p = 2$ and knot vector $\Xi^* = (0, 0, 0, 1, 1, 1)$, displayed on the left. The control polygon of the original curve is red, while the ones of the refined curves are black.

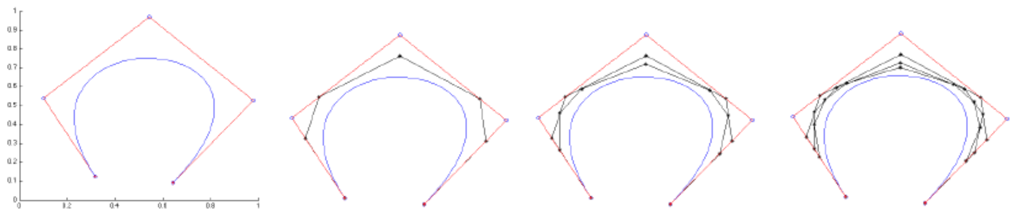


Figure 1.9: Three steps of p-refinement applied to the simple B-spline curve of $p = 3$ and knot vector $\Xi^* = (0, 0, 0, 0, 0.5, 1, 1, 1, 1)$, displayed on the left. The control polygon of the original curve is red, while the ones of the refined curves are black.

We can construct multivariate B-spline basis functions and spline surfaces starting from univariate B-spline basis functions using the tensor product. Let's consider two univariate spline functions spaces, namely $\mathcal{S}(\mathbb{P}_{m-1}, M, U)$ and $\mathcal{S}(\mathbb{P}_{n-1}, N, V)$, with knot vectors $U = (u_1, \ldots, u_k)$ and $V = (v_1, \ldots, v_l)$, and multiplicity vectors $M = (m_1, \ldots, m_k)$ and $N = (n_1, \ldots, n_l)$, such that $K = \sum_{i=1}^{k} m_i$ and $L = \sum_{i=1}^{l} n_i$.

If $\{P_{i,j}\}$, $i = 1, \ldots, m + K$, $j = 1, \ldots, n + L$ is a control net of points in

$\mathbb{R}^d$, a spline surface of order (m, n) is defined as

$$S(\hat{u}, \hat{v}) = \sum_{i=1}^{m+K} \sum_{j=1}^{n+L} N_{i,m}(\hat{u}) N_{j,n}(\hat{v}) P_{i,j}, \tag{1.20}$$

where $N_{i,m}$ and $N_{j,n}$ are the univariate B-spline basis functions of order m and n associated with the knot vectors U and V.

If all $P_{i,j} \in \mathbb{R}^2$, $S(\hat{u}, \hat{v})$ is on the XY plane, while if $P_{i,j} \in \mathbb{R}^3$, then $S(\hat{u}, \hat{v})$ can be a surface in $\mathbb{R}^2$.

Similarly, we can define solids. Given three knot vectors $U = (u_1, \dots, u_k)$, $V = (v_1, \dots, v_l)$ and $Z = (z_1, \dots, z_s)$, and a control lattice $\{P_{i,j,t}\}$, $i = 1, \dots, m + K,\ j = 1, \dots, n + L,\ k = 1, \dots, o + S$, a B-spline solid of order (m, n, o) is defined by

$$S(\hat{u}, \hat{v}, \hat{w}) = \sum_{i=1}^{m+K} \sum_{j=1}^{n+L} \sum_{t=1}^{o+S} N_{i,m}(\hat{u}) N_{j,n}(\hat{v}) N_{t,o}(\hat{w}) P_{i,j,t}. \tag{1.21}$$

The properties of surfaces and solids are bivariate and trivariate generalizations of those of curves. The basis functions are linearly independent, pointwise non-negative, locally supported and form a partition of unity.

1.5 Bernstein polynomials and Bézier curves

We will now introduce a set of basis functions $\{B_{i,p}(\hat{x})\}_{i=0}^{p}$ given by the classical pth-degree Bernstein basis polynomials, defined as:

$$B_{i,p}(\hat{x}) = \binom{p}{i} \hat{x}^i (1 - \hat{x})^{p-i}, \tag{1.22}$$

where $\hat{x} \in [0, 1]$ and $\binom{p}{i}$ is the binomial coefficient:

$$\binom{p}{i} = \frac{p!}{i!(p-i)!}. \tag{1.23}$$

The Bernstein basis polynomials $\{B_{i,p}(\hat{x})\}_{i=0}^{p}$ form a basis for $\mathbb{P}_p$, the space of polynomials of degree $\leq p$. A linear combination of Bernstein basis polynomials with coefficients $c_i \in \mathbb{R}$

$$b(\hat{x}) = \sum_{i=0}^{p} B_{i,p}(\hat{x}) c_i \tag{1.24}$$

is a Bernstein polynomial.

As we used B-spline basis functions to define spline curves in Eq. (1.19), we can use the Bernstein polynomials to define parametric Bézier curves as follows:

$$C(\hat{x}) = \sum_{i=0}^{p} B_{i,p}(\hat{x}) P_i, \tag{1.25}$$

where the geometric coefficients $P_i \in \mathbb{R}^d$ are called control points.

We remark that a B-spline basis function $N_{i,p+1}$ of order $p+1$, restricted to the interval $[0,1]$ without internal knots, coincides with a Bernstein basis polynomial or degree p. Thus, the set of B-spline basis functions $\{N_{i,p+1}\}_{i=1}^{p+1}$ reduces to the set $\{B_{i,p}(\hat{x})\}_{i=0}^{p}$ when we consider the extended knot vector of the form:

$$\Xi^* = \{\underbrace{0, \ldots, 0}_{p+1}, \underbrace{1, \ldots, 1}_{p+1}\}. \tag{1.26}$$

Therefore, Bézier curves defined in Eq. (1.25) are particular cases of B-spline curves defined in Eq. (1.19).

A spline geometry can always be decomposed into C^0-connected Bézier segments, that are called Bézier patches if the geometry represents a surface. Algorithms to get the piecewise Bézier form of a spline geometry can be found in [47]. They use the knot insertion process described in Section 1.3: each interior knot is inserted until it has multiplicity p. We will use Bézier curves in Chapter 4 to approximate the solution of some boundary value problems.

1.6 NURBS

NURBS are a popular modeling primitive suitable to represent both analytic shapes such as conics and quadrics, and free-form geometries like car parts, engines or ship hulls. They are a generalization of polynomial splines and of rational and non-rational Bézier curves.

In the following, we will introduce NURBS curves and surfaces. For the creation and manipulation of NURBS geometries in our figures and examples, we used the NURBS toolbox created by Mark Spink [54], in its extended version created for GeoPDEs [20].

1.6.1 NURBS curves

We will now define Non-Uniform Rational B-Splines (NURBS), which are a generalization of B-splines.

Let $\Xi = \{x_i\}_{i=1}^k$ be a knot vector for the interval $[a,b] \subset \mathbb{R}$. Let $M = (m_1, \dots, m_k)$ be the vector of multiplicities associated to Ξ, with $K = \sum_{i=1}^k m_i$ and let $\{N_{i,m}(x)\}_{i=1}^{m+K}$ be the B-spline basis functions.

Given a vector of weights $W = (w_1, \dots, w_k)$ such that $w_i > 0$, we define a NURBS function as:

$$r(\hat{x}) = \frac{\sum_{i=1}^{m+K} c_i w_i N_{i,m}(\hat{x})}{\sum_{i=1}^{m+K} w_i N_{i,m}(\hat{x})}, \quad c_i \in \mathbb{R}. \tag{1.27}$$

The space of the NURBS functions, denoted with $\mathcal{R}(\mathbb{P}_{m-1}, M, \Xi, W)$, has dimension $m + K$.

Let $\{P_i\}_{i=1}^{m+K}$ be a set of control points in $\mathbb{R}^d$, a parametric NURBS curve of order m is defined by:

$$C(\hat{x}) = \frac{\sum_{i=1}^{m+K} w_i P_i N_{i,m}(\hat{x})}{\sum_{i=1}^{m+K} w_i N_{i,m}(\hat{x})}. \tag{1.28}$$

Setting the rational B-spline of order m:

$$R_{i,m}(\hat{x}) = \frac{w_i N_{i,m}(\hat{x})}{\sum_{i=1}^{m+K} w_i N_{i,m}(\hat{x})},$$

Eq. (1.28) becomes:

$$C(\hat{x}) = \sum_{i=1}^{m+K} R_{i,m}(\hat{x}) P_i. \tag{1.29}$$

The set of rational B-Splines $\{R_{i,m}(\hat{x})\}_{i=1}^{m+K}$ constitutes a basis for the space $\mathcal{R}(\mathbb{P}_{m-1}, M, \Xi, W)$. They possess all the properties of the B-spline basis functions, and if all the weights w_i are equal to 1, they reduce to B-splines, i.e. $R_{i,m}(\hat{x}) \equiv N_{i,m}(\hat{x})$.

From a geometrical point of view, NURBS entities in $\mathbb{R}^d$ are projective transformations of B-spline entities in $\mathbb{R}^{d+1}$.

If all the weights of the control points of a curve are equal, they have equal influence on the shape of the curve. Increasing the weight of one control point gives it more influence and has the effect of "pulling" the curve toward that point, as shown in Fig. 1.10.

NURBS can be used to represent conic sections exactly. For instance, the top left curve of Fig. 1.10 is a NURBS circle, defined on the knot vector $\Xi^* = (0, 0, 0, 0.25, 0.25, 0.5, 0.5, 0.75, 0.75, 1, 1, 1)$.

1.6.2 NURBS surfaces

Analogously to the curves case, the rational basis functions for surfaces and solids are defined respectively as:

$$R_{i,j}^{m,n}(\hat{u}, \hat{v}) = \frac{w_{i,j} N_{i,m}(\hat{u}) M_{j,n}(\hat{v})}{\sum_{i=1}^{n+L} \sum_{j=1}^{m+K} w_{i,j} N_{i,m}(\hat{u}) M_{j,n}(\hat{v})} \tag{1.30}$$

and

$$R_{i,j,k}^{m,n,o}(\hat{u}, \hat{v}, \hat{w}) = \frac{N_{i,m}(\hat{u}) M_{j,n}(\hat{v}) L_{k,o}(\hat{w}) w_{i,j,k}}{\sum_{i=1}^{m+K} \sum_{j=1}^{n+L} \sum_{k=1}^{o+S} N_{i,m}(\hat{u}) M_{j,n}(\hat{v}) L_{k,o}(\hat{w}) w_{i,j,k}}. \tag{1.31}$$

To have a compact notation that does not depend on the dimension of the manifold, we will denote NURBS basis functions by $R_{\mathbf{i}}(\hat{\mathbf{x}})$, where $\mathbf{i}$ is a multi-index and $\hat{\mathbf{x}}$ is a point of the parameter space $\hat{\Omega} \subset \mathbb{R}^k$. Given a set of control points $B_i \in \mathbb{R}^d$, a NURBS geometry is a piecewise rational function and it is defined by the parameterization:

$$\begin{aligned} F : \hat{\Omega} &\longrightarrow \Omega \\ \hat{\mathbf{x}} &\longmapsto F(\hat{\mathbf{x}}) = \sum_{\mathbf{i}} R_{\mathbf{i}}(\hat{\mathbf{x}}) B_i. \end{aligned} \tag{1.32}$$

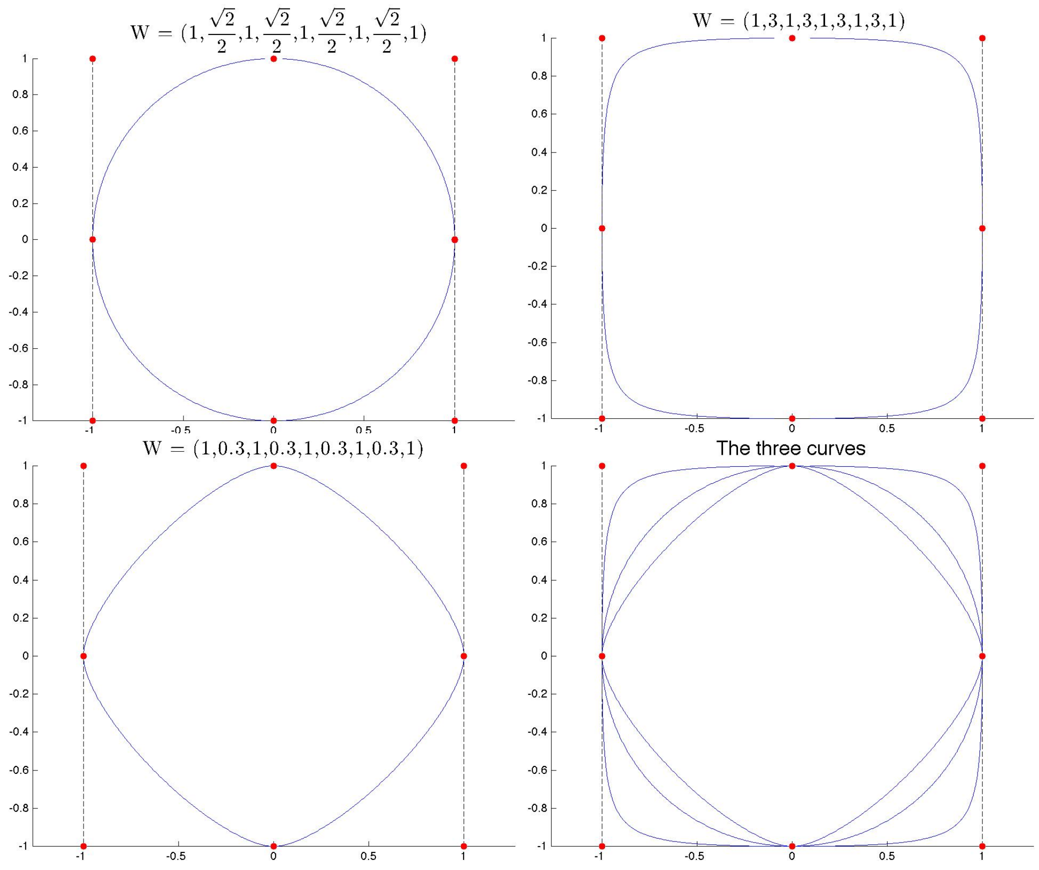


Figure 1.10: The geometric meaning of weights. Three NURBS curves of degree 2 that have the same knot vector and control points, with different weight vector W. Control points with greater weights pull the curves toward them.

This notation avoids distinguishing between curves, surfaces and solids.

A parametric NURBS surface of order m along the u direction and n along the v direction is a bivariate rational vector function of the form:

$$S(\hat{u},\hat{v}) = \frac{\sum_{i=1}^{n+L}\sum_{j=1}^{m+K} w_{ij}P_{i,j}N_{i,n}(\hat{u})N_{j,m}(\hat{v})}{\sum_{i=1}^{n+L}\sum_{j=1}^{m+K} w_{ij}N_{i,n}(\hat{u})N_{j,m}(\hat{v})} \tag{1.33}$$

Defining the rational basis functions as in Eq.(1.30), we can rewrite the surface as:

$$S(\hat{u},\hat{v}) = \sum_{i=1}^{m+K}\sum_{j=1}^{n+L} R_{i,j}^{m,n}(\hat{u},\hat{v})P_{i,j}. \tag{1.34}$$

Designing shapes using NURBS geometries is very intuitive. Commercial and free CAD system offer many simple and effective tools to model surfaces. Here we will briefly describe three tools that can be used to design surfaces starting from curves: extrusion surfaces, ruled surfaces and revolution surfaces.

<u>Extrusion surfaces</u>
Given a vector E and a NURBS curve $C(\hat{x}) = \sum_{i=1}^{m+K} R_{i,m}(\hat{x})P_i$ with knot vector U^* and weights $\{w_i\}_{i=1}^{m+K}$, an extrusion surface is obtained by sweeping the curve along the vector. It is defined as:

$$S(\hat{u},\hat{v}) = \sum_{i=1}^{m+K}\sum_{j=1}^{2} R_{i,j}^{m,2}(\hat{u},\hat{v})(\hat{v})P_{i,j}, \tag{1.35}$$

it has degree $(m-1,1)$, control points $P_{i,1} = P_i$ and $P_{i,2} = P_i + E$ with weights $w_{i1} = w_{i2} = w_i$, and knot vector U^* along the u direction and $V^* = [0,0,1,1]$ along the v direction.

<u>Ruled surfaces</u>
Let $C(\hat{x}) = \sum_{i=1}^{m+K} R_{i,m}(\hat{x})B_i$ and $C_1(\hat{x}) = \sum_{i=1}^{m+K} R_{i,m}(\hat{x})T_i$ be two NURBS curves, both with degree $m-1$ and knot vector U^*, and weights respectively $\{w_i^{[1]}\}_{i=1}^{m+K}$ and $\{w_i^{[2]}\}_{i=1}^{m+K}$. The ruled surface between C and C_1 is obtained by linear interpolation between every two corresponding points on C and C_1.

It has degree $(m-1,1)$, and it is defined as:

$$S(\hat{u},\hat{v}) = \sum_{i=1}^{m+K}\sum_{j=1}^{2} R_{i,j}^{m,2}(\hat{u},\hat{v})P_{i,j}, \tag{1.36}$$

where the control points of the surface are $P_{i,1} = B_i$ and $P_{i,2} = T_i$, the weights are $w_{i,1} = w_i^{[1]}$ and $w_{i,2} = w_i^{[2]}$ and the knot vectors are U^* along the u direction and $V^* = [0,0,1,1]$ along the v direction.

Two curves that have the same degree and knot vector like C and C_1, are said to be compatible. There are algorithms that turn non-compatible curves into compatible ones using knot insertion and degree elevation, thus making it possible to build ruled surfaces between non-compatible curves.

Revolution surfaces

Given a curve on the ZX-plane $C(\hat{x}) = \sum_{j=1}^{m+K} R_{j,m}(\hat{x})P_j$ with knot vector V^*, revolving it of 360 degrees around the Z-axis gives a revolution surface of degree $(2,m-1)$, defined as:

$$S(\hat{u},\hat{v}) = \sum_{i=1}^{9}\sum_{j=1}^{m+K} R_{i,j}^{3,m}(\hat{u},\hat{v})P_{i,j}. \tag{1.37}$$

The control points $B_{i,j}$ of the surface are obtained rotating the control points P_j of the curve around the Z-axis of $45(i-1)$ degrees, the weights are $w_{i,1} = w_i w_j$, with $w_i = [1, \frac{\sqrt{2}}{2}, 1, \frac{\sqrt{2}}{2}, 1, \frac{\sqrt{2}}{2}, 1, \frac{\sqrt{2}}{2}, 1]$. The knot vectors are $U^* = [0,0,0,\frac{1}{4},\frac{1}{4},\frac{1}{2},\frac{1}{2},\frac{3}{4},\frac{3}{4},1,1,1]$ along the u direction and V^* along the v direction.

Fig. 1.11 shows an example of each kind of surface described above.

1.7 Quad meshes

Polygonal meshes are simple and very common representations of geometries in computer graphics. A large number of applications in geometric

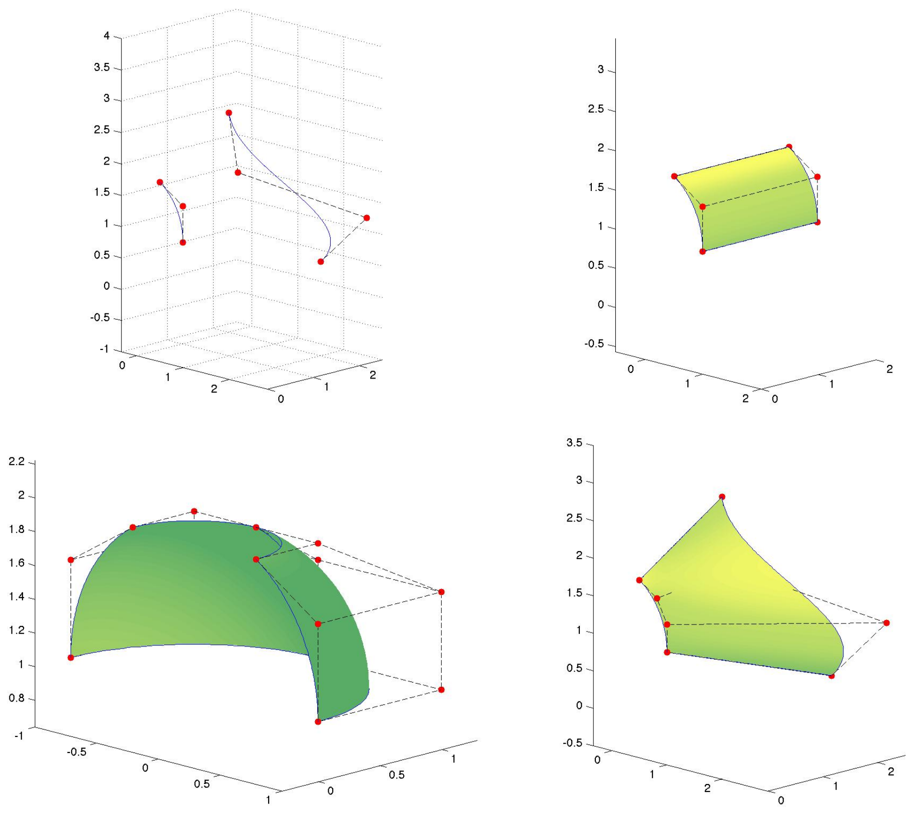


Figure 1.11: The two curves $c1$ and $c2$ (top left), an extrusion surface using $c1$ (top right), a revolution surface obtained by revolving $c1$ around the Z-axis(bottom left) and a ruled surface between $c1$ and $c2$ (bottom right).

modeling, computer graphics, mechanical engineering, simulation and architecture are based on polygonal meshes [6], and FEM is one of these applications.

The constituents of a polygonal mesh are:

- Vertices, that are points in $\mathbb{R}^3$. They can also be referred to as nodes.
- Edges, that are straight-line segments connecting two vertices.
- Facets, that are polygons bounded by a closed set of edges. They can also be referred to as faces, or elements in the FEM context.

We assume all our meshes to be conforming meshes, in which any two facets may share either a single vertex, or an entire edge. In other words, there cannot be T-junctions between two edges.

An edge shared by two incident facets is said to be internal, while an edge with just one incident facet is said to be boundary. A vertex of a boundary edge is also said to be boundary, otherwise it is said to be internal.

Discrete functions can be defined on a mesh by associating values either to the vertices or to the facets.

A triangle mesh is a mesh in which all facets are triangles, while a quad mesh is a mesh in which all the facets are quadrilateral.

The literature for triangle meshes is extensive. In the last several years the advantages of quadrilateral meshes for many applications have been pointed out, and significant progress was made in quadrilateral mesh generation and processing [7, 8, 18].

Eyeshot uses only triangular meshes in the computer graphics and geometry processing context, while, as we will see in Section 6.2, in the FEM context it supports also quad meshes. Fig. 1.12 shows three examples of polygonal meshes.

In the numerical examples of this thesis we will use quad meshes. To apply isogeometric analysis on them, we will interpret meshes as NURBS

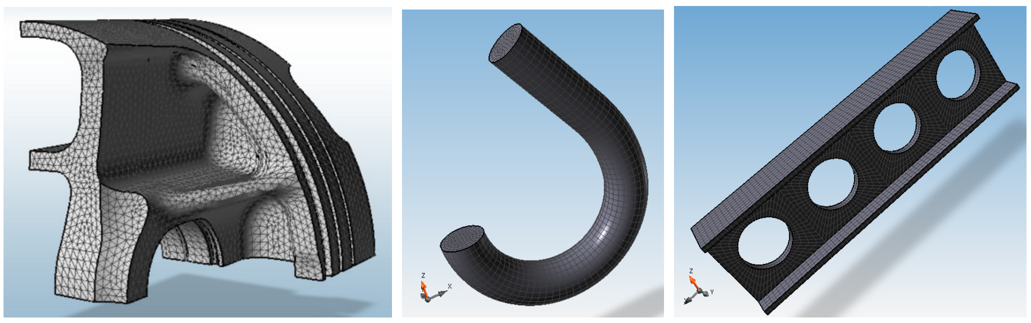


Figure 1.12: A triangular mesh (left) and two quadrilateral meshes (center and right).

surfaces of degree (1,1), since it is always possible to describe a quad mesh as the union of one or more such NURBS surfaces.

Among the three refinement techniques introduced in Section 1.3, h-refinement is commonly applied also to meshes. h-refinement can be performed by splitting the existing faces into smaller ones: it increases the number of nodes and elements of a mesh.

1.8 Extended B-rep

In several circumstances, some complicated models can be generated by combining multiple patches of simpler geometries. Geometries partitioned into two or more non-overlapping subdomains are called multipatch geometries and can be described using boundary representations, often abbreviated as B-rep, in which a solid is represented as a collection of connected surface elements.

In a B-rep model a solid is represented in terms of its bounding entities, such as:

- Vertices, that are points in $\mathbb{R}^3$.

- Edges, that are bounded, non-self-intersecting curves:
$$C : I \longrightarrow \mathbb{R}^3,$$
where $I = [t_0, t_1] \subset \mathbb{R}$.
Every edge is bounded by two vertices, $C(t_0)$ and $C(t_1)$, that can also be coincident.

- Loops, that are ordered alternating sequences of vertices and edges. Each loop defines a non-self-intersecting closed space curve, which may be a boundary of a face.

- Faces, that are finite, connected, non-self-intersecting regions of a closed, orientable surface bounded by one or more loops (at least an outer loop, and one or more inner loops if the face has holes).

A B-rep data structure B is formed by:

- a set of geometric data $G = (V, E, F)$ containing information of the vertices V (points in $\mathbb{R}^3$), edges E (curves in $\mathbb{R}^3$) and faces F (surfaces in $\mathbb{R}^3$),

- a set of topological data T providing the relationships among the geometric objects.

Fig. 1.13 shows an example of B-rep and some of its constituents.

In many situations we have hybrid geometries, in which some patches are described by mesh representations and other patches are NURBS. For this kind of geometries, we introduce a representation called Extended B-rep, that aims at closing the gap between parametric and discrete geometry in the representation of solid objects.

We will define a Mesh-Face as an open mesh delimited by a closed polygonal curve. This polygonal curve will be considered as the loop of the Mesh-Face and all the lines of the polygonal curve will be edges of the B-Rep data structure.

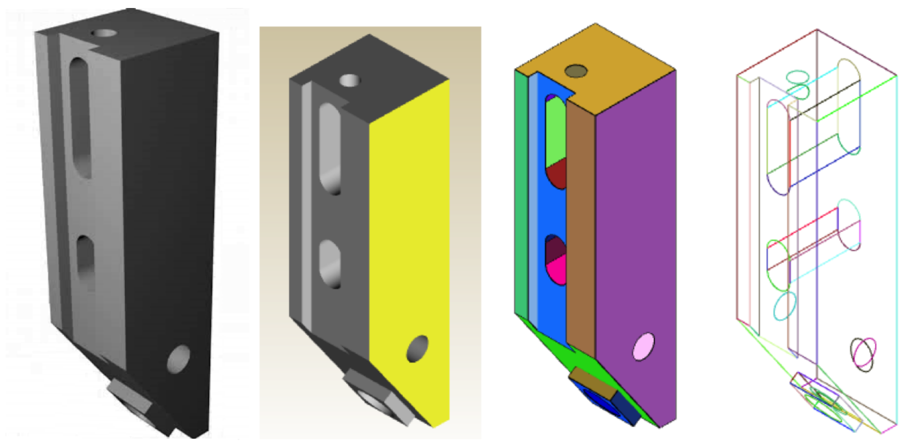


Figure 1.13: From left to right: a B-rep model, one face of the model is displayed in yellow, the faces of the model are displayed in different colors, the edges of the model are displayed in different colors.

An Extended B-Rep data structure is a representation scheme

$$B_e = (G_e, T)$$

where $G_e = (V, E, F_e)$ and the set of possible faces F_e admits also Mesh-Faces.

Fig. 1.14 shows an example of the usage of Extended B-reps to create a prototype that replicates a real cuneiform tablet.

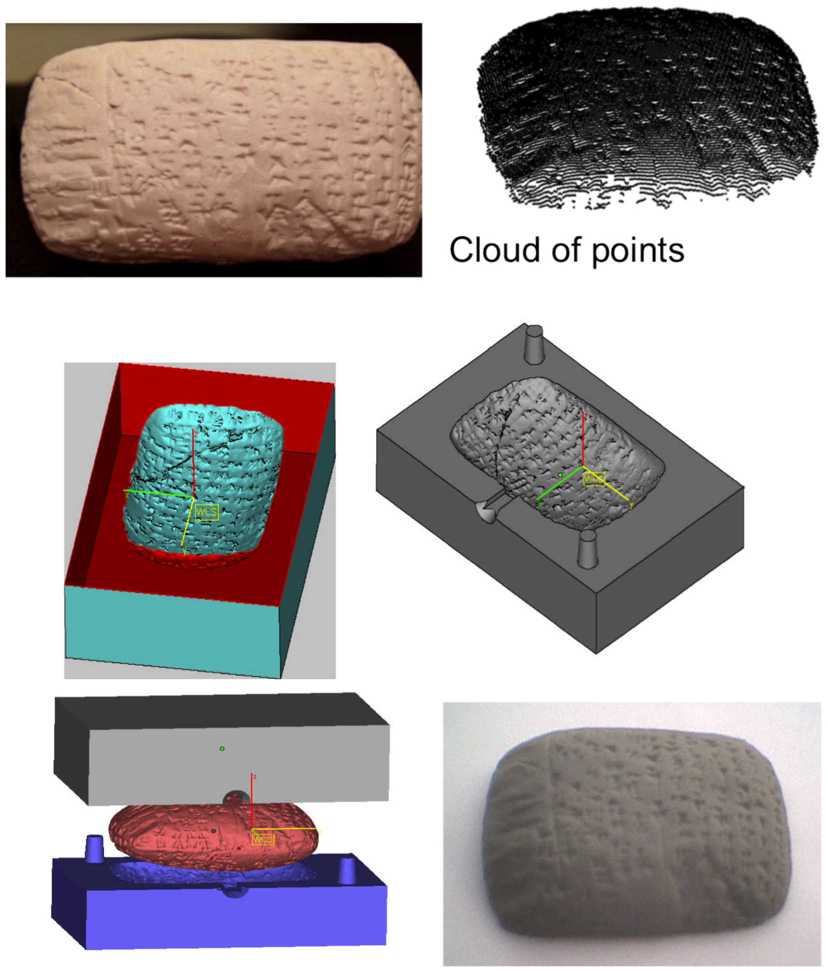


Figure 1.14: Top left: the original cuneiform tablet. Top right: the point cloud produced by a 3D scanner. Center: the molds for the prototype are Extended B-reps. Bottom: the prototype that replicates the cuneiform tablet.

Chapter 2

Boundary value problems and the Galerkin method

A Boundary Value Problem (BVP) is described by a differential equation and a set of additional constraints that the solution must satisfy on the boundary of the domain in which the problem is specified.

These problems arise in many physical and engineering fields, and they can be used to model several physical phenomena.

In this chapter we will introduce three elliptic BVPs: the Poisson problem, linear elasticity problems and the Laplace-Betrami problem.

2.1 The Poisson's equation

We will see how a BVP can be transformed into its weak form through the simple case of the Poisson's equation,

$$-\Delta u = f, \tag{2.1}$$

where Δ is the Laplace operator, that in two dimensions is defined by

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

and f is a function defined on a bounded domain Ω of $\mathbb{R}^2$, which is a bounded, open and connected set of $\mathbb{R}^2$. When f is the identically zero function, the equation (2.1) is called Laplace's equation.

These equations have both infinitely many solutions unless some boundary conditions are defined on the boundary $\partial\Omega$ of Ω. There are different kinds of boundary conditions, such as the Dirichlet condition and the Neumann condition. A Dirichlet BVP has the form

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega \\ u &= g \quad \text{on } \partial\Omega. \end{aligned} \tag{2.2}$$

A Neumann BVP has the form

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega \\ \frac{\partial u}{\partial n} &= h \quad \text{on } \partial\Omega, \end{aligned} \tag{2.3}$$

where n is the outward pointing normal vector to $\partial\Omega$, $\frac{\partial u}{\partial n}$ is the normal derivative of u on $\partial\Omega$, defined by

$$\frac{\partial u}{\partial n} = \nabla u(x,y) \cdot n(x,y), \tag{2.4}$$

and ∇u is the gradient of u

$$\nabla u(x,y) = \begin{bmatrix} \frac{\partial u}{\partial x}(x,y) \\ \frac{\partial u}{\partial y}(x,y) \end{bmatrix}.$$

A BVP can have mixed boundary conditions: if $\Gamma_D \subset \partial\Omega$ and $\Gamma_N \subset \partial\Omega$ form a partition of $\partial\Omega$, there are Dirichlet conditions on Γ_D and Neumann conditions on Γ_N. In general, the strong formulation of an elliptic BVP is:

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega \\ u &= g \quad \text{on } \Gamma_D \\ \frac{\partial u}{\partial n} &= h \quad \text{on } \Gamma_N, \end{aligned} \tag{2.5}$$

where $\Gamma_D \cup \Gamma_N = \partial\Omega$ and $\Gamma_D \cap \Gamma_N = \emptyset$, that is, Γ_D and Γ_N form a non overlapping partition of the boundary $\partial\Omega$.

The Poisson and Laplace's equations are used to model a variety of physical situations such as the heat transfer, electrostatics and gravitation. In the heat conduction context, the unknown u represents the distribution of temperature on a heat-conducting material occupying a domain $\Omega \subset \mathbb{R}^2$. The plate is insulated on the top and bottom, so the problem is two-dimensional. Since the equations does not depend on time, the problems modeled are the stationary heat flow for the Laplace's equation, and the stationary heat flow with a heat source or sink for the Poisson's one.
Dirichlet boundary conditions indicate that at each point $(x, y) \in \Omega$ the temperature is held fixed at $g(x, y)$. Neumann boundary conditions indicate that the heat flux (i.e. the flow of heat energy) across the boundary is equal to h. In particular, homogeneous Neumann conditions imply that also the boundary $\partial\Omega$ is insulated.

The accurate description of a physical material requires the introduction of the thermal conductivity κ in the non-dimensional equations presented in (2.5). From now on we will consider the following form of the Poisson's equation:

$$-\kappa\Delta u = f \quad \text{in } \Omega \tag{2.6}$$

in which κ is positive by definition. For heterogeneous materials, the thermal conductivity is not constant and the PDE in (2.6) is replaced by the following non-linear equation:

$$-\nabla \cdot (\kappa\nabla u) = f \quad \text{in } \Omega \tag{2.7}$$

where $\nabla\cdot$ is the divergence operator defined by

$$\nabla \cdot u = \frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}.$$

Thus, the strong formulation of a non-linear BVP is:

$$\boxed{\begin{aligned} -\nabla \cdot (\kappa \nabla u) &= f \quad \text{in } \Omega \\ u &= g \quad \text{on } \Gamma_D \\ \frac{\partial u}{\partial n} &= h \quad \text{on } \Gamma_N. \end{aligned}} \tag{2.8}$$

The divergence of the gradient is the laplacian

$$\nabla \cdot \nabla u = \nabla \cdot \begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \end{bmatrix} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

so, when κ is constant

$$-\nabla \cdot (\kappa \nabla u) = -\kappa \nabla \cdot \nabla u = -\kappa \Delta u.$$

While the Dirichlet BVP has a unique solution, either existence or uniqueness fails for a Neumann BVP of the form

$$\begin{aligned} -\kappa \Delta u &= f \quad \text{in } \Omega \\ \kappa \frac{\partial u}{\partial n} &= h \quad \text{on } \partial\Omega. \end{aligned}$$

The equation indicates that the heat energy added or taken away from the interior of Ω is regulated by f and the boundary condition states that the heat flux into Ω across $\partial\Omega$ is regulated by h. Since the system is in equilibrium, the total amount of heat flux must be zero, and this is expressed by the compatibility condition

$$\int_\Omega f + \int_{\partial\Omega} h = 0.$$

If the compatibility condition is not satisfied, the existence fails. On the other hand, if there is a solution u for the BVP, then $u + C$ is also a solution for any constant C because the derivatives in the equations are not affected by addictive constants.

2.2 The weak form of a BVP

In this section we derive the weak form of a boundary value problem and we will start by reminding some theorems and definitions that will be fundamental for this derivation.

The divergence theorem states that if F is a vector field defined on the closure $\bar{\Omega}$ of a domain $\Omega \subset \mathbb{R}^2$ that has a smooth or piecewise smooth boundary, then

$$\int_\Omega \nabla \cdot F = \int_{\partial\Omega} F \cdot n, \tag{2.9}$$

where n is the outward-pointing unit normal vector to $\partial\Omega$. This theorem relates a quantity defined on the interior of a domain with another quantity defined on its boundary.

The Green's identity is obtained by integrating over Ω both sides of the product rule in multiple dimensions:

$$\nabla \cdot (v\nabla u) = \nabla v \cdot \nabla u + v\Delta u$$
$$\int_\Omega \nabla \cdot (v\nabla u) = \int_\Omega \nabla v \cdot \nabla u + \int_\Omega v\Delta u.$$

Applying the divergence theorem we get

$$\int_{\partial\Omega} v\nabla u \cdot n = \int_\Omega \nabla v \cdot \nabla u + \int_\Omega v\Delta u,$$

and then replacing $\nabla u \cdot n$ with $\frac{\partial u}{\partial n}$ as in (2.4) we obtain the Green's identity

$$-\int_\Omega v\Delta u = \int_\Omega \nabla v \cdot \nabla u - \int_{\partial\Omega} v\frac{\partial u}{\partial n}. \tag{2.10}$$

Let $u : \Omega \longrightarrow \mathbb{R}$ be a locally integrable function (i.e. integrable over every compact subset of Ω), u is said to be weakly differentiable with respect to x if there exists a locally integrable function g defined on Ω such that

$$\int_\Omega gv = -\int_\Omega u\frac{\partial v}{\partial x} \quad \forall v \in C_0^\infty(\Omega). \tag{2.11}$$

In this case g is called the weak partial derivative with respect to x of u and is denoted by $\frac{\partial u}{\partial x}$. The weak partial derivative with respect to y is defined similarly.

We introduce three spaces of functions: the space of square-integrable functions

$$L^2(\Omega) = \left\{ v : \Omega \longrightarrow \mathbb{R} \, : \, \int_\Omega v^2 < \infty \right\},$$

the Sobolev space

$$H^1(\Omega) = \left\{ v \in L^2(\Omega) \, : \, \frac{\partial v}{\partial x}, \frac{\partial v}{\partial y} \in L^2(\Omega) \right\}, \tag{2.12}$$

and its subset

$$H_0^1(\Omega) = \left\{ v \in H^1(\Omega) \, : \, v = 0, \ \text{on}\, \partial\Omega \right\}.$$

We will derive the weak form of the Poisson BVP with homogeneous Dirichlet conditions:

$$\begin{aligned} -\nabla \cdot (\kappa \nabla u) = f \quad & \text{in } \Omega \\ u = 0 \quad & \text{on } \partial\Omega, \end{aligned} \tag{2.13}$$

that is problem (2.8) with $g = 0$ and $\Gamma_D = \partial\Omega$. If f is continuous and u is a solution of (2.13), we expect that u and its first and second order partial derivatives are continuous on Ω. And because of the boundary conditions u must be zero on $\partial\Omega$. So it is natural to look for a u in the subspace

$$C_D^2(\bar{\Omega}) = \left\{ u \in C^2(\bar{\Omega}) \, : \, u = 0 \quad \text{on } \partial\Omega \right\}.$$

Let v be a function defined on Ω that we will call a test function. Multiplying by v both sides of the Poisson's PDE in (2.13) and integrating over Ω we obtain

$$-\int_\Omega \nabla \cdot (\kappa \nabla u) v = \int_\Omega f v \tag{2.14}$$

We will show that if Eq.(2.14) holds for every test function v for a sufficiently large set, then the Poisson's PDE must hold.

Let $\mathbf{B}_\delta(x_0, y_0)$ be the ball centered at $(x_0, y_0) \in \Omega$ of radius $\delta > 0$ small enough that $\mathbf{B}_\delta(x_0, y_0) \subset \Omega$ and consider $v \in C_D^2(\bar{\Omega})$ such that

(i) $v(x,y) > 0$ for all $(x,y) \in \mathbf{B}_\delta(x_0,y_0)$

(ii) $v(x,y) = 0$ for all $(x,y) \notin \mathbf{B}_\delta(x_0,y_0)$

(iii) $\int_\Omega v = \int_{\mathbf{B}_\delta(x_0,y_0)} v = 1$.

Then for the two members of (2.14) we have

$$\int_\Omega fv = \int_{\mathbf{B}_\delta(x_0,y_0)} fv \xrightarrow[\delta\to 0]{} f(x_0,y_0)$$

and

$$-\int_\Omega \nabla\cdot(\kappa\nabla u)v = \int_{\mathbf{B}_\delta(x_0,y_0)} \nabla\cdot(\kappa\nabla u)v \xrightarrow[\delta\to 0]{} \nabla\cdot(\kappa(x_0,y_0)\nabla u(x_0,y_0)).$$

The integrals become exact equations in the limit. Therefore, if the space of test functions contains all such functions v, and (2.14) holds for all test functions, then the original PDE holds at every (x_0,y_0) in Ω. Taking $C^2_D(\bar{\Omega})$ as space of test functions, we will have that (2.14) holds for some $u \in C^2_D(\bar{\Omega})$ and for all test functions if and only if u is a solution for the BVP (2.13).

Applying the Green's identity to the left-hand side of (2.14) and using the fact that v vanishes on $\partial\Omega$, we get

$$-\int_\Omega \nabla\cdot(\kappa\nabla u)v = \int_\Omega \kappa\nabla u\cdot\nabla v - \int_{\partial\Omega}\kappa v\frac{\partial u}{\partial n} = \int_\Omega \kappa\nabla u\cdot\nabla v.$$

This leads to an equivalent form of the BVP (2.13):

$$\text{Find} \quad u\in C^2_D(\bar{\Omega}) \quad \text{s.t.} \int_\Omega \kappa\nabla u\cdot\nabla v = \int_\Omega fv \quad \forall\, v\in C^2_D(\bar{\Omega}) \tag{2.15}$$

called weak form. A function u satisfies one if and only if it satisfies the other.

The term weak is used because we want to make the weakest possible assumptions on the functions involved, so as to include as many solutions as possible. While the original PDE in (2.13) suggests that $u \in C^2(\Omega)$, the weak form refers only to the first derivatives of u and v. In the weak form, it is only necessary that u, v and their weak partial derivatives are in $L^2(\Omega)$, that is $u, v \in H^1(\Omega)$, and since they both need to satisfy the Dirichlet boundary condition, they must be in $H_0^1(\Omega)$. The weak form derived from the strong form (2.8) with homogeneous Dirichlet boundary condition ($g = 0$ and $\Gamma_D = \partial\Omega$), can be rewritten as:

$$\boxed{\text{Find} \quad u \in H_0^1(\Omega) \quad \text{s.t.} \int_\Omega \kappa \nabla u \cdot \nabla v = \int_\Omega f v \quad \forall\, v \in H_0^1(\Omega)} \tag{2.16}$$

where f is assumed to be in $L^2(\Omega)$. Since the requirements on f and on the solution u have been considerably weakened, this is called the weak form.

Let's now consider the Poisson BVP with inhomogeneous Dirichlet conditions

$$\begin{aligned} -\nabla \cdot (\kappa \nabla u) &= f \quad \text{in } \Omega \\ u &= g \quad \text{on } \partial\Omega, \end{aligned} \tag{2.17}$$

that is problem (2.8) with $\Gamma_D = \partial\Omega$. Let G be a known function in $H^1(\Omega)$, then $w = u - G$ is a function in $H_0^1(\Omega)$ and the solution u has the form $u = w + G$. Proceeding as in the homogeneous case, we multiply by a test function v and integrate over Ω

$$-\int_\Omega \nabla \cdot (\kappa(\nabla w + \nabla G))v = \int_\Omega f v \quad \forall\, v \in H_0^1(\Omega),$$

and applying the Green's identity we get

$$\int_\Omega \kappa(\nabla w + \nabla G) \cdot \nabla v - \int_{\partial\Omega} \kappa v \left(\frac{\partial w}{\partial n} + \frac{\partial G}{\partial n}\right) = \int_\Omega f v \quad \forall\, v \in H_0^1(\Omega).$$

Since $v \in H_0^1(\Omega)$, the integral over the boundary vanishes leaving

$$\int_\Omega \kappa(\nabla w + \nabla G) \cdot \nabla v = \int_\Omega f v \quad \forall\, v \in H_0^1(\Omega),$$

that can be rewritten as

$$\int_\Omega \kappa \nabla w \cdot \nabla v = \int_\Omega f v - \int_\Omega \kappa \nabla G \cdot \nabla v \quad \forall\, v \in H_0^1(\Omega).$$

Thus the weak form of the inhomogeneous Dirichlet problem (2.8) with $\Gamma_D = \partial\Omega$ is:

$$\boxed{\begin{aligned}&\text{Find } u = w + G,\ w \in H_0^1(\Omega),\\ &\qquad\int_\Omega \kappa \nabla w \cdot \nabla v = \int_\Omega fv - \int_\Omega \kappa \nabla G \cdot \nabla v \quad \forall\, v \in H_0^1(\Omega).\end{aligned}} \tag{2.18}$$

We will now derive the weak form of the Neumann problem

$$\begin{aligned} -\nabla \cdot (\kappa \nabla u) &= f \quad \text{in } \Omega \\ \kappa \frac{\partial u}{\partial n} &= h \quad \text{on } \partial\Omega, \end{aligned} \tag{2.19}$$

which is the strong form (2.8) with $\Gamma_N = \partial\Omega$. Multiplying by a test function $v \in H^1(\Omega)$ and integrating over Ω we obtain

$$-\int_\Omega \nabla \cdot (\kappa \nabla u) v = \int_\Omega fv \quad \forall\, v \in H^1(\Omega),$$

and applying the Green's identity we get

$$\int_\Omega \kappa \nabla u \cdot \nabla v - \int_{\partial\Omega} \kappa v \frac{\partial u}{\partial n} = \int_\Omega fv \quad \forall\, v \in H^1(\Omega). \tag{2.20}$$

So the weak form of the inhomogeneous Neumann problem is:

$$\boxed{\text{Find} \quad u \in H^1(\Omega) \quad \text{s.t.} \int_\Omega \kappa \nabla u \cdot \nabla v = \int_\Omega fv + \int_{\partial\Omega} vh \quad \forall\, v \in H^1(\Omega).} \tag{2.21}$$

If instead of inhomogeneous boundary conditions we consider homogeneous Neumann conditions, the second integral of the right-hand side of (2.20) would vanish because $\kappa\frac{\partial u}{\partial n} = 0$ on $\partial\Omega$ because of the boundary conditions. Then, the weak form of the homogeneous Neumann problem (2.8) with $h = 0$ and $\Gamma_N = \partial\Omega$ is:

$$\boxed{\text{Find} \quad u \in H^1(\Omega) \quad \text{s.t.} \int_\Omega \kappa \nabla u \cdot \nabla v = \int_\Omega fv \quad \forall\, v \in H^1(\Omega).} \tag{2.22}$$

While the Dirichlet condition appears explicitly in the weak forms (2.16) and (2.18) through the definition of the space $H_0^1(\Omega)$, the Neumann condition does not appear but is implied in (2.21) and (2.22). This is why the Dirichlet conditions are called essential and the Neumann conditions are called natural boundary conditions.

We will now consider a BVP with homogeneous mixed boundary conditions, like the problem in Eq. (2.8), with $g = 0$ and $h = 0$. The space of test functions is defined by

$$V := H^1_{0,\Gamma_D}(\Omega) = \{v \in H^1(\Omega) \, : \, v = 0 \text{ on } \Gamma_D\},$$

as usual, to derive the weak form, we multiply the PDE for a test function v and integrate over Ω

$$-\int_\Omega \nabla \cdot (\kappa \nabla u) v = \int_\Omega fv \quad \forall\, v \in V,$$

then we apply the Green's identity and get

$$\int_\Omega \kappa \nabla u \cdot \nabla v - \int_{\partial\Omega} \kappa v \frac{\partial u}{\partial n} = \int_\Omega fv \quad \forall\, v \in V.$$

The boundary integral can be written as

$$\int_{\partial\Omega} \kappa v \frac{\partial u}{\partial n} = \int_{\Gamma_D} \kappa v \frac{\partial u}{\partial n} + \int_{\Gamma_N} \kappa v \frac{\partial u}{\partial n},$$

where both integrals of the right-hand side vanish because of the boundary conditions. The weak form of problem (2.8) with $g = 0$ and $h = 0$ is therefore

$$\boxed{\text{Find} \quad u \in V \quad \text{s.t.} \int_\Omega \kappa \nabla u \cdot \nabla v = \int_\Omega fv \quad \forall\, v \in V.} \tag{2.23}$$

In an analogous manner, it can be derived that the weak form of the problem in (2.8), with $g = 0$ and $h \neq 0$ is

$$\boxed{\text{Find } u \in V \quad \text{s.t.} \int_\Omega \kappa \nabla u \cdot \nabla v = \int_\Omega fv + \int_{\Gamma_N} hv \quad \forall\, v \in V.} \tag{2.24}$$

The weak form of the inhomogeneous mixed boundary condition problem, given by (2.8), with $g \neq 0$ and $h \neq 0$ is

$$\boxed{\begin{aligned} &\text{Find } u = w + G, \ w \in V, \\ &\int_\Omega \kappa \nabla w \cdot \nabla v = \int_\Omega fv - \int_\Omega \kappa \nabla G \cdot \nabla v + \int_{\Gamma_N} hv \quad \forall\, v \in V. \end{aligned}} \tag{2.25}$$

The space V is a Hilbert space (i.e. a complete inner product space), and the weak forms (2.21), (2.22), (2.23) and (2.25) can be written in the form

$$\text{Find } u \in V,\ a(u,v) = l(v) \quad \forall\, v \in V, \tag{2.26}$$

where $a(\cdot,\cdot)$ is a symmetric bilinear form on V and l is a continuous linear functional on V.

The Riesz representation theorem, guarantees the existance of a unique solution u for the Dirichlet and mixed BVP. In case of Neumann BVP, if a solution exists, it is not unique [30].

2.3 Linear isotropic elasticity

We will now analyze another stationary elliptic BVP used to describe the mechanics of solids and structures that are stressed, i.e. subjected to loads or forces. Stresses lead to strains, which can be observed as deformations or displacements; the aim of structural and solid mechanics is to understand the relationships between all these variables.

Materials can be isotropic, meaning that their properties and elastic response is the same in every direction, or anisotropic, if they vary with direction. If a model problem is subject to very small deformations, the relationship between loads and deformations is linear. The problems that Eyeshot deals with, and that we will introduce here in two dimensions, are linear, isotropic and elastic.

The model is constituted by an elastic membrane that occupies a domain Ω on which a load f is applied. We want to know how the membrane will respond to that load, we want to know the entity of the deformations that will occur. So the unknown is the displacement of each point $(x, y) \in \Omega$, described by the vector-valued function

$$u(x,y) = \begin{bmatrix} u_1(x,y) \\ u_2(x,y) \end{bmatrix}.$$

An isotropic material is described by two scalar quantities μ and λ, called the Lamé moduli, that are constants if the membrane is homogeneous and that must be measured experimentally for each material. The PDE that describes our model, written in vector form is

$$\boxed{\begin{aligned} -\nabla\cdot\sigma = f \quad &\text{in } \Omega \\ \sigma &= 2\mu\epsilon + \lambda \mathrm{tr}(\epsilon) I, \\ \epsilon &= \frac{1}{2}(\nabla u + \nabla u^T). \end{aligned}} \tag{2.27}$$

The quantity ϵ is called strain tensor, it is a measure of the local deformation of the membrane and it can be written as

$$\epsilon = \frac{1}{2}(\nabla u + \nabla u^T) = \frac{1}{2}\begin{bmatrix} 2\frac{\partial u_1}{\partial x} & \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \\ \frac{\partial u_2}{\partial x} + \frac{\partial u_1}{\partial y} & 2\frac{\partial u_2}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial u_1}{\partial x} & \frac{1}{2}(\frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x}) \\ \frac{1}{2}(\frac{\partial u_2}{\partial x} + \frac{\partial u_1}{\partial y}) & \frac{\partial u_2}{\partial y} \end{bmatrix}.$$

The tensor σ is the stress tensor and measures the elastic response to the deformation described by the strain. Using the above expression for the strain, σ becomes

$$\sigma = 2\mu\epsilon + \lambda\mathrm{tr}(\epsilon)I = 2\mu \begin{bmatrix} \frac{\partial u_1}{\partial x} & \frac{1}{2}(\frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x}) \\ \frac{1}{2}(\frac{\partial u_2}{\partial x} + \frac{\partial u_1}{\partial y}) & \frac{\partial u_2}{\partial y} \end{bmatrix} + \lambda\left(\frac{\partial u_1}{\partial x} + \frac{\partial u_2}{\partial y}\right)\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$
$$= \begin{bmatrix} 2\mu\frac{\partial u_1}{\partial x} + \lambda(\frac{\partial u_1}{\partial x} + \frac{\partial u_2}{\partial y}) & \mu(\frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x}) \\ \mu(\frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x}) & 2\mu\frac{\partial u_2}{\partial y} + \lambda(\frac{\partial u_1}{\partial x} + \frac{\partial u_2}{\partial y}) \end{bmatrix}.$$

The divergence of a tensor is a vector whose components are the divergencies of the rows of the tensor, so, if λ and μ are constants, (2.27) is equivalent to:

$$\begin{aligned} -(2\mu+\lambda)\frac{\partial^2 u_1}{\partial x^2} - \mu\frac{\partial^2 u_1}{\partial y^2} - (\mu+\lambda)\frac{\partial^2 u_2}{\partial x\partial y} &= f_1, \\ -(\mu+\lambda)\frac{\partial^2 u_1}{\partial x\partial y} - \mu\frac{\partial^2 u_2}{\partial x^2} - (2\mu+\lambda)\frac{\partial^2 u_2}{\partial y^2} &= f_2. \end{aligned}$$

Rearranging the addends of these equalities, we have

$$-\mu(\frac{\partial^2 u_1}{\partial x^2}+\frac{\partial^2 u_1}{\partial y^2})-(\mu+\lambda)(\frac{\partial^2 u_1}{\partial x^2}+\frac{\partial^2 u_2}{\partial x \partial y})=f_1,$$
$$-\mu(\frac{\partial^2 u_2}{\partial x^2}+\frac{\partial^2 u_2}{\partial y^2})-(\mu+\lambda)(\frac{\partial^2 u_1}{\partial x \partial y}+\frac{\partial^2 u_2}{\partial y^2})=f_2,$$

that leads to the more concise equivalent form

$$\mu\Delta u+(\mu+\lambda)\nabla(\nabla\cdot u)=-f. \tag{2.28}$$

The right-hand side of the PDE, f, represents a body force applied on the interior of the domain Ω. Often a load is introduced by a traction on the boundary through Neumann boundary conditions.

In practical applications it is very common to use, instead of the Lamé moduli μ and λ, the Young modulus E and the Poisson ratio ν. Both pairs of constants can easily be obtained from the other using conversion formulas.

We will now derive the weak form of the BVP (2.27) with mixed homogeneous boundary conditions

$$\begin{aligned} -\nabla\cdot\sigma &= f \quad \text{in } \Omega \\ u &= 0 \quad \text{on } \Gamma_D, \\ \sigma n &= 0 \quad \text{on } \Gamma_N. \end{aligned} \tag{2.29}$$

For this derivation it will be fundamental an alternate version of the Green's identity given by

$$-\int_\Omega(\nabla\cdot\sigma)\cdot v=\int_\Omega \sigma\cdot\epsilon_v-\int_{\partial\Omega} v\cdot(\sigma n). \tag{2.30}$$

As space of test functions, we will use

$$V=\{v\in(H^1(\Omega))^2\ :\ v=0 \text{ on } \Gamma_D\},$$

where

$$(H^1(\Omega))^2=\{v:\Omega\longrightarrow\mathbb{R}^2\ :\ v_1,v_2\in H^1(\Omega)\}.$$

We multiply both members of the PDE by a test function v and integrate over Ω

$$-\int_\Omega (\nabla \cdot \sigma_u) \cdot v = \int_\Omega f \cdot v \quad \forall\, v \in V,$$

and applying (2.30), we get

$$\int_\Omega \sigma_u \cdot \epsilon_v - \int_{\partial\Omega} v \cdot (\sigma_u n) = \int_\Omega f \cdot v \quad \forall\, v \in V.$$

The boundary integral can be decomposed into two integrals over Γ_D and Γ_N that both vanish because v is zero on Γ_D and $\sigma_u n$ is zero on Γ_N, thanks to the boundary conditions. So we have

$$\int_\Omega \sigma_u \cdot \epsilon_v = \int_\Omega f \cdot v \quad \forall\, v \in V,$$

which is equivalent to

$$\int_\Omega (2\mu\epsilon_u + \lambda \mathrm{tr}(\epsilon_u) I) \cdot \epsilon_v = \int_\Omega f \cdot v \quad \forall\, v \in V,$$

and since $I \cdot \epsilon_v = \mathrm{tr}(\epsilon_v)$, the weak form of the BVP is

$$\boxed{\text{Find } u \in V, \int_\Omega (2\mu\epsilon_u \cdot \epsilon_v + \lambda \mathrm{tr}(\epsilon_u)\mathrm{tr}(\epsilon_v)) = \int_\Omega f \cdot v \quad \forall\, v \in V.} \tag{2.31}$$

Similarly, considering only homogeneous Dirichlet conditions and inhomogeneous Neumann conditions ($\sigma n = h$ on Γ_N), the weak form becomes

$$\boxed{\text{Find } u \in V, \int_\Omega (2\mu\epsilon_u \cdot \epsilon_v + \lambda \mathrm{tr}(\epsilon_u)\mathrm{tr}(\epsilon_v)) = \int_\Omega f \cdot v + \int_{\Gamma_N} h \cdot v \quad \forall\, v \in V.} \tag{2.32}$$

2.4 The Galerkin method

The Galerkin method solves BVPs by computing the best approximation to the true solution in a given finite-dimensional subspace. The Galerkin method is based on the projection theorem, that states that if V is an inner product space with inner product $(\cdot, \cdot)$, V_h is a finite-dimensional subspace of V with dimension N_h and $u \in V$, then:

(i) there is a unique vector $w \in V_h$ such that

$$\|u - w\| < \|u - z\| \quad \forall\, z \in V_h, \quad z \neq w$$

and w is called the best approximation to u from V_h.

(ii) a vector w is the best approximation to u from V_h if and only if it satisfies the following orthogonality condition:

$$w \in V_h, \quad (u - w, z) = 0 \quad \forall\, z \in V_h \tag{2.33}$$

The subscript h in V_h is related to the size of the spacial discretization of the domain Ω.

Let $\{v_1, \dots, v_{N_h}\}$ be a basis of V_h, if $w = \sum_{j=1}^{N_h} \alpha_j v_j$ satisfies (2.33), then taking $z = v_i$ we have that

$$(u - w, z) = \left(u - \sum_{j=1}^{N_h} \alpha_j v_j, v_i \right) = (u, v_i) - \sum_{j=1}^{N_h} \alpha_j (v_j, v_i) = 0, \quad i = 1, \dots, N_h$$

which means that w is determined by the following system of linear equations in the unknowns $\alpha_1, \dots, \alpha_{N_h}$:

$$\sum_{j=1}^{N_h} (v_j, v_i) \alpha_j = (u, v_i) \quad i = 1, \dots, N_h. \tag{2.34}$$

As we showed before, the weak form of a BVP can be written as:

$$\text{Find } u \in V,\ a(u, v) = l(v) \quad \forall\, v \in V, \tag{2.35}$$

where $a(\cdot, \cdot)$ is a symmetric bilinear form on V and l is a continuous linear functional on V. When the BVP is elliptic, $a(\cdot, \cdot)$ defines an alternate inner product on V, called energy inner product, that is used by the Galerkin method to find the best approximation w to the exact solution u from a finite-dimensional subspace V_h of V.

In matrix-vector form, the linear system of equations (2.34) can be rewritten as

$$KU = F, \tag{2.36}$$

where K is a $N_h \times N_h$ matrix called stiffness matrix and defined by

$$K_{ij} = a(v_j, v_i), \quad i, j = 1, \dots, N_h$$

and F is the load vector defined by

$$F_i = l(v_i),$$

the approximate solution is given by

$$w = \sum_{i=1}^{N_h} U_i v_i. \tag{2.37}$$

Because of (2.33), w satisfies

$$a(w, v_i) = a(u, v_i) = l(v_i) \quad i = 1, \dots, n,$$

and since $\{v_1, \dots, v_{N_h}\}$ is a basis of V_h, we can rewrite (2.35) replacing V with its subspace V_h, obtaining the Galerkin form:

$$\text{Find } w \in V_h,\ a(w, v) = l(v) \quad \forall\, v \in V_h. \tag{2.38}$$

The existence and uniqueness of a solution is guaranteed by the Lax-Milgram theorem.

Theorem 2.4.1 (Lax-Milgram thorem)**.** *Let V be a Hilbert space and let $a(\cdot, \cdot)$ be a bounded and V-elliptic bilinear form on V. Then:*

1. there exists $\alpha > 0$ such that $a(v, v) \geqslant \alpha \|v\|^2 \quad \forall\, v \in V$

2. there exists $\beta > 0$ such that $a(w, v) \leqslant \beta \|w\| \|v\| \quad \forall\, w, v \in V$.

Given a continuous linear form $l(\cdot)$, there exists a unique solution $u \in V$ such that:

$$a(u, v) = l(v) \quad \forall\, v \in V.$$

The approximate solution found by the Galerkin method is the best approximation in the energy norm [1], and not in the original norm $\|\cdot\|$ of V. That is, for $w \in V_h$, $\|u-w\|_E = \min\limits_{v\in V_h} \|u-v\|_E$, where u is the exact solution.

A fundamental result for the estimation of the discretization error is Cea's theorem, which states that:

$$\|u-w\| \leqslant \frac{\beta}{\alpha}\|u-v\| \quad \forall\, v \in V_h,$$

In other words, the subspace solution w is the best approximation of u in V_h, up to the constant $\frac{\beta}{\alpha}$.

2.5 The Laplace-Beltrami problem

The Laplace operator can be generalized into the Laplace-Beltrami operator, that operates on functions defined on arbitrary Riemannian manifolds.

The Laplace-Beltrami operator allows to generalize the Poisson equation to Riemannian manifolds of dimension k embedded in the physical space $\mathbb{R}^d$, with $d > k \geq 1$. A problem described by this generalized equation is called Laplace-Beltrami problem, and it can be applied on lower dimensional manifolds, such as surfaces embedded in $\mathbb{R}^3$.

2.5.1 Tangential differential operators

In this section we will introduce the Laplace-Beltami operator on a manifold Ω of dimension k embedded in the physical space $\mathbb{R}^d$, with $d > k \geq 1$.

Let $\phi : \Omega \longrightarrow \mathbb{R}$ be a function on Ω, and let $\tilde{\phi}$ be the smooth extension of ϕ to a neighborhood $\mathcal{U}$ of Ω, so that $\tilde{\phi}|_\Omega = \phi$.

For each $\xi \in \mathcal{U}$, the tangential projector operator $\mathrm{P}(\xi) \in \mathbb{R}^{d\times d}$ is defined as

$$\mathrm{P}(\xi) := \mathrm{I} - \mathrm{n}_\Omega(\xi) \otimes \mathrm{n}_\Omega(\xi), \tag{2.39}$$

[1]The energy norm is denoted by $\|\cdot\|_E$ and defined as $\|v\|_E = \sqrt{a(v,v)}$

where I is the identity tensor in $\mathbb{R}^{d\times d}$, and $\mathrm{n}_\Omega(\xi)$ is the unit normal to Ω in $\mathbb{R}^d$.

This allows to define the tangential gradient of a function $\phi \in C^1(\Omega)$ as the projection of the gradient of its smooth extension $\tilde{\phi}$ onto the tangent space of Ω. That is:

$$\nabla_\Omega \phi(\xi) := \mathrm{P}(\xi)\nabla\tilde{\phi}(\xi), \tag{2.40}$$

where ∇ represents the usual gradient operator in $\mathbb{R}^d$.

Finally, we introduce the Laplace-Beltrami operator associated to the manifold Ω for a function $\phi \in C^2(\Omega)$ as:

$$\Delta_\Omega \phi(\xi) := \nabla_\Omega \cdot (\nabla_\Omega \phi(\xi)), \tag{2.41}$$

where the divergence operator $\nabla_\Omega\cdot$ for a vector valued function $\mathrm{v} : \Omega \longrightarrow \mathbb{R}^d$ of class C^1 is defined as the trace of its tangential gradient:

$$\nabla_\Omega \cdot \mathrm{v}(\xi) = \mathrm{tr}(\nabla_\Omega \mathrm{v}(\xi)). \tag{2.42}$$

If we have a parametric representation of the manifold Ω like the one defined in Section 1.1 with the metric $\hat{g}$ defined in (1.5), we can use its geometrical mapping $F : \hat{\Omega} \longrightarrow \Omega$ to rewrite the tangential gradient and the Laplace-Beltrami operator in terms of quantities in the parameter space $\hat{\Omega}$.

Recalling the definitions of Jacobian J and first fundamental form G of F given in Eq. (1.2) and (1.4) respectively, the gradient of $\tilde{\phi}$ in the physical space is given by:

$$\nabla\tilde{\phi}(\xi) = \left[\hat{J}(\hat{\xi})\hat{G}^{-1}(\hat{\xi})\hat{\nabla}\hat{\phi}(\hat{\xi})\right] \circ F^{-1}(\hat{\xi}), \tag{2.43}$$

where $\hat{\nabla}\hat{\phi} : \hat{\Omega} \longrightarrow \mathbb{R}^k$ is the gradient operator in the parameter space.

The tangential gradient of ϕ is:

$$\nabla_\Omega\phi(\xi) = \left[\hat{J}(\hat{\xi})\hat{G}^{-1}(\hat{\xi})\hat{\nabla}\hat{\phi}(\hat{\xi})\right] \circ F^{-1}(\hat{\xi}). \tag{2.44}$$

The Laplace-Beltrami operator of a function $\phi : \Omega \longrightarrow \mathbb{R}$ can be rewritten as:

$$\Delta_\Omega \phi(\xi) = \left[\frac{1}{\hat{g}(\hat{\xi})} \hat{\nabla} \cdot \left(\hat{g}(\hat{\xi}) \hat{G}^{-1}(\hat{\xi}) \hat{\nabla} \hat{\phi}(\hat{\xi}) \right) \right] \circ F^{-1}(\hat{\xi}), \tag{2.45}$$

where g is the determinant of the first fundamental form, defined in Eq. (1.5).

Finally, in view of the definition of integrals, the differential $d\xi$ used in integrals is transformed as $g(\hat{\xi})d\xi$.

2.5.2 The Laplace-Beltrami equation

Having defined the tangential differential operators, we can formulate a BVP on the manifold Ω that generalizes the Poisson problem. Given a source function $f \in L^2(\Omega)$, the strong formulation of the Laplace-Beltrami problem with homogeneous boundary conditions is:

$$\boxed{\begin{aligned} -\Delta_\Omega u = f &\quad \text{in } \Omega \\ u = 0 &\quad \text{on } \Gamma_D \\ n_\Gamma \cdot \nabla_\Omega u = 0 &\quad \text{on } \Gamma_N, \end{aligned}} \tag{2.46}$$

where n_Γ is the unit vector normal to the boundary Γ.

Introducing the symmetric bilinear form:

$$a(v, w) := \int_\Omega \nabla_\Omega v \cdot \nabla_\Omega w \; d\xi, \tag{2.47}$$

the linear functional form:

$$l(v) := \int_\Omega f v \; d\xi, \tag{2.48}$$

and the test function space:

$$V := \{ v \in H^1(\Omega) \, : \, v|_{\Gamma_D} = 0 \}, \tag{2.49}$$

the weak form of the Laplace-Beltrami reads:

$$\boxed{\text{Find } u \in V, \quad a(u,v) = l(v) \quad \forall\, v \in V.} \tag{2.50}$$

We observe that, if the manifold Ω is not endowed with a boundary, then the Laplace-Beltrami problem (2.50) is ill-posed.

To rewrite Eq.(2.50) in the parameter space $\hat{\Omega}$, we can use again the geometrical mapping $F : \hat{\Omega} \longrightarrow \Omega$, together with Eq(2.44). The weak form of problem (2.46) is:

$$\boxed{\text{Find } \hat{u} \in \hat{V}, \quad \hat{a}(\hat{u},\hat{v}) = \hat{l}(\hat{v}) \quad \forall\, \hat{v} \in \hat{V},} \tag{2.51}$$

where $\hat{V} := \{\hat{v} \in H^1(\hat{\Omega}) \,:\, \hat{v}|_{\hat{\Gamma}_D} = 0\}$,

$$\hat{a}(\hat{v},\hat{w}) := \int_{\hat{\Omega}} \hat{\nabla}\hat{v} \cdot \left(\hat{G}^{-1}\hat{\nabla}\hat{w}\right) \hat{g}\; d\hat{\xi}, \tag{2.52}$$

and

$$\hat{l}(\hat{v}) := \int_{\hat{\Omega}} \hat{f}\hat{v}\hat{g}\; d\hat{\xi}. \tag{2.53}$$

If the requirements of the Lax-Milgram lemma are satisfied, the solution to the Laplace-Beltrami problem exists and is unique.

A priori error estimates under h-refinement for the Laplace-Beltrami problem and other BVP on lower dimensional manifolds are given in [19].

Chapter 3

Isogeometric analysis

Isogeometric Analysis is an approximation method based on the isoparametric concept. It was introduced in 2005 in [34] with the purpose of unifying the worlds of CAD and FEA. Simulations are performed in FEA systems, which require the conversion of designs, made with CAD systems, into analysis-suitable formats leading to finite element meshes. This conversion process is very laborious, and takes more than 80% of the analysis time. Isogeometric Analysis was created to address this problem by developing a new paradigm for FEA that uses geometric designs made by CAD systems, without converting them into meshes. These geometric models are suitable for both design and analysis.

Since its introduction, Isogeometric Analysis has become a focus of research within both the fields of FEA and CAD and it has attracted a lot of interest from the scientific community.

In this chapter we will present the IGA framework, using NURBS geometries to represent both the geometry and the solutions of BVP.

3.1 The Finite Element Method

FEM is a technique developed during the 1960s and 1970s that is widely used to find approximate solutions to boundary value problems for partial differential equations. As discussed in the last section, to solve a BVP, the Galerkin method requires the solution of the linear system (2.36). For an implementation of the method, it is crucial to choose the finite-dimensional approximating subspace V_h in a way that guarantees that the stiffness matrix K, the load vector F and the solution of the system can be computed efficiently. In addition, the computed solution should be a good approximation of the true solution. When in the Galerkin problem the space of piecewise polynomial function is chosen as approximating subspace V_h, we obtain the finite element method.

This choice implies that the domain Ω is discretized by a mesh of polygonal facets (or polyhedral for tridimensional domains) denoted by T_h. In the FEM context, the polygonal facets are usually called elements and their vertices are called nodes.

Let T_h be a discretization of the domain Ω, where $h \in \mathbb{R}$ is referred to as size of the mesh, a piecewise polynomial is a function that is defined by a polynomial on each element K_j of the mesh. Therefore the space V_h is the space of piecewise polynomials of degree p on Ω is defined as:

$$P\mathbb{P}_p^h(\Omega) = \{v_h \in C^0(\bar{\Omega}) : v_h|_{K_j} \in \mathbb{P}_p,\ \forall K_j \in T_h\}, \tag{3.1}$$

where $\mathbb{P}_p$ is the space of polynomials of degree $\leq p$ and $\bar{\Omega}$ denotes the closure of Ω.

The simplest spaces of continuous piecewise polynomial functions are the space of piecewise linear functions $P\mathbb{P}_1^h$, together with $P\mathbb{P}_2^h$, which are often chosen as approximating subspace in practical applications. Typically, Lagrange polynomials are used as basis functions.

Using continuous piecewise polynomial functions as approximating subspace is very advantageous: working with piecewise polynomials is easy because evaluations, differentiations and integrations are simple, the resulting stiffness matrix is sparse, and the linear system can be solved in an efficient way.

Since the Bernstein basis $B_{i,p}$ introduced in Section 1.5 constitutes a basis for $\mathbb{P}_p$, we will use it to define the space of piecewise polynomials $P\mathbb{P}_p^h(\Omega)$ defined in Eq. (3.1) and to simulate the standard FEM approximation method. In fact, in standard FEM, $P\mathbb{P}_p^h(\Omega)$ is often chosen as approximating space V_h.

In the IGA context, the approximating space can be the space of NURBS functions $\mathcal{R}(\mathbb{P}_{m-i}, M, \Xi, W)$ or the space of spline functions $\mathcal{S}(\mathbb{P}_{m-1}, M, \Xi)$, defined in Section 1.6 and 1.2 respectively.

Since NURBS functions for the solution can be defined on the partition of the domain given by the knot vectors of the geometrical description, IGA does not need to discretize the physical domain. This is one great advantage of IGA over FEM.

3.2 NURBS basis for analysis

As mentioned, piecewise polynomials are widely used in classical FEA as basis functions for the approximating space V_h. This is because they are easy to understand, program and use in the theoretical setting.

But they are not the only possible basis. A basis that is C^1 on the element interiors, C^0 on the element boundaries and complete, satisfies sufficient conditions for convergence for a wide class of problems [33]. Since NURBS basis functions meet these requirements, they ensure that isogeometric analysis will result in convergent methods.

Let's consider the Poisson BVP in (2.8) and let F be the NURBS parameterization from the parameter space $\hat{\Omega}$ to the physical space Ω:

$$\begin{aligned} F : \hat{\Omega} &\longrightarrow \Omega \\ \hat{\xi} &\longmapsto \xi := F(\hat{\xi}) = \sum_i R_i(\hat{x}) P_i, \end{aligned} \tag{3.2}$$

where $R_i(\hat{x})$ represents a suitable NURBS basis defined in Section 1.6.

A function defined over the parametric domain $\hat{u}_h : \hat{\Omega} \longrightarrow \mathbb{R}$ can be defined in terms of the same NURBS bases as:

$$\hat{u}_h(\hat{\xi}) = \sum_i R_i(\hat{\xi}) d_i, \tag{3.3}$$

where the coefficients d_i are called control variables. As seen in Eq.(1.9), we can define the function over the physical domain $u_h : \Omega \longrightarrow \mathbb{R}$ as $u_h = \hat{u}_h \circ F^{-1}$, and avoid to distinguish between $\hat{u}_h$ and u_h.

An element in the physical space is defined as the image of a knot span under the NURBS mapping F. We denote the knot spans in the parameter space by $\hat{\Omega}_k$ and their image in the physical space as Ω_k, where $k = 1, \ldots, N_e$, with N_e being the total number of elements. An element in the physical domain, and the corresponding element in the parameter space are highlighted in Fig. 3.1.

To solve the problem in (2.8), numerical methods aim to find an approximation u_h of the solution u. Different numerical methods are different techniques for finding d_i such that u_h is the best approximation of the solution u.

3.3 The Galerkin method in IGA

The Galerkin method for IGA takes the same form as in Section 2.4, but using NURBS basis functions instead of piecewise polynomials for the solution space V_h.

Let's consider the Poisson BVP in (2.8) with homogeneous Dirichlet boundary conditions defined on a domain $\Omega \subset \mathbb{R}^2$. The weak form of

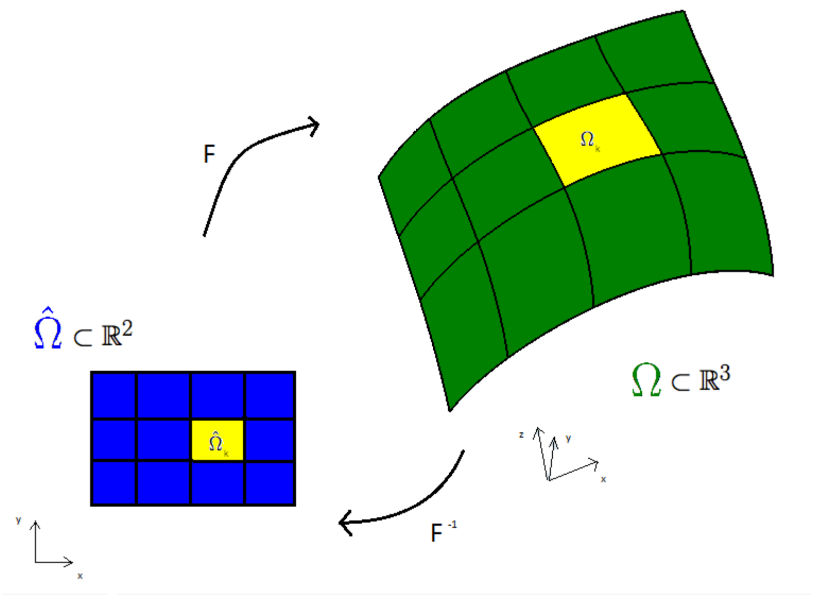

F
$\hat{\Omega} \subset \mathbb{R}^2$
$\hat{\Omega}_k$
Ω_k
$\Omega \subset \mathbb{R}^3$
F^{-1}
x
y
z

Figure 3.1: An element $\hat{\Omega}_k$ on the parameter space and its image Ω_k in the physical space.

the problem is given in Eq. (2.24), where the space of test function V is $H^1_{0,\Gamma_D}(\Omega) = \{v \in H^1(\Omega) \,:\, v = 0, \;\text{on}\,\Gamma_D\}$.

The Galerkin method consists of approximating the infinite-dimensional space V by a finite-dimensional space V_h. The weak form of the discrete problem is:
Find $u_h \in V_h$ such that:

$$\int_\Omega \kappa(\xi)\nabla u_h \cdot \nabla v_h \, d\xi = \int_\Omega f v_h \, d\xi + \int_{\Gamma_N} h v_h \, d\Gamma \quad \forall\, v_h \in V_h, \tag{3.4}$$

where the discrete space V_h is defined as:

$$V_h = \{v_h \in H^1_{0,\Gamma_D}(\Omega) \,:\, v_h = \hat{v}_h \circ F^{-1},\, \hat{v}_h \in \hat{V}_h\} \tag{3.5}$$

and $\hat{V}_h$ is the discrete space in the parametric domain, that has to be chosen.

Let N_h be the dimension of the spaces $\hat{V}_h$ and V_h, and let $\{\hat{v}_i\}_{i=1}^{N_h}$ be a basis for $\hat{V}_h$. Then we can define a basis for V_h as

$$\{v_i = \hat{v}_i \circ F^{-1}\}_{i=1}^{N_h}. \tag{3.6}$$

Thus, the trial functions of V_h can be expressed as linear combinations of the elements of the basis:

$$u_h = \sum_{j=1}^{N_h} \alpha_j v_j = \sum_{j=1}^{N_h} \alpha_j (\hat{v}_j \circ F^{-1}), \tag{3.7}$$

and their gradients are:

$$\nabla u_h = \sum_{j=1}^{N_h} \alpha_j \nabla v_j = \sum_{j=1}^{N_h} \alpha_j \hat{J}^{-T} (\nabla \hat{v}_j \circ F^{-1}), \tag{3.8}$$

where $\hat{J}$ is the $d \times k$ Jacobian matrix of the parameterization F defined in Eq. (1.2), and $(\hat{J})^{-T}$ denotes its inverse transposed.

The expressions (3.7) and (3.8) can be used to rewrite Eq. (3.4).

As we explained in Section 2.4, it is sufficient that Eq. (3.4) is verified for any test function v_i of the basis (3.6) for V_h, which yields the solution of the linear system of equations:

$$\int_\Omega \kappa(\xi) \sum_{j=1}^{N_h} \alpha_j \nabla v_j \cdot \nabla v_i \, d\xi = \int_\Omega f v_i \, d\xi + \int_{\Gamma_N} h v_i \, d\Gamma \quad \text{for } i = 1, \dots, N_h, \tag{3.9}$$

where f and h are the source and boundary terms, that contribute to the right-hand side of the linear system (2.36).

Let $\hat{\mathcal{K}}_h := \{\hat{\Omega}_k\}_{k=1}^{N_e}$ be the partition of the parameter domain $\hat{\Omega}$, defined by the knot spans. This partition induces a partition of the physical domain Ω given by $\mathcal{K}_h = \{\Omega_k\}_{k=1}^{N_e} = \{F(\hat{\Omega}_k)\}_{k=1}^{N_e}$.

The integrals on Ω are mathematically equivalent to the sum of the integrals on the N_e elements of the partition of $\mathcal{K}_h$ defined by the knot spans. By the transformation rules, the integrals on Ω in Eq. (3.4) can be expressed in the parametric space as :

$$\int_\Omega \kappa(\xi) \nabla u_h \cdot \nabla v_h \, d\xi = \sum_{k=1}^{N_e} \int_{\hat{\Omega}_k} \kappa(\hat{\xi}) \left(\hat{J}^{-T}(\hat{\xi}) (\nabla \hat{u}_h \circ F^{-1}) \right) \left(\hat{J}^{-T}(\hat{\xi}) (\nabla \hat{v}_h \circ F^{-1}) \right) |\det \hat{J}(\hat{\xi})| \, d\hat{\xi} \tag{3.10}$$

and

$$\int_{\Omega} f v_h \, d\xi = \sum_{k=1}^{N_e} \int_{\hat{\Omega}_k} f(\hat{\xi})(\hat{v}_h(\hat{\xi}) \circ F^{-1}(\hat{\xi})) |\det \hat{J}(\hat{\xi})| \, d\hat{\xi}, \tag{3.11}$$

since $d\xi = |\det \hat{J}(\hat{\xi})| \, d\hat{\xi}$ because we assumed $\Omega \subset \mathbb{R}^2$.

To compute the coefficients of the stiffness matrix and of the source and boundary terms, the integrals of Eq. (3.9) must be numerically approximated by quadrature rules, such as for example the Gaussian quadrature rules. For simplicity, we will use the partition $\mathcal{K}_h$ to define them: a quadrature rule is defined on every element $\hat{\Omega}_k$, and it is determined by a set of n_k nodes and their corresponding weights, denoted by $\{\hat{\xi}_{l,k}\} \subset \hat{\Omega}_k$ and $\{w_{l,k}\} \subset \mathbb{R}$, for $l = 1, \dots, n_k$ respectively.

Using the parameterization F, the integral of a function $\phi \in L^1(K_k)$ can be approximated as follows:

$$\int_{\Omega_k} \phi d\xi = \int_{\hat{\Omega}_k} \phi(F(\hat{\xi})) |det(\hat{J}(\hat{\xi}))| d\hat{\xi} \simeq \sum_{l=1}^{n_k} w_{l,k} \phi(\xi_{l,k}) |det(\hat{J}(\hat{\xi}_{l,k}))|, \tag{3.12}$$

where $\xi_{l,k} := F(\hat{\xi}_{l,k})$.

By applying Eq.s (3.10), (3.11), (3.12) and the quadrature rules, the integrals on Ω of Eq. (3.9) can be numerically approximated and the coefficients of the stiffness matrix and of the source term can be computed as:

$$\begin{aligned} A_{i,j} &\simeq \sum_{k=1}^{N_e} \sum_{l=1}^{n_k} \kappa(\xi_{l,k}) w_{l,k} \nabla v_j(\xi_{l,k}) \cdot \nabla v_i(\xi_{l,k}) |det(\hat{J}(\hat{\xi}_{l,k}))| \\ &\simeq \sum_{k=1}^{N_e} \sum_{l=1}^{n_k} \kappa(\hat{\xi}_{l,k}) w_{l,k} \left(\hat{J}^{-T}(\hat{\xi}_{l,k}) \nabla \hat{v}_j(\hat{\xi}_{l,k}) \right) \left(\hat{J}^{-T}(\hat{\xi}_{l,k}) \nabla \hat{v}_i(\hat{\xi}_{l,k}) \right) |det(\hat{J}(\hat{\xi}_{l,k}))| \end{aligned} \tag{3.13}$$

and

$$\begin{aligned} f_i &\simeq \sum_{k=1}^{N_e} \sum_{l=1}^{n_k} f(\xi_{l,k}) w_{l,k} v_i(\xi_{l,k}) |det(\hat{J}(\hat{\xi}_{l,k}))| \\ &\simeq \sum_{k=1}^{N_e} \sum_{l=1}^{n_k} f(\hat{\xi}_{l,k}) w_{l,k} \hat{v}_i(\hat{\xi}_{l,k}) |det(\hat{J}(\hat{\xi}_{l,k}))|, \end{aligned} \tag{3.14}$$

for $i, j = 1, \dots, N_h$. To compute the boundary term h_i in Eq. (3.9), we need a quadrature rule for boundary integrals, which is inherited from the one defined on the entire domain that we just described.

Let Ω be a surface, assuming that each side of the parametric domain is completely mapped into Γ_D or Γ_N, we can define a mapping $F_b : [0,1] \longrightarrow \Gamma_N$ as the restriction of F to the boundary. Denoting the quadrature nodes in the reference interval by their parametric coordinates $t_{l,k}$, and using for the rest the same notation as before with a superscript or subscript b, we get the approximation of a boundary line integral:

$$\int_{\Omega_k^b} \phi d\Gamma = \int_{\hat{\Omega}_k} \phi(F_b(t))|F_b'(t))|dt \simeq \sum_{l=1}^{n_k^b} w_{l,k}^b \phi(\xi_{l,k}^b)|F_b'(t_{l,k}))|. \qquad (3.15)$$

Thus, the boundary term is given by:

$$h_i \simeq \sum_{k=1}^{N_e} \sum_{l=1}^{n_k} h(\xi_{l,k}^b) w_{l,k}^b v_i(\xi_{l,k}^b)|F_b'(t_{l,k}))|. \qquad (3.16)$$

Let $\Omega \subset \mathbb{R}^3$ be a two-dimensional manifold embedded in $\mathbb{R}^3$. The same computations for the weak form of the Laplace-Beltrami problem in Eq. (2.50) can be carried out analogously, taking into account Eq. (2.45) and the fact that, when $k < d$ the differential $d\xi$ is transformed to the parametric space as $\hat{g}(\hat{\xi})\, d\hat{\xi}$ instead of as $|\det \hat{J}(\hat{\xi})|\, d\hat{\xi}$.

3.4 IGA on multipatch geometries

Typically, geometric models cannot be represented by only one surface with rectangular topology. CAD systems use joined NURBS surfaces, called NURBS patching, to create more complex models, that can be represented by the B-reps introduced in Section 1.8. In this context, a single surface is called patch, and the geometry is said to be multipatch. One curve that is shared by two adjacent surfaces is called interface. Fig. 3.2 shows an example of a multipatch geometry representing a pipe created in Eyeshot.

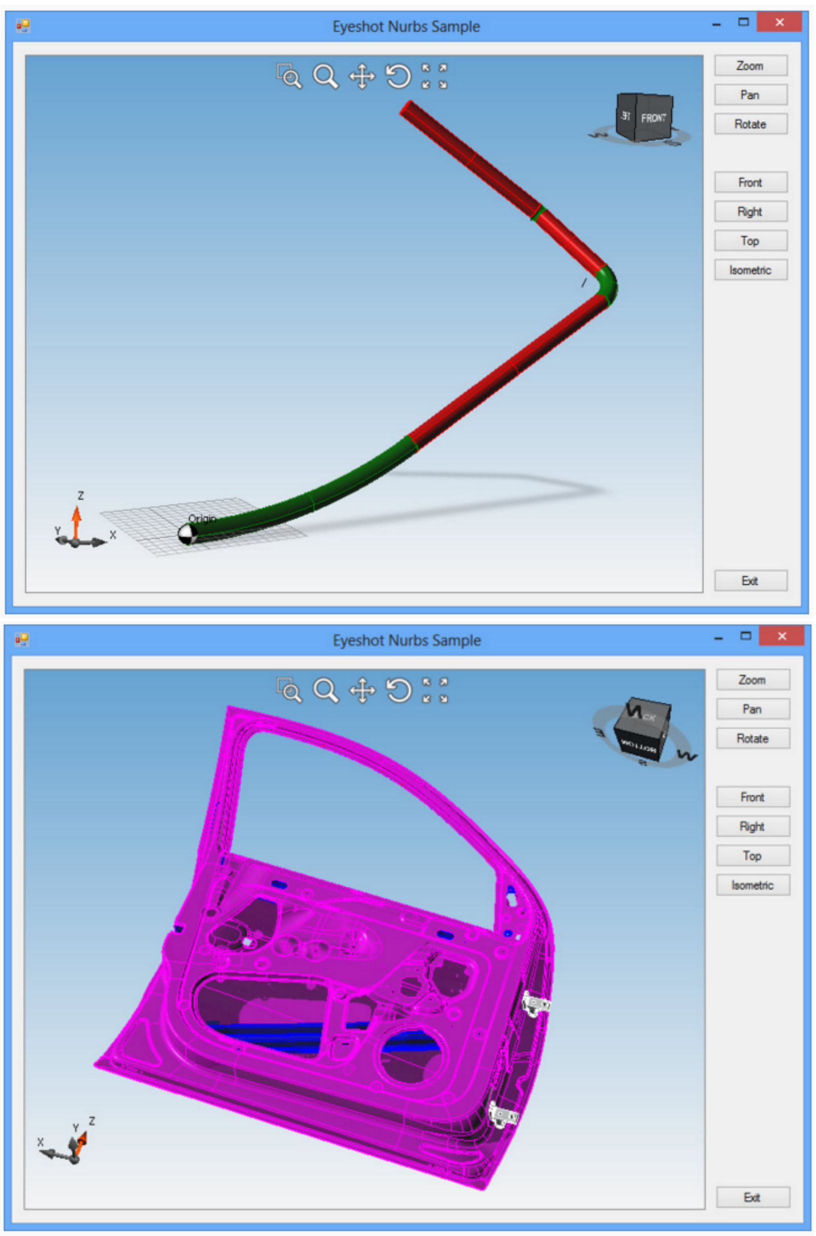


Figure 3.2: Multipatch geometries created in Eyeshot. Top: the model represents a pipe in which the green patches are toroidal surfaces and the red patches are cylindrical surfaces. Bottom: a more complex model representing a car door.

It is possible to apply IGA also on this kind of models [20, 23, 37]. Let Ω be a multipatch domain formed by n_p patches $P_1, \dots, P_{n_p}$, we have that $\overline{\Omega} = \cup_{l=1}^{n_p} \overline{P_l}$, with $P_l \cap P_m = \emptyset$ for each $l \neq m$. The patches are defined by n_p parameterizations $F_l : \hat{\Omega} \longrightarrow P_l$ like the one defined in Eq. (1.1), that have a common parameter domain $\hat{\Omega}$.

To conduct IGA analyses on multipatch geometries, it is required that the patches of the model are compatible. This means that, if two patches P_l and P_m have a common interface $I_{l,m} = \overline{\Omega}_l \cap \overline{\Omega}_m \neq \emptyset$, the degree, knot vectors and control points of two adjacent surfaces must coincide on their interface $I_{l,m}$. Fig. 3.3 shows a planar multipatch domain in which $n_p = 3$.

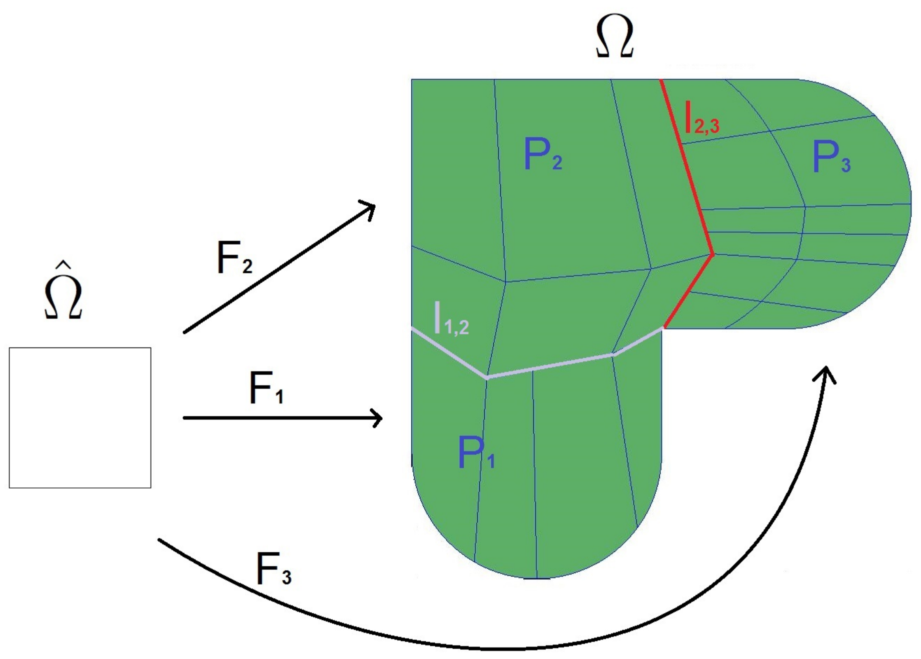


Figure 3.3: Planar multipatch geometry composed of three patches and two interfaces.

Therefore, the parametric domains of two adjacent patches are equal at least along one of the two directions, and refinements of one patch can propagate from that patch to the next. Also the basis functions are the same for both patches. Hence, creating a connectivity array which identifies the matching basis functions on each patch in one single function in the global domain, we can build a global stiffness matrix and solve a unique linear system that gives the results on all the patches.

Along each interface, the two solutions from two adjacent patches are equal, and they are glued together with C^0 continuity. Higher continuity has also been implemented by applying constraint equations in the normal direction [16].

3.4.1 IGA on multipatch geometries with GeoPDEs

GeoPDEs, the tool that we used for the applications, includes a package to perform IGA analyses on multipatch geometries [20]. It can read multipatch geometries from .txt files that contain information on patches, interfaces, subdomains and boundaries. Since it is common to have adjacent patches that have one edge in common but are not compatible, we extended GeoPDEs with the function `nrbmakecompatible` that creates two compatible surfaces starting from non-compatible ones. Here is a list of the tools we implemented for handling multipatch geometries as physical domains:

- `mp_geo_substitute_patches`: reads a .txt file with a geometry and creates a new file allowing the user to substitute one or more patches of the pre-existing geometry with new ones. Since the method doesn't update the interfaces and boundaries, it can be used only if the interfaces and boundaries don't change or are changed manually.

- `nrbmakepolyline`: creates a constant length polyline passing through the given vertices, represented by a NURBS curve of degree 1. We used this kind of curves to create NURBS surfaces with at least one direction of degree 1.

- `nrbmakecompatible`: starting from two nurbs surfaces (both made of a single patch) that have one edge in common, creates two new surfaces that are compatible along that edge. In other words, the output surfaces will have same degree, knot vector and control points along the shared interface.
  The signature of the method is:

  ```
  [comp1, comp2] = nrbmakecompatible (srf1, srf2, side1, side2, orientation)
  ```

  The input parameters are:

  – the two non-compatible surfaces `srf1` of degree $(pU1, pV1)$ and `srf2` of degree $(pU2, pV2)$,

  – the indices `side1` and `side2` of the edges of each surface corresponding to the shared interface,

  – and the `orientation` of the interface (1 if the two edges have the same direction, 0 otherwise).

  The steps of the algorithm are:

  – based on `side1` and `side2`, the routine identifies in each surface which parametric direction (u or v) will be modified. Let $w1$ and $w2$ be the identified directions for `srf1` and `srf2` respectively.

  – If $w1$ and $w2$ don't have the same degree, a degree elevation along the direction w is applied to the surface with smaller degree, so that both surfaces have the same degree along w, equal to $max(pW1, pW2)$.

  – Create a common knot vector merging the two knot vectors. If `orientation` is 0, one of the two knot vectors is reversed twice in this phase.

  – The missing knots of each knot vector are then inserted by `nrbkntins` along the directions $w1$ and $w2$, creating the two compatible surfaces `comp1` and `comp2`, that are returned as output parameters.

If the edges corresponding to `side1` and `side2` are not overlapping, the method works anyway, creating two surfaces that have one compatible direction.

We extended the application of IGA on multipatch domains applying IGA also on hybrid domains, made of both mesh and NURBS patches. This was possible because we used NURBS surfaces of degree (1,1) to represent the mesh patches. Our examples of IGA to multipatch geometries are collected in Chapter 5.

Chapter 4

Numerical examples: IGA on a single patch

In this chapter we present the results of the application of Isogeometric Analysis to solve BVPs on different physical domains Ω represented by a single patch in $\mathbb{R}^2$ or $\mathbb{R}^3$.

In order to solve these problems we used the tool GeoPDEs [20]. It allows to treat the geometry and the approximating subspace independently, and thus to use non-isoparametric approaches, in which the physical space and the solution space do not coincide.

We used three different strategies to represent the physical domain Ω:

- The first one is through a NURBS surface.
- The second is an analytic definition of the domain.
- The third is a polygonal quad mesh defined by a NURBS surface of degree (1,1). We used this approach in order to adopt GeoPDEs as a solver in the classical FEM setting.

In the following, we will refer to these three different representations of the domain respectively as *nurbs*, *analytic* and *mesh*.

For the solution space we used NURBS, spline functions or piecewise polynomial functions represented in Bézier form. And we will refer to these choices of solution space as *nurbs*, *spline* and *bezier*, respectively.

A solution in which we used *mesh* for the physical domain, and B-spline as basis functions of the solution's space will be denoted by $(mesh \to spline)$. Sometimes we will also add a number after "spline" and "nurbs" that indicates the degree of the B-spline or NURBS basis functions. For instance, the case in which we used a NURBS domain of degree $(3, 3)$ for the physical domain, and B-spline of degree 5 as basis functions of the solution's space, will be denoted by $(nurbs3 \to spline5)$.

We compared different choices of physical domain and solution space under h-refinement and p-refinement, and we measured the accuracy of the approximate solution in terms of the number of degrees of freedom $ndof$, which is the dimension of the solution space V_h.

Let's define the function err on Ω as the difference between the exact and computed solutions:

$$err(x, y) = u_{ex}(x, y) - u(x, y) \qquad \forall (x, y) \in \Omega. \tag{4.1}$$

The L^2 error is defined as:

$$\|err\|_{L^2(\Omega)} = \left(\int_\Omega |err|^2 \right)^{\frac{1}{2}}, \tag{4.2}$$

and it will be denoted by err_{L^2}, and the H^1 error is defined as:

$$\|err\|_{H^1(\Omega)} = \left(\|err\|^2_{L^2(\Omega)} + \|err'\|^2_{L^2(\Omega)} \right)^{\frac{1}{2}}, \tag{4.3}$$

and it will be denoted by err_{H^1}.

In the discrete settings, where the parametric domain $\hat{\Omega}$ is discretized by a set of grid points $(x_i, y_j) \in \hat{\Omega}$ with $i, j = 1, \cdots, N$, we consider the matrix

$$err(x_i, y_j) = u_{ex}(F(x_i, y_j)) - u(F(x_i, y_j)) \qquad \forall (x_i, y_j) \in \hat{\Omega}, \tag{4.4}$$

where F the geometrical mapping defined in Eq.(1.1). We define the error err_∞ as

$$err_\infty := \|err\|_\infty = \max_i(err(i)), \tag{4.5}$$

where err the vector form of the matrix defined in Eq.(4.4), and err_2 as

$$err_2 := \|err\|_2 = \left(\frac{1}{N^2}\sum_{i=1}^{N^2} err(i)^2\right)^{1/2} \tag{4.6}$$

which is a weighted Euclidean vector norm that makes the error independent of the size of the evaluation grid.

With the following examples we investigate some aspects of isogeometric analysis techniques and compare them with classic FEA techniques. Using GeoPDEs, we simulate the classical FEA by using *mesh* as physical domain Ω (NURBS with degree (1,1)) and Bézier piecewise polynomials as solution space.

The examples are organized as follows:

- Examples 1, 2 and 3: we applied h-refinement, p-refinement on 2D physical domains, $\Omega \subset \mathbb{R}^2$.
- Example 4: we examined the influence of parameterizations, applying h-refinement on two different parameterizations of the same 2D physical domain $\Omega \subset \mathbb{R}^2$, and we compared the results.
- Example 5: solution of the Laplace-Beltrami problem on a surface embedded in $\mathbb{R}^3$.

4.1 Example 1

The first problem that will be analyzed is the BVP:

$$\begin{aligned} \Delta u &= f \quad \text{in } \Omega \\ u &= g \quad \text{on } \partial\Omega, \end{aligned} \tag{4.7}$$

where $f(x,y) = 2\pi^2 \sin(\pi x)\sin(\pi y)$ and $\Omega \subset \mathbb{R}^2$.

The exact solution of (4.7) is given by

$$u_{ex}(x,y) = \sin(\pi x)\sin(\pi y), \tag{4.8}$$

which also defines the non-homogeneous Dirichlet boundary conditions g.

In this example the domain Ω is a quarter of a ring with internal radius 1 and external radius 2.

The *analytic* physical domain is given by the analytic function:

$$F(u,v) = ((u+1)\cos(\pi v/2), (u+1)\sin(\pi v/2)) \quad \text{with} \quad 0 < u, v < 1.$$

We obtained the *nurbs* domain revolving the line from $(0,1)$ to $(0,2)$ around the Z-axis and then performing a degree elevation in the v direction of the surface in order to get a NURBS surface $S(u,v)$ of degree 2 in both directions. Using the NURBS toolbox [54] the procedure is the following:

```
% BUILD NURBS PHYSICAL DOMAIN
line = nrbline ([1 0], [2 0]);
S = nrbrevolve (line, [0 0 0], [0 0 1], pi/2);
S = nrbdegelev (S, [0 1]);
```

The code generates a surface of degree $(2,2)$, with open knot vectors that have no internal knots, and a 3×3 control point grid:

```
pU = 2
pV = 2
knotVectorU = [0 0 0 1 1 1]
knotVectorV = [0 0 0 1 1 1]
```

```
controlPoints = [ (1, 0, 1), (1/sqrt(2), 1/sqrt(2), 1/sqrt(2)),
(0, 1, 1), (1.5, 0, 1), (3/(2*sqrt(2)), 3/(2*sqrt(2)), 1/sqrt(2)),
(0,1.5, 1), (2, 0, 1), (2/sqrt(2), 2/sqrt(2), 1/sqrt(2)), (0, 2, 1)]
```

Since the physical domain is on the XY plane, the third component of the above control points is the weight.

For this example, we created the geometry using procedural methods However, GeoPDEs also allows the user to import a geometry from a .txt file. The following frame shows the file .txt that, read with the command `geo_load`, produces the same surface S (the lines that start with # are comments).

```
# dimension of the geometry and number of patches
   2   1
PATCH 1
# degree along U and V directions
   2   2
# dimensions of the matrix of control points
   3   3
# knot vector along U
0   0   0   1   1   1
# knot vector along V
0   0   0   1   1   1
# X coordinates of all the control points
1   0.707106781186548    0     1.5   1.060660171779821    0      2   1.414213562373095   0
# Y coordinates of all the control points
0   0.707106781186547    1     0      1.060660171779821   1.5    0   1.414213562373095   2
# Weights of all the control points
1   0.707106781186548    1     1     0.707106781186548    1      1   0.707106781186548   1
```

4.1.1 *h*-refinement

We compare five different test cases: $(mesh \to spline1)$, $(mesh \to spline2)$, $(nurbs2 \to spline2)$, $(nurbs2 \to nurbs2)$ and $(analytic \to spline2)$ under h-refinement.

At every step of h-refinement, a new knot is inserted by the knot insertion procedure in the middle of each knot span, thus splitting the elements in four (two for each direction) equal parts in the parametric domain. For this geometry, this is equivalent to splitting each side of an element in half in the physical space.

We performed five iterations of h-refinement. Before starting the refinement process, we applied a few knot insertions to the surface described above, in order to obtain a physical space that is subdivided into 4×8 elements, with less elements on the radial direction. Thus the knot vectors at the first iteration are:

```
knotVectorU = [0  0  0  0.125  0.25  0.375  0.5  0.625  0.75  0.875  1  1  1]
knotVectorV = [0  0  0  0.25  0.5  0.75  1  1  1]
```

Fig. 4.1 shows the subdivision of the *nurbs* and *mesh* physical spaces at the first, third and fifth iterations of h-refinement.

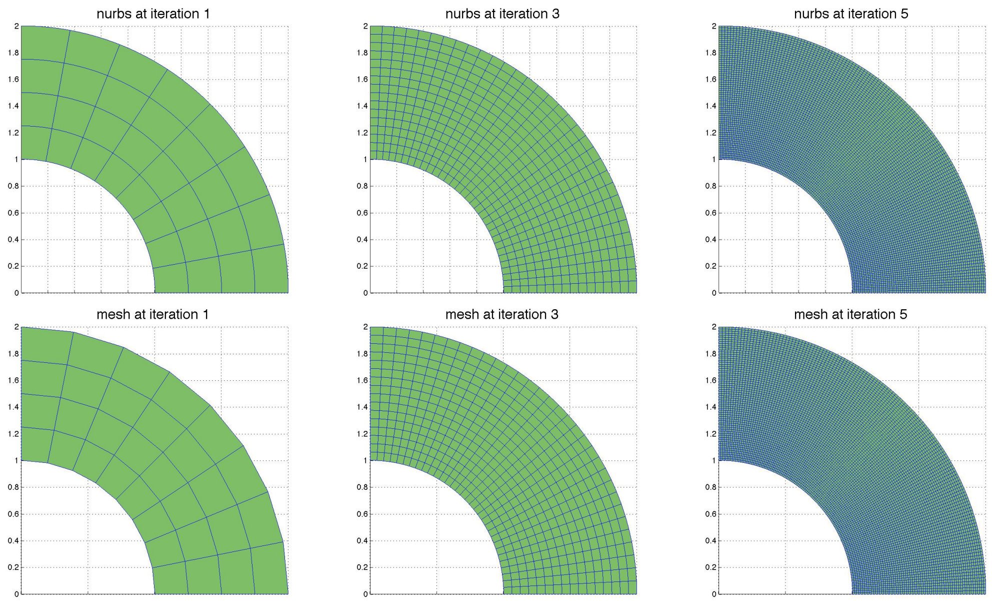


Figure 4.1: h-refinement of the physical domain: *nurbs* in the upper row, *mesh* in the lower row.

As can be noted in Fig. 4.1, in case of *mesh* physical domain, when we increase the number of knots, we add the new nodes (which in this case coincide with the control points of the mesh) along the curved sides of the domain instead of in the midpoint of the segment between two existing knots. This is not what knot insertion on a NURBS of degree (1,1) would do, but we want to simulate a FEM solution, and this approach reflects better mesh refinement in that context.

Fig.4.2 shows the exact solution and the computed solution at the first step of h-refinement for the $(nurbs2 \to nurbs2)$ case.

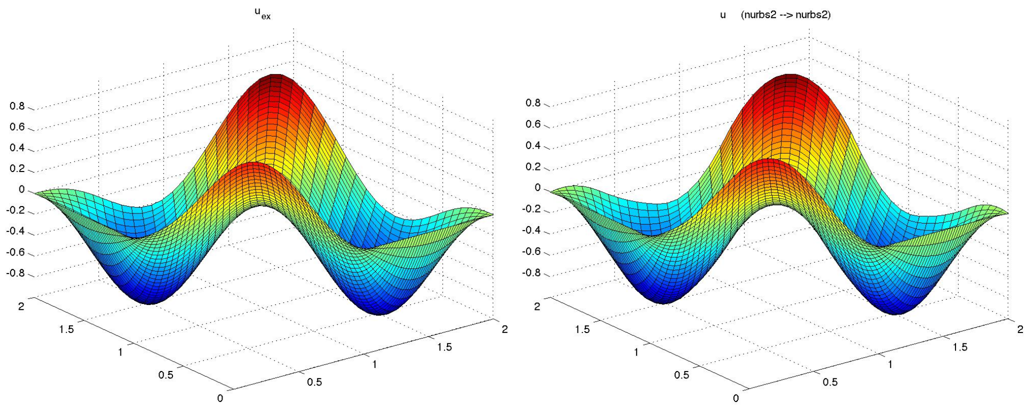


Figure 4.2: Exact solution (left) and computed solution at the first step of h-refinement for the $(nurbs2 \to nurbs2)$ case (right).

In Fig. 4.3 we show the error function err defined in Eq. (4.1) at the last iteration of the h-refinements. The maximum errors, computed on a 50×50 points grid, are $err_\infty = 4.304 \times 10^{-5}$ for $(mesh \to spline2)$ and $err_\infty = 5.609 \times 10^{-6}$ for $(nurbs2 \to nurbs2)$.

Throughout the h-refinement process, the number of degrees of freedom when the solution has degree 2 passes from 32 at the first iteration to 8192 at the last one. The value of $ndof$ represents the dimension of the stiffness matrix of the linear system required to solve the BVP, and it is given by the dimension of the solution space minus the number of degrees of freedom on the boundaries, since we applied Dirichlet boundary conditions. Since the dimension of the polynomial spline and NURBS spaces are given by $m + K$,

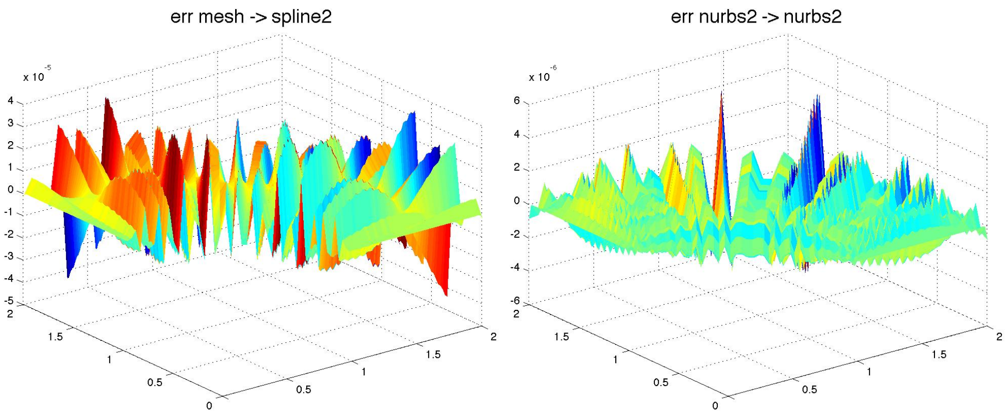


Figure 4.3: Plot of err, defined in Eq. (4.1), for the ($mesh \to spline2$) (left) and ($nurbs2 \to nurbs2$) (right) cases.

where m is the order of the splines and K is the sum of the multiplicities of the internal knots, the $ndof$ for the first iteration of the ($nurbs2 \to spline2$) case, is given by: $ndof = (3+3) \times (7+3) - 28 = 32$.

The log-log plot of the convergence curves for the five performed tests is depicted in Fig. 4.4. The L^2 error is shown as function of the number of degrees of freedom $ndof$. The ($analytic \to spline2$), ($nurbs2 \to spline2$) and the ($nurbs2 \to nurbs2$) cases give similar results.

Table 4.1 reports the numerical results of h-refinement.

From the corresponding rows of Tables 4.1 (a) and (b), we notice that the basis functions of the solution spaces have the same internal knots, but different degree, and the meshes representing the physical domain are the same. The number of degrees of freedom $ndof$ increases with the increase of the degree of the solution, but the obtained result is more accurate.

The results in Tables 4.1 (c), (d) and (e) show that using NURBS or analytic geometries, that allow to represent the physical domain Ω exactly, improves the solution substantially with respect to the use of mesh for the representation of the geometric domain. Both the errors err_{L^2} and err_∞ for those cases are smaller.

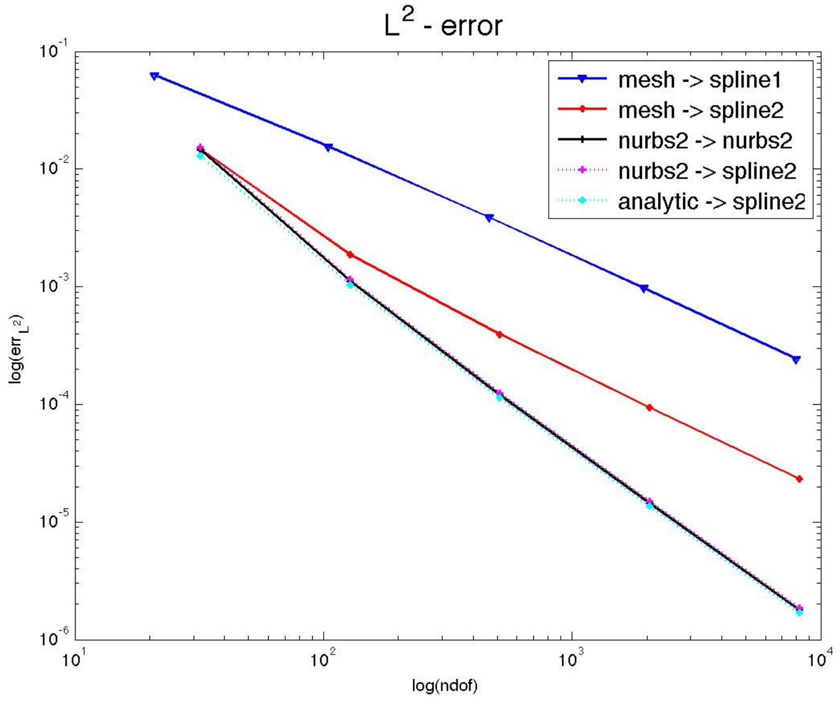


Figure 4.4: L^2 error through h-refinement.

($mesh \rightarrow spline1$) h-refinement		
$ndof$	err_{L^2}	err_∞
21	6.2124e-02	1.4178e-01
105	1.5450e-02	3.6480e-02
465	3.8625e-03	9.3578e-03
1953	9.6572e-04	2.2068e-03
8001	2.4144e-04	5.6303e-04

(a)

($mesh \rightarrow spline2$) h-refinement		
$ndof$	err_{L^2}	err_∞
32	1.4727e-02	3.5963e-02
128	1.8721e-03	4.0572e-03
512	3.9198e-04	7.4176e-04
2048	9.3903e-05	1.9048e-04
8192	2.3231e-05	4.3039e-05

(b)

($nurbs2 \rightarrow spline2$) h-refinement		
$ndof$	err_{L^2}	err_∞
32	1.5425e-02	4.1380e-02
128	1.1618e-03	3.4999e-03
512	1.2471e-04	3.4906e-04
2048	1.4965e-05	4.7711e-05
8192	1.8513e-06	5.7747e-06

(c)

($nurbs2 \rightarrow nurbs2$) h-refinement		
$ndof$	err_{L^2}	err_∞
32	1.4553e-02	3.9571e-02
128	1.1136e-03	3.3951e-03
512	1.2018e-04	3.4436e-04
2048	1.4443e-05	4.6713e-05
8192	1.7874e-06	5.6091e-06

(d)

($analytic \rightarrow spline2$) h-refinement		
$ndof$	err_{L^2}	err_∞
32	1.2939e-02	3.2295e-02
128	1.0349e-03	2.9520e-03
512	1.1332e-04	3.2569e-04
2048	1.3675e-05	4.2752e-05
8192	1.6940e-06	5.0108e-06

(e)

Table 4.1: Tables of results for h-refinement.

4.1.2 p-refinement

In this example we make a comparison between $(mesh \rightarrow spline)$, $(nurbs \rightarrow spline)$, $(nurbs \rightarrow nurbs)$ and $(analytic \rightarrow spline)$ under p-refinement. The solutions are computed using spline or NURBS geometries of varying degrees as basis functions. We consider five iterations of p-refinement, from degree 2 to degree 7.

Degree elevation at each step is applied only to the solution space, and not to the physical domain. Therefore the degree of the physical domain is always (2,2) for *nurbs* and *analytic* domains. The physical domains *nurbs* and *mesh* are depicted in Fig. 4.5. We inserted 4 knots in each direction in the nurbs domain and 5 knots in the mesh domain, in order to have at the first iteration of p-refinement a number of degrees of freedom equal to 25 for both the considered representations.

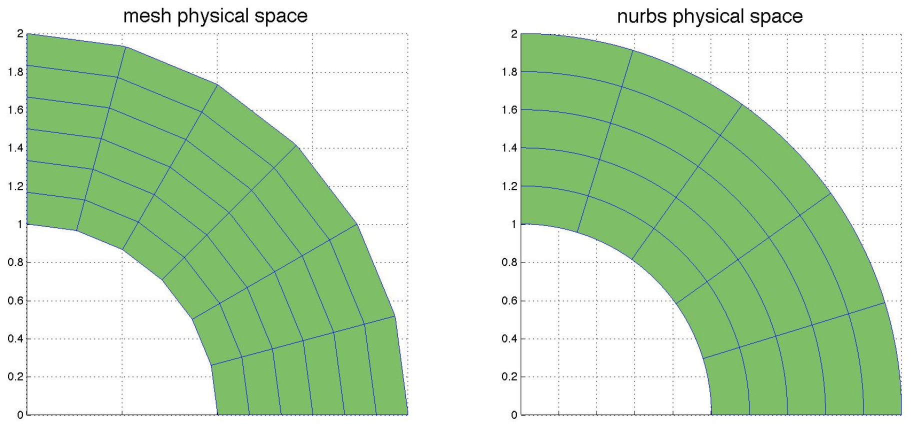


Figure 4.5: The physical domains used for p-refinement for the *mesh* (left) and *nurbs* (right) cases.

A graph of the error $err = u_{ex}(x, y) - u(x, y)$, at the last iteration of the p-refinement procedure, is shown in Fig. 4.6. The maximum errors are $err_\infty = 3.7157 \times 10^{-6}$ for $(mesh \rightarrow spline)$ and $err_\infty = 5.480 \times 10^{-6}$ for $(nurbs \rightarrow nurbs)$.

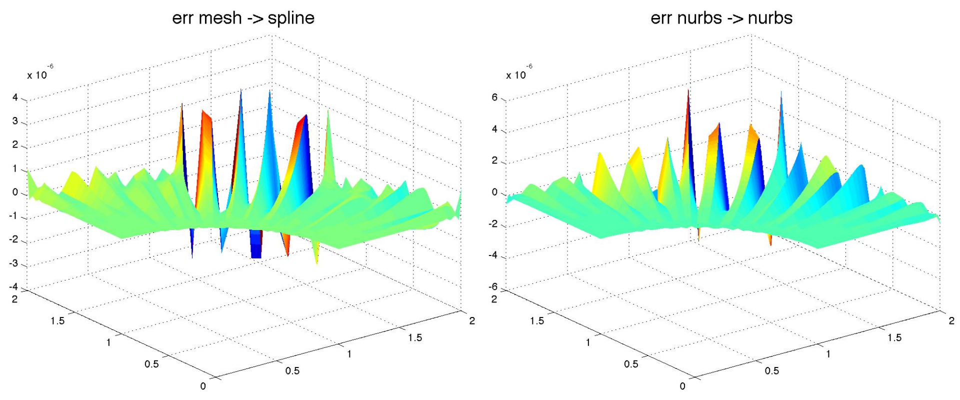


Figure 4.6: Plot of err at the last step of p-refinement for the $(mesh \rightarrow spline)$ (left) and $(nurbs \rightarrow nurbs)$ (right) cases.

The loglog plot of the L^2 error is depicted in Fig. 4.7. Comparing the results in Fig. 4.4 with those in Fig. 4.7, we can conclude that for this problem using p-refinement on the *nurbs* or *analytic* domain, we can reach the same accuracy of h-refinement with less degrees of freedom, which means solving a smaller system of equations.
With the $(analytic \rightarrow spline)$ case we obtain results even more precise than those of $(nurbs \rightarrow nurbs)$ and $(nurbs \rightarrow spline)$.

Table 4.2 contains the numerical results on p-refinement in which degree elevation is applied only to the solution space. The test cases in which the domain describes the geometry exactly (*nurbs* and *analytic*) give the best results using a smaller number of degrees of freedom.

We also tried to apply a p-refinement process in which we elevate at each step both the physical domain and the solution space. For all the analyzed cases, the results are identical to the ones obtained elevating only the degree of the solution space. This is of course because degree elevation does not modify the geometry.

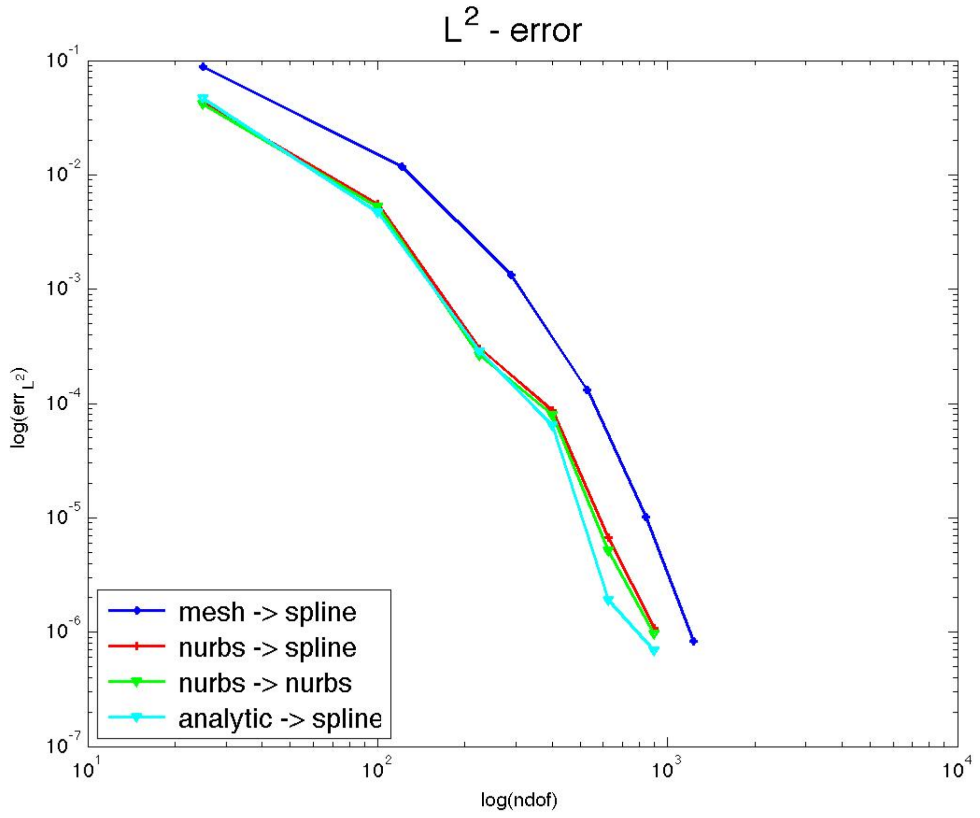


Figure 4.7: L^2 error through p-refinement. The ($analytic \rightarrow spline$) case for higher degrees gives the most accurate results.

$(mesh \to spline)$ p-refinement		
$ndof$	err_{L^2}	err_∞
25	8.8384e-02	1.8381e-01
121	1.1751e-02	3.1091e-02
289	1.3200e-03	3.5102e-03
529	1.3037e-04	4.9178e-04
841	1.0184e-05	3.3377e-05
1225	8.3506e-07	3.7157e-06

(a)

$(nurbs \to spline)$ p-refinement		
$ndof$	err_{L^2}	err_∞
25	4.2863e-02	6.5495e-02
100	5.5509e-03	1.6371e-02
225	3.0268e-04	1.0025e-03
400	8.7199e-05	3.8942e-04
625	6.6786e-06	3.1572e-05
900	1.0824e-06	6.1180e-06

(b)

$(nurbs \to nurbs)$ p-refinement		
$ndof$	err_{L^2}	err_∞
25	4.1666e-02	6.4849e-02
100	5.2579e-03	1.5779e-02
225	2.6160e-04	7.5462e-04
400	7.9765e-05	3.6001e-04
625	5.1340e-06	2.5415e-05
900	9.6371e-07	5.4802e-06

(c)

$(analytic \to spline)$ p-refinement		
$ndof$	err_{L^2}	err_∞
25	4.6611e-02	7.0483e-02
100	4.7207e-03	1.3201e-02
225	2.8526e-04	4.6913e-04
400	6.4233e-05	2.5708e-04
625	1.8959e-06	1.0208e-05
900	6.8627e-07	3.3736e-06

(d)

Table 4.2: Tables of results for p-refinement. Degree elevation is applied only to the solution space, from degree 2 (first rows) to degree 7 (last rows).

4.2 Example 2

We now solve problem (4.7) on a different domain $\Omega \subset \mathbb{R}^2$: a ruled surface between a line and a semicircle (see Fig. 4.8).

For this geometry we don't have an analytic description, so we only used the *nurbs* and *mesh* representations for the physical space. In addition to *nurbs* and *spline*, for the solution space we use also *bezier*, which is obtained in GeoPDEs using splines in which all the internal knots have multiplicity equal to the degree.

To obtain the *nurbs* domain, we built a ruled surface between a semicircle and a line, and then performed a degree elevation in the v direction in order to get a NURBS surface $S(u, v)$ of degree 2 in both directions.

```
% BUILD NURBS PHYSICAL DOMAIN
arc = nrbcirc(1 , [0 0 0] , -pi/2 , pi/2);
line = nrbline([-1 -1] , [-1 1]);
S = nrbruled(arc , line);
S = nrbdegelev (S, [0 1]);
```

It generates a NURBS surface of degree $(2, 2)$, with a 5×3 control point grid:

```
pU = 2
pV = 2
knotVectorU = [0 0 0 0.5 0.5 1 1 1]
knotVectorV = [0 0 0 1 1 1]
controlPoints = [ (0, -1, 1), (sqrt(2)/2, -sqrt(2)/2, sqrt(2)/2),
(1, 0, 1), (sqrt(2)/2, sqrt(2)/2, sqrt(2)/2), (0, 1, 1), (-0.5, -1, 1),
(-0.146446609406726, -0.603553390593274, 0.853553390593274),
(0, 0, 1), (-0.146446609406726, 0.603553390593274, 0.853553390593274),
(-0.5, 1, 1), (-1, -1, 1), (-1, -0.5, 1), (-1, 0, 1), (-1, 0.5, 1),
(-1, 1, 1) ]
```

4.2.1 *h*-refinement

As in the previous example, we applied h-refinement inserting a new knot in the middle of each knot span and we compared the results performing five h-refinement iterations. For the $(mesh \rightarrow bezier2)$ case, at each iteration the new knots are inserted with multiplicity $m = p = 2$ in order to ensure a C^0 basis everywhere.

We inserted in the basic parameterization reported above three knots along the v direction and two knots in each knot span of the u direction.

Thus the knot vectors at the first iteration are:

```
knotVectorU = [0  0  0  0.1667  0.3333  0.5  0.5  0.6667  0.8333  1  1  1]
knotVectorV = [0  0  0  0.25  0.5  0.75  1  1  1]
```

A representation of the *nurbs* and *mesh* physical domains at iteration 1, 3 and 5 are illustrated in Fig. 4.8.

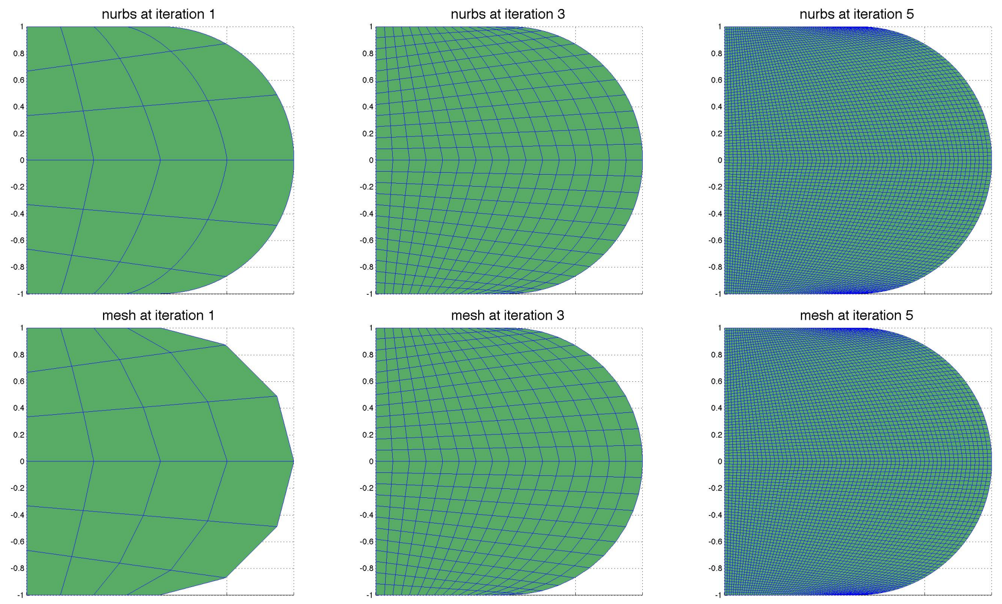


Figure 4.8: h-refinement of the ruled physical domain: *nurbs* in the upper row and *mesh* in the lower row.

Fig.4.9 shows the exact solution and the computed solutions at the first step of h-refinement for the $(mesh \to spline1)$ and $(mesh \to bezier2)$ cases. The solution of $(mesh \to spline1)$ on a coarse mesh is visibly worse than the one of $(mesh \to bezier2)$, in which the solution space is made of piecewise polynomials of degree 2.

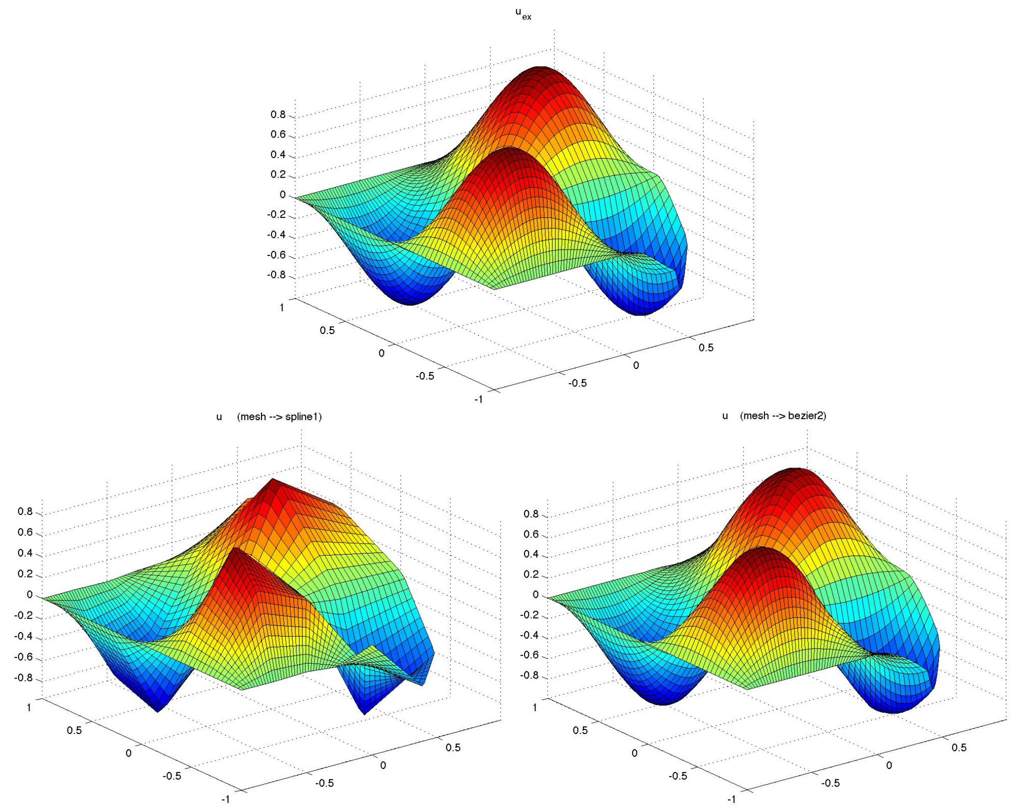


Figure 4.9: Exact solution (top) and computed solutions at the first step of h-refinement for $(mesh \to spline1)$ (bottom left) and $(mesh \to bezier2)$ (bottom right) cases.

The loglog plot of the L^2 error for h-refinement is depicted in Fig. 4.10. The $(nurbs2 \to spline2)$ and the $(nurbs2 \to nurbs2)$ cases give similar results. The $(mesh \to bezier2)$ case reaches comparable error values with a bigger number of degrees of freedom. This is also confirmed by the results reported in Table 4.3.

Throughout the h-refinement process, the number of degrees of freedom passes from around 20 to around 6000 in 5 iterations for the $(mesh \to spline1)$, $(mesh \to spline2)$, $(nurbs2 \to nurbs2)$ and $(nurbs2 \to spline2)$ cases. The same h-refinement steps on a classical FEM configuration, $(mesh \to bezier2)$, result in more degrees of freedom: from 77 to 24257 degrees of freedom.

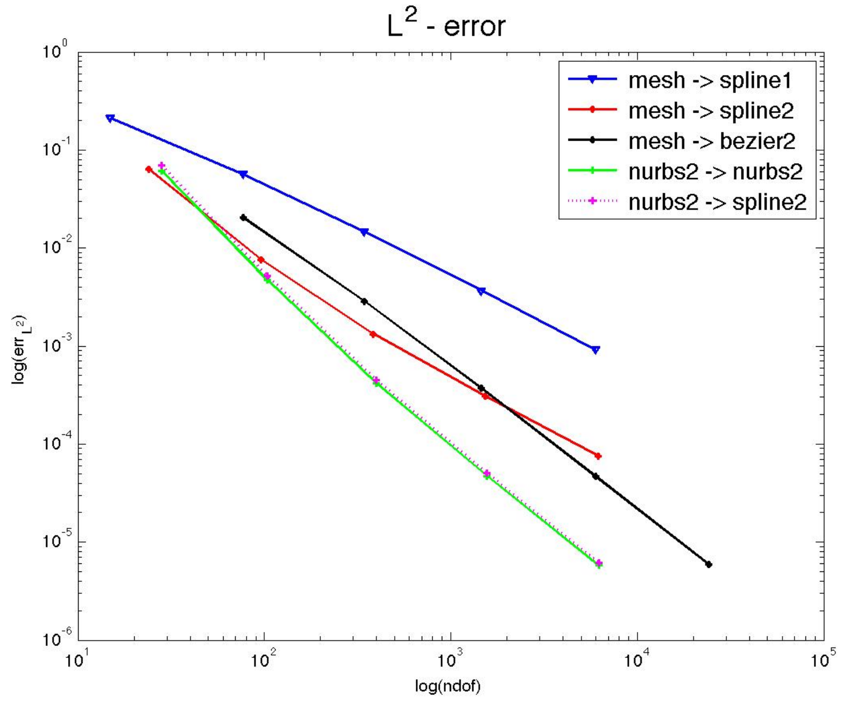


Figure 4.10: L^2 error under h-refinement.

Tables 4.3 (a), (b) and (c) refer to h-refinement on $mesh$ domains. They show that, in general, a C^1 solution on Ω (b) is better than a C^0 solution (c), when they have similar number of degrees of freedom and the same degree of the piecewise polynomials ($p = 2$). Using a solution space of degree $p = 1$ (a), as expected, gives the worst results. Also on this problem, the best results are obtained by the isogeometric approaches.

For the cases $(mesh \to bezier2)$ and $(nurbs2 \to nurbs2)$, that represent classical FEM and IGA respectively, we also analyzed the sparsity and the

$(mesh \to spline1)$ h-refinement		
$ndof$	err_{L^2}	err_∞
15	2.1023e-01	3.5515e-01
77	5.6677e-02	1.0685e-01
345	1.4550e-02	2.8722e-02
1457	3.6640e-03	7.1585e-03
5985	9.1770e-04	1.9116e-03

(a)

$(mesh \to spline2)$ h-refinement		
$ndof$	err_{L^2}	err_∞
24	6.4027e-02	1.4153e-01
96	7.6342e-03	1.6598e-02
384	1.3246e-03	2.7618e-03
1536	3.0747e-04	7.0997e-04
6144	7.5748e-05	1.2321e-04

(b)

$(mesh \to bezier2)$ h-refinement		
$ndof$	err_{L^2}	err_∞
77	2.0500e-02	4.5280e-02
345	2.8633e-03	6.0926e-03
1457	3.7111e-04	1.0100e-03
5985	4.6869e-05	1.0307e-04
24257	5.8743e-06	1.2939e-05

(c)

$(nurbs2 \to nurbs2)$ h-refinement		
$ndof$	err_{L^2}	err_∞
28	6.1601e-02	1.1949e-01
104	4.7049e-03	9.5583e-03
400	4.1640e-04	1.0734e-03
1568	4.7279e-05	1.0352e-04
6208	5.7634e-06	1.1541e-05

(d)

$(nurbs2 \to spline2)$ h-refinement		
$ndof$	err_{L^2}	err_∞
28	6.9129e-02	1.3173e-01
104	5.1732e-03	1.0095e-02
400	4.4804e-04	1.1332e-03
1568	5.0565e-05	1.1046e-04
6208	6.1543e-06	1.2222e-05

(e)

Table 4.3: Tables of results for h-refinement.

condition number $K(A) = \|A\|_2\|A^{-1}\|_2$, where A is the stiffness matrix given in Eq.(3.13). The graph in Fig. 4.11 shows how the condition number grows as the mesh is refined. Throughout the h-refinement process, the stiffness matrix tends to become ill-conditioned and this makes the linear system more difficult to solve, in a way that direct methods are less accurate and iterative methods are less efficient. For a fixed number of degrees of freedom, the linear system derived from IGA is worse conditioned with respect to the linear system derived from FEM.

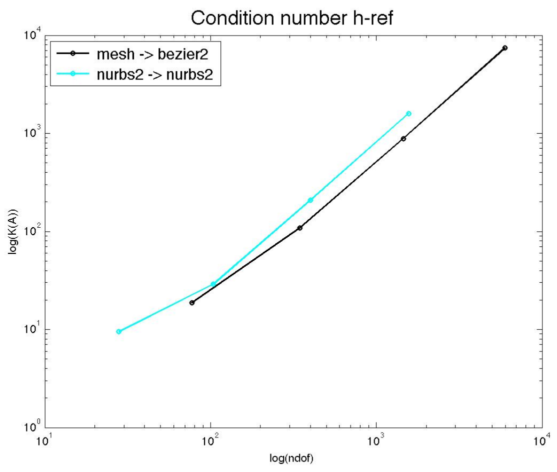


Figure 4.11: Condition number $K(A)$ of the stiffness matrix in the first four steps of h-refinement.

In both cases the stiffness matrices are banded, sparse, and structured matrices. Fig.4.12 shows the non-zero elements of the stiffness matrices during h-refinement in the ($mesh \rightarrow bezier2$) and ($nurbs2 \rightarrow nurbs2$) cases. As expected, the first one has bigger matrices with more non-zero elements.

The support of the B-spline functions of order m is always m knot spans. Thus, higher-order functions have support over much larger portions of the domain than do classical FEA functions. But this does not lead to an increase of bandwidth when using B-spline instead of FEA basis. In fact, the bandwidth of the matrices is the same.

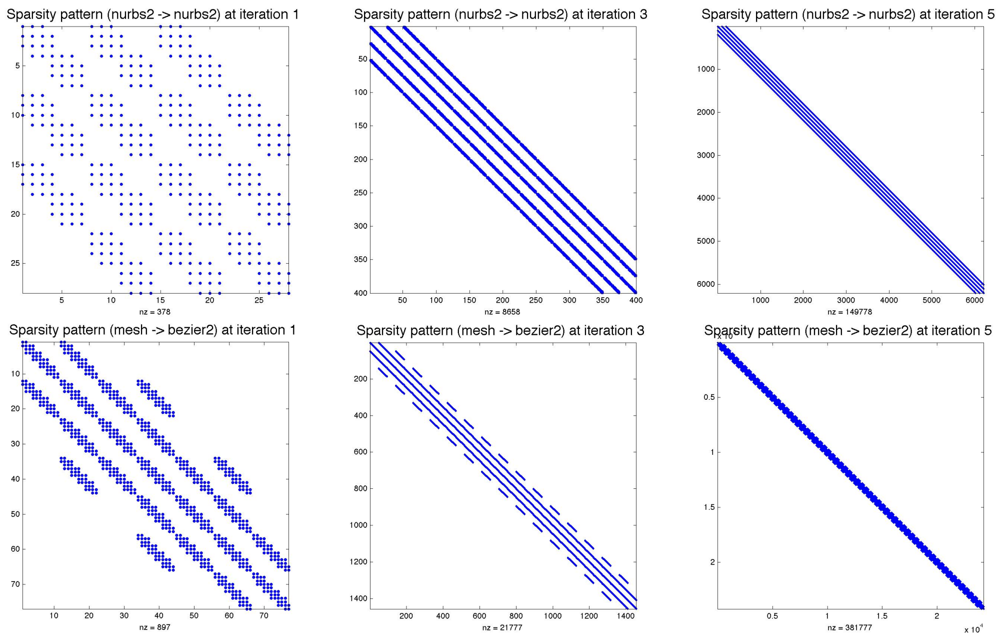


Figure 4.12: Sparsity patterns of the stiffness matrices under h-refinement: $(nurbs2 \rightarrow nurbs2)$ in the upper row and $(mesh \rightarrow bezier2)$ in the bottom row.

4.2.2 *p*-refinement

We also analyzed the behavior of this problem under *p*-refinement for the $(mesh \to bezier)$, $(nurbs \to nurbs)$ and $(nurbs \to spline)$. At each iteration we apply degree elevation only to the solution space. Fig. 4.13 shows the *mesh* and *nurbs* domains that we maintained fixed along the *p*-refinement process.

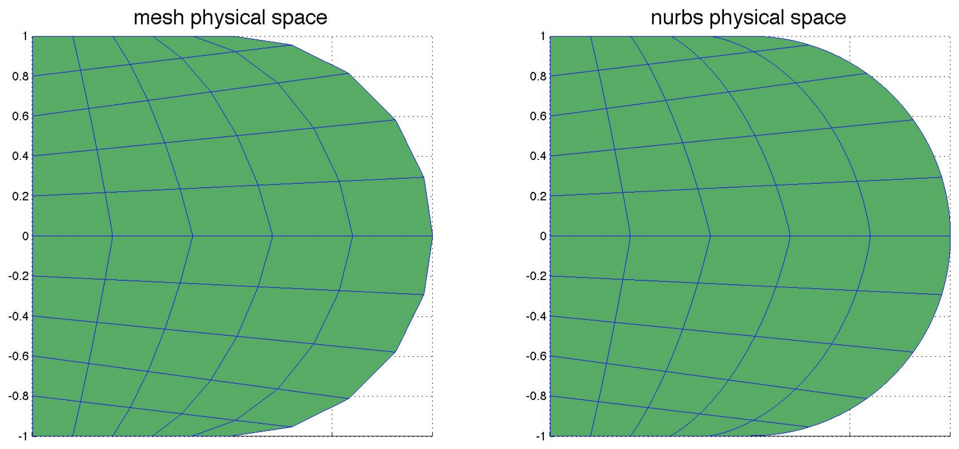


Figure 4.13: The physical domains used for *p*-refinement for the *mesh* (left) and *nurbs* (right) cases.

The resulting plot of the L^2 error for *p*-refinement is in Fig. 4.14, and Table 4.4 contains the numerical results. The $(nurbs \to spline)$ and the $(nurbs \to nurbs)$ cases give similar results, the $(mesh \to bezier)$ case takes more degrees of freedom.

The graph in Fig. 4.15 shows how the condition number grows as the degree of the solution space increases in the $(mesh \to bezier)$ and $(nurbs \to nurbs)$ cases. Also for *p*-refinement, for a fixed number of degrees of freedom, the linear system derived from IGA is worse conditioned with respect to the linear system derived from FEM.

Fig.4.16 shows the non-zero elements of the banded, sparse and structured stiffness matrices during *p*-refinement in the $(mesh \to bezier)$ and

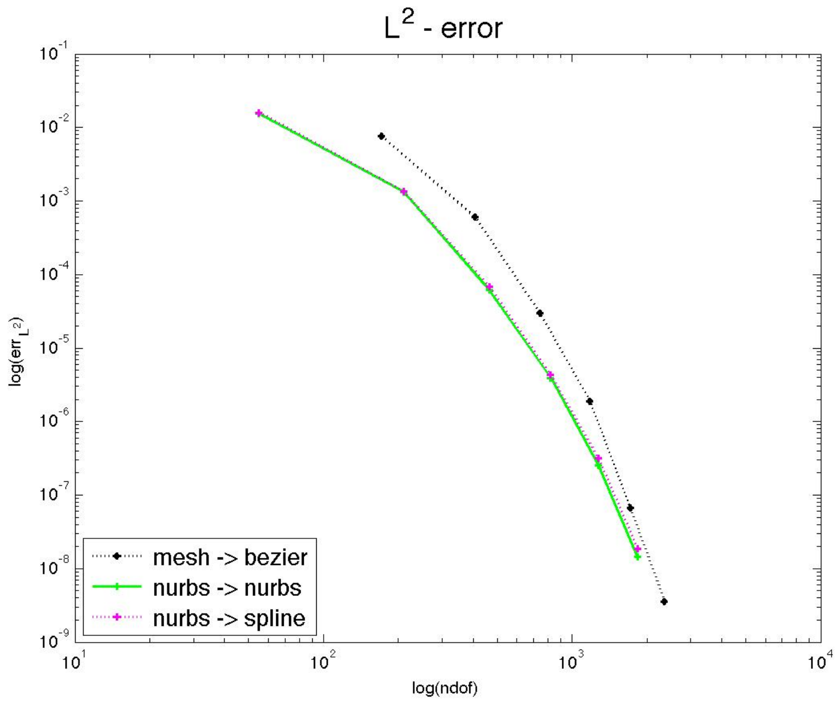


Figure 4.14: L^2 error under p-refinement.

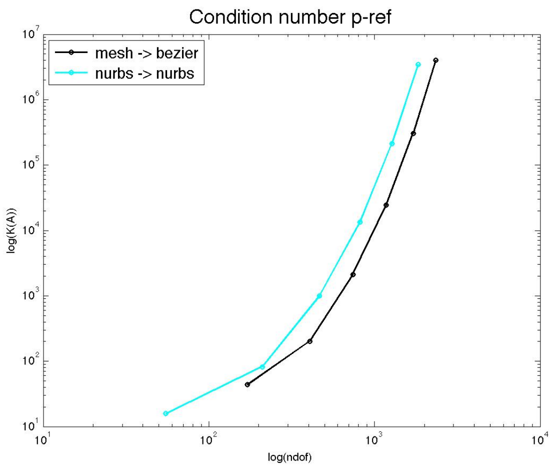


Figure 4.15: Condition number $K(A)$ of the stiffness matrix under p-refinement.

$(mesh \to bezier)$ p-refinement		
$ndof$	err_{L^2}	err_∞
171	7.5961e-03	1.5257e-02
406	6.0573e-04	1.8032e-03
741	3.0040e-05	8.7155e-05
1176	1.9065e-06	6.3871e-06
1711	6.6850e-08	1.6772e-07
2346	3.5597e-09	1.4448e-08

(a)

$(nurbs \to spline)$ p-refinement		
$ndof$	err_{L^2}	err_∞
55	1.5974e-02	3.5228e-02
210	1.3533e-03	2.9206e-03
465	6.8343e-05	1.6552e-04
820	4.3123e-06	1.4118e-05
1275	3.1826e-07	7.9948e-07
1830	1.8632e-08	9.8769e-08

(b)

$(nurbs \to nurbs)$ p-refinement		
$ndof$	err_{L^2}	err_∞
55	1.5283e-02	3.2647e-02
210	1.3295e-03	2.8108e-03
465	6.0949e-05	1.5610e-04
820	3.9210e-06	1.1961e-05
1275	2.5594e-07	6.1232e-07
1830	1.4390e-08	7.5115e-08

(c)

Table 4.4: Tables of results for p-refinement. Degree elevation is applied from degree 2 (first rows), to degree 7 (last rows).

($nurbs \to nurbs$) cases. As expected, the first one has bigger matrices with more non-zero elements. At the fist iteration the bandwidth is 25 for both cases.

Both cases get to an err_{L^2} smaller than 10^{-2} at the third iteration: the ($mesh \to bezier$) case takes more degrees of freedom than the ($nurbs \to nurbs$) case ($741 > 465$), and the stiffness matrix is more dense (non-zero elements: $22969 > 18109$).

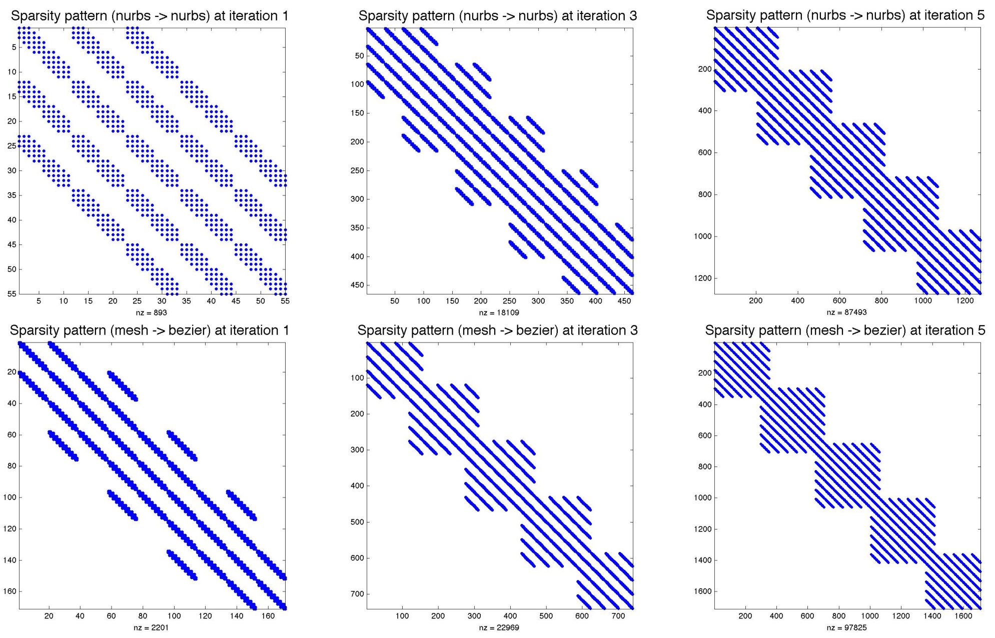


Figure 4.16: Sparsity patterns of the stiffness matrices under p-refinement: ($nurbs \to nurbs$) in the upper row and ($mesh \to bezier$) in the lower row.

4.3 Example 3

In this example, the BVP (4.7) is solved on a circular domain. The classic parameterization of the disk, obtained revolving a line on the XY plane around the Z axis, is problematic because one of the boundaries is collapsed to the center of the disk, and two other boundaries coincide with the revolved line.

We used an alternative parameterization, shown in Fig. 4.17, in which the four boundaries are the four quarters of circumference.

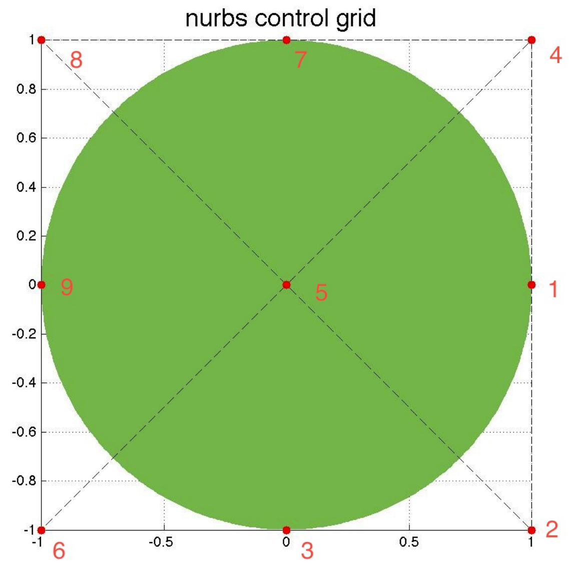


Figure 4.17: The circular domain and its control grid. The control points are numbered in red.

The basic *nurbs* parameterization, displayed at top left of Fig. 4.18, is a NURBS surface of degree $(2, 2)$, with a 3×3 control point grid, defined as follows:

```
pU = 2
pV = 2
knotVectorU = [0 0 0 1 1 1]
knotVectorV = [0 0 0 1 1 1]
controlPoints = [ (1, 0, 1),
(1/sqrt(2), -1/sqrt(2), 1/sqrt(2)), (0, -1, 1),
(1/sqrt(2), 1/sqrt(2), 1/sqrt(2)), (0, 0, 1/sqrt(2)),
(-1/sqrt(2), -1/sqrt(2), 1/sqrt(2)), (0, 1, 1),
(-1/sqrt(2), 1/sqrt(2), 1/sqrt(2)), (-1, 0, 1)]
```

As in the previous sections, we realized h-refinement iterations, placing a new knot in the middle of each knot span and compared the results. We performed five h-refinement iterations. Fig. 4.18 shows the *nurbs* and *mesh* physical domains at iteration 1, 3 and 5.

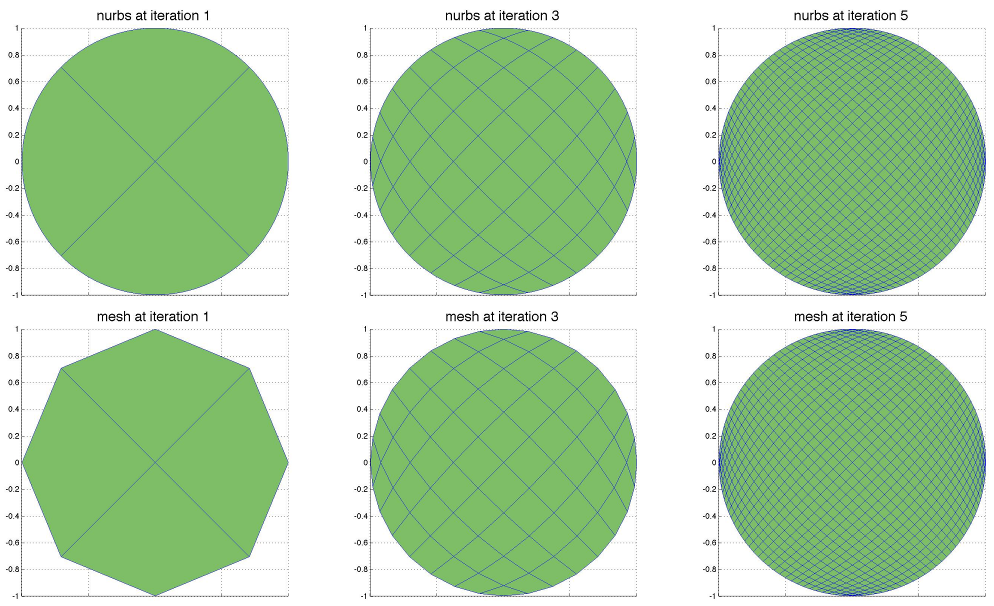


Figure 4.18: h-refinement of the disk physical domain: *nurbs* in the upper row and *mesh* in the lower row.

The loglog plot of the L^2 error for h-refinement is depicted in Fig. 4.19. The $(nurbs2 \rightarrow spline2)$ and the $(nurbs2 \rightarrow nurbs2)$ cases give similar results. The $(mesh \rightarrow bezier2)$ case, for a given number of degrees of freedom, presents the worst accuracy among the solutions of degree $p = 2$.

Table 4.5 reports the numerical results on h-refinement. We observe that, for the *mesh* domain, the solutions of degree 2 give more accurate results than the one of degree 1. In addition, if we compare similar numbers of degrees of freedom, $(mesh \rightarrow spline2)$, which is C^1 on Ω, is better than $(mesh \rightarrow bezier2)$, which is only C^0.

In general, the IGA approach applied on *nurbs* domains (that can repre-

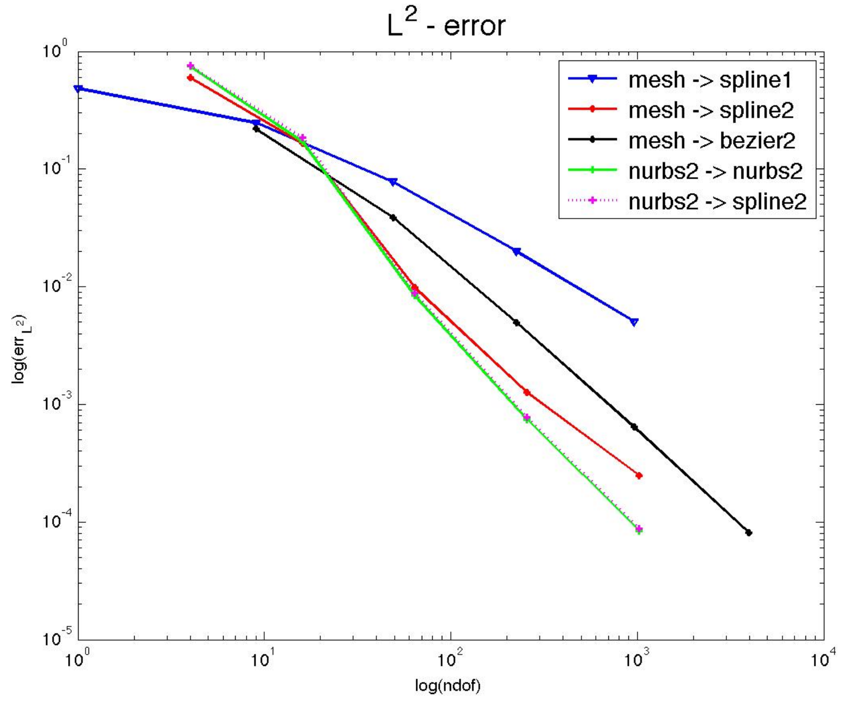


Figure 4.19: L^2 error under h-refinement.

sent the circular geometry exactly) gives the best results.

We also analyzed the behavior of this problem under p-refinement for the $(mesh \to bezier)$, $(nurbs \to nurbs)$, $(nurbs \to spline)$ and $(nurbs \to bezier)$ cases. At each iteration we apply degree elevation only to the solution space. Fig. 4.20 shows the *mesh* and *nurbs* parameterizations that we considered.

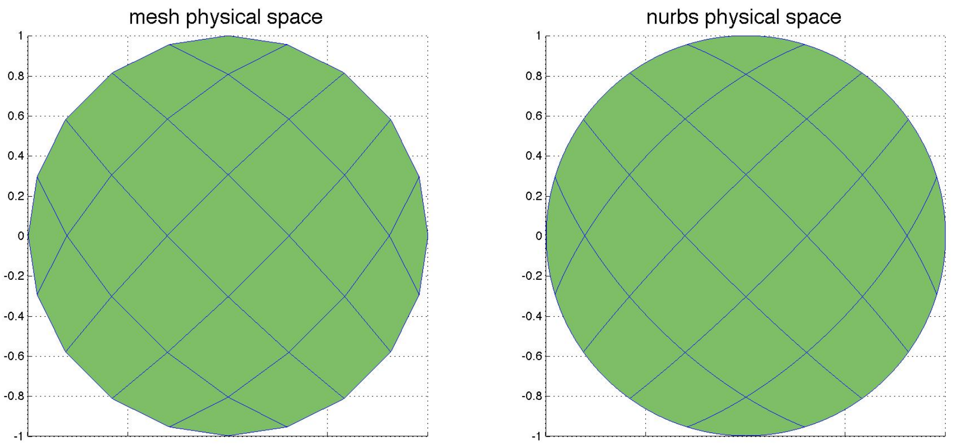


Figure 4.20: The physical spaces used for p-refinement for the *mesh* (left) and *nurbs* (right) cases.

The resulting plot of the L^2 error for p-refinement is illustrated in Fig. 4.21. The $(nurbs \to spline)$ and the $(nurbs \to nurbs)$ cases give similar results, the $(mesh \to bezier)$ takes more degrees of freedom.

$(mesh \to spline1)$ h-refinement		
$ndof$	err_{L^2}	err_∞
01	4.8530e-01	4.9941e-01
09	2.4671e-01	3.1456e-01
49	7.7755e-02	1.2001e-01
225	2.0074e-02	3.3111e-02
961	5.0586e-03	8.0507e-03

(a)

$(mesh \to spline2)$ h-refinement		
$ndof$	err_{L^2}	err_∞
04	5.9744e-01	7.3020e-01
16	1.6624e-01	2.6541e-01
64	9.8864e-03	1.7882e-02
256	1.2565e-03	2.0324e-03
1024	2.4885e-04	4.2204e-04

(b)

$(mesh \to bezier2)$ h-refinement		
$ndof$	err_{L^2}	err_∞
09	2.2124e-01	3.1734e-01
49	3.8860e-02	7.9256e-02
225	4.9830e-03	1.0301e-02
961	6.4370e-04	1.3842e-03
3969	8.1138e-05	1.6450e-04

(c)

$(nurbs2 \to nurbs2)$ h-refinement		
$ndof$	err_{L^2}	err_∞
04	7.4444e-01	7.9930e-01
16	1.7102e-01	2.6135e-01
64	8.3844e-03	1.4084e-02
256	7.4595e-04	1.5457e-03
1024	8.4261e-05	1.7148e-04

(d)

$(nurbs2 \to spline2)$ h-refinement		
$ndof$	err_{L^2}	err_∞
04	7.5887e-01	8.1477e-01
16	1.8592e-01	2.8018e-01
64	8.8963e-03	1.4789e-02
256	7.8165e-04	1.6058e-03
1024	8.7921e-05	1.7849e-04

(e)

Table 4.5: Tables of results for h-refinement.

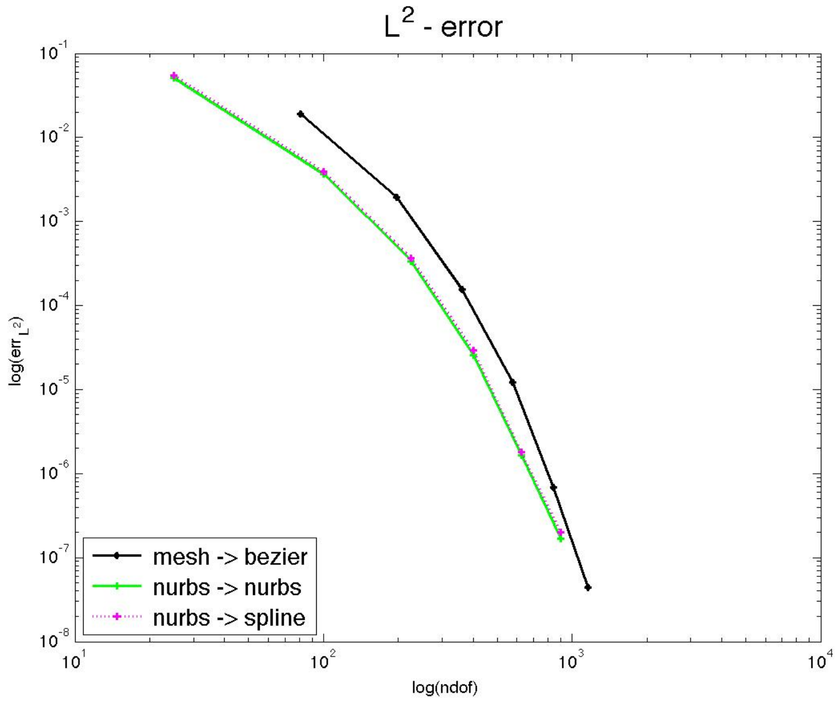


Figure 4.21: L^2 error for p-refinement.

$(mesh \rightarrow bezier)$ p-refinement		
$ndof$	err_{L^2}	err_{∞}
81	1.8873e-02	3.8785e-02
196	1.9588e-03	5.6540e-03
361	1.5450e-04	3.5886e-04
576	1.2182e-05	4.3281e-05
841	6.8095e-07	1.8201e-06
1156	4.4306e-08	1.4464e-07

(a)

$(nurbs \rightarrow spline)$ p-refinement		
$ndof$	err_{L^2}	err_{∞}
25	5.4446e-02	8.1206e-02
100	3.8786e-03	7.8453e-03
225	3.6517e-04	5.7762e-04
400	2.9033e-05	7.4289e-05
625	1.7911e-06	3.2319e-06
900	2.0141e-07	4.1683e-07

(b)

$(nurbs \rightarrow nurbs)$ p-refinement		
$ndof$	err_{L^2}	err_{∞}
25	5.1339e-02	7.7404e-02
100	3.6933e-03	7.6208e-03
225	3.3844e-04	5.4578e-04
400	2.5899e-05	6.8338e-05
625	1.6569e-06	3.0598e-06
900	1.6921e-07	3.6482e-07

(c)

Table 4.6: Tables of results for p-refinement. Degree elevation is applied from degree 2 (first rows), to degree 7 (last rows).

4.4 Example 4

We now solve problem (4.7) with a different source term:

$$f(x,y) = e^x((x^2+y^2-1)\sin(xy) - 2y\cos(xy)). \tag{4.9}$$

on a Γ-shaped domain Ω, represented in Fig. 4.22.

The exact solution is given by

$$u_{ex}(x,y) = e^x \sin(xy), \tag{4.10}$$

and we use it to set the non-homogeneous Dirichlet boundary conditions g.

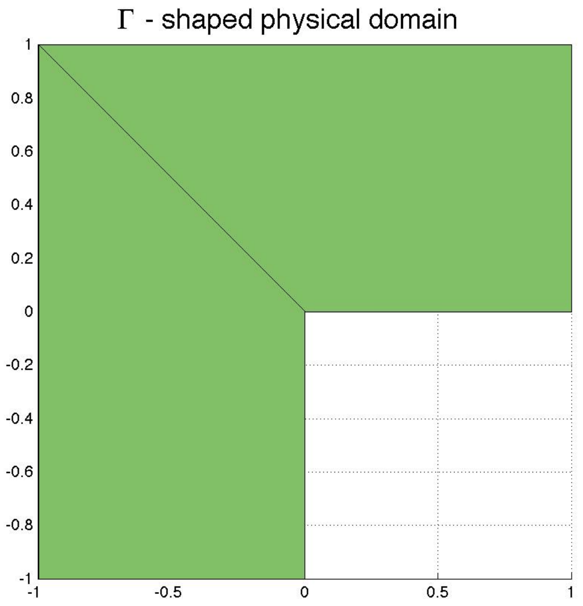


Figure 4.22: Γ-shaped domain.

We considered two different parameterizations of this domain, shown in Fig. 4.23: in the first (left), called *geoK*, we repeat the internal knots to obtain the C^0-continuity in the corners, and in the second (right), named *geoCP*, we use repeated control points, which allows for maintaining C^1 continuity in the corners. They both represent a NURBS surface of degree $(2,2)$, with 5×3 and 4×3 control point grids respectively.

Here is a detailed description of *geoK*:

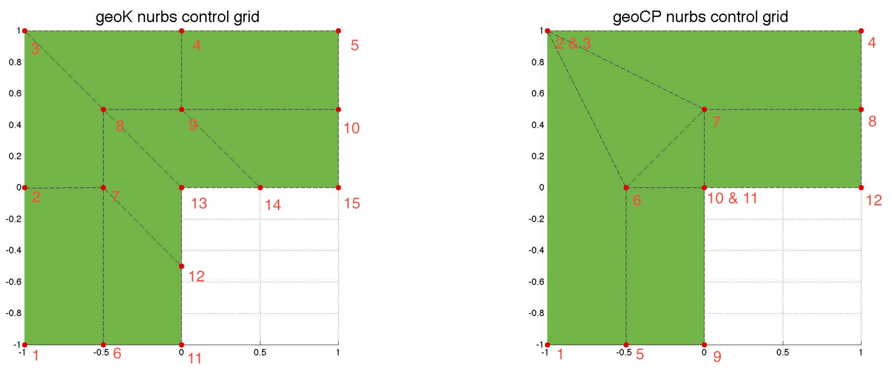


Figure 4.23: The domains *geoK* and *geoCP* with their control grids. The control points are numbered in red.

```
pU = 2
pV = 2
knotVectorU = [0 0 0 0.5 0.5 1 1 1]
knotVectorV = [0 0 0 1 1 1]
controlPoints = [ (-1, -1, 1), (-1, 0, 1), (-1, 1, 1),
(0, 1, 1), (1,1,1), (-0.5, -1, 1), (-0.5, 0, 1),
(-0.5, 0.5, 1), (0, 0.5, 1), (1, 0.5, 1), (0, -1, 1),
(0, -0.5, 1), (0, 0, 1), (0.5, 0, 1), (1, 0, 1) ]
```

The *geoCP* domain parameterization is given by:

```
pU = 2
pV = 2
knotVectorU = [0 0 0 0.5 1 1 1]
knotVectorV = [0 0 0 1 1 1]
controlPoints = [ (-1, -1, 1), (-1, 1, 1), (-1, 1, 1),
(1,1,1), (-0.5, -1, 1), (-0.5, 0, 1), (0, 0.5, 1),
(1, 0.5, 1), (0, -1, 1), (0, 0, 1), (0, 0, 1), (1, 0, 1) ]
```

An analytic description of this geometry is not known, so we only used the *nurbs* and *mesh* definitions for the physical domain. For the solution space we used *nurbs*, *spline* and *bezier*.

4.4.1 h-refinement on $geoK$ and $geoCP$

We performed five h-refinement iterations. A representation of the $nurbs$ and $mesh$ physical spaces at iteration 1, 3 and 5 can be seen in Fig. 4.24 for $geoK$ geometry and in Fig. 4.25 for $geoCP$ geometry.

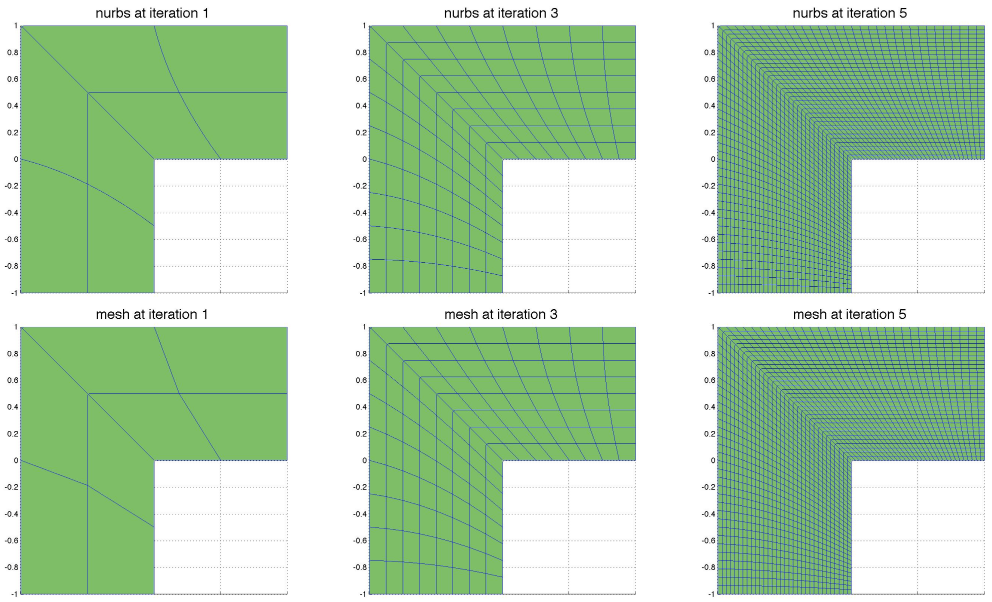


Figure 4.24: h-refinement of the Γ-shaped physical domain $geoK$: $nurbs$ in the upper row and $mesh$ in the lower row.

In Fig. 4.26 the exact solution and the solution obtained at the first iteration of h-refinement in the ($mesh \rightarrow bezier2$) case are illustrated for $geoK$ geometry.

For both $geoK$ an $geoCP$, throughout the h-refinement process, the number of degrees of freedom passes from around 10 to around 2000 in 5 iterations in the ($mesh \rightarrow spline1$), ($nurbs2 \rightarrow nurbs2$) and ($nurbs2 \rightarrow spline2$) cases. Adding the same knots in a C^0-continuous domain results in much more degrees of freedom: ($mesh \rightarrow bezier2$) goes from 21 to 8001 degrees of freedom.

The loglog plot of the L^2 error for h-refinement is depicted in Fig. 4.27 for

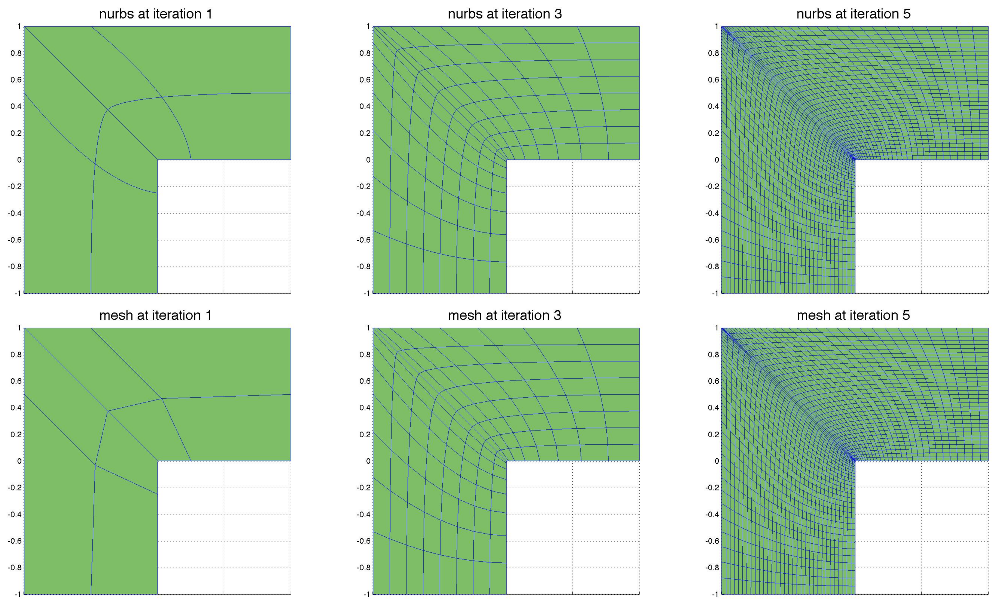


Figure 4.25: h-refinement of the Γ-shaped physical domain $geoCP$: *nurbs* in the upper row and *mesh* in the lower row.

$geoK$ and $geoCP$ geometry. The $(nurbs2 \to spline2)$ and the $(nurbs2 \to nurbs2)$ cases give identical results. The $(mesh \to bezier2)$ and $(nurbs2 \to bezier2)$ cases have the biggest number of degrees of freedom and the smallest error.

Tables 4.7 and 4.8 report the numerical results of h-refinement for $geoK$ and $geoCP$ geometries, respectively. The solutions using IGA on *nurbs* domains obtain better results, even if for this example also the *mesh* domains represent the geometry exactly. In all the previous examples we noted that the $(nurbs2 \to spline2)$ and $(nurbs2 \to nurbs2)$ cases gave similar results: for this example, the two results are identical, for both the geometries $geoK$ and $geoCP$. This is because these parameterizations are non-rational, that is, all the control points have unitary weights, thus the *nurbs* and *spline* solution spaces coincide.

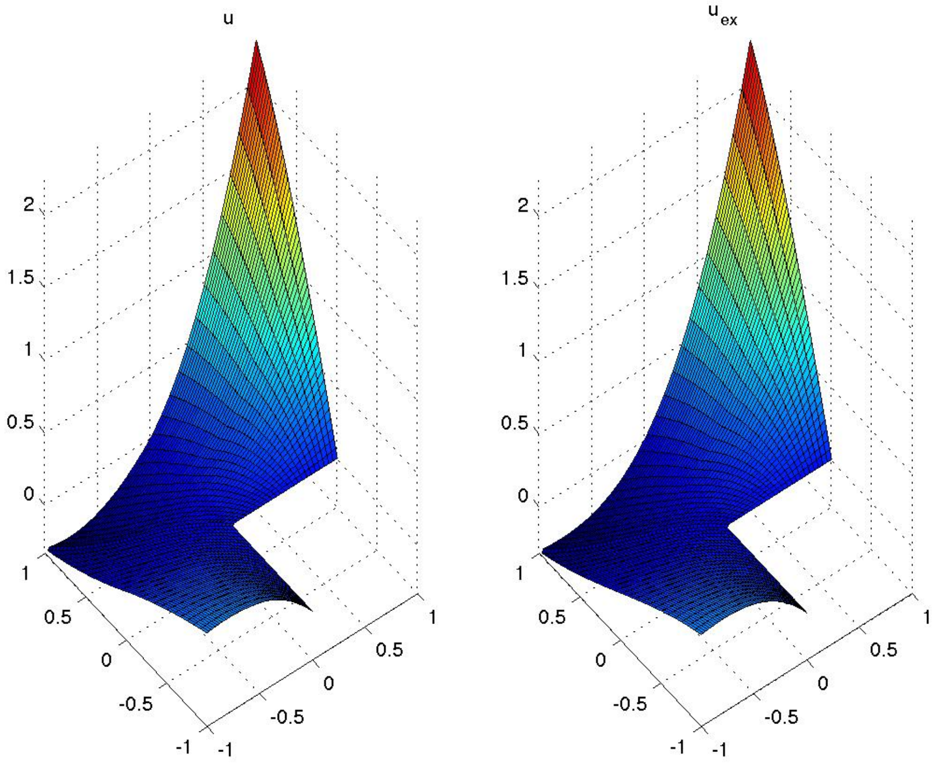
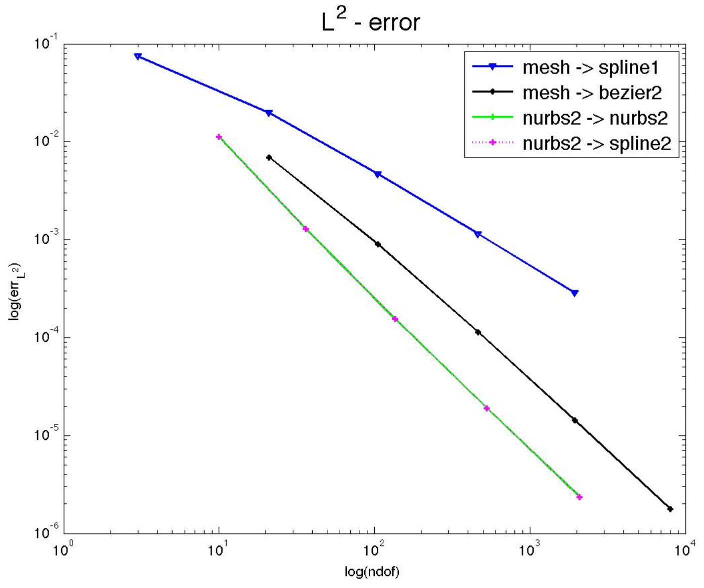


Figure 4.26: Computed ($mesh \rightarrow bezier2$) and exact solutions for $geoK$ geometry at the first iteration of h-refinement.



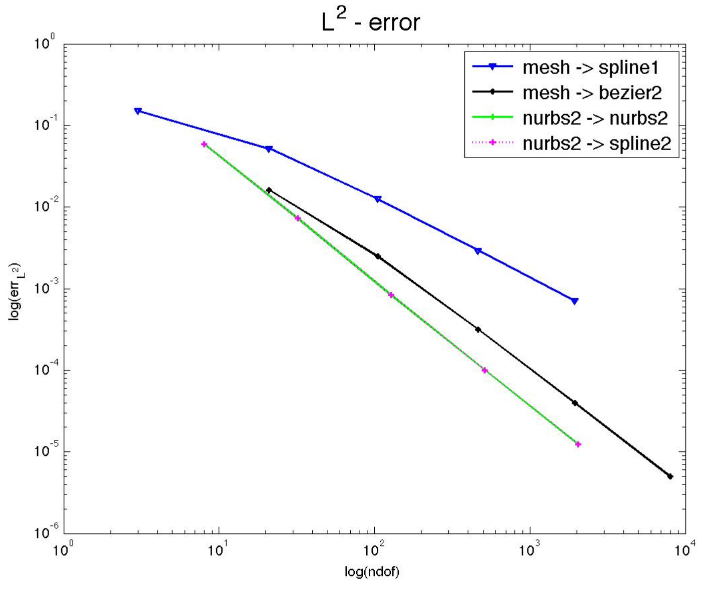


Figure 4.27: L^2 error under h-refinement on $geoK$ (top) and $geoCP$ (bottom).

$(mesh \to spline1)$ h-refinement		
$ndof$	err_{L^2}	err_∞
03	7.4730e-02	1.9888e-01
21	1.9657e-02	8.0192e-02
105	4.6932e-03	2.2104e-02
465	1.1443e-03	5.2775e-03
1953	2.8350e-04	1.1461e-03

(a)

$(mesh \to bezier2)$ h-refinement		
$ndof$	err_{L^2}	err_∞
21	6.9296e-03	1.5115e-02
105	9.0330e-04	2.1441e-03
465	1.1380e-04	2.5802e-04
1953	1.4249e-05	3.1893e-05
8001	1.7814e-06	4.4507e-06

(b)

$(nurbs2 \to nurbs2)$ h-refinement		
$ndof$	err_{L^2}	err_∞
10	1.1130e-02	2.5985e-02
36	1.2958e-03	3.4674e-03
136	1.5463e-04	3.8367e-04
528	1.9026e-05	4.7227e-05
2080	2.3676e-06	6.2163e-06

(c)

$(nurbs2 \to spline2)$ h-refinement		
$ndof$	err_{L^2}	err_∞
10	1.1130e-02	2.5985e-02
36	1.2958e-03	3.4674e-03
136	1.5463e-04	3.8367e-04
528	1.9026e-05	4.7227e-05
2080	2.3676e-06	6.2163e-06

(d)

Table 4.7: Tables of results for h-refinement on *geoK* parameterization.

$(mesh \to spline1)$ h-refinement		
$ndof$	err_{L^2}	err_∞
03	1.4957e-01	4.1091e-01
21	5.1749e-02	1.7265e-01
105	1.2544e-02	5.9303e-02
465	2.9198e-03	1.2417e-02
1953	7.0155e-04	3.0016e-03

(a)

$(mesh \to bezier2)$ h-refinement		
$ndof$	err_{L^2}	err_∞
21	1.6172e-02	3.8833e-02
105	2.4988e-03	7.1787e-03
465	3.1703e-04	8.3576e-04
1953	3.9679e-05	1.0967e-04
8001	4.9778e-06	1.7838e-05

(b)

$(nurbs2 \to nurbs2)$ h-refinement		
$ndof$	err_{L^2}	err_∞
08	5.9258e-02	1.7738e-01
32	7.2541e-03	2.3754e-02
128	8.2921e-04	2.4444e-03
512	1.0031e-04	2.9442e-04
2048	1.2394e-05	4.1533e-05

(c)

$(nurbs2 \to spline2)$ h-refinement		
$ndof$	err_{L^2}	err_∞
08	5.9258e-02	1.7738e-01
32	7.2541e-03	2.3754e-02
128	8.2921e-04	2.4444e-03
512	1.0031e-04	2.9442e-04
2048	1.2394e-05	4.1533e-05

(d)

Table 4.8: Tables of results for h-refinement on $geoCP$ parameterization.

4.4.2 Comparison between $geoK$ and $geoCP$

In Fig. 4.28 we compare the L^2 errors obtained with the two different parameterizations of the Γ-shaped domain throughout the h-refinement process. In all the cases, the parameterization with repeated knots $geoK$ obtains the best results.

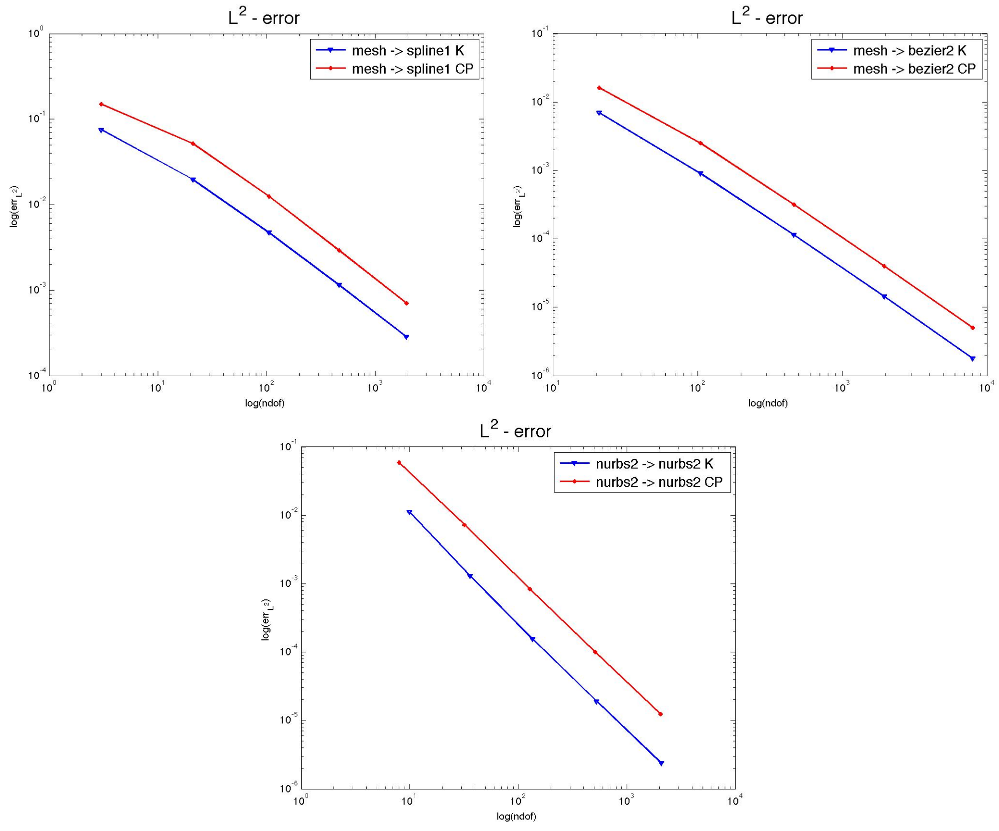


Figure 4.28: L^2 error for h-refinement in $(mesh \rightarrow spline1)$, $(mesh \rightarrow bezier2)$ and $(nurbs2 \rightarrow nurbs2)$ cases.

As expected, $(mesh \rightarrow spline1)$ provides the less accurate results. Let an accuracy threshold be fixed at 10^{-5}. We can observe that $(mesh \rightarrow spline1)$ does not reach the threshold. The IGA solution $(nurbs2 \rightarrow nurbs2)$ reaches the given threshold exploiting less degrees of freedom with respect to the FEM classical solution $(mesh \rightarrow bezier2)$.

We will not go through the details of the p-refinement process. We only report, in Fig. 4.29, the comparison of the L^2 errors obtained with the two different parameterizations $geoK$ and $geoCP$ under p-refinement from degree 2 to degree 7. Also in this case, the parameterization with repeated knots always obtains the best results.

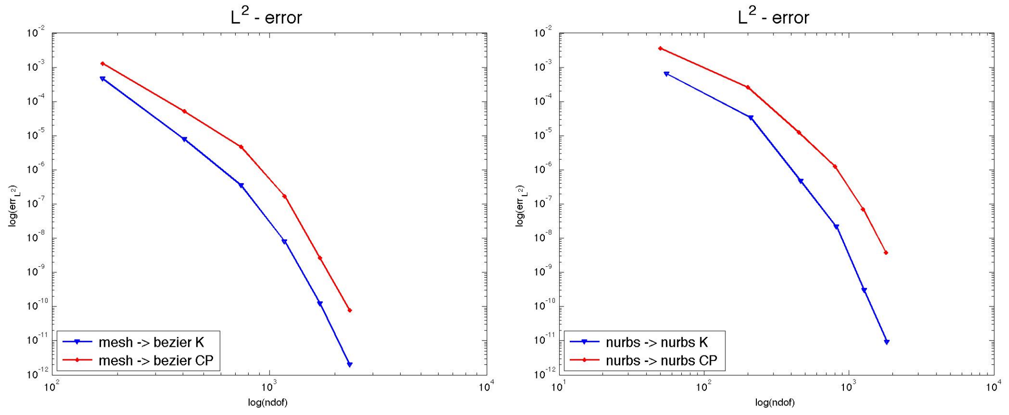


Figure 4.29: L^2 error for p-refinement in $(mesh \to bezier)$ and $(nurbs \to nurbs)$ cases.

4.5 Example 5

In this section we will solve the Laplace-Beltrami problem (2.46) on a single patch surface representing a piece of the unitary sphere embedded in $\mathbb{R}^3$. We used a beta version of GeoPDEs that allows to solve BVPs on lower dimensional manifolds.

We set the source function:

$$f(\phi,\theta) = \sin(\alpha\phi)\sin(\beta\theta)\left[\frac{\alpha^2}{\sin^2(\theta)} + \beta^2 - \beta\frac{\cos(\theta)\cos(\beta\theta)}{\sin(\theta)\sin(\beta\theta)}\right], \tag{4.11}$$

where $\phi := \text{atan2}\left(\frac{x}{y}\right)$, $\theta := \arccos\left(\frac{z}{r}\right)$. The exact solution in spherical coordinates is given by:

$$u(\phi,\theta) = \sin(\alpha\phi)\sin(\beta\theta), \tag{4.12}$$

and we used it to impose Dirichlet boundary conditions. In particular, we solved the problem for $\alpha = 3$, $\beta = 4$ and $r = 1$.

For the physical domain, we used a NURBS surface of degree (2,2) to represent the *nurbs* domain and a NURBS surface of degree (1,1) to represent the *mesh* domain. We consider three cases: $(mesh \to spline2)$, $(mesh \to bezier2)$ and $(nurbs \to spline2)$.

We performed five iterations of h-refinement and compared the results. Fig. 4.30 shows different subdivisions of the physical domains.

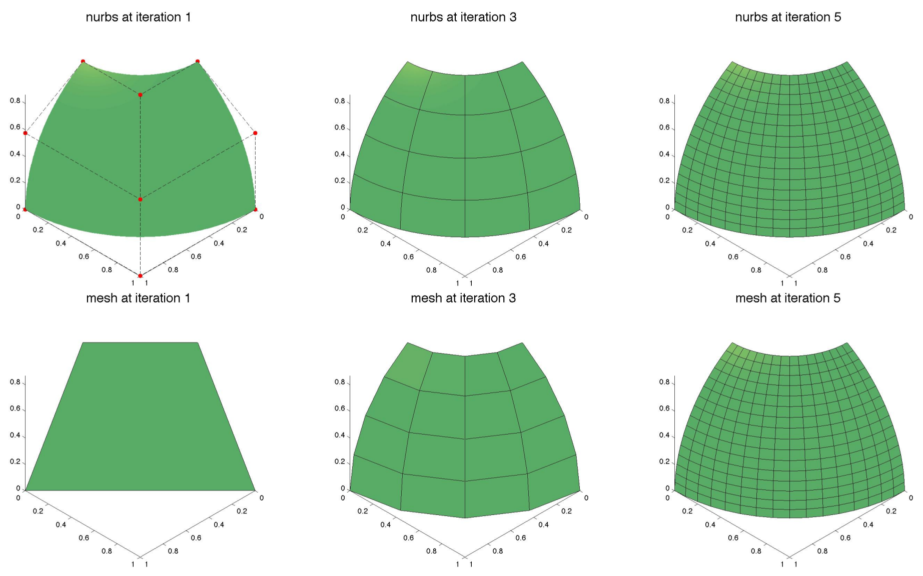


Figure 4.30: *nurbs* (top row) and *mesh* (bottom row) physical domains at different iterations of the h-refinement process. In the top left image, the red points are the control points of the NURBS surface.

The exact and computed solutions are shown in Fig. 4.31. It can be noted that, as expected, the solutions computed at the first iteration are not precise in all the three cases.

Fig. 4.32 and Table 4.9 show the errors at different steps of the h-refinement process. As expected, IGA applied on the *nurbs* representation gives better results.

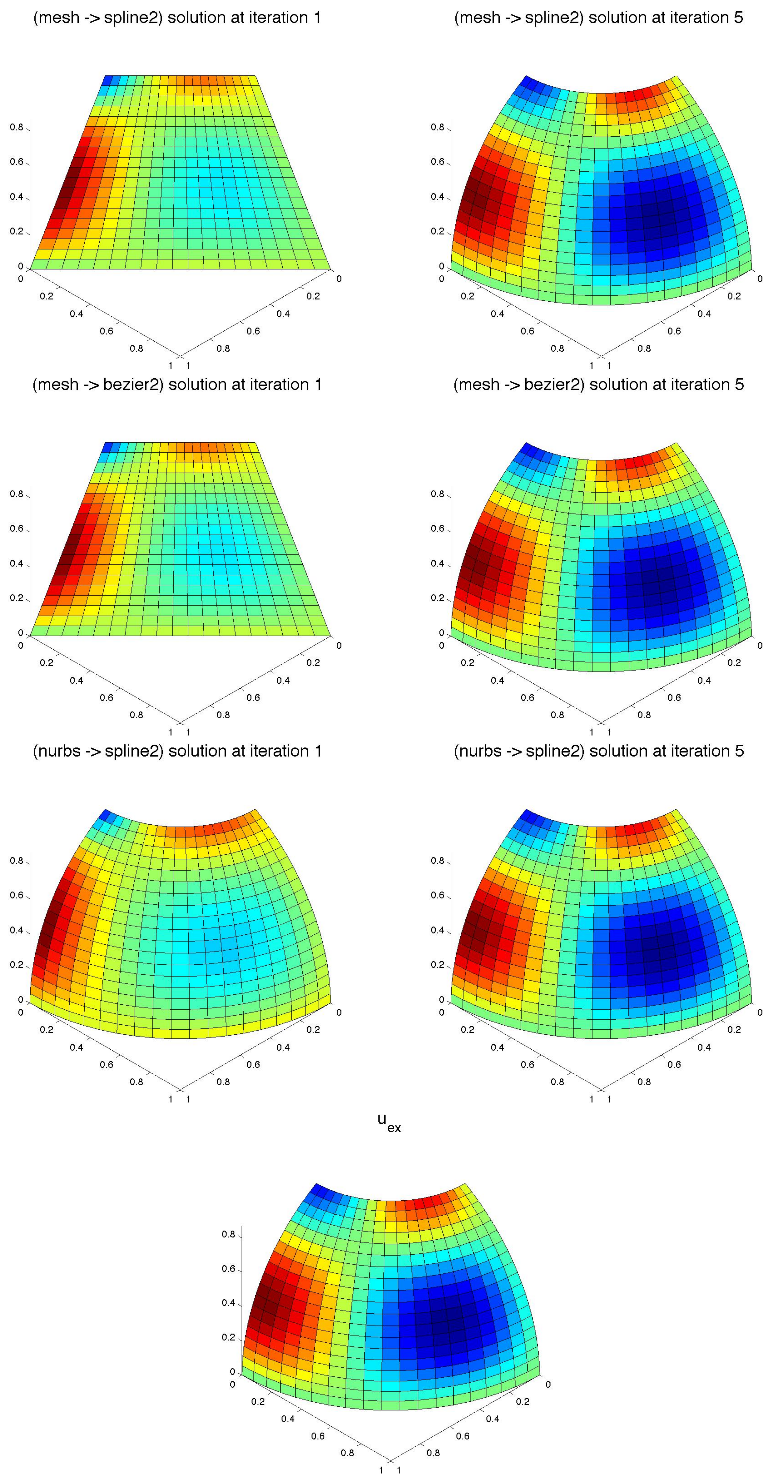


Figure 4.31: Computed solutions at the first and last iterations of the h-refinement process for ($mesh \rightarrow spline2$) (first row), ($mesh \rightarrow bezier2$) (second row), and ($nurbs \rightarrow spline2$) (third row). The exact solution is illustrated in the fourth row.

To get an error smaller than 10^{-3}, the $(nurbs \rightarrow spline2)$ case takes 3 iterations and 64 degrees of freedom, the $(mesh \rightarrow spline2)$ case takes 4 iterations and 256 degrees of freedom and the $(mesh \rightarrow bezier2)$ case is the worst, with 4 iterations and 961 degrees of freedom.

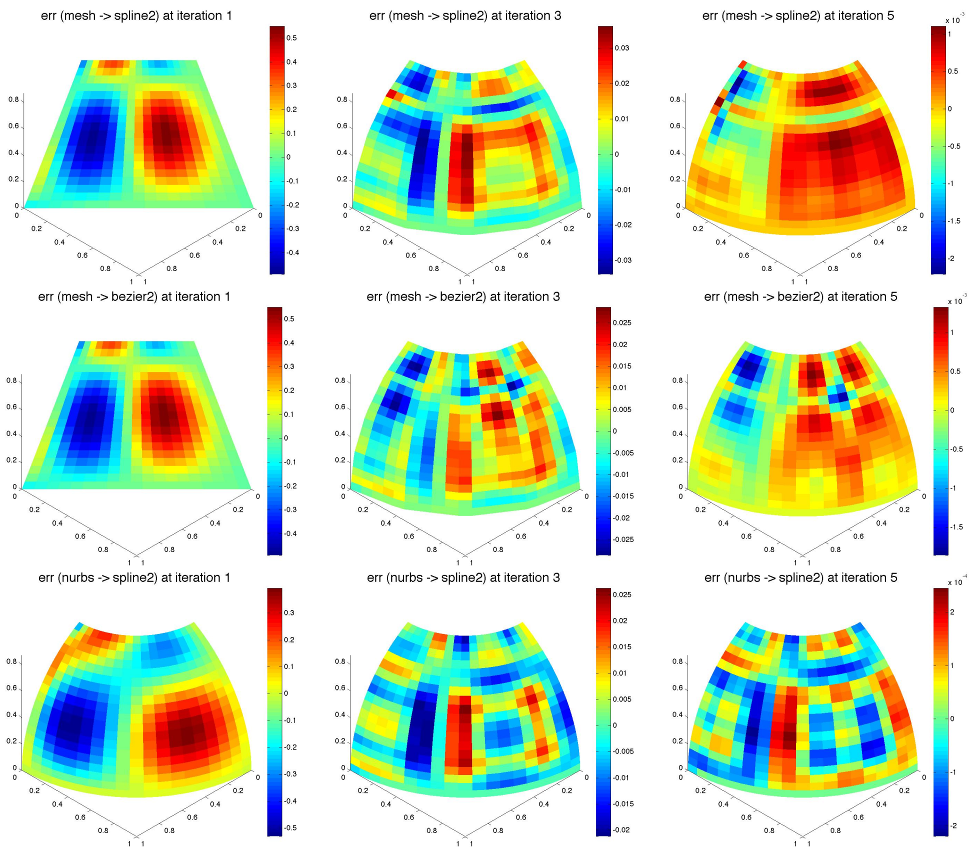


Figure 4.32: Plot of *err* values at different iterations of the *h*-refinement process for $(mesh \rightarrow spline2)$ (first row), $(mesh \rightarrow bezier2)$ (second row), and $(nurbs \rightarrow spline2)$ (third row).

$(mesh \to spline2)$ h-refinement		
$ndof$	err_{L^2}	err_∞
01	2.0957e-01	5.4916e-01
04	7.9492e-02	1.9033e-01
16	1.5335e-02	3.6164e-02
64	2.6998e-03	8.0745e-03
256	6.1944e-04	2.2081e-03

$(mesh \to bezier2)$ h-refinement		
$ndof$	err_{L^2}	err_∞
01	2.0957e-01	5.4916e-01
09	4.8563e-02	1.3174e-01
49	1.1232e-02	2.8589e-02
225	2.6000e-03	8.8861e-03
961	6.2974e-04	1.8611e-03

$(nurbs \to spline2)$ h-refinement		
$ndof$	err_{L^2}	err_∞
01	2.0201e-01	5.3050e-01
04	9.1471e-02	1.7894e-01
16	1.0039e-02	2.6220e-02
64	8.6420e-04	2.2633e-03
256	9.6604e-05	2.4482e-04

Table 4.9: Table of results for h-refinement.

Chapter 5

Numerical examples: IGA on multipatch hybrid geometries

In this chapter we apply IGA to solve BVPs on multipatch domains. The multipatch representation corresponds to a decomposition of the computational domain into non-overlapping subdomains also called patches in the geometrical framework. In the examples of Sections 5.1 and 5.2 we investigate the solution of BVPs on multipatch domains in which the patches are all represented by the same kind of geometry (*mesh* or *nurbs*). In particular, in Example 1 we introduce the formalism for the multipatch configuration in GeoPDEs. Then, in Sections 5.3 and 5.4, we propose to use multipatch hybrid physical domains, made of both *mesh* and *nurbs* patches, handling a C^0 join between patches. Finally, in Sections 5.5, 5.6 and 5.7, we solve a Laplace-Betrami problem on three 3D hybrid domains representing two-dimensional manifolds embedded in $\mathbb{R}^3$. The requirement of compatibility between patches along the interfaces will be satisfied in all the cases.

To compute a total accuracy error, we consider the 2-norm of the vector of the errors on each patch. For instance, the L^2 error in a multipatch example on a domain Ω made of n patches will be computed as:

$$err_{L^2} := \|err\|_{L^2(\Omega)} = \left(\sum_{i=1}^{n} \|err\|^2_{L^2(P_i)} \right)^{\frac{1}{2}}, \tag{5.1}$$

where $\|err\|_{L^2(P_i)}$ is the L^2 error on the i-th patch, defined by Eq. (4.2). The global errors err_{H^1}, err_∞ and err_2 are obtained analogously from the individual errors on the patches.

GeoPDEs can read multipatch geometries from .txt files that contain information on patches, interfaces, subdomains and boundaries.

For the sake of clarity, in our examples the domain partition used to set the quadrature rule always coincides with the knot-spans in the geometry read from the files.

As solution space, we used the space of spline functions of degree (2, 2). This implies that, unless there are knots with double multiplicity, the solution is of class C^2 on both directions inside each patch. Along each interface, the two solutions from two adjacent patches are equal, and they are glued together with C^0 continuity.

For the multipatch case, the number of degrees of freedom $ndof$ is the sum of the number of degrees of freedom of the patches, minus the degrees of freedom along each interface, which must be counted only once. If Dirichlet boundary conditions are applied, also the degrees of freedom along the boundary need to be subtracted from the total count.

5.1 Example 1

We consider the BVP in form (4.7) with the source term given in Eq. (4.9), defined on the same domain of the example in Section 4.4, illustrated in Fig. 4.22. The Γ-shaped domain will be represented through a multipatch geometry composed of three non- overlapping patches: $P1$, $P2$ and $P3$ as shown in Fig. 5.1. This partition implies the presence of two interfaces: $I1$ shared by $P1$ and $P2$, and $I2$ shared by $P2$ and $P3$.

The boundary of the domain is composed by all the edges of the patches that are not interfaces, and we applied Dirichlet boundary conditions.

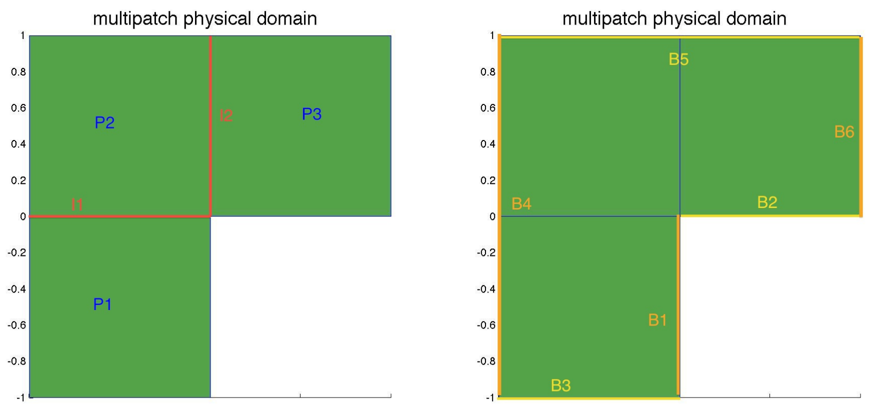


Figure 5.1: Γ-shaped multipatch domain: patches and interfaces on the left, boundaries on the right.

We solved the BVP on the *mesh* and *nurbs* physical domains shown in Fig. 5.2, and compared the results obtained using $spline2$ as solution space, i.e. the space of spline functions of degree (2, 2).

The *mesh* multipatch domain is constituted by three NURBS patches of degree (1,1). To create the *mesh* domain, we simply obtained each patch by the extrusion of a line. We chose to always extrude a left-to-right line in a bottom-to-top direction. Therefore all the patches have the same structure, with the u direction along the x-axis and the v direction along the y-axis. Since the parametric directions on the patches coincide, the orientation flag of the two interfaces is equal to 1.

Then we refined the three patches to obtain the domain shown in Fig.5.2 (left). To refine this domain in a fashion that satisfies the compatibility requirement, it is not necessary to refine all the three patches in the same way. To make sure that the meshes of two patches coincide on the shared

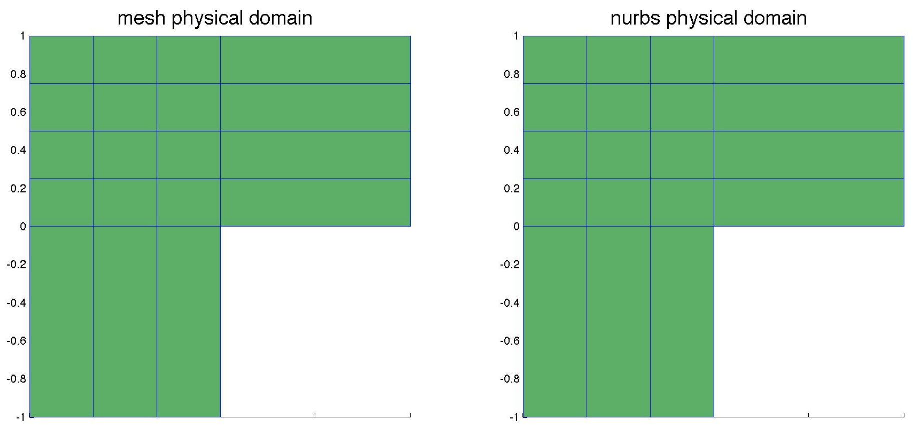


Figure 5.2: Γ-shaped physical domain: *mesh* on the left and *nurbs* on the right, together with the isocurves at the knot values (u, v).

interface, it is sufficient to apply the same refinement to patches $P1$ and $P2$ along the u direction, and to patches $P2$ and $P3$ along the v direction. Therefore $P2$ is a mesh of 4×5 vertices, while the meshes $P1$ and $P3$ have respectively $4 \times i$ and $j \times 5$ vertices, with arbitrary i and j. For this example we set i and j both equal to 2,

To obtain the *nurbs* domain, we extruded the same three lines and then we elevated the degree of each patch in both directions before performing the knot insertion, in order to get internal knots with single multiplicity (see Fig.5.2 (right)).

We will now inspect the geometry file `gamma.txt` for the *nurbs* physical domain. The first line is:

```
2 3 2 1
```

These four numbers respectively represent:

- the dimension of the geometry,
- the number of patches that constitute the domain,
- the number of interfaces, each one connecting two patches,

- the number of subdomains, formed by the union of patches. In our examples it is always equal to 1.

Then the three patches are defined. They are:

```
# PATCH P1
   2   2      # degree U       degree V
   5   3      # ctrl pts U     ctrl pts V
0   0   0   0.33   0.67   1   1   1     #knotVectorU
0   0   0   1   1   1                   #knotVectorV
# control points: X coordinate in the first row, Y on the second, weight W on the third
-1  -0.833  -0.5  -0.167  0  -1    -0.833  -0.5  -0.167  0   -1  -0.833  -0.5  -0.167  0
-1  -1      -1    -1     -1  -0.5  -0.5    -0.5  -0.5   -0.5  0  0       0     0      0
 1   1       1     1      1   1     1       1     1      1    1  1       1     1      1
# PATCH P2
   2   2      # degree U       degree V
   5   6      # ctrl pts U     ctrl pts V
0   0   0   0.33   0.67   1   1   1               #knotVectorU
0   0   0   0.25   0.5   0.75   1   1   1         #knotVectorV
# control points: X coordinate in the first row, Y on the second, weight W on the third
-1 -0.833 -0.5   -0.167 0 -1    -0.833 -0.5  -0.167  0    -1    -0.833 -0.5  -0.167 0
 0  0      0      0     0  0.125 0.125  0.125 0.125  0.125 0.375 0.375  0.375 0.375 0.375...
 1  1      1      1     1  1     1      1     1      1     1     1      1     1     1

  -1    -0.833 -0.5   -0.167  0    -1    -0.833 -0.5  -0.167  0    -1 -0.833 -0.5 -0.167 0
...0.625 0.625  0.625  0.625  0.625 0.875 0.875  0.875 0.875  0.875 1  1      1    1     1
   1     1      1      1      1     1     1      1     1      1     1  1      1    1     1
# PATCH P3
   2   2      # degree U       degree V
   3   6      # ctrl pts U     ctrl pts V
0   0   0   1   1   1                             #knotVectorU
0   0   0   0.25   0.5   0.75   1   1   1         #knotVectorV
# control points: X coordinate in the first row, Y on the second, weight W on the third
0 0.5 1  0      0.5    1      0      0.5    1      0      0.5    1      0      0.5
0 0    0  0.125  0.125  0.125  0.375  0.375  0.375  0.625  0.625  0.625  0.875  0.875 ...
1 1    1  1      1      1      1      1      1      1      1      1      1      1


   1     0  0.5 1
...0.875 1  1   1
   1     1  1   1
```

After the definition of the patches, in the file `gamma.txt` there is the definition of the two interfaces. To define an interface, five integer values are needed:

- `patch1`: the index of the first patch to which the edge belongs
- `side1`: the local index of the edge in the first patch
- `patch2`: the index of the second patch to which the edge belongs
- `side2`: the local index of the edge in the second patch
- `orientation`: a flag equal to 1 if the two edges have the same orientation, 0 otherwise

To assign the `side` index to an edge, GeoPDEs uses an edge numbering of the parametric domain shown in Fig. 5.3, which is inherited by the physical domain.

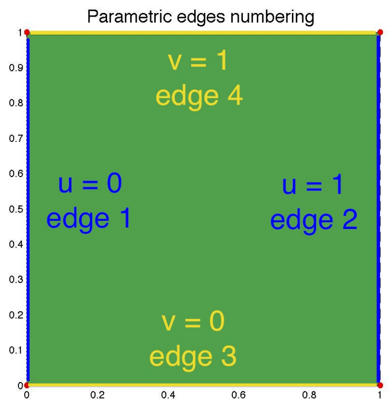


Figure 5.3: Edge numbering in the parametric domain, used to define interfaces and boundaries.

Since interface $I1$ is shared by edge 4 of $P1$ and edge 3 of $P2$, interface $I2$ is shared by edge 2 of $P2$ and edge 1 of $P3$ and in each interface the edges have the same direction, the file reads like this:

```
# INTERFACE I1
1 4
```

```
2 3
1
# INTERFACE I2
2 2
3 1
1
```

We only have one subdomain that contains all the patches:

```
# SUBDOMAIN 1
1 2 3
```

At last, the boundaries are defined. All the edges that are not part of an interface, are part of a boundary. Each boundary is constituted by one or more edges from different patches, and the definition of boundaries in the geometry file is similar to the one of interfaces, with the only difference that boundaries don't need the orientation flag.

```
# BOUNDARY 1
1
1 2
# BOUNDARY 2
1
3 3
# BOUNDARY 3
1
1 3
# BOUNDARY 4
2
1 1
2 1
# BOUNDARY 5
2
2 4
3 4
# BOUNDARY 6
```

```
1
3 2
```

As expected, since this geometry is not curved, *mesh* and *nurbs* give identical results, shown in Fig. 5.4 and listed in Table 5.1.

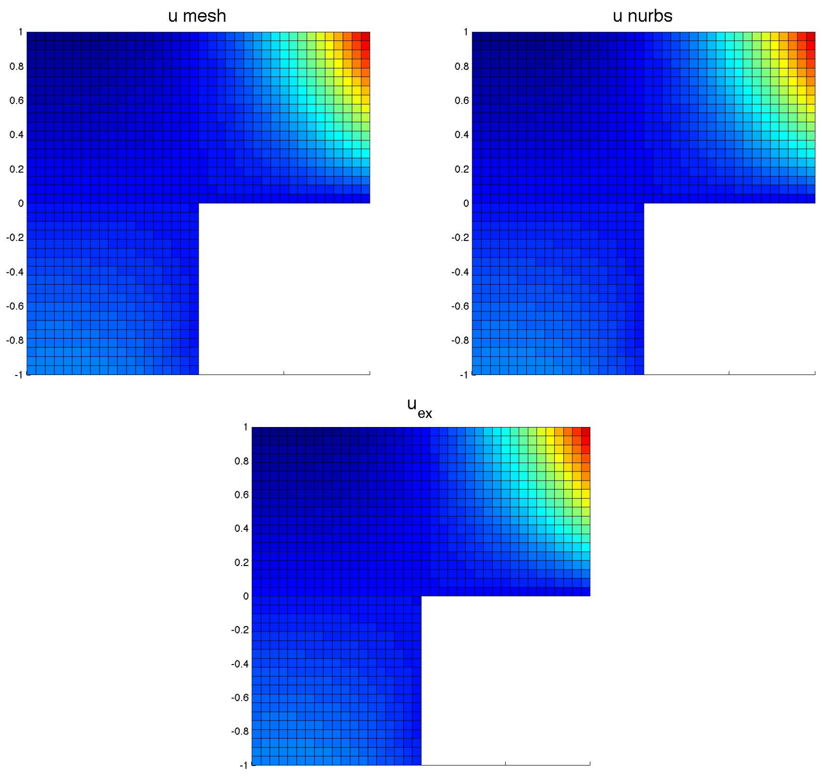


Figure 5.4: Computed solutions for *mesh* (top left) and *nurbs* (top right), and exact solution (bottom).

The number of degrees of freedom $ndof$ is given by the sum of the degrees of freedom on each patch, minus the degrees of freedom along the interfaces $I1$ and $I2$ (5+6=11). Since we applied Dirichlet boundary conditions, we also need to subtract the number of degrees of freedom on the boundaries. Recalling that the dimension of a spline space is given by the sum $m + K$, where m is the order of the splines and K is the sum of the multiplicities of

	$(mesh \to spline2)$	$(nurbs \to spline2)$
$ndof$	26	26
err_{L^2}	8.0781e-03	8.0781e-03
err_{H^1}	7.5858e-02	7.5858e-02
err_∞	2.2469e-02	2.2469e-02
err_2	9.6084e-03	9.6084e-03

Table 5.1: Table of results for *mesh* and *nurbs* physical domains.

the knots, we have that the degrees of freedom are 15 for $P1$, 30 for $P2$ and 18 for $P3$. Thus, the number of degrees of freedom is: $ndof = 63-11-26 = 26$.

5.2 Example 2

In this section we solve the BVP in form (4.7) with the source term given in Eq. (4.9) on a physical domain that, unlike the Γ-shaped domain used in Section 5.1, has curved boundaries.

We used the *mesh* and *nurbs* physical domains shown in Fig. 5.5, and compared the results obtained using *spline*2 as solution space in both cases.

The structure of the B-Rep of this physical domain is the same of the one in Example 1: that is, the three patches share the same two interfaces and six boundaries. Therefore, in order to change only the geometry of each patch, we used the routine `mp_geo_substitute_patches`, introduced in Section 3.4.

We impose that, as in Example 1, the u direction is from left to right and the v direction is from bottom to top in all the three patches. We created patch $P1$ making a ruled surface between a semicircle and a line, patch $P2$ extruding a left-to-right line in a bottom-to-top direction and patch $P3$ making a ruled surface between a bottom-to-top line and a bottom-to-top semicircle. Since the surfaces created by `nrbruled` are ruled along the

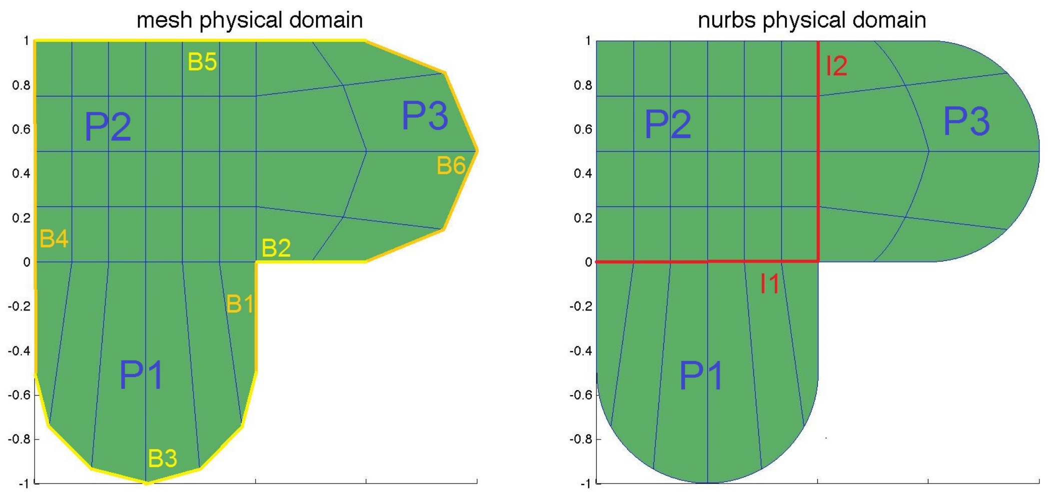


Figure 5.5: Physical domain: *mesh* on the left and *nurbs* on the right. We highlighted the boundaries in the *mesh* image and the interfaces in the *nurbs* image.

v direction, $P3$ had initially the directions swapped with respect to the ones of $P3$ of Section 5.1. In order to get the same orientation and to use `mp_geo_substitute_patches`, we swapped the directions of $P3$ through `nrbtransp`.

As in Section 5.1, the v direction of $P1$ and the u direction of $P3$ are not constrained by the joining and can change without affecting the geometry of the other patches and interfaces. In this case we added an internal knot to the u direction of $P3$.

The approximated solutions of the BVP on this multipatch domain are shown in Fig. 5.6 and th error results are reported in Table 5.2. The solution computed considering the *nurbs* domain gives better results than the one on the *mesh* domain, especially along the curved part of the domain. Fig. 5.7 shows the error function err, defined in Eq. (4.1), for the *mesh* and *nurbs* cases.

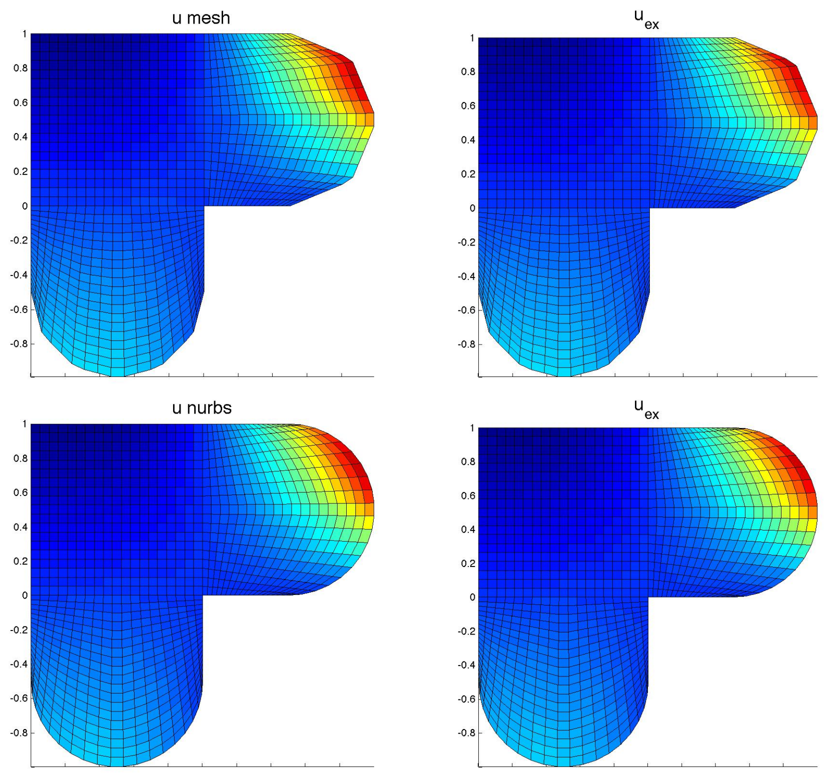


Figure 5.6: Computed solutions for *mesh* (top left) and *nurbs* (bottom left), and exact solution for *mesh* (top right) and *nurbs* (bottom right).

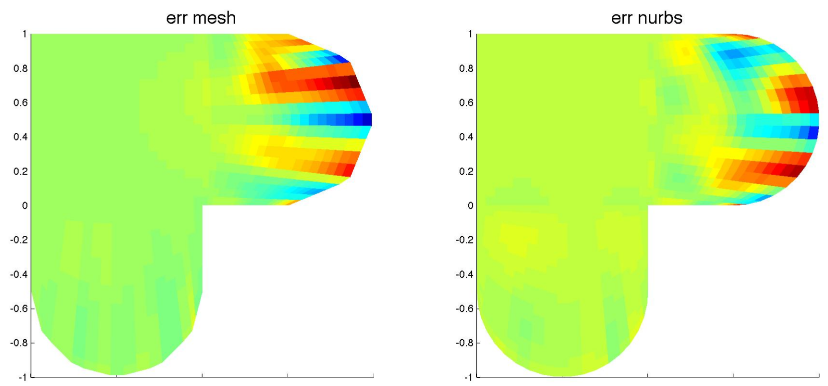


Figure 5.7: Plot of *err*, defined in Eq. (4.1) as the difference between the exact and computed solutions, for the *mesh* (left) and *nurbs* (right) cases.

	mesh	*nurbs*
$ndof$	48	64
err_{L^2}	1.7305e-02	6.6116e-03
err_{H^1}	3.2304e-01	1.6259e-01
err_{∞}	5.9361e-02	3.0097e-02
err_2	1.6338e-02	7.0207e-03

Table 5.2: Table of results for *mesh* and *nurbs* physical domains.

5.3 Example 3

In this section we solve the BVP in form (4.7) with the source term given in Eq. (4.9) on a hybrid physical domain, that is a geometry made by *mesh* and *nurbs* patches joined together, shown in Fig. 5.8.

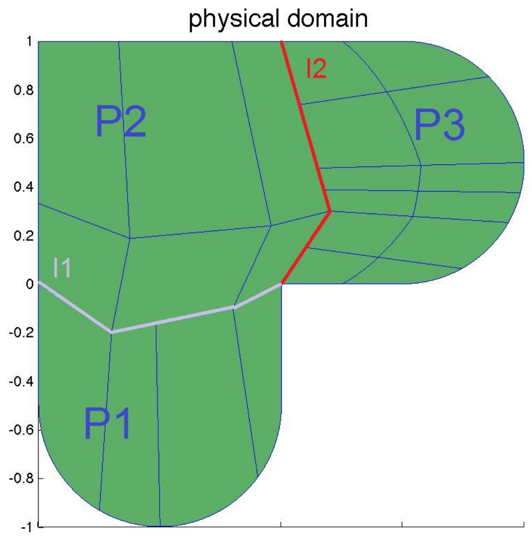


Figure 5.8: Patches and interfaces of the physical domain. $P1$ and $P3$ are *nurbs* patches of degree (2,1) and (2,2) respectively, and $P2$ is a *mesh* patch.

In order to build the geometry in Fig. 5.8 we started from the six curves shown in Fig. 5.9, generated using `nrbline`, `nrbcirc` and `nrbmakepolyline`.

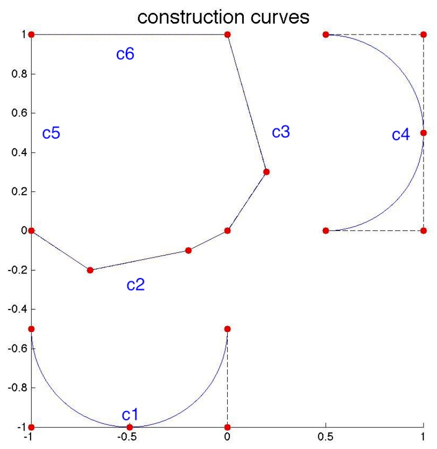


Figure 5.9: Six curves used to build the patches.

The code below creates from the curves three non-compatible patches with two overlapping edges, that are shown in Fig. 5.10:

```
P1 = nrbruled( c1 , c2 ) ;
P2 = nrbcoons(c2, c6, c5, c3);
srf = nrbruled( c3, c4 ) ;
srf= nrbdegelev (srf, [0 1]);
srf = nrbtransp(srf);
[~, ~, knots] = kntrefine (srf.knots, [1 1], [2 2], [1 1]);
P3 = nrbkntins(srf, knots);
```

Patch $P1$ is obtained making a ruled surface between the semicircle $c1$ and the polyline $c2$. Patch $P2$ is a bilinearly blended Coons surface patch created with `nrbcoons` from the curves $c2$, $c6$, $c5$ and $c3$, that define the boundaries. Patch $P3$ is a ruled surface between and the polyline $c3$ and the semicircle $c4$ to whom we applied `nrbtransp` to swap the directions and `nrbkntins` to refine the knot vectors.

From those patches we built three compatible patches using:

```
[P1comp, P2comp1] = nrbmakecompatible(P1,P2, 4, 3,1);
[P3comp, P2comp2] = nrbmakecompatible(P3,P2comp1, 1, 2,1);
```

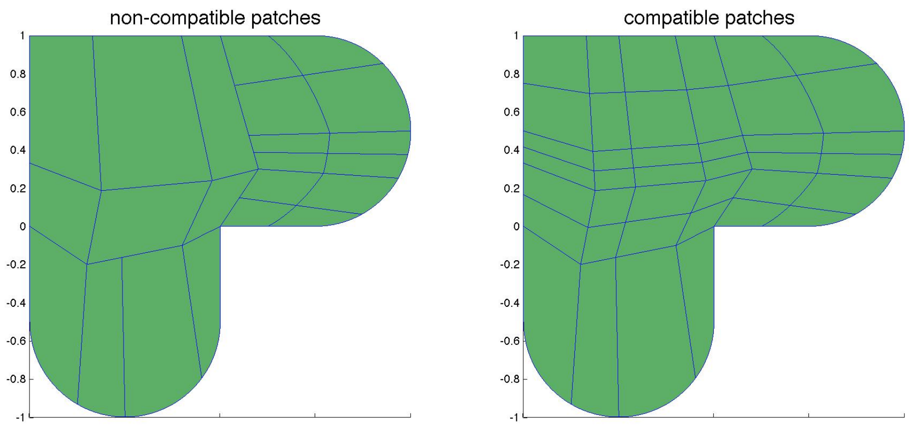


Figure 5.10: Physical domain, before (left) and after (right) applying the compatibility routine.

Then we created a new file `compatibleGeo.txt` for the new hybrid geometry starting from a copy of the `gamma.txt` file containing the physical domain of Example 1:

```
mp_geo_substitute_patches('copy_gamma.txt',...
                 'compatibleGeo.txt', [1 2 3], [P1comp P2comp2 P3comp])
```

The results obtained by solving the BVP on the geometry represented by `compatibleGeo.txt` are listed in Table 5.3 and shown in Fig. 5.11.

$ndof$	94
err_{L^2}	3.7972e-03
err_{H^1}	1.1037e-01
err_{∞}	1.8716e-02
err_2	4.2991e-03

Table 5.3: Table of results.

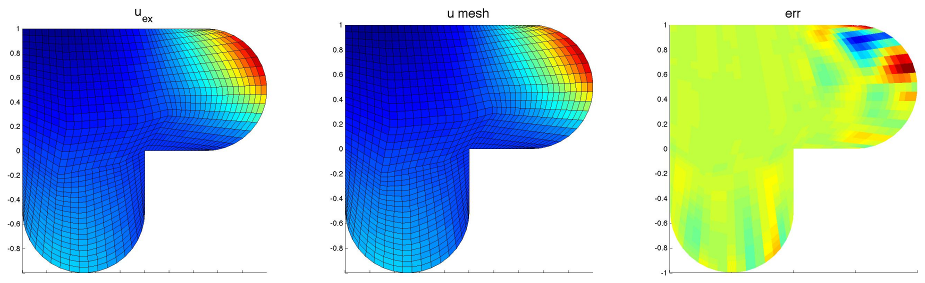


Figure 5.11: Exact solution (left), computed solution (center) and their difference *err* (right) on a hybrid domain.

5.4 Example 4

In general, non-trivial geometries require to apply the compatibility procedure more than once for each interface. This example describes such a scenario.

We solve the BVP in form (4.7) with the source term given in Eq. (4.9) on the physical domain represented in Fig. 5.12.

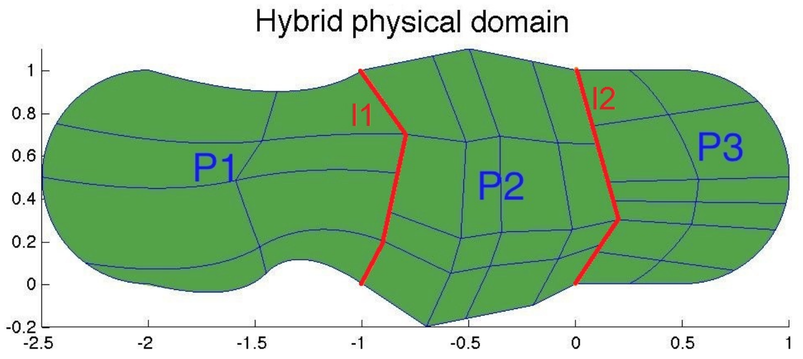


Figure 5.12: Patches and interfaces of the physical domain. $P1$ and $P3$ are both *nurbs* patches of degree (2,2), and $P2$ is a *mesh* patch.

The non compatible patches $P1$, $P2$ and $P3$ that constitute the physical domain are obtained from the nine curves illustrated in Fig 5.13 using the following code:

```
c1 = nrbcirc( 0.5 , [ -2 0.5 0 ] , pi/2 , 3*pi/2) ;
pnts = [-2 -1.3 -1 ;
        1   0.8  1 ;
        0    0   0 ];
c2 = nrbmak(pnts,[0 0 0 1 1 1]);
pnts = [-2 -1.6 -1.3  -1 ;
        0  -0.1  0.2  0 ;
        0   0    0     0 ];
c3 = nrbmak(pnts,[0 0 0 0.5 1 1 1]);
c4 = nrbmakepolyline([[-1,1]; [-0.8,0.7];  [-0.9,0.2]; [-1,0]]);
P1 = nrbcoons( c1, c4, c2, c3);

c5 = nrbreverse(c4,1);
c6 = nrbmakepolyline([[-1 1]; [-0.5 1.1]; [0 1]]) ;
c7 = nrbmakepolyline([[-1,0]; [-0.7,-0.2];  [-0.2,-0.1]; [0,0]]);
c8 = nrbmakepolyline([[0,0]; [0.2, 0.3]; [0,1]]);
P2 = nrbcoons(c7, c6, c5, c8);

c9 = nrbcirc( 0.5 , [ 0.5 0.5 0 ] , -pi/2 , pi/2) ;
srf = nrbruled( c8, c9 ) ;
srf= nrbdegelev (srf, [0 1]);
srf = nrbtransp(srf);
[~, ~, knots] = kntrefine (srf.knots, [1 1], [2 2], [1 1]);
P3 = nrbkntins(srf, knots);
```

The physical domain Ω is thus composed of $P1$ and $P3$ which are *nurbs* patches of degree $(2, 2)$, and $P2$ which is a *mesh* patch (represented by a NURBS with degree (1,1)). This model contains two interfaces:

- $I1$, shared by $P1$ and $P2$. $I1$ is along $c4$ and $c5$, which are overlapping sides of patches with opposite directions. This implies that the

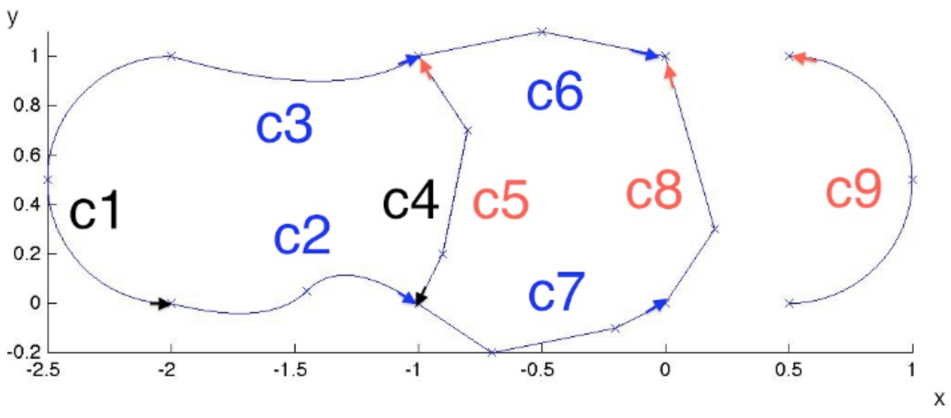


Figure 5.13: Nine curves used to build patches $P1$, $P2$ and $P3$. The arrows indicate the direction of the curves: $c4$ and $c5$ are overlapping with opposite directions. c1, c2, c3 and c4 are used to create $P1$, c5, c6, c7 and c8 are used for $P2$, c8 and c9 for $P3$.

`orientation` flag of $I1$ in the geometry file is 0.

- $I2$ shared by $P2$ and $P3$ along $c8$. Its `orientation` flag in the geometry file is 1.

To get three compatible patches out of the three patches created by the code above, it is not sufficient to apply `nrbmakecompatible` twice like in the previous example. Three calls to `nrbmakecompatible` are necessary:

```
[P1comp, P2comp] = nrbmakecompatible(P1, P2, 4, 1, 0);
[P3comp, P2comp1] = nrbmakecompatible(P3, P2comp, 1, 2, 1);
[P1comp1, P2comp2] = nrbmakecompatible(P1comp, P2comp1, 4, 1, 0);
```

Fig. 5.14 shows the effects on the non-compatible patches (top left) after each application of the routine `nrbmakecompatible`. The first call produces two compatible surfaces `P1comp` and `P2comp`, which are not compatible with `P3` (top right). After the second call, `P2comp1` and `P3comp` are compatible, but not compatible with `P1` (bottom left). The third call makes all the patches compatible (bottom right).

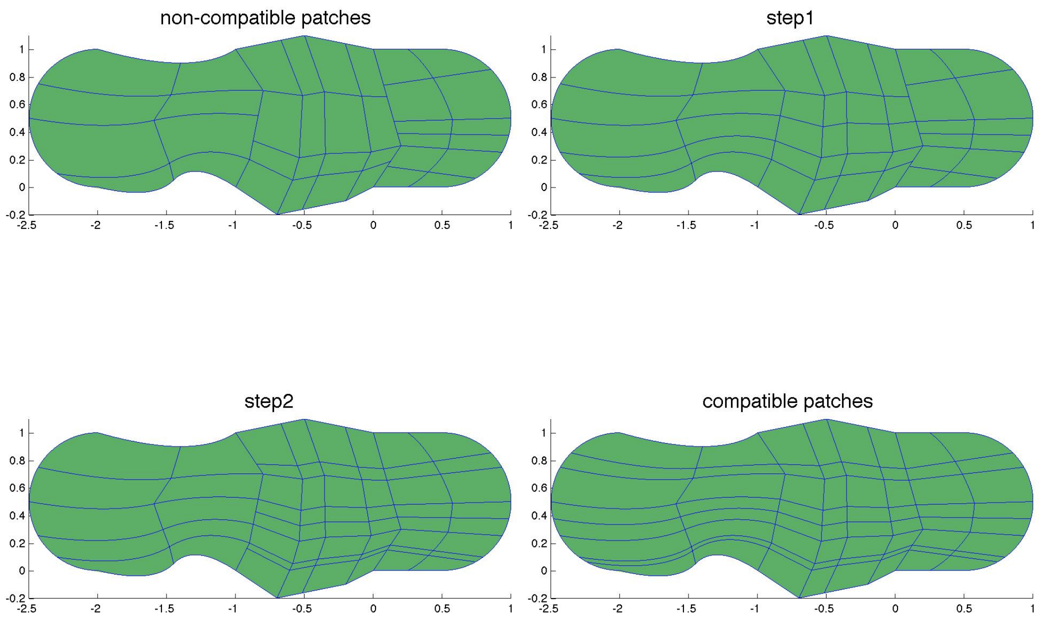


Figure 5.14: Step by step results of multiple applications of `nrbmakecompatible` to provide the compatibility of the three patches.

The obtained results are listed in Table 5.4 and shown in Fig. 5.15.

5.5 Example 5

In this section we will solve a Laplace-Beltrami problem on a multipatch hybrid geometry, representing a quarter of cylindrical surface of unitary radius embedded in $\mathbb{R}^3$.

The physical domain is composed of three patches:

- $P1$ is a *nurbs* surface of degree (2, 1) which describes a cylindrical geometry exactly.

$ndof$	120
err_{L^2}	3.5305e-03
err_{H^1}	9.0607e-02
err_∞	1.4286e-02
err_2	3.6923e-03

Table 5.4: Table of results.

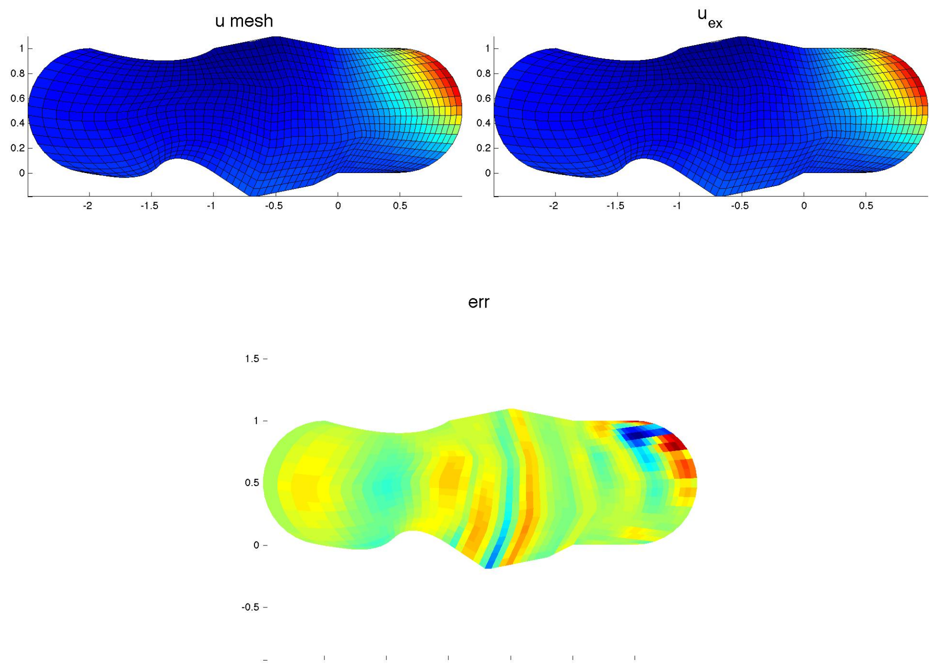


Figure 5.15: Computed solution (top left), exact solution (top right) and their difference err (bottom) on a hybrid domain.

- $P2$ is a narrow blending surface of degree (2, 1) that connects $P1$ and $P3$.
- $P3$ is a *mesh* patch. It has degree (2, 1) because of the compatibility on the interface, but all the knots along the u direction are repeated two times, thus $P3$ is geometrically a quad mesh.

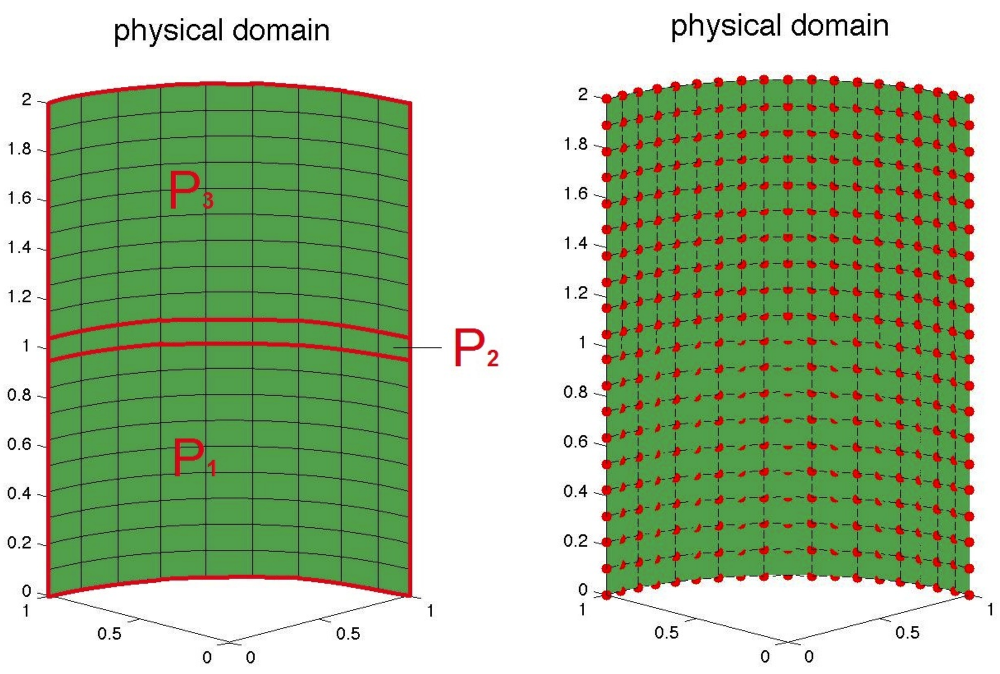


Figure 5.16: Physical domain represented by hybrid multipatch geometry.

The elements grids are 9×1 for $P2$ and 9×9 for $P1$ and $P3$.

The source function of this Laplace-Beltrami problem is:

$$f(x,y,z) = \beta\left(\left(\frac{\alpha\pi}{L}\right)^2 g_1(x,y) - g_2(x,y)\right) g_3(z), \qquad (5.2)$$

where

- $g_1(x,y) = (1-\cos(\arctan\left(\frac{x}{y}\right)))(1-\sin(\arctan\left(\frac{x}{y}\right))),$
- $g_2(x,y) = \cos(\arctan\left(\frac{x}{y}\right))+\sin(\arctan\left(\frac{x}{y}\right))-4\cos(\arctan\left(\frac{x}{y}\right))\sin(\arctan\left(\frac{x}{y}\right)),$

- $g_3(z) = \sin(\left(\frac{\alpha\pi}{L}\right) z)$

- $\alpha = 5$, $\beta = 1$ and $L = 1$.

As a basis for the solution space, we adopted B-spline functions of degree 2. The exact solution is:

$$u(x, y, z) = \beta g_1(x, y) g_3(z), \tag{5.3}$$

and it is displayed in Fig. 5.17, together with the computed solution.

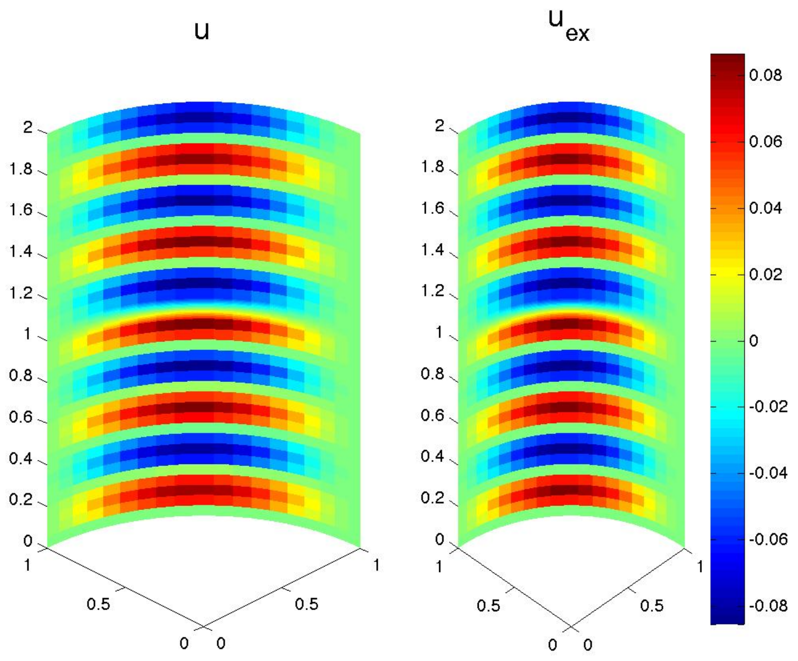


Figure 5.17: Computed (left) and exact (right) solutions for the Laplace-Betrami problem.

Fig. 5.18 shows the error $err := u_{ex} - u$ and Tables 5.6 and 5.5 show all the other errors.

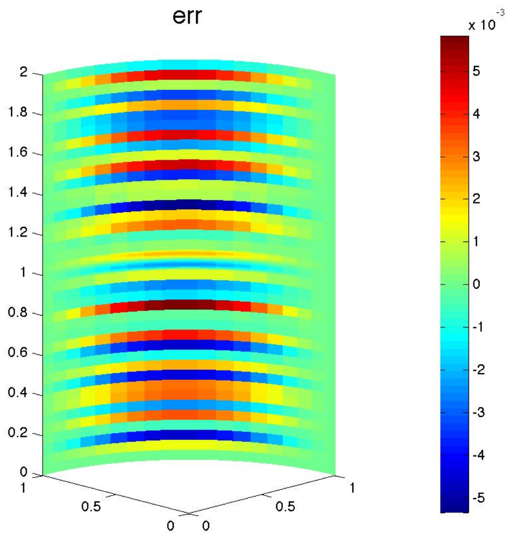


Figure 5.18: Plot of *err* for the Laplace-Betrami problem.

$patch$	err_{L^2}	err_{H^1}	err_∞	err_2
1	2.2124e-03	9.9587e-02	5.8165e-03	1.7789e-03
2	3.5514e-04	2.9490e-02	2.3178e-03	1.0061e-03
3	2.2092e-03	9.9671e-02	5.3368e-03	1.7799e-03

Table 5.5: Table of results on each patch.

$ndof$	357
err_{L^2}	3.1467e-03
err_{H^1}	1.4395e-01
err_∞	8.2271e-03
err_2	2.7102e-03

Table 5.6: Table of results.

5.6 Example 6

We solve the same Laplace-Beltrami problem of Example 5, on a multi-patch hybrid geometry, composed by an open free-form surface patch and a mesh patch.

The *nurbs* patch is a NURBS surface of degree (3,3) generated as a bi-linearly blended Coons surface patch from the four NURBS curves shown in Fig. 5.19, that define its boundary. More information on Coons surfaces can be found in [47].

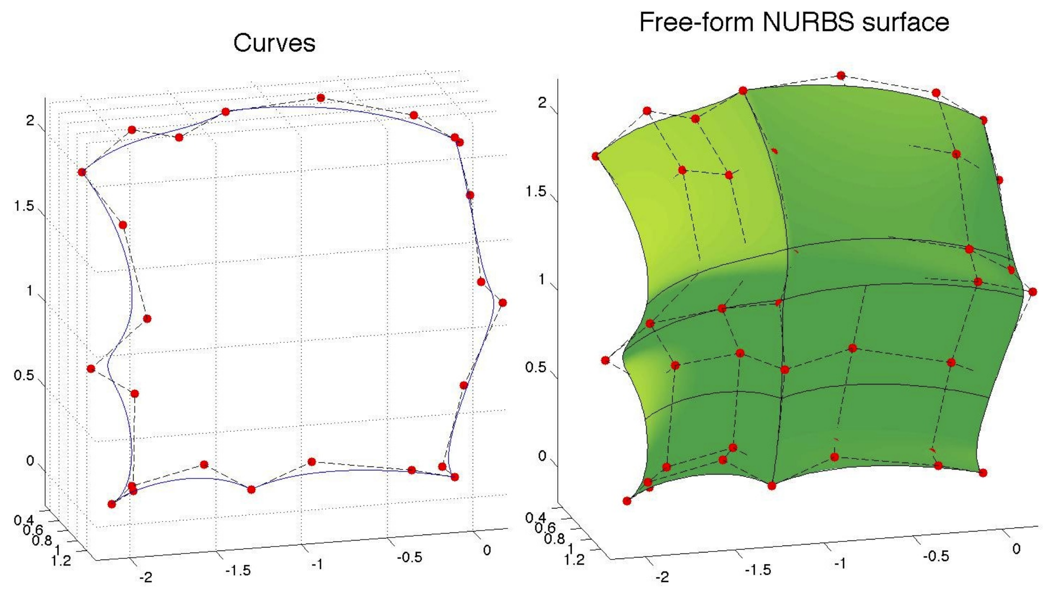


Figure 5.19: Curves (left) used to build a Coons surface (right).

The mesh patch has only three points of the boundary in common with the boundary of the surface, as shown in Fig. 5.20.

To apply IGA, the patches need to be coincident and compatible along the interfaces. In this case, instead of creating a third patch to join the two existing ones (like the blending surface of Example 5), we slightly changed the geometry of the *nurbs* patch to create a common interface. To do so, we elevated the degree of the *mesh* patch and translated the control points

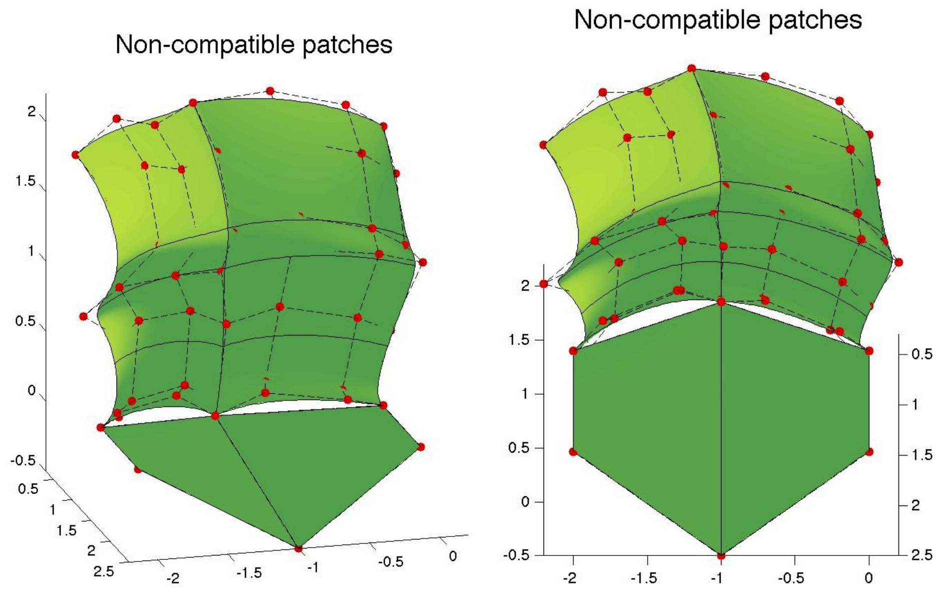


Figure 5.20: Two views of the hybrid multipatch geometry: the *mesh* and *nurbs* patches are not compatible.

along the boundary of the *nurbs* patch in order to make them coincident with the control points along the boundary of the *mesh* patch. This results in two compatible patches shown in Fig. 5.21.

We performed knot insertions in each knot span, in order to have a 10×10 elements grid on each Bézier patch, as in Fig. 5.22.

We applied IGA on the hybrid physical domain in Fig. 5.22, imposing homogeneous Dirichlet boundary conditions.

The result of the analysis is displayed in Fig. 5.23. Since we don't have an exact solution on this geometry, we cannot provide an analysis of the error similar to the ones carried out in the previous examples.

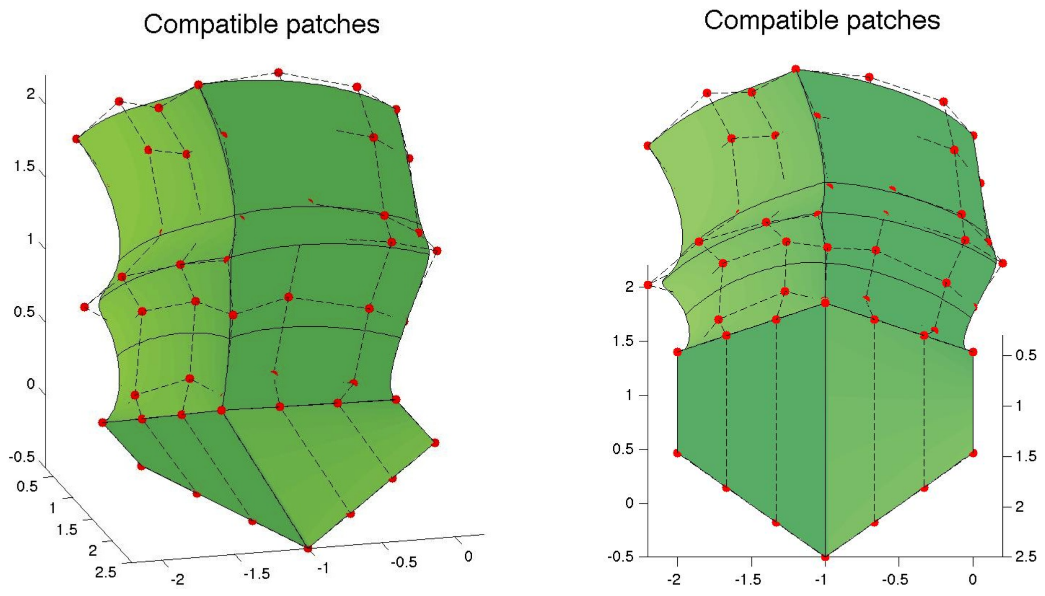


Figure 5.21: Two views of the hybrid multipatch geometry: the *mesh* and *nurbs* patches are compatible.

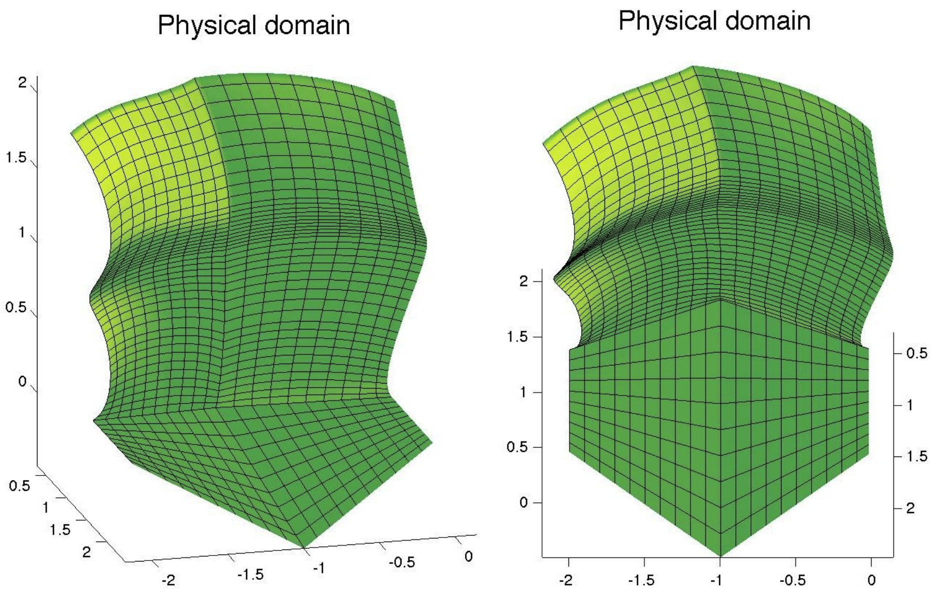


Figure 5.22: Two views of the refined hybrid multipatch physical domain.

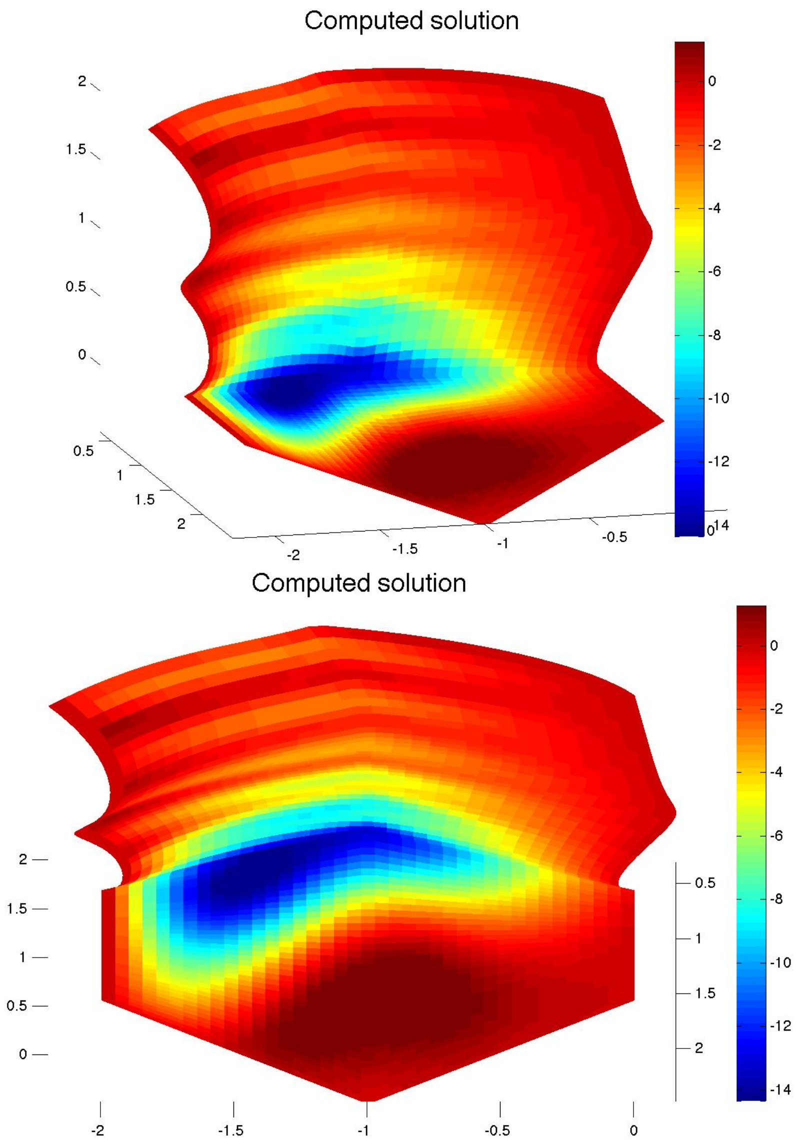


Figure 5.23: Two views of the result of IGA applied to the hybrid multipatch physical domain.

5.7 Example 7

In this section we solve a more realistic example of the application of IGA on a multipatch hybrid geometry. The BVP is the same Laplace-Beltrami problem of Example 5, with different values of the constants, namely $\alpha = 5$, $\beta = 1$ and $L = 10$.

The geometry on which we applied the problem is composed by a lid represented by an open free-form NURBS surface, with a knob represented by a mesh patch. Supposing that the geometry has an axis of symmetry, we solved the problem on half of the model, which is a technique used often in the analysis of symmetric models.

The mesh patch representing one half of the knob is shown in Fig. 5.24.

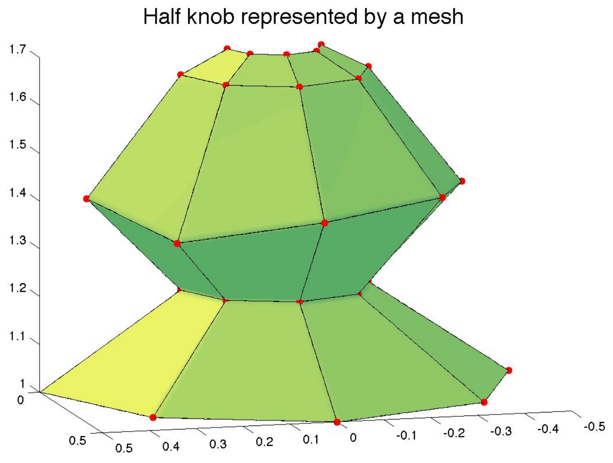


Figure 5.24: The *mesh* patch representing one half of the knob.

The *nurbs* patch is a NURBS surface of degree (3,3) generated by the revolution of a degree 3 NURBS curve around the Z axis. The two patches are not compatible and the mesh patch has only three points of the boundary in common with the boundary of the surface, as shown in Fig. 5.25.

To make the patches compatible in order to apply IGA, we slightly changed the geometry of the *nurbs* patch to create a common interface. To do so, we elevated the degree of the *mesh* patch and translated the control

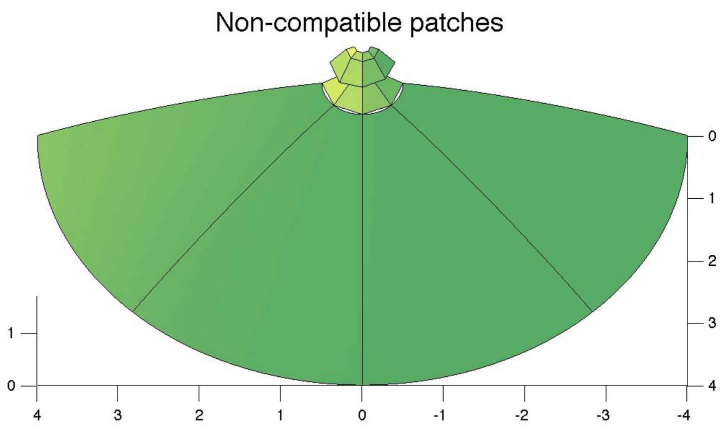


Figure 5.25: The *mesh* and *nurbs* patches are not compatible.

points along the boundary of the *nurbs* patch in order to make them coincident with the control points along the boundary of the *mesh* patch. This results in two compatible patches shown in Fig. 5.26.

We applied IGA on the hybrid physical domain in Fig. 5.26, imposing homogeneous Dirichlet boundary conditions.

The result of the analysis is displayed in Fig. 5.27. Since we don't have an exact solution on this geometry, we cannot provide an analysis of the error similar to the ones carried out in the previous examples.

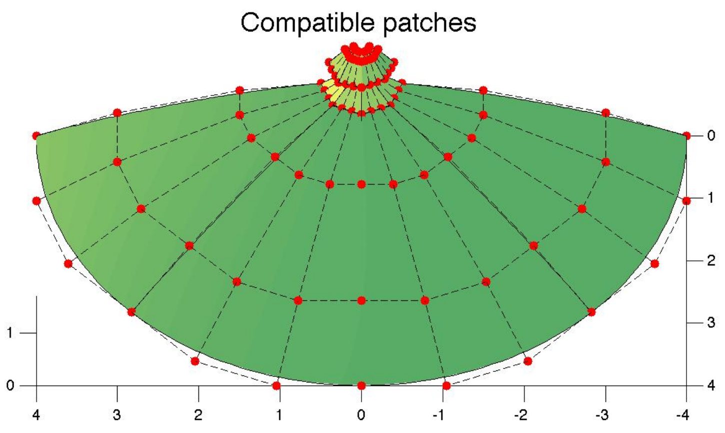


Figure 5.26: The *mesh* and *nurbs* patches are made compatible.

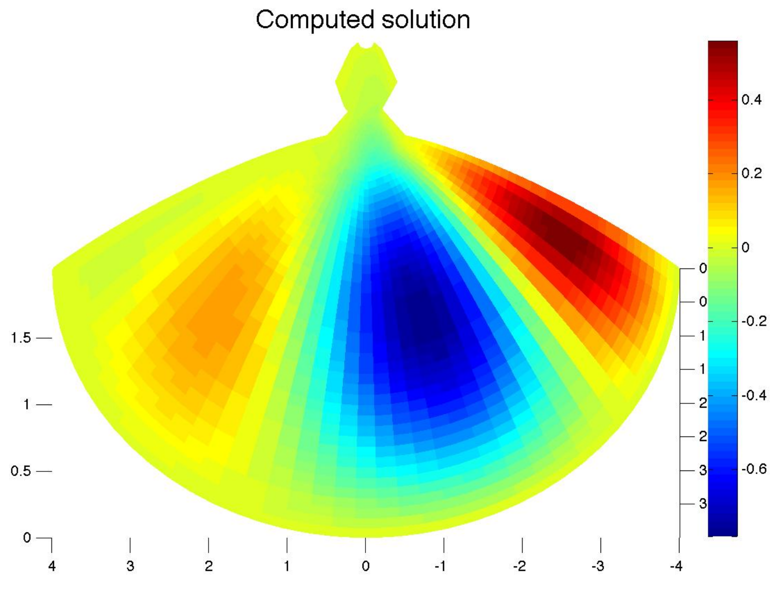


Figure 5.27: The result of IGA applied to the hybrid multipatch physical domain.

Chapter 6

CAD and FEM at devDept Software

Since January 2012 I have been working at devDept Software, a software house founded in 2006. devDept Software's flagship product, Eyeshot, is a 3D graphics, CAD and FEM control for .NET Framework.

With different product editions, Eyeshot allows to use Mesh, Solid, and NURBS surface modeling technologies. Mesh and NURBS surfaces can co-exist in the same model as individual entities. Eyeshot's library does not have hybrid functions to make mesh and NURBS entities interact with each other, nor a structure that can contain both kind of entities to represent an extended B-rep like the ones introduced in Section 1.8. Therefore, users typically adopt either mesh or NURBS modeling, depending on their specific needs.

Eyeshot FEM edition performs classical FEA to solve linear elasticity problems, introduced in Section 2.3.

During these years at devDept Software, I contributed to the growth of Eyeshot NURBS and FEM libraries by designing and implementing new features while enhancing pre-existing ones. I also reinforced the unit testing system and handled support requests submitted by Eyeshot's users, that are programmers from across the world.

Listing all my contributions to the library would be a difficult (and probably pointless) task. This chapter presents an overview on some of the features and algorithms to which I dedicated my time at work, in the past years during the Ph.D. program.

6.1 Eyeshot CAD

The CAD and NURBS library is the part of Eyeshot in which, being a mathematician, I obtained the best results and that I found more interesting. In the following sections we will see how curves and surfaces can be represented in Eyeshot, and some fundamental methods to manipulate them.

6.1.1 Geometry representations in Eyeshot

In Eyeshot there are different ways to define curves and surfaces [25].

The interface `ICurve` groups all the properties and methods of the classes of entities that represent curves, which are:

- `Point`,
- `Line`,
- `Circle`, with the derived class `Arc`,
- `Ellipse`, with the derived class `EllipticalArc`,
- `Curve`, that defines a NURBS curve,
- `LinearPath`, that defines a polyline,
- `CompositeCurve`, that groups a set of subsequent ICurves.

These classes share many properties (domain, start and end points, start and end tangent) and methods, such as `PointAt()` to evaluate a curve at a parameter value, `Project()` to find the orthogonal projection of a 3D point on a curve, or `TrimAt()` that trims a curve at a parameter value.

Even if all these entities could be described using NURBS curves, keeping them in separated classes is very helpful since some algorithms, such as the evaluation or the projection of a point, are quicker and more precise if implemented for special curves than for the more general NURBS curves. It is always possible to convert a special curve into a NURBS curve using the method `GetNurbsForm()`.

Analogously, there are different classes of surface entities. They all derive from the class of NURBS surfaces called `Surface` and they are:

- `PlanarSurface`,
- `TabulatedSurface`, that defines surfaces obtained by extruding a curve called `Directrix` along a vector called `Generatrix`,
- `RevolvedSurface`, that defines surfaces obtained by revolving a curve called `Generatrix` around an `Axis`. It has the following derived classes:
  - `SphericalSurface`,
  - `ToroidalSurface`,
  - `ConicalSurface`,
  - `CylindricalSurface`.

A diagram of the `Surface` class and its dependencies is shown in Fig. 6.1.

Also for special surfaces, it is possible to obtain the generic NURBS form using the method `GetGeneric()`. The opposite operation, i.e. getting a special curve or surface from a NURBS one, is not always possible and it is performed by the method `Promote()`.

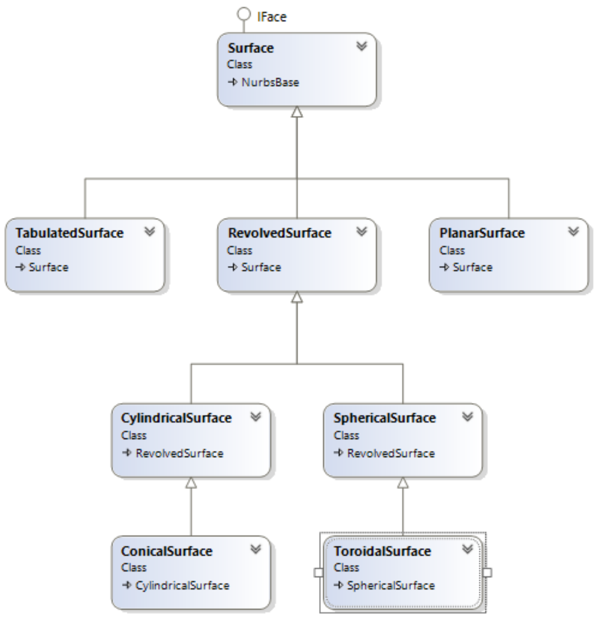


Figure 6.1: Class diagram of Eyeshot's surface classes.

To represent a single surface in Eyeshot, a user can also employ the mesh representation. The `Mesh` entities are triangular meshes that can be defined through an array of 3D points called `Vertices` and an array of indices called `Triangles`, that defines the triangles of the mesh pointing to the array of its vertices.

A solid model composed of many NURBS surface patches can be represented as a B-rep using the entity `Solid3D`.

Eyeshot does not have a structure to represent extended B-reps made of both `Mesh` and `Surface` entities, introduced in Section 1.8. Since Eyeshot is not a hybrid system, the CAD libraries for `Mesh` and `Surface` entities are completely independent. The library for surfaces is richer: some methods, such as intersection, offset and fillet, can be performed among surfaces, but not among meshes.

6.1.2 Point projection and inversion on curves and surfaces

Projecting a 3D point P on an entity e (that can be a curve denoted by C or a surface denoted by S) is the process of finding the closest point Q on e such that the segment connecting P and Q is perpendicular to e at Q.

Point inversion is applied to a point P that is assumed to be on e to find the parameter of the domain of e corresponding to P. For a curve C the domain is an interval $D = [u_0, u_1] \subset \mathbb{R}$, and the output of point inversion is the parameter value $u \in D$ such that $C(u) = P$. For a surface S the domain is $D_U \times D_V = [u_0, u_1] \times [v_0, v_1] \subset \mathbb{R}^2$, and the output of point inversion is a parametric pair $(u, v) \in D_U \times D_V$ such that $S(u, v) = P$.

Point projection and point inversion are important processes in geometric modeling and computer graphics, they are widely used by more complex algorithms and they have numerous applications in CAD related topics. In Eyeshot both operations can be performed by the method `Project()`.

For instance, when a user needs to interactively select one entity from among several others represented on screen, he/she clicks somewhere near that entity, and the system finds out which entity has been selected by projecting the clicked point on all the entities and choosing the entity closest to the point. Other contexts in which point projection and inversion are fundamental are fitting and intersection problems, reconstructing curves, collision detection and shape registration [10, 39, 46].

As pointed out in [35], point projection algorithms experience stability and performance issues, and there still is no perfect algorithm that satisfies high accuracy, robustness and efficient computation time simultaneously.

6.1.2.1 Point projection for curves

Our algorithm uses Newton iterative method to find the parameter value u such that minimizes the distance between P and $C(u)$. If this distance is smaller than a defined tolerance `tol1`, we consider P to be on C.

Newton's method, also known as Newton-Raphson method, aims to find successively better approximations to the roots of a real-valued equation or system of equations. In our case, $C(u)$ is an orthogonal projection of P on C if u is a zero of the dot product function:

$$f(u) = C'(u) \cdot (C(u) - P). \tag{6.1}$$

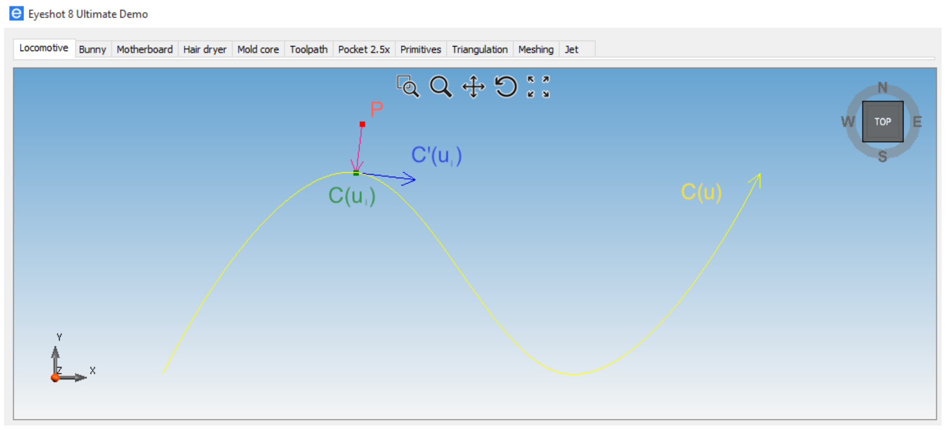


Figure 6.2: Projection of the point P on the curve C: the vectors $C'(u)$ and $(C(u) - P)$ are perpendicular at the orthogonal projection.

The method needs a start value u_0 and if we denote by u_i the parameter obtained at the ith Newton iteration, the $(i+1)$th iteration gives:

$$u_{i+1} = u_i - \frac{f(u_i)}{f'(u_i)} = u_i - \frac{C'(u_i) \cdot (C(u_i) - P)}{C''(u_i) \cdot (C(u_i) - P) + |C'(u_i)|^2}. \tag{6.2}$$

The iteration method stops if one of the following convergence criteria is met:

- point coincidence, that can only happen if P is on C:

$$|(C(u_i) - P)| \leq \texttt{tol1}, \tag{6.3}$$

- the cosine between the tangent and the vector $C(u_i) - P$ is zero (within a small tolerance `tol2`):

$$\frac{|C'(u_i) \cdot (C(u_i) - P)|}{|(C(u_i) - P)||C'(u_i)|} \leq \texttt{tol2}, \tag{6.4}$$

- the parameter does not change significantly:

$$|(u_{i+1} - u_i)C'(u_i)| \leq \texttt{tol1}. \tag{6.5}$$

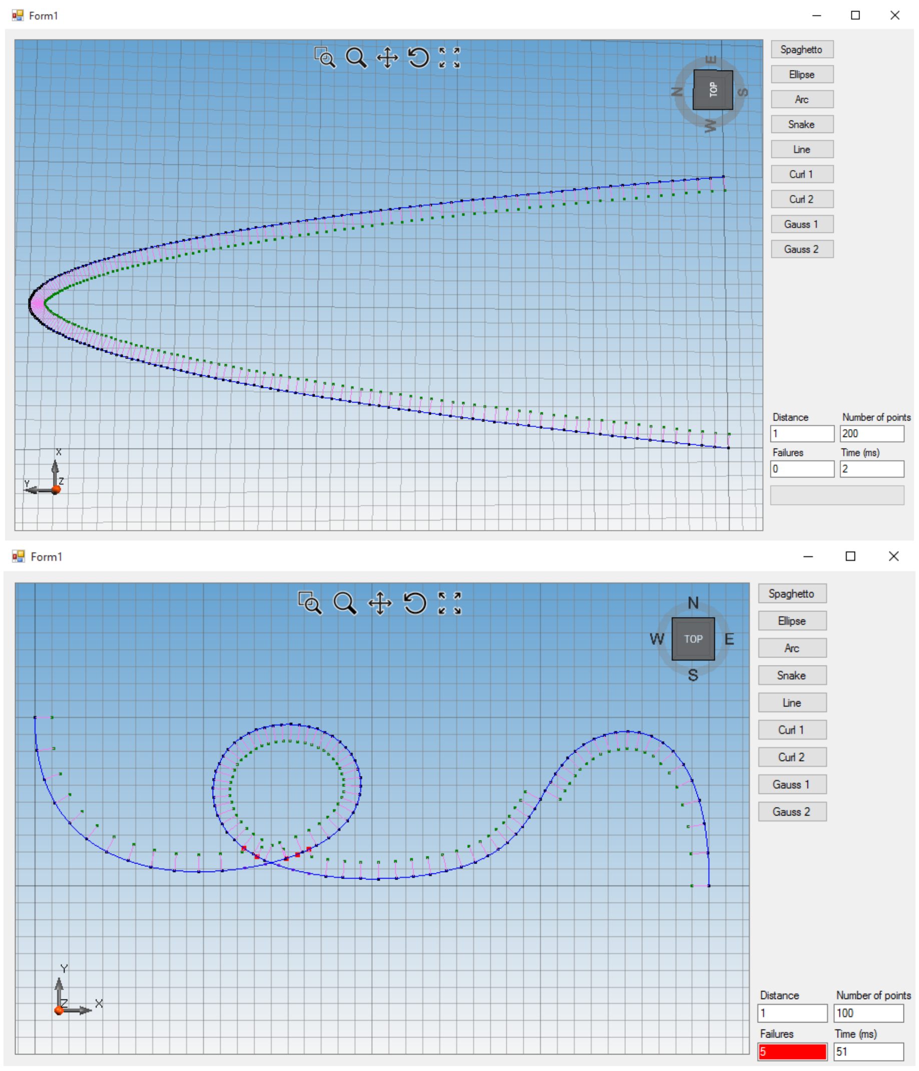
The success of the method heavily depends on the initial value, and choosing a good initial value is a fundamental but not easy problem. As suggested in [47], we solve this problem by repeating the Newton iteration starting from different initial values. In particular, we decompose the curve C in its Bézier patches and perform the iteration starting from five equally spaced initial points on each patch. Then we return as result the parameter corresponding to the closest orthogonal projection.

Special algorithms are implemented for ICurves such as Line, Circle, Arc, Ellipse and EllipticalArc.

To test our method, we built an application that evaluates the normal vectors $(n_1, \ldots, n_m)$ to C at m different parameters $(u_1, \ldots, u_m)$, and given a distance d, it computes the points $(C(u_1) + dn_1, \ldots, C(u_m) + dn_m)$ and projects them back on the curve C. We can check the result of the projection method by checking if the return parameters are within tolerance from $(u_1, \ldots, u_m)$. Some screenshots of this application are shown in Fig. 6.3.

6.1.2.2 Point projection for surfaces

Point projection and inversion for surfaces are analogous. Given a 3D point P, we want to find a point (u, v) in the parametric space such that the segment between $S(u, v)$ and P is perpendicular to S, or has zero length in the case of point inversion.



Distance 1
Number of points 200
Failures 0
Time (ms) 2

Distance 1
Number of points 100
Failures 5
Time (ms) 51

Figure 6.3: Eyeshot's application to test point projection on curves. The user can decide for each curve the number of points to project and the distance of the points form the curve. The points are drawn in green, and they are connected to the projections (black points) by a pink segment. If one point is not projected at the parameter used to compute the normal, a red point is drawn, and the failure count is incremented. In the bottom image, the application reports five failures, but looking at the picture we see that the red points are all close to the curve's self-intersection point, and that the projections are all correct.

Defining the vector function:

$$r(u,v) = S(u,v) - P, \tag{6.6}$$

we use the Newton's method to find the zeros of the non-linear system of equations:

$$\begin{cases} f(u,v) = r(u,v) \cdot S_u(u,v) = 0 \\ g(u,v) = r(u,v) \cdot S_u(u,v) = 0. \end{cases} \tag{6.7}$$

Defining:

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

$$J_i = \begin{bmatrix} f_u & f_v \\ g_u & g_v \end{bmatrix} = \begin{bmatrix} |S_u|^2 + r \cdot S_{uu} & S_u \cdot S_v + r \cdot S_{uv} \\ S_u \cdot S_v + r \cdot S_{vu} & |S_v|^2 + r \cdot S_{vv} \end{bmatrix}$$

and

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

at the ith iteration we must solve the following linear system of equations in the unknown δ_i

$$J_i \delta_i = k_i. \tag{6.8}$$

From the solution of the system, we get the $(i+1)$th iteration as:

$$\begin{aligned} u_{i+1} &= \Delta u + u_i \\ v_{i+1} &= \Delta v + v_i. \end{aligned} \tag{6.9}$$

The convergence criteria are:

- point coincidence, that can only happen if P is on C:

$$|(S(u_i, v_i) - P)| \leq \texttt{tol1}, \tag{6.10}$$

- the cosine between the tangents and the vector $S(u_i, v_i) - P$ are zero (within a small tolerance `tol2`):

$$\frac{|S_u(u_i, v_i) \cdot (S(u_i, v_i) - P)|}{|(S(u_i, v_i) - P)||S_u(u_i, v_i)|} \leq \texttt{tol2}, \tag{6.11}$$

$$\frac{|S_v(u_i, v_i) \cdot (S(u_i, v_i) - P)|}{|(S(u_i, v_i) - P)||S_v(u_i, v_i)|} \leq \texttt{tol2}, \tag{6.12}$$

- the parameters do not change significantly:

$$|(u_{i+1} - u_i)S_u(u_i, v_i) + (v_{i+1} - v_i)S_v(u_i, v_i)| \leq \texttt{tol1}. \tag{6.13}$$

Also for surfaces, we used to apply the decomposition in Bézier patches and perform the iteration starting from a grid of 3×3 equally spaced initial points on each patch. This was not sufficient to have good results on some of the surfaces that we tested, so we tried to increase the number of initial points on each patch. But this slowed down the process dramatically and did not guarantee an accurate result on all surfaces.

Then we proposed to iteratively subdivide the patches into two children and perform the Newton iteration on each child. The iteration stops if the result is equal to the one obtained on the parent patch or if we reached the heuristically determined maximum of eighth level of subdivision. In the end we compare the results and return the pair of parameters that corresponds to the closest orthogonal projection.

To reach this level of accuracy, the method became really slow. In order to avoid applying this method unless it is strictly necessary, we implemented special projection methods for special surfaces: exploiting the geometric properties of planar, tabulated and revolved surfaces we obtained quick and reliable algorithms. In addition, to speed up the method on generic surfaces, we tested some criteria to eliminate a patch: we looked for conditions to exclude the possibility that a patch or one of its children could contain a perpendicular projection, if a patch satisfies such conditions we stop subdividing it and avoid to perform many Newton's iterations.

We adopted the criterium proposed by Selimovic in [52], which, combined with a criterion that uses the bounding box of each patch, gives us satisfactory results in terms of both accuracy and computation time.

Fig. 6.4 and 6.5 show some images of the results obtained by an application to test projections analogous to the one for curves. It allows us to check results of point projection and inversion on surfaces.

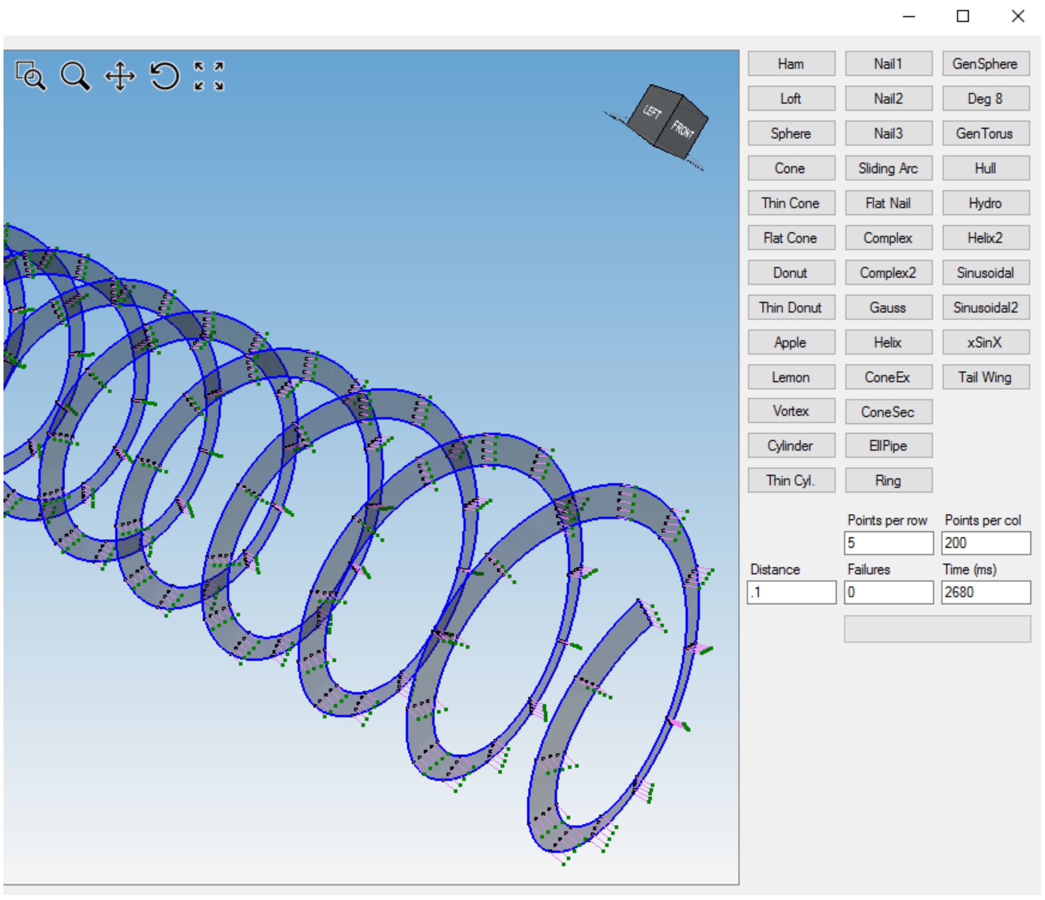
6.1.3 Surface-Surface Intersection and Surface Section

The problem of computing the intersection curves between two surfaces is essential in various CAD, computer graphics, and geometric modeling applications. This problem is often referred to as SSI, which stands for surface-surface intersection. It is considered to be among the basic but difficult problems in CAD, and hence it has been studied extensively to obtain accurate and efficient solutions [2, 36, 42]. Fig. 6.6 shows a typical SSI case.

Eyeshot's algorithm for SSI is a marching method: each intersection curve is obtained by stepping from a given point of the curve in the direction of tangent vectors, collecting a sequence of points on the curve and interpolating them at the end of the process.

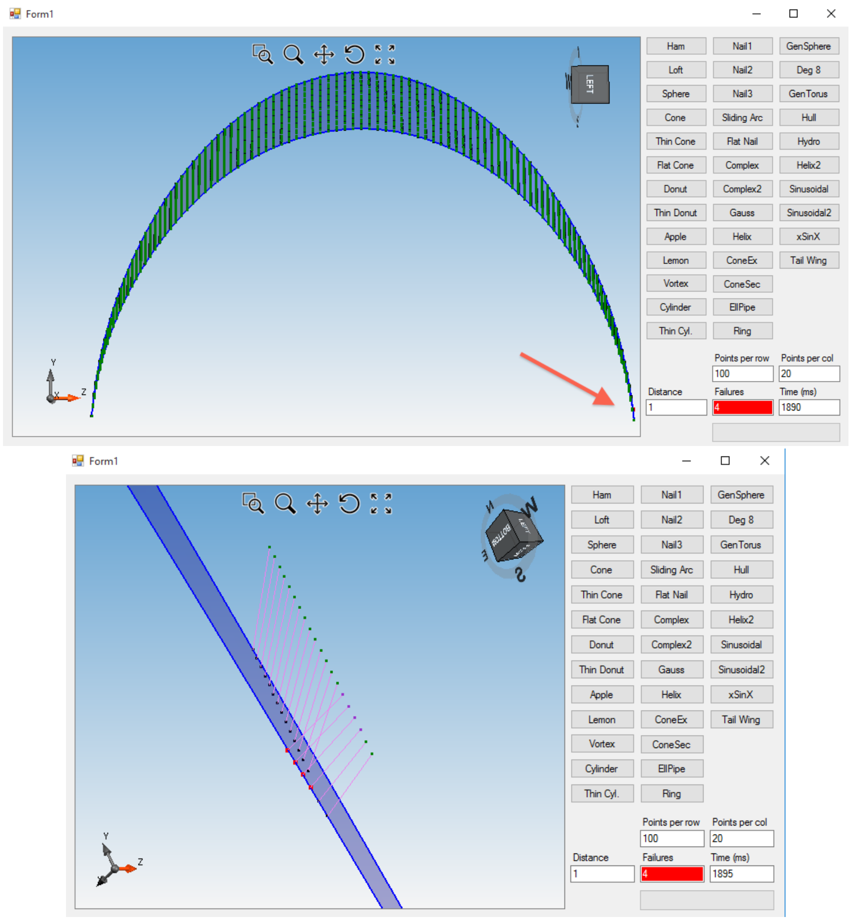
In the following, we will describe Eyeshot's procedure to find the intersection of two surfaces F and G.

The first step of the algorithm reparameterizes the two surfaces changing their 2D parametric domains. We sum the distances between the control points in each row and column of the control lattice, and resize the 2D domain so that the dimension in the u direction is equal to the length of the longest row of control legs, and in the v direction to the length of the longest column. This reparameterization aims to avoid having surfaces with a 2D domain not proportioned with with the 3D dimensions of the surface, which can lead to problems, especially if the intersection process is followed by the trimming of the surfaces.



Points per row: 5
Points per col: 200
Distance: .1
Failures: 0
Time (ms): 2680

Figure 6.4: Eyeshot's application to test point projection on surfaces applied on a ruled surface between two helices. The user can decide for each surface the number of points to project along the u and v directions, and the distance of the points form the surface. The points to project are drawn in green, and they are connected to the projections (black points) by a pink segment. If one point is not projected on the 2D point used to compute the normal, a red point is drawn, and the failure count is incremented.



Points per row 100
Points per col 20
Distance 1
Failures 4
Time (ms) 1890

Points per row 100
Points per col 20
Distance 1
Failures 4
Time (ms) 1895

Figure 6.5: Eyeshot's application to test point projection on surfaces applied on a ruled surface between two curves with the same start and end points. On top, an arrow indicates the part of the surface in which our projection method still has some projection problems. There are four wrong results, displayed in the zoomed bottom image.

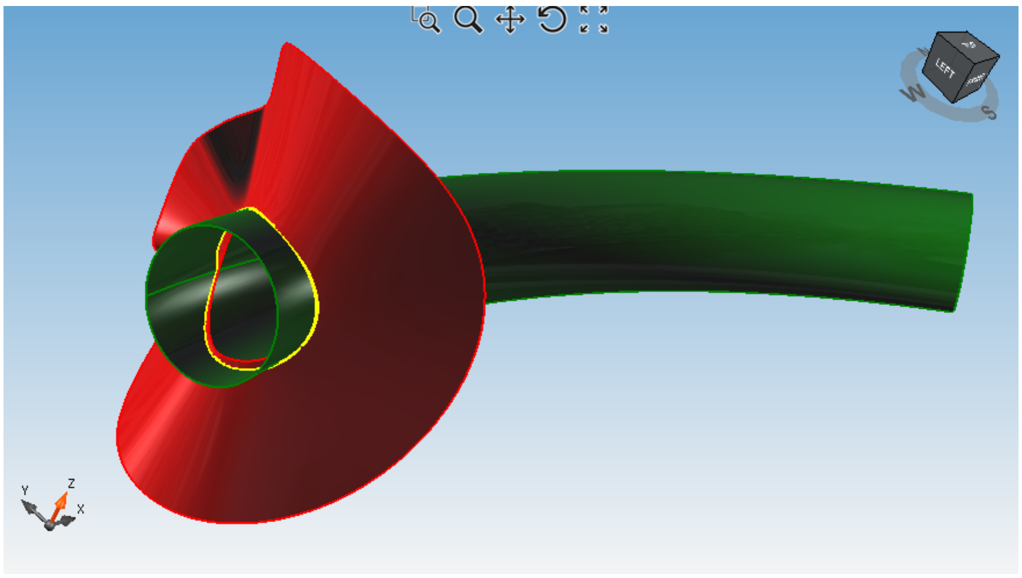


Figure 6.6: Surface-surface intersection between a green toroidal pipe and a red ruled surface. The intersection curve is depicted in yellow.

Then, we find the initial points of the intersection curves. To do so, we intersect all the boundaries of F with G and all the boundaries of G with F. The initial points (and also the marching points) can be approximated points that don't lie on either surface. For this reason, these points go through a process called point refinement, that relaxes the points onto the intersection curve using the Newton-Raphson iteration. More details on this can be found in [2].

Starting from the initial point P_0, successive intersection points on the curve can be generated by stepping in the direction prescribed by the local differential geometry of the curve. Assuming that F and G are not tangent at P_0, the step direction is given by the cross product of the normals of F and G at P_0.

It is also necessary to determine a step length, that represents the distance of each marching point from the previous one along the step vector. This decision is critical: incorrect step size can lead to wrong intersection curves or non-converging point refinement if the step size is too big, and to endless looping or less efficiency if it is too small. Eyeshot uses an adaptive method to compute the step length using curvature estimates and an angular tolerance. The initial step length can be reduced during the tracing.

Every time a new marching point is obtained, we use the segment connecting it to the previous point to check if we passed closed to another initial point. If so, the marching stops and the last point is substituted by the initial point. We interpolate all the points of the sequence to obtain the intersection curve, and if there are other initial points, we restart the marching method from them.

The point refinement process, in addition to providing a 3D point that lies on the intersection curve, gives also the 2D coordinates of the intersection point on each surface. So, if one or both the surfaces need to be trimmed at the intersection, we can interpolate the sequence of 2D points to get the parametric trim curve corresponding to the 3D intersection and use it to cut the surface. At the end of the process, the original parameterization of the surfaces is restored. In Fig. 6.7, the surfaces of Fig.6.6 are trimmed by the intersection curve.

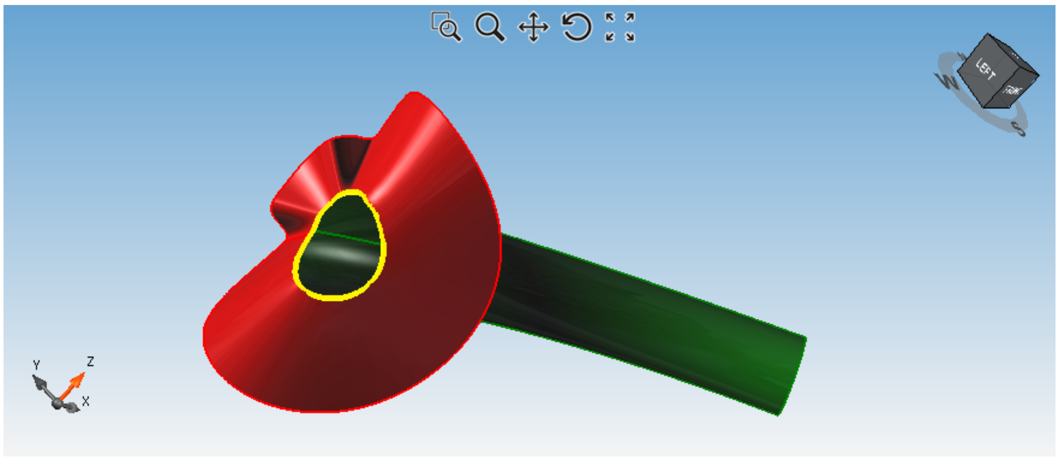


Figure 6.7: The surfaces of Fig.6.6 are trimmed by the intersection curve.

Like many other tracing algorithms, the solution provided by this procedure might be incomplete. In particular, if there is an intersection curve that does not have a starting point on the edge of one of the two surfaces, the algorithm does not find that curve. To include that kind of solutions, a loop detection algorithm is needed: we are currently implementing it and we plan to add it to Eyeshot version 10, that will be released in January 2017.

6.1.3.1 Surface Section

The surface section problem is a special case of SSI, in which a surface is intersected with a plane. The computation of this kind of intersections is required for different objectives in many industries, such as surface slicing for laminated object manufacturing and stereo lithography, NC tool path generation, medical imaging and contour lines for geographical models.

In general, when there is an intersection between a plane `pln` and a surface F, Eyeshot's algorithm for surface section generates from the plane a `PlanarSurface` G that is big enough to intersect F. Then, the SSI algorithm between F and G is applied.

Recently, we developed a new method that handles plane section of special surfaces differently. When possible, we compute analytic intersections between special surfaces and planes avoiding to perform the tracing algorithm. This method returns analytic curves like lines, circles and ellipses instead of NURBS curves that interpolate the tracing points.

For a `PlanarSurface` all the sections are now lines. For a `TabulatedSurface`, if the plane contains the `Generatrix` the section is a line, and if the plane is parallel to the plane of the `Directrix` the section of the same type of the `Directrix`. For a `RevolvedSurface`, if the plane is perpendicular to the axis of revolution, the section is a circle or an arc, and if the plane contains the axis of revolution, the section is of the same type of of the `Generatrix` curve. Finally, for cylinders we can compute analytic sections with any plane: the sections are ellipses or elliptical arcs. Fig. 6.8 shows two examples of surface sections.

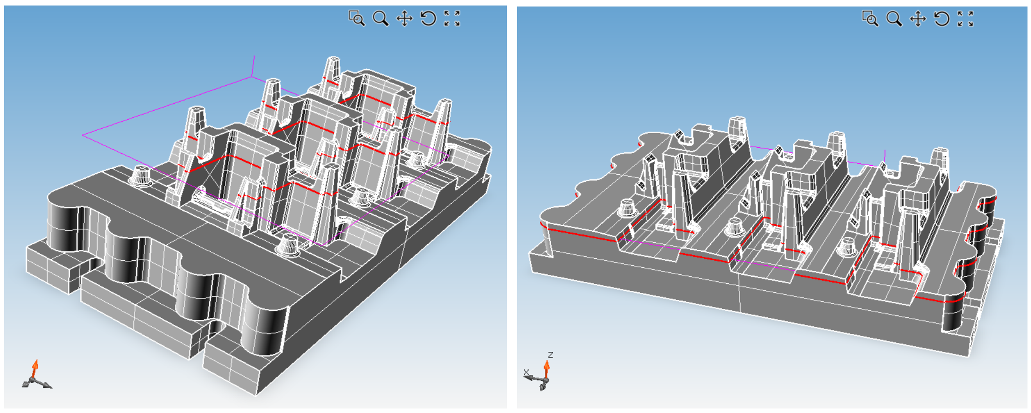


Figure 6.8: Surface section: the section plane is displayed in fuchsia, and the section curves in red.

The algorithms for point projection and SSI where already included in Eyeshot when I started working at devDept Software. Over the years, I modified these algorithms, especially the point projection ones, to improve the speed and accuracy. I also carried out the implementation of all the methods for special curves and surfaces. We will now introduce two algorithms that I added to Eyeshot: the algorithms for offsetting and filleting curves and surfaces.

6.1.4 Offset for curves and surfaces

An offset curve (or surface) is the set of all points which are at constant distance d along the normal vector of the base curve (or surface). Offset generation is an important task of CAD/CAM applications, since offsets play an important role in many areas, such as manufacturing mechanical parts, pocket machining and Computer Numerical Control (CNC) [38, 43].

Given a curve $C(u)$ and a surface $S(u, v)$, their offsets are respectively defined as

$$C^O(u) = C(u) + dn(u) \tag{6.14}$$

and

$$S^O(u,v) = S(u,v) + dn(u,v), \tag{6.15}$$

where d is the offset distance and $n(u)$ and $n(u,v)$ are the unit normal vectors. Imposing a negative offset distance is possible, and produces and offset on the other side of the curve or surface. Fig. 6.9 shows two offsets of the same curve with opposite signs of offset distance.

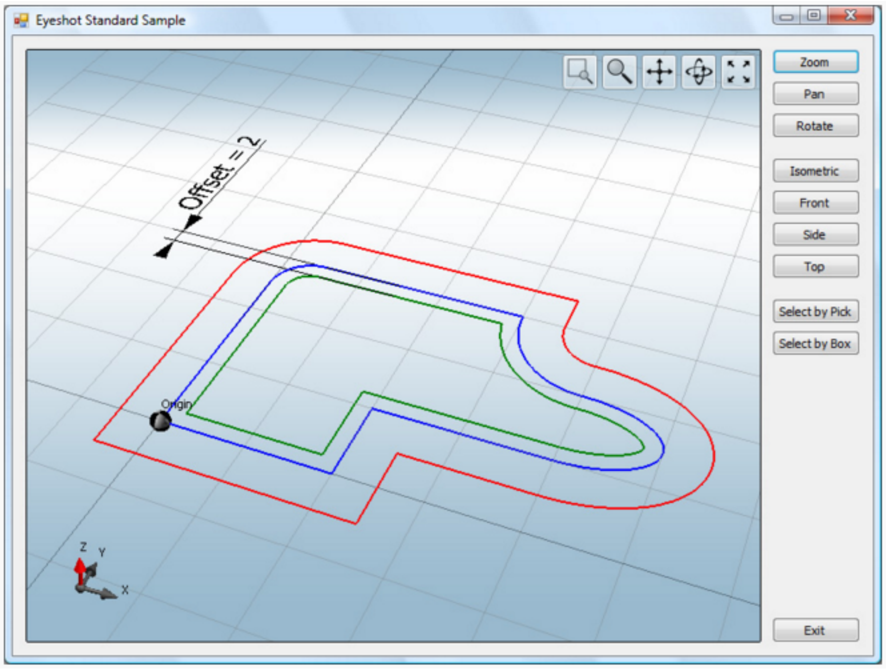


Figure 6.9: The green curve is the offset of the blue curve with distance $d = 2$, the red curve is the offset of the blue curve with distance $d = -3$.

Offsets of special curves and surfaces, such as circles, lines, revolved and tabulated surfaces, can be computed exactly and can be expressed in the same format as their progenitors, thus we developed special methods for these cases. Some examples of offsets of tabulated and revolved surfaces are shown in Fig.6.10.

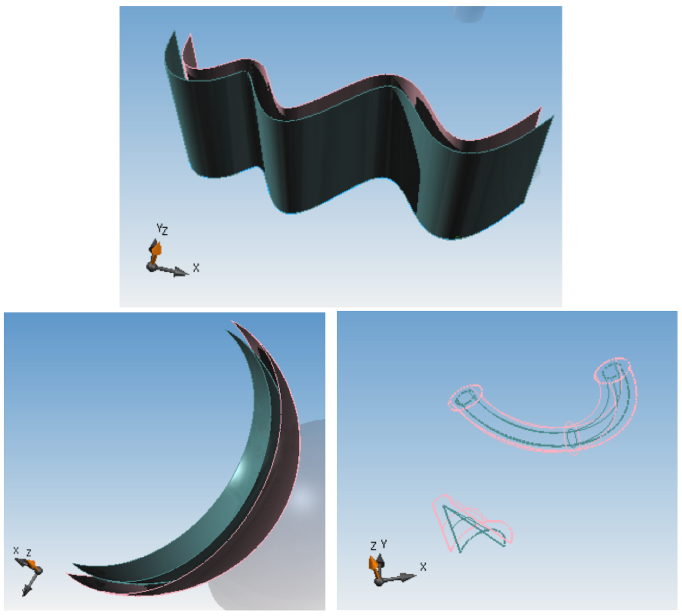


Figure 6.10: Surface offset of tabulated (top) and revolved (bottom) surfaces. The original surfaces are shown in pink, the offsets in blue.

For all the general cases, approximations are needed. We tried different algorithms to compute the offset of free-form curves and surfaces, and the first version of the algorithm added to Eyeshot was similar to the one presented in [45]. The method has been modified later and now it consists of the following steps:

- we compute the offset points at the knots of the original entity,
- we interpolate the offset points, and then check the error at the middle of each knot span, using point projection to measure the distance between the two curves.
- If there are points that are not within tolerance, we add new points where they are needed, and then repeat the interpolation and check until all the intervals are within tolerance.

Fig. 6.11 shows the a NURBS curve and its offset. In general, the number of control points of the offset curve is bigger than the number of control points of the original curve.

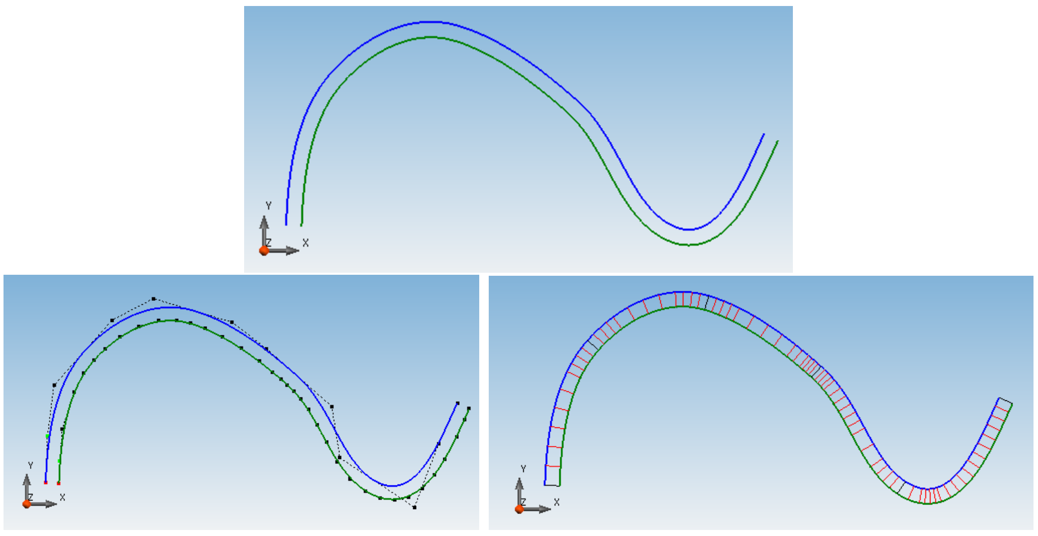


Figure 6.11: Top: the green curve is the offset of the blue one. Bottom left: the curves with their control polygons, the offset curve has more control points than the original curve. Bottom right: the segments connect the points in which we check if the distance between the two curves is within tolerance from the offset distance.

6.1.5 Fillet for curves and surfaces

Fillets are arc-shaped transitions between coplanar curves or between surfaces. They are very important in geometric modeling and are widely used to reduce stress concentrations in load-bearing models, to improve aesthetics and aerodynamics of the parts, and to avoid sharp edges that can be easily damaged or that can cause injury when a part is handled [27].

Unlike other methods described above, Eyeshot's fillet function does not have a different implementation when special curves and surfaces are involved.

The fillet method for two coplanar curves creates an arc of the desired radius r that connects and is tangent to both of the curves. If desired, the curves can be trimmed at the intersection point with the arc. Fig. 6.12 shows the creation of a fillet arc between two curves, which are also trimmed by the arc.

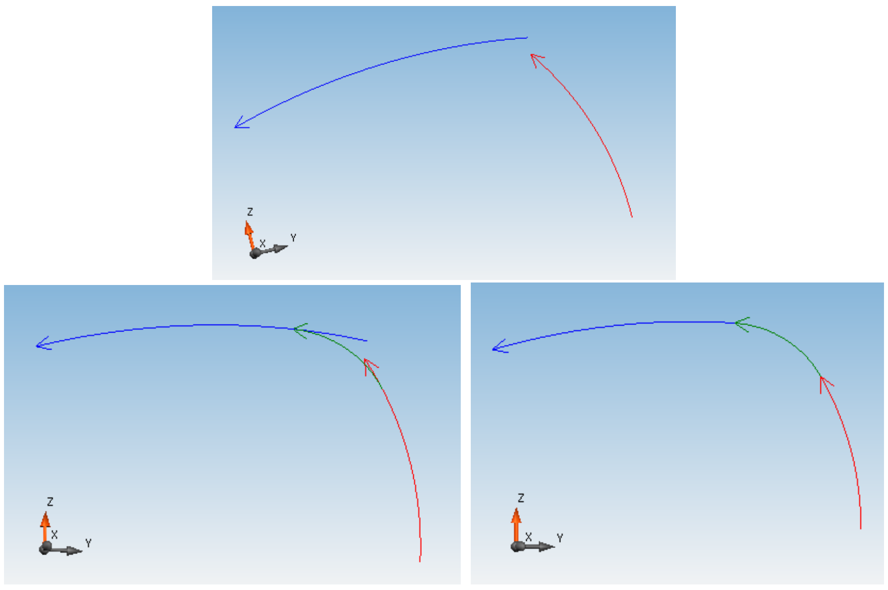


Figure 6.12: Top: the curves to fillet. Bottom left: the green arc is the fillet between the red and blue curves. Bottom right: the original curves are trimmed by the fillet arc.

The center of the fillet arc coincides with the intersection of the offsets of the two curves with distance $d = r$. In general, given a radius value, two intersecting curves can have four different fillet arcs, as shown in Fig. 6.13, depending on the side in which each curve is offsetted.

In Eyeshot, the choice of which of the four fillets will be computed is operated through two booleans that control the side on which each curve is offsetted. The signature of the fillet method for curves is:

```
Fillet(ICurve C1, ICurve C2, double radius, bool flip1, bool flip2, ...
        ... bool trim1, bool trim2, out Arc fillet)
```

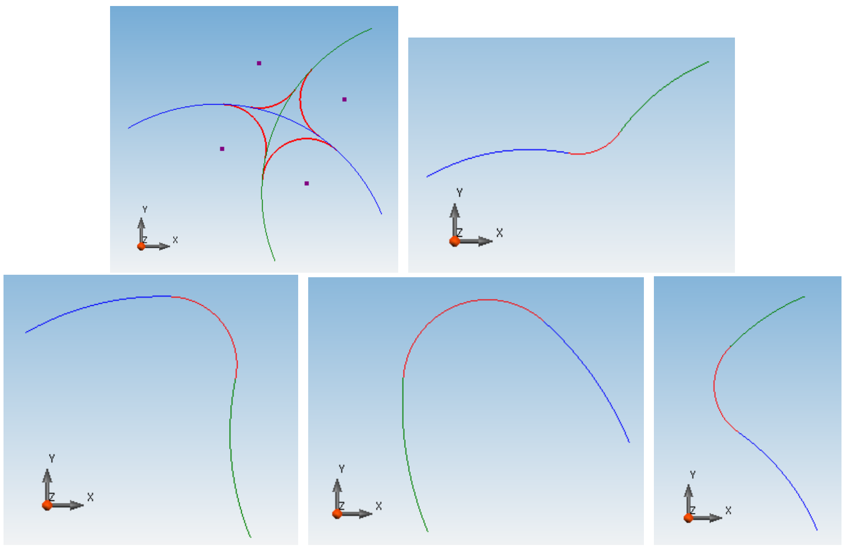


Figure 6.13: Top left: the red arcs are the four possible fillets that can be obtained from the green and blue curves given a radius value. The arcs have the same radius and the purple points are their centers. Top right and bottom: each of the four fillet arcs is used to trim the the original curves.

The input parameters are:

- `C1` and `C2`: the two coplanar curves to fillet,
- `radius`: the desired radius of the fillet arc,
- `flip1` and `flip2`: the booleans to control which fillet to compute,
- `trim1` and `trim2`: the booleans to control whether the curves are trimmed by the fillet arc.

The method returns as out parameter the arc `fillet`.

The fillet method for surfaces creates a rolling-ball blending between two surfaces and it is based on a marching algorithm similar to the one used for SSI introduced in Section 6.1.3. The resulting fillet can be seen as the surface described by an imaginary sphere of radius r rolling along the intersection between two surfaces in such a way that it remains in simultaneous contact with both surfaces at points of tangency. Also for surfaces, the fillet can be used to trim the original surfaces. Fig. 6.14 shows some examples of fillets made with Eyeshot.

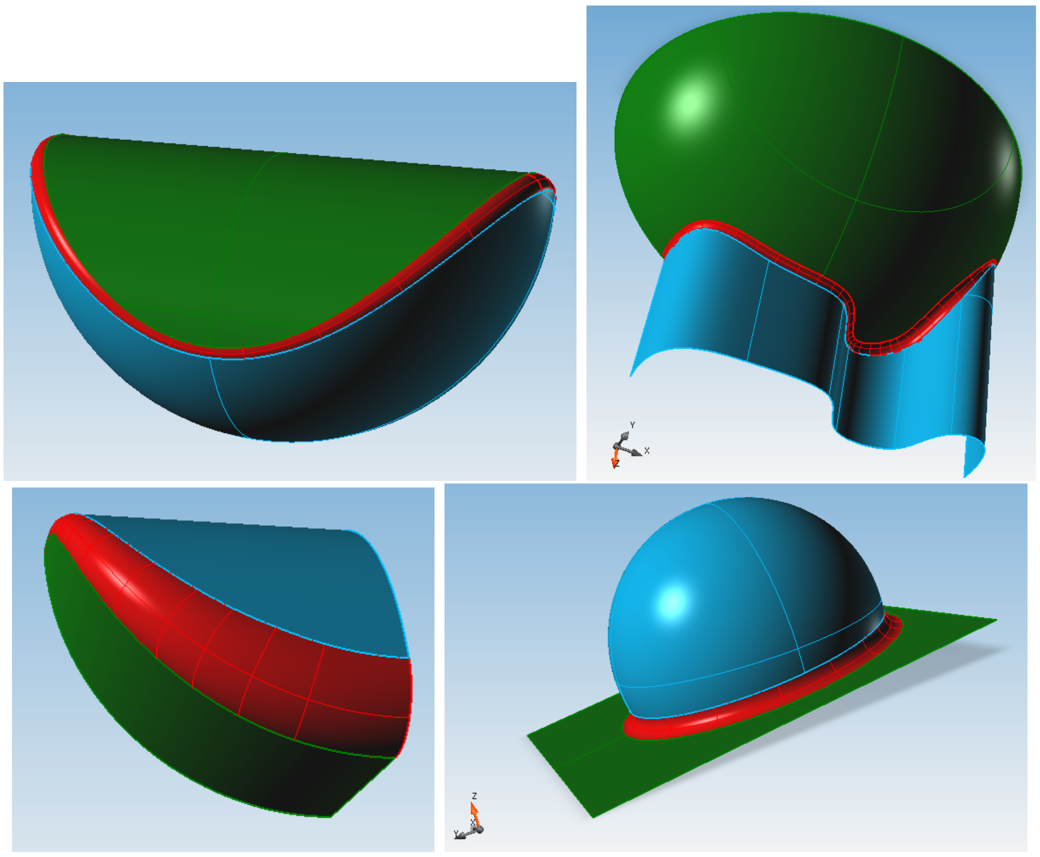


Figure 6.14: Examples of fillets (drawn in red) between two surfaces. Top left: fillet between a cylinder and a sphere. Top right: fillet between a sphere and an extrusion surface. Bottom left: fillet between two cylinders. Bottom right: fillet between a planar surface and a sphere.

Given two surfaces and a radius value, the marching algorithm finds a sequence of arcs of radius r that have their centers along the intersection curve of the offsets to the given surfaces. Once the tracing is complete, the fillet surface is constructed interpolating all the arcs by a process called skinning or lofting [44, 47]. Fig. 6.15 shows the arcs used to build a fillet surface.

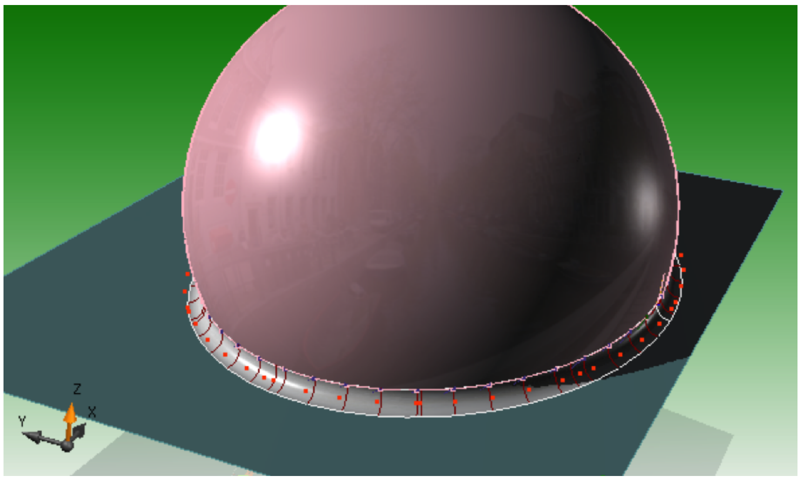


Figure 6.15: Fillet between a spherical and a planar surfaces. The brown curves are the arcs found by the marching algorithm that have been used to build the fillet surface, and the orange points are their centers.

Computing the offsets of two surfaces and then the intersection between them, would make the fillet method very slow. Thus, we use a modified version of the SSI algorithm that allows to find the intersection of the offset surfaces directly from the non-offsetted ones, avoiding to generate the offset surfaces [13, 32].

The signature of the fillet method for surface is similar to the one for curves:

```
Fillet(Surface F, Surface G, double radius, double tol, bool flipNormalF,...
   ...bool flipNormalG, bool trimF, bool trimG, bool flipTrimSideF, bool flipTrimSideG,...
   out Surface[] fillet)
```

where the parameters `F, G, radius, flipNormalF, flipNormalG, trimF` and `trimG` have the same roles of the corresponding ones in the curves' method, and the additional parameters are:

- `tol`: the tolerance to compute the intersection between the offset surfaces,

- `flipTrimSideF` and `flipTrimSideG`: the booleans to control which side of the surfaces F and G are trimmed by the fillet surface.

The out parameter `fillet` is an array of surfaces: one for each curve resulting from the intersection between the offset surfaces.

Eyeshot has two other sightly different versions of the fillet method for surfaces. The first one computes the fillets between two lists of surfaces instead of just two surfaces. It is useful when a user has a B-rep or is building a model composed of many surfaces. The second one is called `VariableFillet` and allows to create variable-radius fillets. The user can decide the start and end radiuses, and if the transition between them is linear or cubic. In this case, the algorithm computes and lofts arcs of different radiuses depending on their position along the intersection. Two examples of variable-radius fillets are shown in Fig. 6.16.

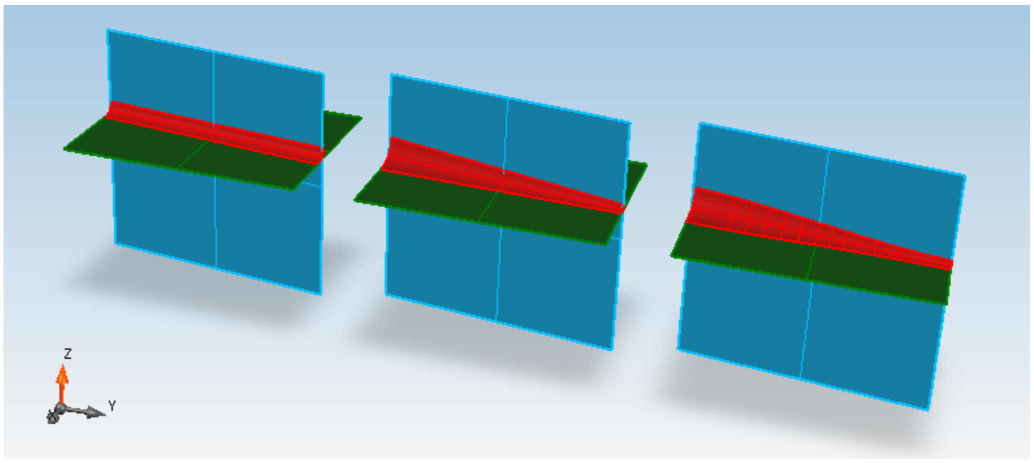


Figure 6.16: Fillet between planar surfaces with constant (left) and variable (center and right) radiuses.

6.1.6 A CAD modeling example

We will now show how the methods introduced in the previous sections can be used in Eyeshot to create the CAD model of a hair dryer shown in

Fig. 6.17.

Since the model is symmetric, we build half of it, and then we mirror it along the symmetry plane to get the full model.

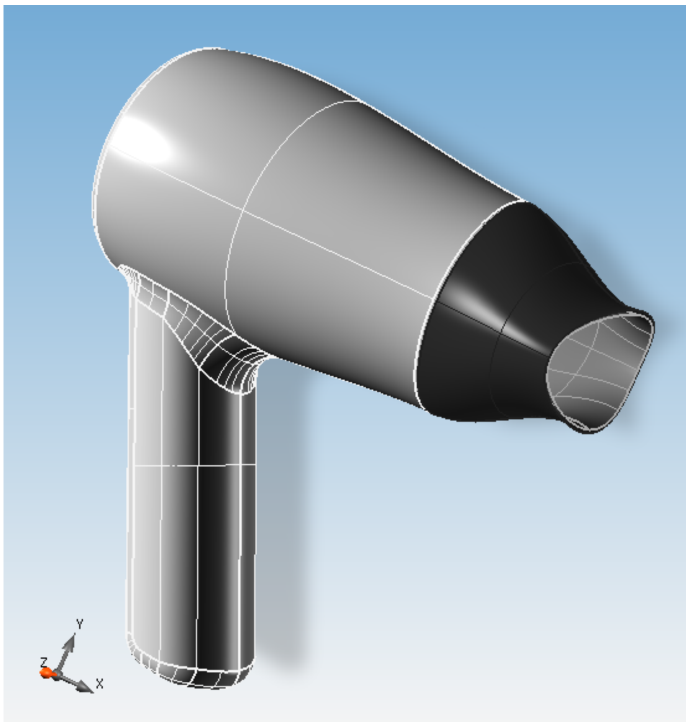


Figure 6.17: Hair dryer model.

Fig. 6.18 shows the creation of the body and rear of the hair dryer, and the fillet between them.

Fig. 6.19 shows the creation of the handle.

In Fig. 6.20 the two parts created in Fig. 6.18 and Fig. 6.19 are joined by a fillet surface.

In Fig. 6.21 an air concentrator is added.

In Fig. 6.22 all the surfaces are offsetted to create the interior part of the model.

It can be noted that the offsets of NURBS surfaces have more isocurves (i.e. more knots and control points) than the original surfaces. This does

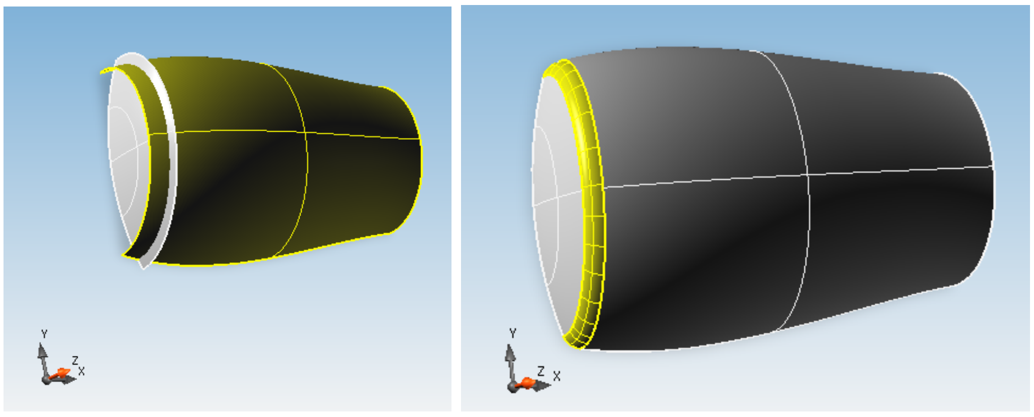


Figure 6.18: Left: the body of the hair dryer (in yellow) is a `RevolvedSurface` obtained by revolving a NURBS curve, and the rear is a `SphericalSurface` obtained by revolving an arc. Right: a fillet of the two surfaces is added and displayed in yellow.

not happen to the offsets of the special surfaces (rear and some surfaces of the handle), that are created with a special procedure.

Fig. 6.23 shows the details of some offset surfaces.

All the surfaces are then mirrored along the symmetry plane, Fig. 6.24 shows two more images of the full model.

6.2 Eyeshot FEM

Eyeshot FEM edition allows the users to solve linear elasticity problems, introduced in Section 2.3, on 2D and 3D domains represented by meshes. A typical problem aims to find the displacements, stresses and strains of an object subject to forces and constraints.

The user builds a model that represents an object and applies forces, constraints, temperatures and fixed displacements on it. Eyeshot solves the problem using the Galerkin method, with Lagrange basis functions as basis for the solution and solving the linear system with a direct or iterative (pre-

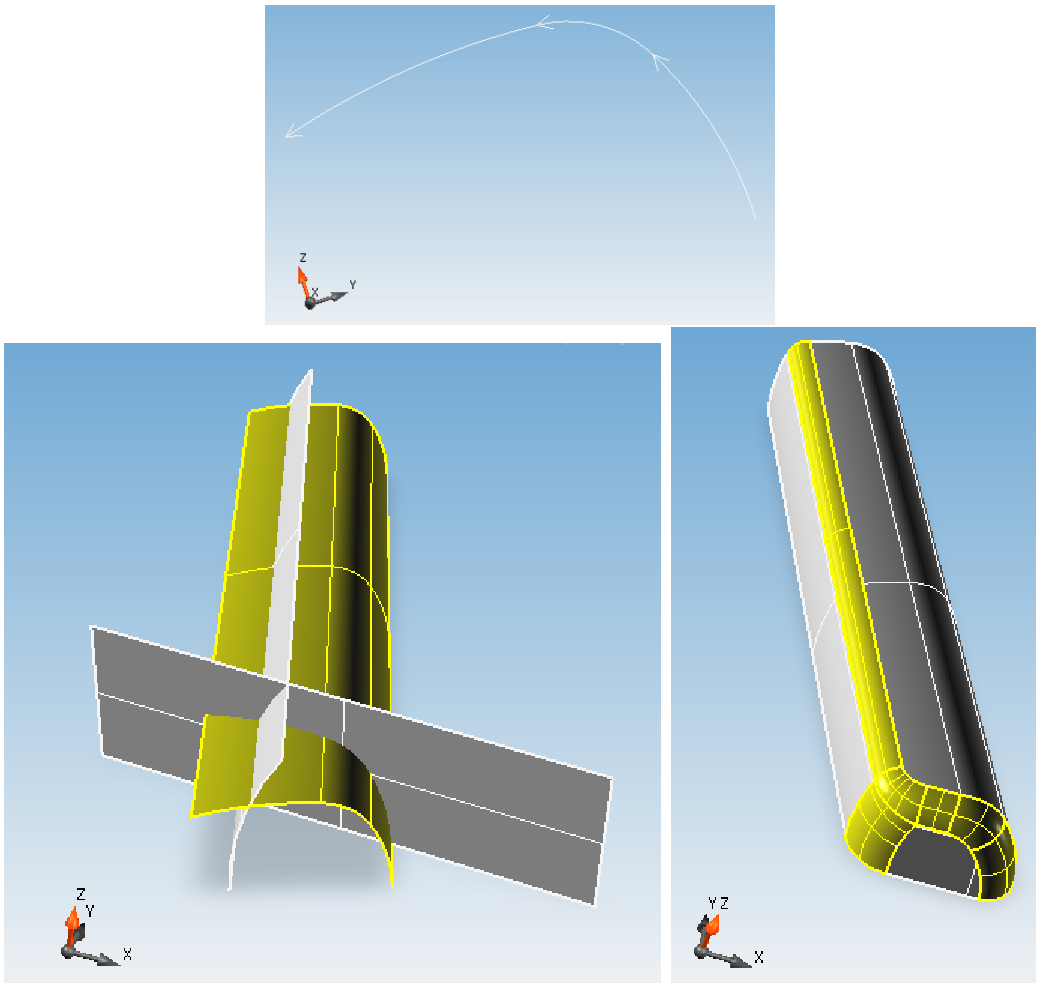


Figure 6.19: Top: the same curves and fillet arc of Fig. 6.12 are joined and extruded to create the yellow `TabulatedSurface` in the bottom left figure. Bottom left: the tabulated, cylindrical and planar surfaces used to create the handle. Bottom right: with two calls of the fillet method (one of which is applied to a list of surfaces) the fillets to complete the handle are created.

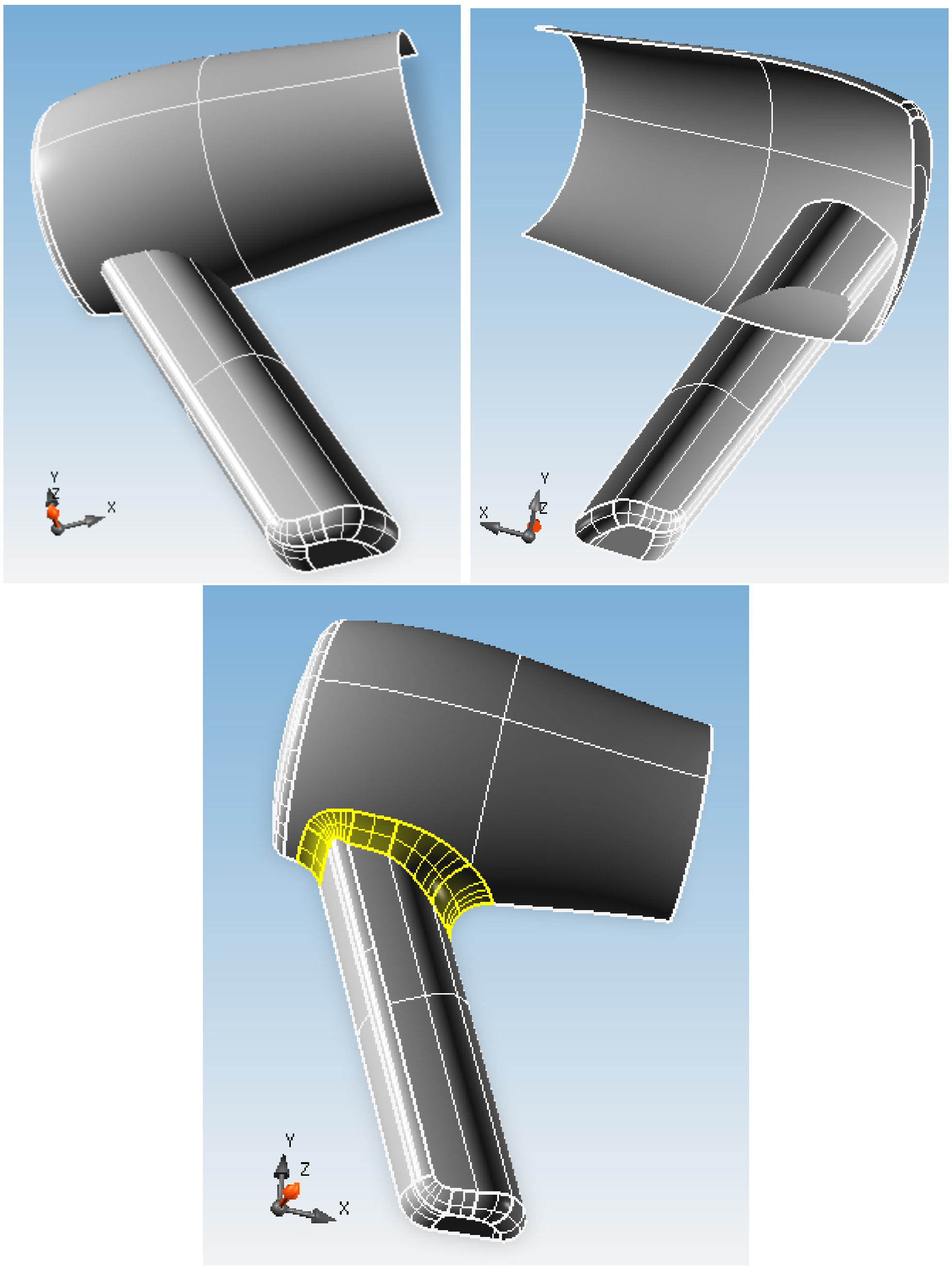


Figure 6.20: Top: the two groups of surfaces created above, seen form two different points of view. Bottom: a fillet between the body and a group of surfaces of the handle is created and displayed in yellow.

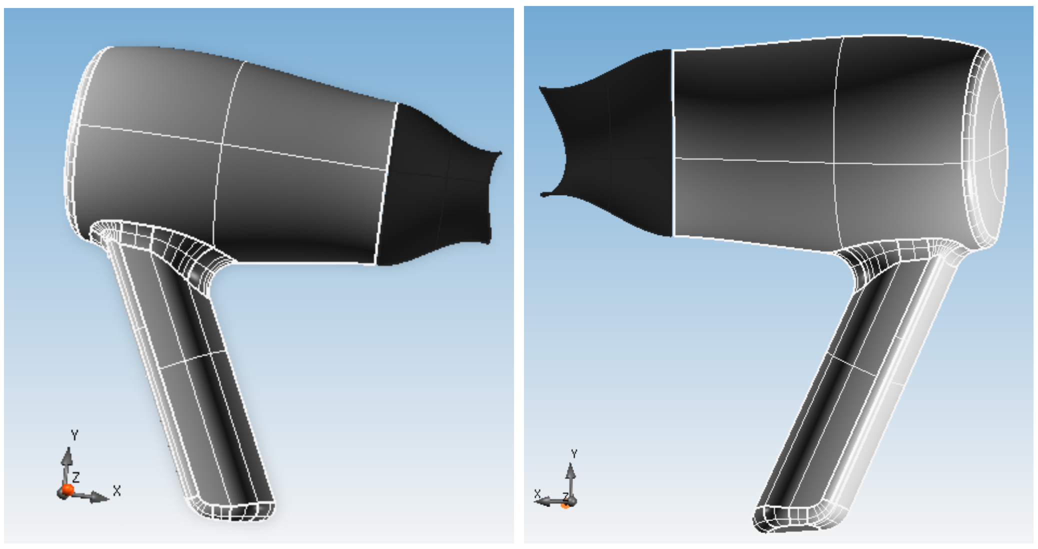


Figure 6.21: A black air concentrator is added, created by lofting a circle and two other NURBS curves.

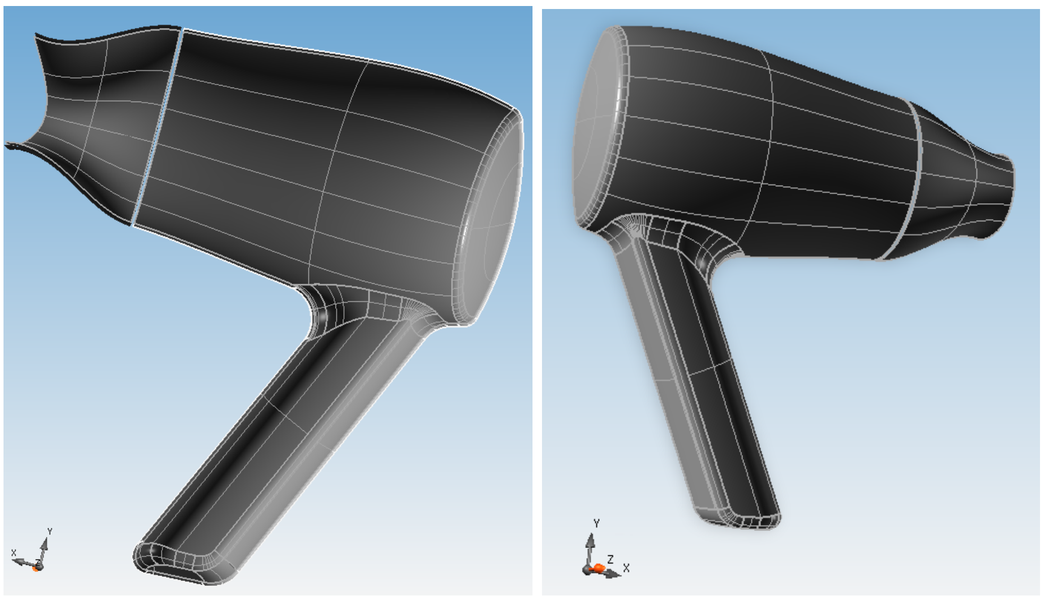


Figure 6.22: A dark gray offset of each surface is created. Right: the original surfaces have been removed to show their offsets. It can be noted that the offsets of NURBS surfaces are more complex than the originals. This does not happen to the offsets of special surfaces.

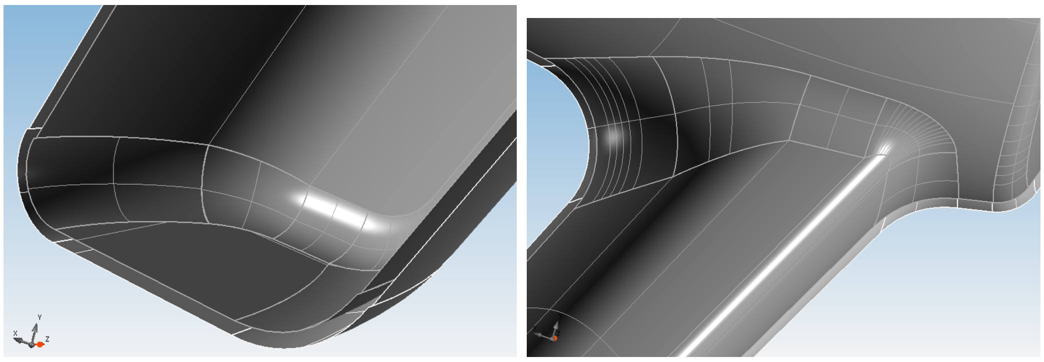


Figure 6.23: Details of some offset surfaces.

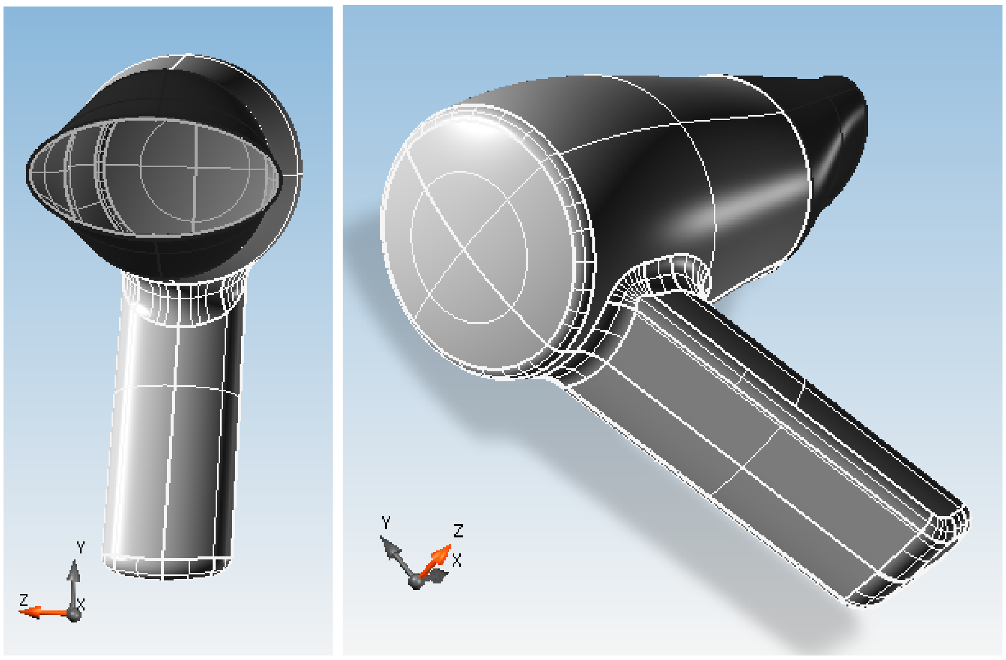


Figure 6.24: Two more views of the complete hair dryer model.

conditioned conjugate gradient) solver. At the end of the calculations, the software displays the results and the user has to interpret them.

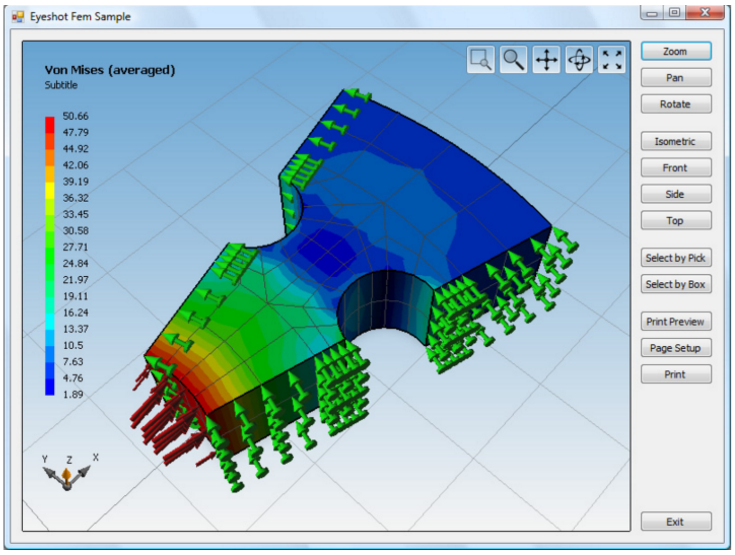


Figure 6.25: FEM analysis. The red arrows represent punctual loads and the green arrows represent the nodal constraints. The colors on the mesh represent the Von Mises stress values, that are scalar stress values combining the principal stresses acting in different directions.

The entities that are used to build a FEM model in Eyeshot are called `FEMMesh`. They are made of elements of various shapes, and they are different from the `Mesh` entities introduced in Section 6.1.1, that are triangular meshes used by the CAD system.

The geometry of a `FEMMesh` is defined by the properties `Elements` and `Vertices`, in which are stored the informations on elements, nodes and connectivity arrays of the model. Analysis on surfaces embedded in $\mathbb{R}^3$ is not supported yet, therefore one can perform 2D-analyses on planar surfaces, or 3D-analyses on solids. Depending on this dimension, the elements are subdivided in `Element2D`, that lie on the XY-plane, or `Element3D`.

The supported 2D elements are:

- `Tria3`: triangular elements with 3 nodes and 2 dof per node,
- `Tria6`: quadratic triangular elements with 6 nodes and 2 dof per node,
- `Quad4`: rectangular elements with 4 nodes and 2 dof per node,
- `Quad8`: quadratic rectangular elements with 8 nodes and 2 dof per node,
- `Truss2D`: linear elements with 2 nodes and 2 dof per node,
- `Beam2D`: linear elements with 2 nodes and 3 dof per node,

and the supported 3D elements are:

- `Tetra4`: tetrahedral elements with 4 nodes and 3 dof per node,
- `Tetra10`: quadratic tetrahedral elements with 10 nodes and 3 dof per node,
- `Hexa8`: hexahedral elements with 8 nodes and 3 dof per node,
- `Hexa20`: quadratic hexahedral elements with 20 nodes and 3 dof per node,
- `Penta6`: pentahedral elements with 6 nodes and 3 dof per node,
- `Penta15`: pentahedral elements with 15 nodes and 3 dof per node,
- `Truss`: linear elements with 2 nodes and 3 dof per node,
- `Beam`: linear elements with 2 nodes and 6 dof per node.

Fig. 6.26 shows a diagram of Eyeshot's elements, and their geometry.

A `Material` is assigned to each element, it stores physical properties such as the Young modulus E, the Poisson ratio ν and the coefficient of expansion α, which, as we saw in Section 2.3, are fundamental for the analysis.

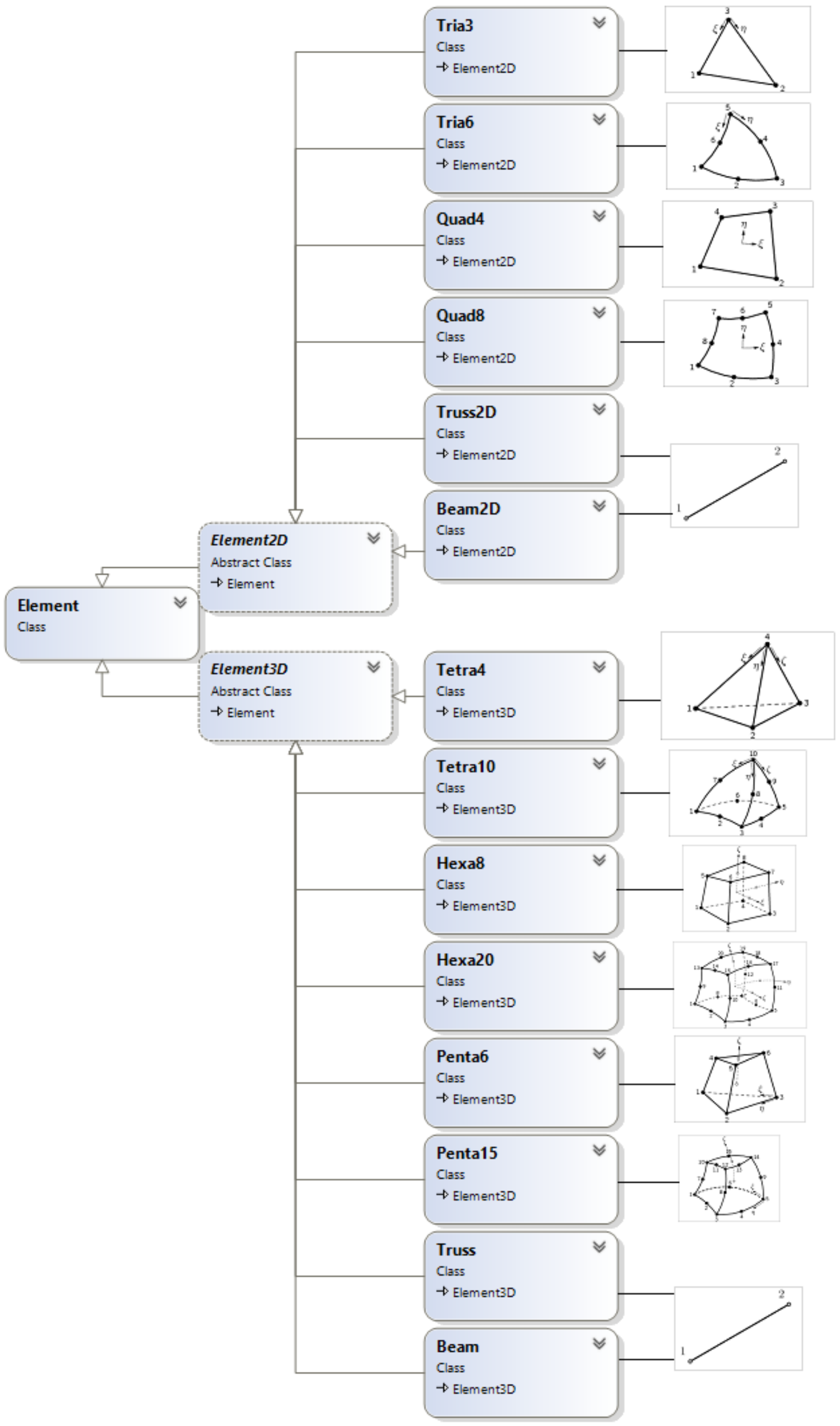


Figure 6.26: FEM elements in Eyeshot.

A `FEMMesh` model can either be imported from a .txt file or it can be built within Eyeshot, that in addition to the mesh definition via nodes and elements offers some mapped meshing functions. These functions allow to fill some planar regions representing simple shapes (rectangles, circles, squares with holes) with `Tria` or `Quad` elements, and create 3D elements by extruding or revolving 2D meshes. A `FEMMesh` made of `Tria3` or `Tria6` elements can also be derived from a `Mesh` entity, using the method `ConvertToFEMMesh()`.

Once the geometric model is ready, loads and constraints can be applied. The supported load types are: punctual forces or temperature loads applied to the nodes, and pressure loads applied to the faces of the elements. Constraints can be applied to a node by blocking all or some of its degrees of freedom: this implies that the displacements of a node after the application of the loads are zero along the constrained directions.

Eyeshot solver has a pre-processing phase in which each element's stiffness matrix and load vector are computed and assembled in the global stiffness matrix K and load vector F, which take into account also the boundary conditions. Then the linear system $KU = F$ is solved using a direct or iterative method. The solution of the system gives a vector U of the displacements of each node of the mesh. From this vector, in the post-processing phase, we compute the strains and stresses and prepare the `FEMMesh` to display the solution. In addition to the graphical display of the solution, Eyeshot can also produce tables with the numeric results for each node and element.

When I first arrived at devDept Software, the FEM component was already been developed, and it's only when I knew that my thesis would have been on Isogeometric Analysis that I started working on it. My main contributions to the FEM component of Eyeshot are:

- modifying the code in order to store the matrices in Compressed Sparse Row (CSR) format, which is more efficient and less memory-consuming,

- developing part of the mapped meshing methods, that allow users to quickly create a `FEMMesh` of a simple shape.
- adding temperature and pressure loads,
- adding a preconditioning scheme to the conjugate gradient method,
- adding the possibility to use a direct solver instead of the iterative one,
- adding `Truss2D` and `Truss` elements,
- adding `Beam2D` and `Beam` elements.

Since the addition of Beam elements has been the latest, longest and most interesting task, we will examine it more in depth.

6.2.1 Beam elements

Beams are slender structural members that offer resistance to forces and bending under applied loads. A beam element differs from a truss element in that a beam resists moments (twisting and bending) at the connections. Thus, beams have rotational degrees of freedom in addition to the translational ones. In particular, `Beam2D` elements have three degrees of freedom: two translational along the X and Y axes, and one rotational around the Z axis. `Beam` elements have six degrees of freedom: one translational and one rotational for each axis (X, Y and Z).

Beams are typically applied in a civil engineering context to describe structural elements as pillars, columns or structural beams, but they can also be used to represent parts of other mechanical or structural systems.

We developed elementary beam theory, formally known as Euler-Bernoulli beam theory, following mainly Bhatti's book [2].

Since beam members are represented as lines, and lines are objects with no inherent orientation, a beam needs a local coordinate system (U, V, W) to control how the cross section is rotated about the beam's longitudinal axis (which by convention is the U axis). In 2D, the local coordinate system is fully determined by the locations of the two nodes: the U axis is parallel to the element's length and the W axis is parallel to the global Z axis. In 3D, we chose to set the V axis perpendicular to the U axis and the global Z axis, and the W axis is obtained by the cross product $U \times V$. When U is parallel to Z, V is made parallel to the global Y axis. Users can change this default setting assigning an arbitrary local coordinate system to beams, and they can see beams with their associated coordinate system activating wireframe visualization mode.

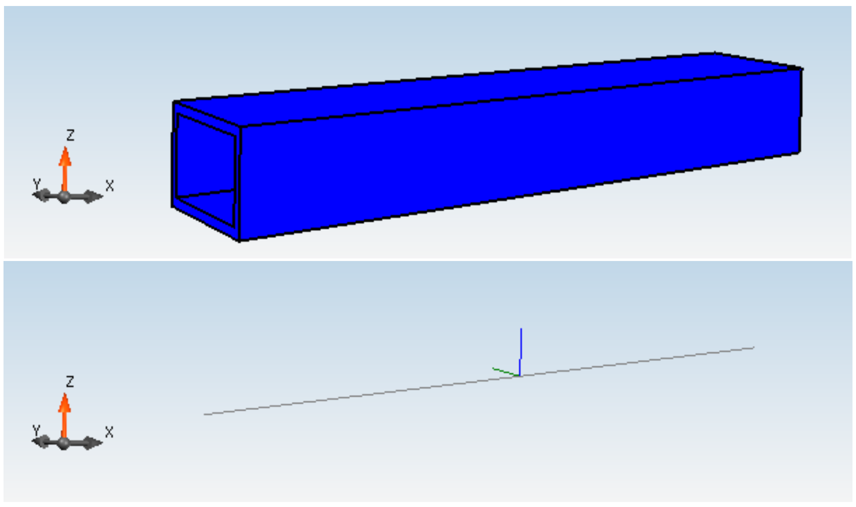


Figure 6.27: A beam element with rectangular hollow cross section (top) and its local coordinate system (bottom): the the green and blue segments are along the V and W axes, respectively.

To each beam are assigned a material and a cross section, which can be general or have a predefined shape such as: square, rectangular, rectangular hollow, circular, circular hollow, I-shaped, T-shaped or C-shaped (the last two cross sections can only be used for 2D analyses, because they have only one axis of symmetry). From the shape of the cross section we automatically compute some properties that are fundamental for the creation of the stiffness matrix and the solution of the beam: cross section area A, moments of inertia I_{V} and I_{W} about V and W axes, and torsional constant J. For beams with general sections, these quantities must be supplied as input by the user.

The procedure to solve a beam problem through Eyeshot is the same as that followed with other elements: a `FEMMesh` must be defined, then loads and constraints are applied, and finally the solver is called. Internally, we compute the stiffness matrix and load vector in local coordinates for each beam, then we convert them in global coordinates and assemble them into the global stiffness matrix and load vector. The solution of the linear system gives the displacements and rotations in global coordinates for each node of the mesh.

From this data, we compute the displacements at different equally spaced points along each beam, in order to display the beams with their deformations. We also compute three orthogonal forces (one axial along U and two shear forces along V and W) and three orthogonal moments (one torsion around U and two bending around V and W) are calculated at each end of each element. The user can decide which of these quantities to show in a model like the one in Fig. 6.28.

Also for beams, tables with the numeric results on nodes and elements can be produced.

After releasing the first implementation of beam elements, some customers asked us to develop also thermal loading, punctual loads applied to an internal point of a beam, and hinge releases, which are equivalent to imposing that the bending moments around some axes are zero. We implemented all these new features, and we are getting the correct results on the nodes of the `FEMMesh`. We are still working on getting good results along the beams when non-nodal loads or hinges are involved.

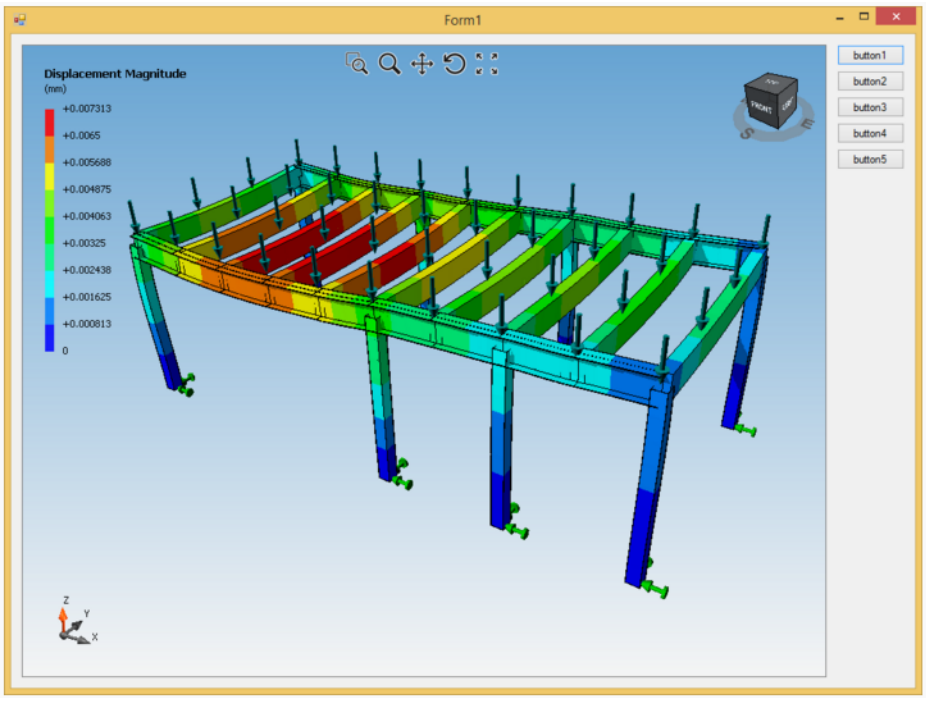


Figure 6.28: Beam analysis. The cyan arrows represent distributed loads and the green arrows represent the nodal constraints. The colors on the mesh represent the displacement magnitude along each beam.

Chapter 7

Conclusions

This thesis presents a detailed study of Isogeometric Analysis, focusing on geometry representations and Computer Aided Design.

We provided several examples of the successful application of IGA to solve problems defined on different kinds of geometries, 2D and 3D, single and multipatch.

Comparing the results obtained by IGA with the ones of a conventional FEM approach, we observed that IGA exhibits superior accuracy than FEM if the same number of degrees of freedom is used. Therefore, FEM needs more degrees of freedom to reach a prescribed tolerance.

Both methods use compactly supported basis functions, thus the stiffness matrices are banded and sparse. Although the B-spline functions used by IGA have support over larger portions of the domain than do classical FEA functions, this does not lead to increased bandwidth of IGA in a numerical method. We observed that, at the same iteration of h and p-refinement, the bandwidths of the stiffness matrices of IGA and FEM are equal, even though the FEM matrix is bigger and has more nonzero elements.

Analyzing the condition number of the stiffness matrices, we noticed that it grows with the refinement for both methods. For a fixed number of degrees of freedom, the condition number of IGA is higher than the one of FEM.

Besides the examples on traditional geometries, we applied IGA on a particular class of physical domains, that we called hybrid domains, which are multipatch models composed of NURBS and mesh patches. Combining precise and approximate geometries, hybrid models are an emergent way to represent solid objects. The development of robust and performant algorithms involving hybrid geometries can be the subject of future research in the CAD field.

As for the application of IGA on hybrid geometries, the results obtained in problems with a known exact solution showed that a satisfactory level of precision can be achieved.

In our examples on 2D and 3D hybrid physical domains, we guaranteed C^0 continuity of the solution along the interfaces between the patches. Further research can be made in order to improve the continuity of the solution between adjacent patches.

Bibliography

[1] Alberty Jochen, Carstensen Carsten, Funken Stefan A., Klose Ronald, *Matlab-implementation of the finite element method in elasticity*, Computing, 69: 239-263, 2002.

[2] Barnhill Robert E., Kersey Scott N., *A Marching Method For Parametric Surface/Surface Intersection*, Computer Aided Geometric Design, 7: 257-280, 1990.

[3] Bazilevs Yuri, Beirão da Veiga Lourenço, Cottrell Austin J., Hughes Thomas J.R., Sangalli Giancarlo, *Isogeometric Analysis: Approximation, stability and error estimates for h-refined meshes*, Mathematical Models and Methods in Applied Sciences, 16: 1031-1090, 2006.

[4] Bazilevs Yuri, Calo Victor M., Cottrell Austin J., Evans John A., Hughes Thomas J.R., Lipton Scott, Scott Michael A., Thomas W. Sederberg, *Isogeometric analysis using T-splines*, Computer Methods in Applied Mechanics and Engineering, 199: 229-263, 2010.

[5] Bhatti Asghar M., *Fundamental finite element analysis and applications*, John Wiley & Sons, 2005.

[6] Botsch Mario, Kobbelt Leif, Pauly Mark, Alliez Pierre, Lvy Bruno, *Polygon Mesh Processing*, CRC press, 2010.

[7] Bommes David, Lempfer Timm, Kobbelt Leif, *Global Structure Optimization of Quadrilateral Meshes*, Computer Graphics Forum, Vol. 30, No. 2, 2011.

[8] Bommes David, Lévy Bruno, Pietroni Nico, Puppo Enrico, Silva Claudio, Tarini Marco, Zorin Denis, *Quad-Meshe Generation and Processing: a survey*, Computer Graphics Forum, Vol. 32, No. 6, 2013.

[9] Buffa Annalisa, Sangalli Giancarlo, Raphael Vázquez, *Isogeometric Methods for Computational Electromagnetics: B-spline and T-spline discretizations*, Journal of Computational Physics, 257, 1291-1320, 2014.

[10] Chen Xiao-Diao, Yong Jun-Hai, Wang Guozhao, Paul Jean-Claude, Xu Gang, *Computing the minimum distance between a point and a NURBS curve*, Computer Aided Geometric Design, 40: 1051-1054, 2008.

[11] Chen Xiao-Diao, Xu Gang, Yong Jun-Hai, Wang Guozhao, Paul Jean-Claude, *Computing the minimum distance between a point and a clamped B-spline surface*, Graphical Models, 71: 107-112, 2009.

[12] Chen Jianjun, Cao Bingwan, Zheng Yao, Xie Lijun, Li Chenfeng, Xiao Zhoufang, *Automatic surface repairing, defeaturing and meshing algorithms based on an extended B-rep*, Advances in Engineering Software, 86: 55-69, 2015.

[13] Choi Byoung K., Ju S. Y., *Constant-radius blending in surface modeling*, Computer-Aided Design, 21: 213-220, 1989.

[14] Cohen Elaine, Martin Tobias, Kirby Robert M., Tom Lyche, Riesenfeld Richard F., *Analysis-aware modeling: Understanding quality considerations in modeling for isogeometric analysis*, Computer Methods in Applied Mechanics and Engineering, Vol. 199, 334-356, 2010.

[15] Cohen Elaine, Tom Lyche, Riesenfeld Richard F., *Discrete B-splines and subdivision techniques in computer aided geometric design and computer graphics. Computer Graphics and Image Processing*, Computer Graphics and Image Processing, 14, 87-111, 1980.

[16] Cottrell Austin J., Hughes Thomas J.R., Bazilevs Yuri, *Isogeometric Analysis: Toward integration of CAD and FEA*, John Wiley & Sons, 2009.

[17] Cottrell Austin J., Hughes Thomas J.R., Reali Alessandro, *Studies of Refinement and Continuity in Isogeometric Structural Analysis*, Computer Methods in Applied Mechanics and Engineering Vol. 196, 4160-4183, 2007.

[18] D'Azevedo Eduardo F., *Are bilinear quadrilaterals better than linear triangles?*, SIAM Journal on Scientific Computing, Vol. 22, No. 1, 2000.

[19] Dedè Luca, Quarteroni Alfio, *Isogeometric Analysis for second order Partial Differential Equations on surfaces*, Computer Methods in Applied Mechanics and Engineering, Vol. 284, 2015

[20] De Falco Carlo, Reali Alessandro, Vázquez Rafael, *GeoPDEs: a research tool for Isogeometric Analysis of PDEs* Advances in Engineering Software, 42(12): 1020-1034, 2011.

[21] De Lorenzis Laura, Wriggers Peter, Hughes Thomas J.R., *Isogeometric contact: A review*, GAMM Mitteilungen, Vol 37, No. 1, 2014.

[22] Do Carmo Manfredo P., *Differential Geometry of Curves and Surfaces*, Prentice Hall, 1988.

[23] Dokken Tor, Quak Ewald, Skytt Vibeke, *Requirements from Isogeometric Analysis for Changes in Product Design Ontologies*, Proceedings of

the Focus K3D Conference on Semantic 3D Media and Content, Sophia Antipolis, 2010.

[24] Dokken Tor, Skytt Vibeke, Haenisch Jochen, Bengtsson Kjell, *Isogeometric Representation and Analysis - Bridging the Gap between CAD and Analysis*, 47th AIAA Aerospace Sciences Meeting Including The New Horizons Forum and Aerospace Exposition, Orlando, Florida, 5 - 8 January 2009.

[25] Eyeshot's documentation:
`http://documentation.devdept.com/90/webframe.html#topic1.html`.

[26] Farin Gerald, Hoschek Josef, Kim Myung-Soo, *Handbook of Computer Aided Geometric Design*, Elsevier Science B.V., 2002.

[27] Farouki Rida T., Sverrisson Ragnar, *Approximation of rolling-ball blends for free-form parametric surfaces*, Computer-Aided Design, 28: 871-878, 1996.

[28] Filkins Peter C., Tuohy Seamus T., Patrikalakis Nicholas M., *Computational methods for blending surface approximation*, Engineering With Computers, 9: 49-62, 1993.

[29] Giannelli Carlotta, Jüttler Bert, Kleiss Stefan K., Mantzaflaris Angelos, Simeon Bernd, Špeh Jaka, *THB-splines: an effective mathematical technology for adaptive refinement in geometric design and isogeometric analysis*, Computer Methods in Applied Mechanics and Engineering, 299, 337-365, 2016.

[30] Gockenbach Mark S., *Understanding and Implementing the Finite Element Method*, Siam, 2006.

[31] Gravesen Jens, Evgrafov Anton, Nguyen Dang-Manh, Nøortoft Peter, *Planar Parametrization in Isogeometric Analysis*, Lecture Notes in Computer Science, Vol. 8177, 189-212, 2014.

[32] Huang Ling, Zhu Xinxiong, *Construction of blending surfaces*, Technical Report HZ-TMSurf-Huang04, Beijing University of Aeronautics and Astronautics, 2000.

[33] Hughes Thomas J.R., *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*, Dover Publications, 2000.

[34] Hughes Thomas J.R., Cottrell Austin J., Bazilevs Yuri, *Isogeometric Analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement*, Computer Methods in Applied Mechanics and Engineering, Vol. 194, No. 39-41, 2005.

[35] Ko Kwangee, Sakkalis Takis, *Orthogonal projection of points in CAD/ CAM applications: an overview*, Journal of Computational Design and Engineering, Vol. 1, No. 2, 2014.

[36] Kriezis George A., Patrikalakis Nicholas M., Wolter Franz-Erich, *Topological and differential-equation methods for surface intersections.* Computer-Aided Design, Vol. 24, No. 1: 41-55, 1992.

[37] Langer Ulrich, Mantzaflaris Angelos, E. Moore Stephen, Toulopoulos Ioannis, *Multipatch Discontinuous Galerkin Isogeometric Analysis*, Technical Report 18, NFN Geometry + Simulation, 2014.

[38] Liu Xu-Zheng, Yong Jun-Hai, Zheng Guo-Qin, Sun Jia-Guang, *An offset algorithm for polyline curves*, Computers in Industry, 58, 240-254, 2007.

[39] Ma Ying Liang, Hewitt Terry W., *Point inversion and projection for NURBS curve and surface: Control polygon approach*, Computer Aided Geometric Design, 20: 79-99, 2003.

[40] Micula Gheorghe, Micula Sanda, *Handbook of Splines*, Mathematics and its applications, Vol. 462, Kluwer Academic Publishers, 1999.

[41] Nguyen Thien, Karčiauskas Keçstutis, Peters Jörg, *A Comparative Study of Several Classical, Discrete Differential and Isogeometric Methods for Solving Poissons Equation on the Disk*, Axioms, 3(2): 280-299, 2014.

[42] Patrikalakis Nicholas M., *Surface-to-Surface Intersections*, IEEE Computer Graphics and Applications, Vol. 13, No. 1, 1993.

[43] Pham Binh, *Offset curves and surfaces: a brief survey*, Computer-Aided Design, 24: 223-229, 1992.

[44] Piegl Les, Tiller Wayne, *Algorithm for approximate NURBS skinning*, Computer-Aided Design, 28: 699-706, 1996.

[45] Piegl Les, Tiller Wayne, *Computing offsets of NURBS curves and surfaces*, Computer-Aided Design, 31: 147-156, 1999.

[46] Piegl Les, Tiller Wayne, *Parametrization for surface fitting in reverse engineering*, Computer-Aided Design, 33: 593-603, 2001.

[47] Piegl Les, Tiller Wayne, *The NURBS book*, Springer-Verlag, New York, 1997.

[48] Quarteroni Alfio, *Modellistica numerica per problemi differenziali*, Third edition, Springer-Verlag Italia, Milano 2008.

[49] Schmidt Robert, Wüchner Roland, Bletzinger Kai-Uwe, *Isogeometric analysis of trimmed NURBS geometries*, Computer Methods in Applied Mechanics and Engineering, 241-244: 93-111, 2012.

[50] Scott Michael A., Xin Li, Thomas W. Sederberg, Thomas J.R. Hughes, *Local refinement of analysis suitable T-splines*, Computer Methods in Applied Mechanics and Engineering, 213-216: 206-222, 2012.

[51] Scott Michael A., Derek C. Thomas, Emily J. Evans, *Isogeometric spline forests*, Computer Methods in Applied Mechanics and Engineering, 269: 222-264, 2014.

[52] Selimovic Ilijas, *Improved algorithms for the projection of points on NURBS curves and surfaces*, Computer Aided Geometric Design, 23: 439-445, 2006.

[53] Selimovic Ilijas, *On NURBS using tangent cones*, Computer Aided Geometric Design, 26: 772-778, 2009.

[54] Spink Mark, NURBS toolbox,
`http://www.aria.uklinux.net/nurbs.php3`,
`http://octave.sourceforge.net/nurbs/index.html`.

[55] Tierney Christopher, Nolan Declan, Robinson Trevor, Armstrong Cecil, *Using mesh-geometry relationships to transfer analysis models between CAE tools*, Engineering with Computers, 31: 465-481, 2015.

[56] Vida Janos A., Martin Ralph R., Varady Tamas, *A survey of blending methods that use parametric surfaces*, Computer-Aided Design, 26: 341-365, 1994.

[57] Vuong Ahn-Vu, Heinrich Christoph, Simeon Bernd, *ISOGAT: A 2D Tutorial Matlab Code for Isogeometric Analysis*, Computer Aided Geometric Design, 27: 644-655, 2010.