A New Contact Method for Simcenter Madymo

Contact Method based on IsoGeometric Analysis

Jingya Li

Delft University of Technology

TU Delft

SIEMENS

A New Contact Method for Simcenter Madymo

Contact Method based on IsoGeometric Analysis

by

Jingya Li

Student Name	Student Number
Jingya Li	5246091

Submitted in partial fulfilment of the requirements for the degree of
Master of Science
in
Applied Mathematics
at Delft University of Technology

Thesis Committee:	Dr. M. Möller (TUDelft supervisor)
	El Hassan Tazammourti (Siemens supervisor)
	Göktürk Kuru (Siemens supervisor)
	Prof. Dr. A. W. Heemink
Defense Date:	January 19, 2023
Faculty:	Faculty of Electrical Engineering, Mathematics and Computer Science

Style: TU Delft Report Style, with modifications by Daan Zwaneveld

TU Delft SIEMENS

Summary

Finite element analysis and multibody dynamics are popular mathematical modeling techniques for several mechanical applications. The finite element method is a costly method from a runtime standpoint. Moreover, both multibody dynamics and finite element methods require approximating the geometry of the application either by meshing or by analytical surfaces. Therefore, IsoGeometric Analysis (IGA) has been proposed as an alternate method for parameterizing geometric surfaces. IGA enables a unified mathematical representation of both geometries and solution fields using B-Splines or NURBS [12]. NURBS are the de-facto standard in modern Computer-Aided Design (CAD) workflows, but their use for analysis in finite elements is rare. By directly using the geometric information for the analysis, IGA reduces the cost of mesh generation, therefore enabling a more efficient design process.

The IGA-based rigid body contact algorithm offers several advantages over the traditional FEM-based approach. First, it provides an accurate representation of the geometry. This improves the efficiency of the design process and reduces the need for mesh refinement. Second, by looking for the local orthogonal projection in the parameter space rather than the coordinate space, the algorithm simplifies the definition of boundary conditions. This makes the process of defining boundary conditions easier and more intuitive. Third, the algorithm approximates the contact area by mapping it from the parameter space to the physical space. This allows for the specification of the level of accuracy of the contact area, which can permit more precise solutions.

The paper compares the IGA-based rigid body contact algorithm with the existing rigid FE mesh-based contact method in Simcenter Madymo. The results show that for the same level of accuracy, the IGA-based method can largely reduce execution time. This suggests that IGA could be a valuable alternative for use in the field of contact simulation.

Acknowledgement

I would like to express my sincere gratitude to my daily supervisor Dr. M. Möller for his constant support, guidance, and valuable feedback. His dedication and attention to detail have helped me to refine and improve my work. I would also like to thank Göktürk Kuru and El Hassan Tazammourti; thank you for suggesting this project topic, it's been an enlightening journey, and I appreciate your guidance throughout the process. And I am deeply grateful for the time and mentorship provided by Dr. L. Coradello and Dr. B. Pinto. I am very thankful to Prof. Dr. A. Heemink for agreeing to be a member of my thesis committee.

I want to express my heartfelt gratitude to my family and friends for their unwavering support during my master's degree and this project. Special thanks to my friends, Zhimin Cheng and Falco van der Meulen for their understanding and help. Their contributions have been invaluable to me.

Jingya Li
Delft, January 2023

Contents

List of Figures

List of Tables

Notations

Abbreviations

Abbreviation	Definition
CAD	Computer-Aided Design
CAE	Computer Aided Engineering
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
FEM	Finite Element Method
IGA	IsoGeometric Analysis
MADYMO	MAthematical DYnamic MOdels (Simcenter Madymo)
NURBS	Non-Uniform Rational B-Splines

Greek Letters

Symbol	Definition
ϵ	Radius of the neighborhood around each point
λ_{max}	Maximum penetration
λ_i	Penetration for each node
μ	Lagrange multiplier
σ_d	Damping stress function
σ_e	Elastic stress
Ω	Physical space (coordinate space)
$\Omega^{'}$	Parameter space

Roman Letters

Symbol	Definition
A	Contact area
$\hat{A_{\text{base}}}$	Base surface area
$\hat{A_{\text{net}}}$	Control net area
$\hat{A_{\text{master}}}$	Master surface area
A_i	Area for ith cluster in parameter space
B	B-spline basis functions
$\mathbb{B}$	Function space
C_d	Damping coefficient
$C(u)$	Curve evaluated at u
$\hat{C}(u)$	Curve obtained by connecting two control points P_{i-1} and P_i
C_u	Partial derivative of the master surface along u direction
C_v	Partial derivative of the master surface along v direction
F	Total contact force

Symbol	Definition
F_i	Contact force for each node
N	NURBS basis function
$\mathbf{N}$	Unit normal vector for master surface
$\mathbf{N}_p$	Unit normal vector respect to point p
P	Control points
P_{min}	Minimum number of cluster in the neighborhood
$\hat{P}$	Projection of point P
Q	Inserted control points
$S(u, v)$	Surface evaluated at u, v
U, V	Knot vectors
c_j	Centroid of cluster j
d	Dimension of the physical space
f_d	Damping amplification factor
k	Number of clusters
p, q	Degree of the basis function
u, v	Knot value in parameter space $\Omega^{'}$
w	NURBS weights

Sub- and Superscripts

Symbol	Definition
i, j, k	Indices
α	Derivative direction in parametric coordinate
t	Time step
n_c	Number of in-contact nodes
n, m	Number of basis function in u and v direction, respectively

1 Introduction

This introduction gives an overview of a master thesis that is focused on building an IsoGeometric analysis (IGA) by Hughes et al. [12] based contact method for MADYMO.

1.1. Background

MADYMO or Simcenter Madymo is an acronym for MAthematical DYnamic MOdels originated by TNO Automative in the Netherlands. Presently owned, developed, and distributed by Siemens Industry Software Netherlands BV.

The Simcenter Madymo software suite includes the Simcenter Madymo Solver, which is a powerful and flexible multi-physics simulation engine. The solver combines the capabilities of a multi-body (MB), finite element (FE), and computational fluid dynamics (CFD) in a single computer-aided engineering (CAE) solver, which enables the simulation of the complex interactions between different physics in the system during a crash event. This allows for a more accurate simulation of the behavior of the vehicle, the dummy or human body, and the environment during an impact.

Simcenter Madymo (we use MADYMO in the rest of the thesis) is a software program that is used to simulate and analyze the safety of vehicle occupants and pedestrians. It features fast and accurate simulations, tools for designing experiments and optimization studies, and a comprehensive package that includes solvers, dummy and human model processing tools, and token-based licensing.

In this project, we are more interested in the MADYMO dynamic solver module. There is a Multi-Body solver which can be used to model, for example, the crash-test dummies subsystems mechanisms. There's a Finite Element solver that is used for deformable structures.

A crucial aspect of the dynamic analysis of multibody and multiphysics solvers is the modeling of the interactions between bodies with contacting surfaces. MADYMO permits two distinct surface definitions:

- Simple 3-dimensional analytical shapes.
- Finite Element surface definitions of nodes and elements.

Fig. 8.1 is an example of modeling the interaction between bodies. In MADYMO, the software always assumes one of the surfaces to be 'deformable' and another one to be 'rigid' when modeling the interaction between bodies, also known as 'contact .'Clearly, in Fig. 8.1, the airbag is typically modeled as the deformable surface, and the human body is modeled as the rigid surface. The interaction between these surfaces is modeled using the finite element method (FEM).

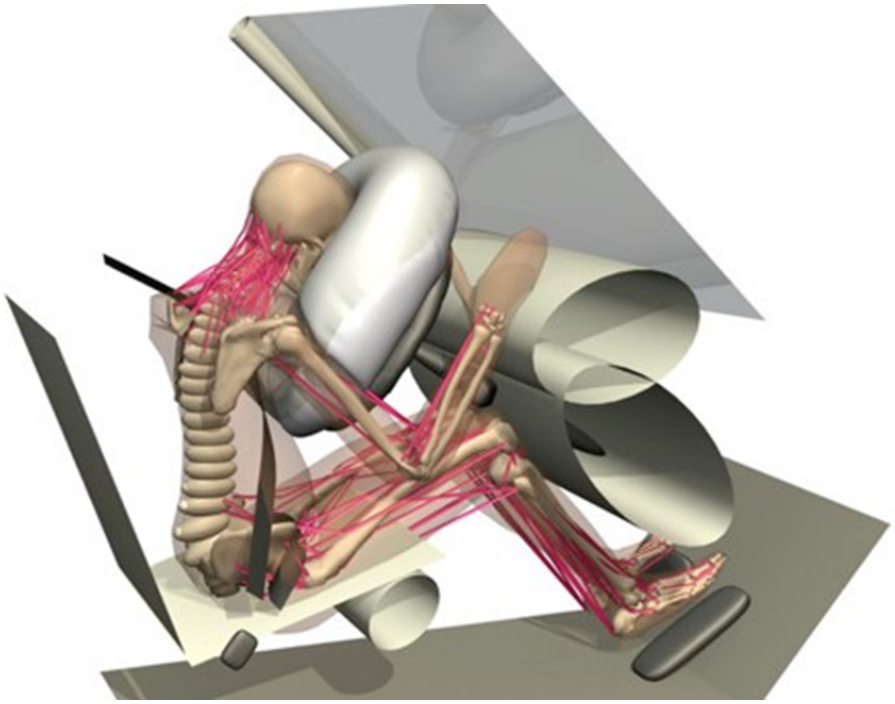


Figure 1.1: MADYMO simulation result [2].

1.2. Current Challenges

As with any simulation method, there are certain limitations to FEM, which is why it is an active area of research to improve the accuracy and efficiency of the simulation. One of the main challenges with using FEM in this context is the ability to accurately model the complex geometries and the non-linear behavior of the deformable surfaces such as airbag, seatbelt, and other restraint systems. There are, in general, three major challenges that remain in the current approaches by using the Finite Element type of surface definitions [13].

Finite Element analysis model creation is a major bottleneck in the industry. It takes a long time to generate Finite Element mesh models in the industry. The traditional workflow for the analysis of structures consists of multiple processes dominated by a CAD design, mesh generation, and then numerical analysis. It is anticipated that around 80% of the time spent on this approach is allocated to analysis-specific model preparation and mesh generation [12].

Finite Element model is not an exact realization of the geometries. It introduces geometry errors, feature removal, geometry clean-up, etc. To facilitate the generation of the computational models, many geometric simplifications, in other words, errors, and sometimes they're extremely influential. That's achieved by removing features and geometry clean-up. The severity is quite great for this problem since it's more time-consuming than the analysis.

The promise of higher-order Finite Elements has not been realized in the industry. The most widely used commercial and industrial Finite Element codes rely on the lowest-degree elements. Specifically, bilinear quadrilaterals, trilinear hexahedra, linear triangles, and tetrahedra.

1.3. Motivations

Recently, IsoGeometric Analysis (IGA) provides another approach to define the contact surface, which is an integrated way to bring exact geometry to the contact problem.

IGA is a method for the numerical analysis of partial differential equations (PDEs) that is based on the use of functions defined over non-uniform rational B-spline (NURBS) spaces. NURBS is a standard technology for representing geometric shapes in computer-aided design (CAD) systems. They are defined by a set of control points and a set of basis functions, which are used to interpolate the control points and generate a smooth curve or surface. NURBS is widely used in the field of engineering and has become the de facto standard for representing geometry in CAD systems. There are several

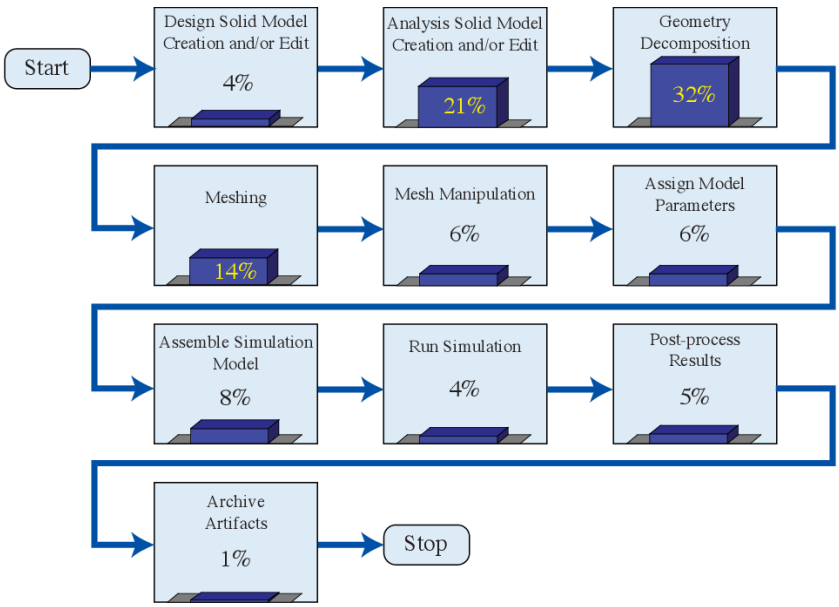


Figure 1.2: Estimation of the relative time costs of each component of Sandia National Laboratories' model creation and analysis procedure. Note that the time spent performing analysis is totally dominated by the time spent developing the model. Taken from Hughes et al. [13]

advantages to using IGA in the integration of computer-aided design (CAD) and finite element analysis (FEA).

IGA was introduced by T. J. R. Hughes and his colleagues in a series of papers published in the early 2000s [12]. IGA has several advantages over traditional FEM, including the ability to represent smooth geometric shapes more accurately, the ability to use higher-order polynomials to represent the solution to a PDE, and the ability to easily incorporate geometric continuity constraints into the analysis. Let's have a more in detailed look at these advantages:

- **Simpler modeling**: With IGA, designers can use the same B-spline or NURBS curves and surfaces that they use in CAD to define the geometry of their finite element models. This can simplify the modeling process and reduce the need for manual meshing and remeshing.
- **Higher accuracy**: IGA allows for an exact representation of geometry and offers a high level of smoothness. This is particularly useful in applications such as contact problems, shell analysis, and membrane analysis, where smoothness is important. In contrast, traditional finite element analysis (FEA) typically uses linear, quadratic, or cubic approximation of the geometry. While these approximations can provide good results in many cases, they may not be as accurate or smooth as NURBS-based representations.
- **Greater flexibility**: IGA allows designers to easily modify the geometry by moving the control points of the B-spline or NURBS curves and surfaces. This can be useful in applications where the geometry needs to be modified frequently or where it is difficult to anticipate the exact shape of the final design. In contrast to the finite element method's refinement principles, the geometry does not change during model refinement in IGA.
- **Higher order continuity**: Compared to the intrinsic C^0-continuity of finite elements, the unique k-refinement in IGA guarantees higher order continuity among the elements of a patch up to a degree of $p-1$.

The surface-based representation of geometry and the high-order approximation and continuity properties of NURBS make IGA an ideal approach for the analysis of rigid body models with curved geometries. IGA has been successfully applied in a wide range of fields, including solids and structural mechanics, fluid mechanics, and contact mechanics, among others.

In solids and structural mechanics, IGA has been used to analyze a variety of structures, including beams, plates, and shells, among others. For example, IGA has been used to analyze the dynamic

response of structures subjected to earthquake excitation [14], the vibration behavior of laminated composite plates [15], and the buckling and post-buckling behavior of thin-walled structures [25].

In fluid mechanics, IGA has been used to analyze a variety of flow phenomena, including laminar and turbulent flow, free-surface flow, and flow in porous media, among others. For example, IGA has been used to analyze the flow around an airfoil, the flow in a tube with periodic constrictions, and the flow in a microfluidic device [1] [10] [11].

In contact mechanics, IGA has been used to analyze a variety of contact problems, including frictional contact, contact with large deformations, and contact between curved surfaces, among others, [3] [23] [6] [4] [24] [20] .

MADYMO is more interested in utilizing the 'IsoGeometric' feature of IGA technique to solve the contact problem. In FEA, contact happens when two surfaces touch each other. There are two contact search algorithms that exist within MADYMO, including the intersection-based contact search algorithm and the penetration-based contact algorithm. The current contact search algorithms in MADYMO are based on the traditional FEA type of surfaces, but with the use of IGA, it's possible to model the contact between rigid NURBS-based surfaces in a more accurate and efficient manner.

Motivated by the potential benefits of IGA for contact analysis, this thesis aims to investigate the possibility of using an IGA-type approach to solve contact problems involving rigid NURBS-based surfaces.

1.4. Research Objectives

The main objective of this thesis is to investigate the possibility of using an IGA-type approach to solve contact problems involving rigid B-spline or NURBS-based surfaces. The goal is to develop a new contact algorithm that is optimized for NURBS-based surfaces and can be used in combination with IGA to improve the accuracy and efficiency of the simulation in MADYMO.

To achieve this goal, the thesis will focus on researching the current limitations of the existing contact algorithms in MADYMO, and developing new algorithms that are specifically designed to work with B-spline or NURBS-based surfaces. This research involves the implementation of the new algorithms in MADYMO, as well as the testing and evaluation of the new algorithm using a set of benchmark problems. And this thesis tries to tackle the main goal step-by-step by solving the following 'sub-goals':

- Gain familiarity with constructing NURBS analytical surfaces.
- Learn the functionality of multibody contact analysis, including the current limitations.
- Develop a novel prototype to demonstrate a NURBS-based contact implementation.
- Generate different test cases of rigid body surfaces and test the models on them.
- Evaluate the performance of the model in comparison to the existing model in MADYMO.

1.5. Structure

The thesis is structured as follows: Chapter 1 provides background information, including the existing limitations in finite element analysis, the rationale for the superiority of IGA, and the research objectives. Chapter 2 presents the principles of IGA, including the definitions of B-splines and NURBS, the representation of NURBS geometries, and the building of an approximation basis and related refinement procedures. Chapter 3 discusses the existing contact algorithm in MADYMO. Chapter 4 presents the proposed methodology for the new IGA-based contact algorithm. Chapter 5 discusses the implementation of NURBS and B-spline methods in the new contact algorithm and includes testing results. Chapter 6 compares the IGA-based contact algorithm to the existing method in MADYMO. Chapter 7 discusses the two contact points grouping algorithm. Finally, Chapter 8 discusses potential further research and provides conclusions on the use of IGA in contact algorithms.

2

IsoGeometric Analysis

This chapter includes a literature review of the principles behind Isogeometric Analysis (IGA) and focuses on the following topics:

- **B-splines and NURBS**: this chapter is going to review the theories about B-splines and NURBS first.
- **Realization of NURBS geometries**: we will discuss the methods used to construct NURBS-based geometries, such as the use of control points and knot vectors.
- **Refinement procedures**: we will introduce the knot insertion technique.

2.1. Introductory Notes

When using CAD to generate a set of smooth curves and surfaces, users have to satisfy a large number of constraints. One traditional method for satisfying these constraints is to use polynomials, but the use of polynomials requires a high degree of approximation. If users want to have a polynomial of degree p, it will have to satisfy p+1 constraints, which can be computationally inefficient, especially when trying to satisfy a large number of constraints. Even a local change can influence the stability and continuity of the global system [8].

As the degree of the polynomial increases, the number of control points needed to represent the curve or surface increases, which can lead to computational inefficiency. Additionally, the higher the degree of the polynomial, the more sensitive the curve or surface becomes to small changes in the control points, which can lead to instability in the system.

One alternative to using polynomials is to use splines, which are piecewise-defined curves or surfaces that can be used to approximate a set of data points or to represent a smooth curve or surface. Splines are constructed using polynomial elements that are joined together in a smooth way, and the level of smoothness can be controlled by the degree of the polynomials and the type of spline used.

B-splines or Basis-Spline is a special type of spline, which is defined by a set of basis functions, each of which is a polynomial of a certain degree. The smoothness of the B-spline curve or surface is determined by the degree of the basis functions, which can range from C^0 (no continuity) to C^{p-1} (continuity up to the derivative of order $p-1$). This allows designers to choose the desired level of smoothness for the curve or surface.

The starting point of the exact modeling of curves and surfaces is B-Splines. B-splines are a very useful tool for modeling curves and surfaces, but they do have some limitations. One of the main limitations of B-splines is that they cannot exactly represent some simple geometric shapes, such as circles or ellipses.

To address this limitation, a new technology called Non-Uniform Rational B-Splines was developed. NURBS is a generalization of B-splines that allows for the exact representation of conic sections (such as circles, ellipses, parabolas, and hyperbolas) as well as other curves and surfaces [8].

2.2. B-splines

One of the benefits of using B-splines is that they can achieve a high degree of smoothness with a relatively low degree of basis functions. This makes them more computationally efficient than high-degree polynomials, especially when a large number of constraints are involved. B-splines also have good local control, meaning that changes to the control points will only affect a local region of the curve or surface rather than the entire curve or surface. This can make them more stable and easier to work with than high-degree polynomials. This section will introduce the basic theories and properties behind B-splines.

2.2.1. Basis Function and Knot Vector

In 1D, a B-Spline basis of polynomial order p is defined by n basis functions $B_{i,p}, i = 1, ..., n$ in the parameter space, denoted $\Omega^{'}$. The **parameter space** is the domain in which the B-spline basis functions are defined. It is the set of all possible parameter values that can be used to define the geometry or solution of a problem. The parameter space is defined by the **knot vector**, U, which specifies the location and multiplicity of the knots in the parameter space.

$$U = u_1, \ldots, u_{n+p+1}, \quad u_1 \leq u_2 \leq \cdots \leq u_{n+p+1}, \tag{2.1}$$

A knot vector U is a vector of $n+p+1$ non-decreasing real numbers $u_i \in \mathbb{R}$ used in the definition of a spline curve or surface. We call u **knot value** and use u_i to represent the ith knot value in the knot vector U.

In the case of a spline curve, the knot vector defines the locations of the control points along the curve and determines how the curve is interpolated between those control points. The knot vector isn't strictly monotonic. Repeated knots are allowed. A knot value has a multiplicity of k means it is repeated in the knot vector k times.

Higher dimensional construction of basis is also based on the one-dimensional set-up. By using a tensor product of a one-dimensional knot vector $U \in [a, b]$, where $a, b \in \mathbb{R}$, we can have the parameter space equal to $[a, b]^d \in \mathbb{R}^d$ with d as the dimension of the space.

The following are two worth-mention properties for the knot vectors:

1. The equally separated knots are called uniform knot vectors. In contrast, the unequally spaced knots are called non-uniform knot vectors.
2. It is possible to have an open knot vector, with the first $p+1$ and last $p+1$ knots repeated.

B-spline basis functions also have a recursive definition that can be used to compute them efficiently, known as *Cox-de Boor*'s algorithm [22]. A B-spline **basis function** is a piecewise polynomial function defined over a specific subinterval of the parameter space. This definition expresses the basis function in terms of a set of coefficients and a set of lower-degree basis functions. The construction of the **B-spline basis** or the **blending functions** $B_{i,p}(u)$ starts from the piecewise constants and polynomial $p = 0$:

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

$$\text{For } p = 1, 2, 3, ... \tag{2.2}$$

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

The following figure is an example of the basis functions (Eqn. 2.2), which takes the open uniform knot vector $U = \{0, 1, 2, 3, 4, ...\}$.

The obtained B-spline curve is defined as the part in the range of d blending functions, and u values outside of the range are not used, i.e., all points on the curve are controlled by d control points. The blending functions are classical nth-degree Bernstein polynomials [5].

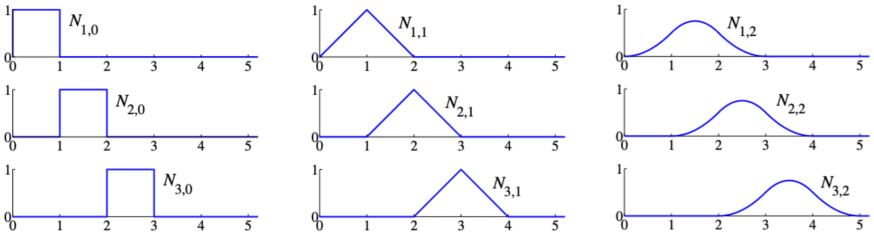


Figure 2.1: Basis function of order 0, 1, 2 for the open and uniform knot vector $U = \{0, 1, 2, 3, 4, ...\}$. Taken from Elguedj et al. [7]

The B-spline basis functions are indeed defined by the knot vector and the polynomial degree. The function space $\mathbb{B}$ of the B-splines is the space of all possible B-spline curves that can be generated by varying the control points. According to function 2.2, given a knot vector U and a polynomial degree p, the function space for the B-splines is defined as:

$$\mathbb{B} \equiv B(U; p) := \text{span}\,\{B_{i,p}\}_{i=1}^{n} \tag{2.3}$$

With the help of *Cox-de-Boor* formula. A higher-dimensional B-Spline realization can be obtained using the tensor product:

$$\mathbb{B} \equiv B(U, V, \ldots; p, q, \ldots) := \text{span}\,\{B_{i,p} \otimes B_{j,q} \otimes \ldots\}_{i,j,\ldots=1}^{n,m,\ldots} \tag{2.4}$$

We define V as the knot vector in v-direction with the polynomial order equal to q, with m basis functions $B_{j,q}, j = 1, ..., m$.

The basis function features the following noteworthy properties:

- **Non-negativity.** $B_{i,p}(u) \geq 0, \forall i, p$. The basis functions are non-negative functions and have compact support.
- **Partition of unity.** $\sum_{i=1}^{n} B_{i,p}(u) = 1$. The summation of all basis functions is equal to 1.
- **Recursive definition (known as Cox-de Boor's algorithm).** $B_{i,p}(u) = (1-u)B_{i,p-1}(u) + uB_{i-1,p-1}(u)$

2.2.2. B-Spline Curve

The B-spline curve is defined as a linear combination of B-spline basis functions. As we stated in the previous session, the degree of the basis function, denoted by p, determines the smoothness of the curve, with a higher degree resulting in a smoother curve. The dimensionality of the control points, denoted by d, determines whether the curve is one-dimensional (curve) or two-dimensional (surface). The **control points**, denoted by P_i, are used to determine how much each basis function contributes to the final shape of the curve. And $a, b \in \mathbb{R}$ are bounds for function space $\mathbb{B}$.

The B-spline curve is defined as:

$$C(u) = \sum_{i=0}^{n-1} P_i B_{i,p}(u), \quad a \leq u \leq b \tag{2.5}$$

The curve denoted by $C(u)$ is a function with a knot value that only takes in one parameter in one dimensional situation. It maps from the parameter space $\Omega^{'}$ to the physical space Ω.

It is possible to compute the derivative of a curve by using the derivatives of the basis functions.

$$C'(u) = \sum_{i=1}^{n} \frac{dB_{i,p}(u)}{du} P_i \quad a \leq u \leq b \tag{2.6}$$

Here are some useful properties for the B-Spline curves:

- **B-spline curves inherit all of the properties from the basis function.** This means that the smoothness, continuity, and other properties of the curve are determined by the properties of the B-spline basis function.

- The **convex hull property** states that a B-spline curve is contained within the convex hull of its control polyline. This means that the curve will always stay within the boundaries defined by the control points, regardless of the shape of the curve. This property is a result of the non-negativity of the B-spline basis function, which ensures that the curve stays within the bounds defined by the control points [9]. More specifically, if u is in the knot span $[u_i, u_{i+1}]$ then $C(u)$ lies within the convex hull spanned by the control points $P_{i-p}, \ldots, P_i$. In Fig. 2.2, Q_0, Q_2, Q_3, Q_5, and Q_7 are control points.

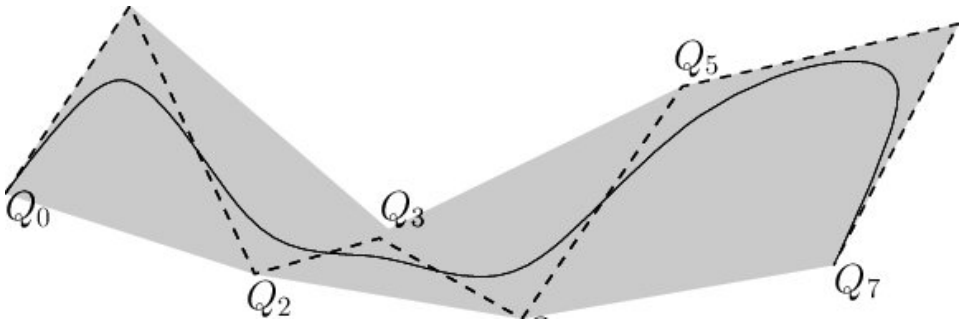


Figure 2.2: Convex Hull property for B-Spline, Q_0, Q_2, Q_3, Q_5, and Q_7 are control points. Taken from Guilbert et al.

- **Variation diminishing property**, also known as the local control point property. It states that as the degree of the B-spline curve is increased, the curve will always stay within the convex hull of its control points [17]. In other words, the B-spline curve will never move outside of the area defined by its control points. The variation diminishing property is a direct result of the fact that the B-spline basis functions have compact support and non-negative values, which means that the B-spline curve is a linear combination of the control points with non-negative coefficients. So as the degree of the B-spline curve is increased, the coefficients of the control points in the linear combination can only decrease, which means that the curve cannot move outside the convex hull of the control points.

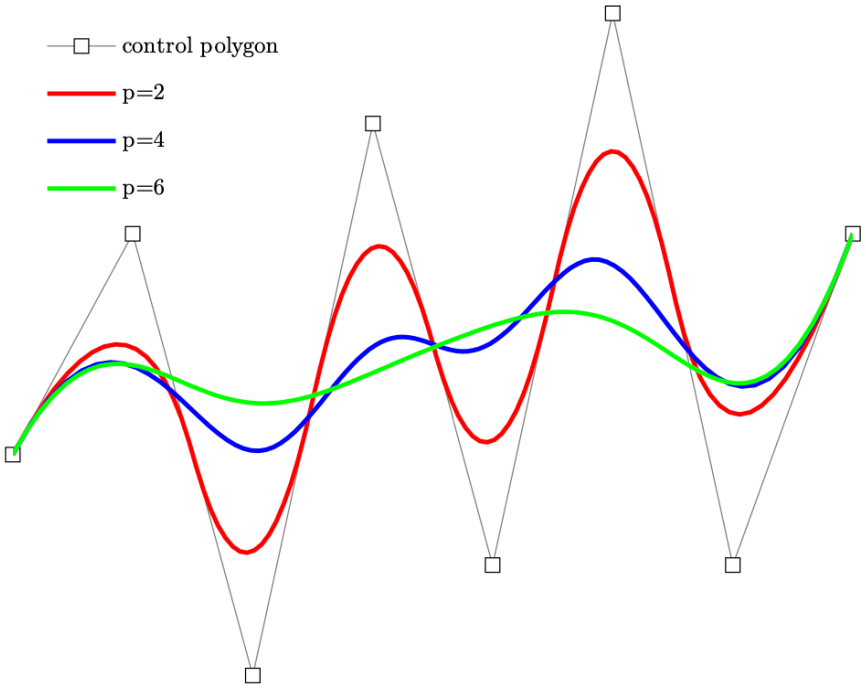


Figure 2.3: As the degree of the curve rises, the variation diminishing property is depicted.

- **Affine preserve property.** This property states that if the control points of a B-spline curve or surface are transformed using an affine transformation (such as a translation, rotation, or scaling), then the resulting curve or surface will also be transformed by the same affine transformation.

2.2.3. B-Spline Surface

The construction of the B-Spline surface is similar to the B-Spline curve. Instead of having only one knot vector, it takes two knot vectors and spans to a surface. The two knot vectors are represented as $U = \{u_1, u_2, \ldots, u_{n+p+1}\}, V = \{v_1, v_2, \ldots, v_{m+q+1}\}$. In the case of a spline surface, the knot vectors define the locations of the control points in the u and v directions and determine how the surface is interpolated between those control points.

The B-Spline surface is described as $S : \Omega^{'} \to \Omega$, with the map defined as taking the tensor product of the basis functions $B_{i,p}(u)$ and $B_{j,q}(v)$.

$$S(u,v) = \sum_{i=1,j=1}^{n,m} B_{i,p}(u) B_{j,q}(v) P_{i,j}. \tag{2.7}$$

2.3. NURBS and NURBS Geometries

One of the main benefits of NURBS is that they allow for the exact representation of a wide range of geometric shapes, which makes them very useful in computer-aided design (CAD) and other applications where precise geometry is important. NURBS also have the advantage of being easy to modify by moving the control points, which allows designers to easily change the shape of the curve or surface while maintaining its smoothness.

Non-uniform rational B-splines (NURBS) are a generalization of B-splines that permit the exact geometric representation of entities such as circles, spheres, and tori that B-splines cannot represent. These conic geometries can be derived from B-splines using a piecewise projective transformation of the B-spline curve that results in rational functions. The following is the definition of the NURBS basis function $N_{i,p}(u)$:

$$N_{i,p}(u) = \frac{w_i \cdot B_{i,p}(u)}{\sum_{j=0}^{n} w_j \cdot B_{j,p}(u)}, \tag{2.8}$$

where $B_{i,p}(u)$ is the B-spline basis function of degree p, w_i is the ith weight associated with the ith control point P_i, and u is the knot value.

The NURBS surface, can obtained from the multivariante NURBS basis function. In analogy to equation 2.7, multivariate NURBS basis functions are created by a d-dimensional tensor product. As an illustration, the 2D NURBS basis function is defined as follows:

$$N_{i,j}^{p,q}(u,v) = \frac{w_{i,j} \cdot B_{i,p}(u) \cdot B_{j,q}(v)}{\sum_{k=0}^{n} \sum_{l=0}^{m} w_{k,l} \cdot B_{k,p}(u) \cdot B_{l,q}(v)}, \tag{2.9}$$

where $B_{i,p}(u)$ is the one-dimensional B-spline basis function of degree p in the u-direction, $B_{j,q}(v)$ is the one-dimensional B-spline basis function of degree q in the v-direction, and $w_{i,j}$ is the weight associated with the control point $P_{i,j}$.

By evaluating the NURBS basis function at a grid of parameter values, designers can generate a smooth, continuous surface that can be easily modified by moving the control points.

The NURBS basis possesses all the qualities of B-spline basis functions, such as partition of unity, continuity, and non-negativeness.

Analogy to the B-spline curve, it is easy to conclude the definition of the NURBS curve:

$$C(u) = \sum_{i=0}^{n} P_i \cdot N_i^p(u), \tag{2.10}$$

The construction of a NURBS surface is also similar to the construction of a B-spline surface, with the exception that NURBS use a set of NURBS basis functions instead of B-spline basis functions. By taking the tensor product with two knot vectors in different directions, we can get the expression of NURBS surface, namely

$$S(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} P_{i,j} \cdot N_{i,j}^{p,q}(u,v) \tag{2.11}$$

2.4. NURBS and B-Splines Refinement

In order to capture the landscape and the analysis the variation of the geometry, it is crucial to consider the refine discretizations.

An interesting property of the B-Spline is that the refinement of the basis does not change the corresponding B-Spline geometry and its parameterization. There are three refinement methods for both B-Spline and NURBS, including knot insertion, order elevation, and $k-$refinement.

Designers can use these methods to tailor the level of accuracy and smoothness of B-splines and NURBS to the needs of their specific application. These techniques are frequently used in CAD and other applications where precise geometry is required.

We only used the knot insertion refinement technique in this project. As a result, we will only address the knot insertion in this section.

Knot insertion is a process of inserting an additional one or more knots into the original knot U vector to generate a refined knot vector $\bar{U}$. By use of the refined knot vector, we could have the refined B-Spline basis corresponding to the new knot vector. This also increases the number of basis functions and control points, which can result in a higher level of detail in the curve or surface.

Recall that to initialize a B-Spline curve, and the following information is needed:

- Degree p: The degree of the B-spline curve determines the smoothness and continuity of the curve. Higher-degree curves can achieve a higher level of smoothness and continuity, but they also require more control points and may be more difficult to modify.
- Knot vector U: The knot vector defines the parameterization of the curve, and it determines the number and placement of the knots. The spacing and arrangement of the knots can have a significant impact on the shape of the curve.
- Control points $P_i, (0 \leq i \leq n)$: The control points help to determine the shape of the curve.

To insert a knot t into a B-spline curve with a knot vector U, designers can follow these steps:

- Determine the index i of the knot span $[u_i, u_{i+1})$ that contains t. This can be done by finding the highest index i such that $u_i \leq t < u_{i+1}$.
- Initialize a new set of control points $Q_0, Q_1, \ldots, Q_n$ as follows:
  - For $j = 0, 1, \ldots, i-p-1$, set $Q_j = P_j$.
  - For $j = i-p, i-p+1, \ldots, n$, set $Q_j = P_{j-1}$.
- For $r = 1, 2, \ldots, x$, do the following:
  - For $j = i-p+r-1, i-p+r, \ldots, n-r$, compute:

$$Q_j = \frac{t - u_j}{u_{j+p+1} - u_j} Q_{j-1} + \frac{u_{j+p+1} - t}{u_{j+p+1} - u_j} P_j$$

- Update the knot vector U by inserting t

Consider the following knot vector with order = 3:

$$U = [0, 0, 0, 0, 1, 1, 1, 1]$$

Two examples of knot insertion are given in the rest of the subsection. The first example is inserting one knot $u_i = 0.50$ into the original knot vector U. First, we need to determine the index i of the knot span that contains u. In this case, the highest index such that $u_i \leq t < u_{i+1}$ is $i = 4$.

Then, initialize a new set of control points $Q_0, Q_1, \ldots, Q_n$ as follows:

- For $j = 0, 1, 2, 3$, set $Q_j = P_j$.
- For $j = 4, 5, 6, 7$, set $Q_j = P_{j-1}$.

Insert the knot u into the new knot vector $\hat{U}$ by adding it $x = 1$ times:

$$\hat{U} = [0, 0, 0, 0, 0.50, 1, 1, 1, 1]$$

Since we are inserting the knot only once, $r = 1$ and we can compute:

$$Q_3 = \frac{t - u_3}{u_{3+p+1} - u_3} Q_2 + \frac{u_{3+p+1} - t}{u_{3+p+1} - u_3} P_3 = \frac{0.5 - 0}{1 - 0} P_2 + \frac{1 - 0.5}{1 - 0}$$

Here is the implementation result for inserting $u = 0.5$ into the knot vector:

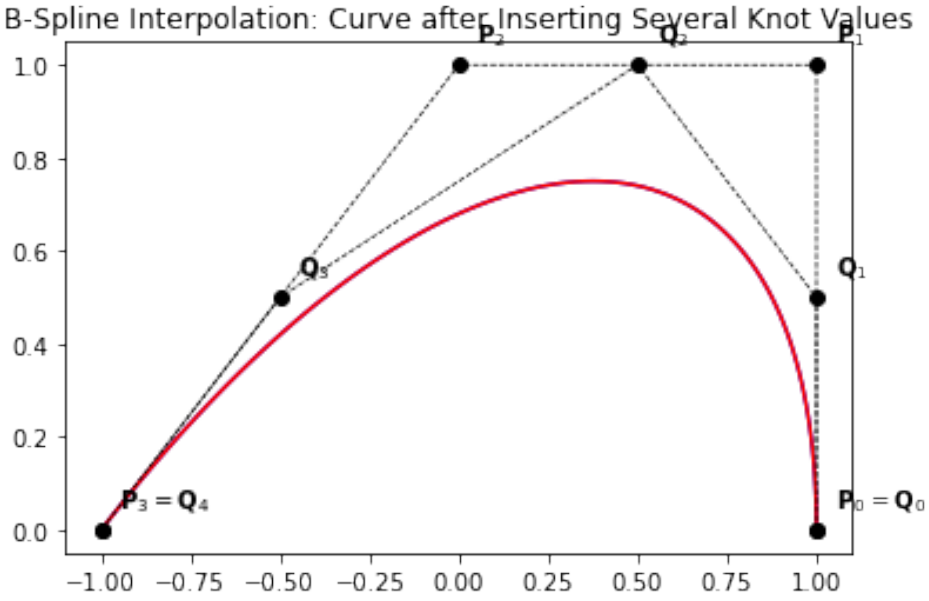


Figure 2.4: Knot Vector after Inserting Knot Value 0.5

As shown in Fig 2.5, it is possible to insert multiple knots into a B-spline curve by repeating the process of knot insertion multiple times. Designers can insert as many knots as they want as long as they follow the correct steps and update the knot vector and control points accordingly.

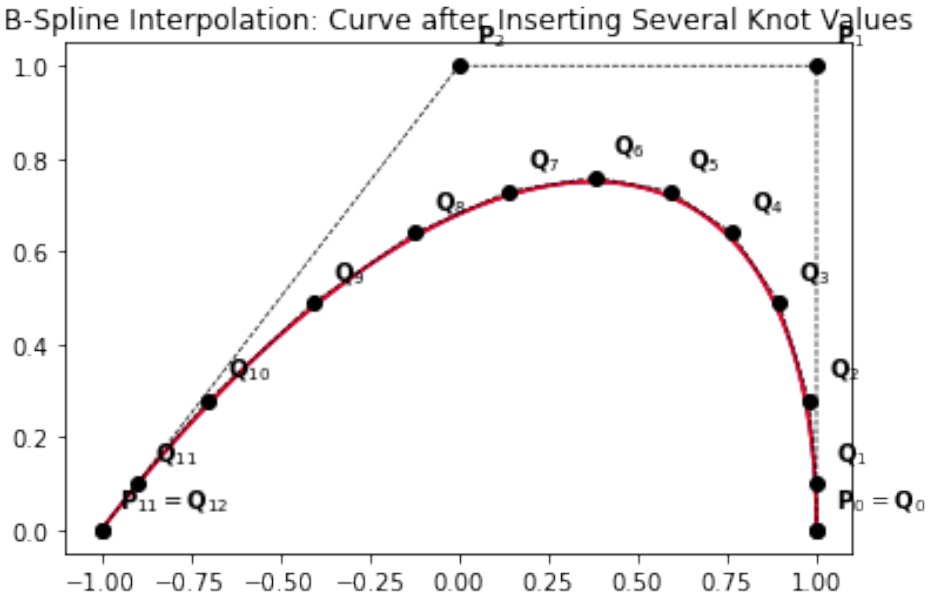


Figure 2.5: Inserting Several Knot Values into the Knot Vector

Or, it is also possible to add a new knot that is equal to the existing knot values. In this case, the multiplicity of that knot is increased by one.

3

Contact Theory in MADYMO

This chapter provides a brief overview of the fundamentals of the contact search method used in MADYMO, as well as a discussion of the derived contact force model. The materials discussed in this chapter are based on the MADYMO Theory Manual [19], which is a comprehensive reference to the principles and algorithms used in MADYMO.

3.1. Contact Search Algorithm based on Intersections

When two separate surfaces interact with each other, they are considered to be in 'contact.' In MADYMO, the contact search algorithms aim to deal with the contact problems, they search for contacts between a **master surface** and a **slave surface**. The master surface is defined as a group of contact segments. The slave surface is defined as a point cloud.

Let's keep using the example from MADYMO in Chapter 1, Fig. 8.1 showcases a contact event between the human body and the airbag. As we stated in the introduction, the airbag is a deformable surface, so it can be regarded as a deformable point cloud. Then the human body in this example is the master surface.

Two kinds of contact search algorithms are available in MADYMO: the intersection based contact search algorithm and the penetration based contact search algorithm. So far, this project only discusses the intersection based contact search algorithm.

The intersection based contact search algorithm used in MADYMO is composed of two phases: the search phase and the detailed search phase.

The **search phase** calculates the intersection between each of the slave nodes and the master segment at the current time step. Figure 3.1 illustrates the procedure of the search phase. The positions of both the contact node and contact segment of the current and previous time steps are used. Accordingly, all of the new contacts in the current time step are detected.

Figure 3.1 depicts a slave point, denoted by '$\circ$,' penetrating a master surface composed of four segments that are spanned by five vertices '$\bullet$.' The grey area roughly represents the contact area, which is not an accurate representation of the contact area and is only meant to demonstrate the procedure of the search phase. The algorithm checks the point position in the current step during the search phase. If the current point is in the contact area, the orthogonal distance between the point and the master surface is calculated. For example, if the current time step is $t = 1$, the slave node in the previous time step $t = 0$ is not in contact with the master surface. As a result of the search algorithm, the penetration needs to be calculated and stored in the memory.

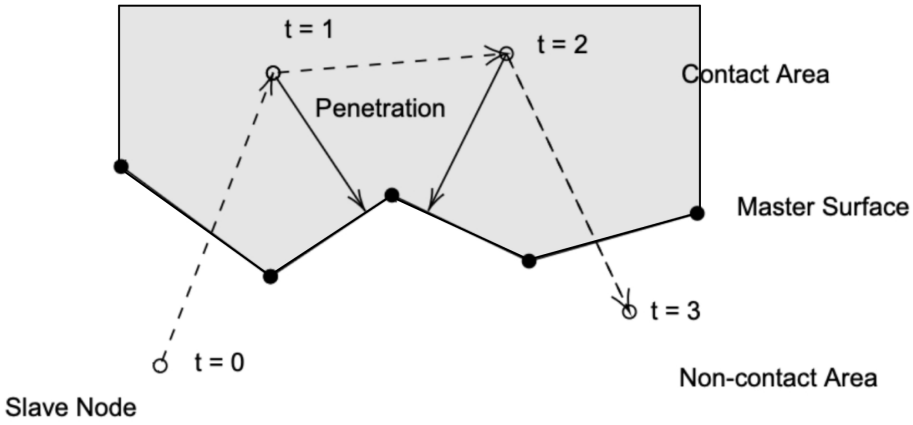


Figure 3.1: Slave Node Penetrating Master Surface

The **detailed search phase** checks both the old contacts (all of the contacts that occurred in the earlier time step) and the new contacts (contacts that were found in the search phase).

Now consider the previous time step to be $t = 2$ and $t = 3$ (Fig. 3.1). Each contact is checked to see if the slave node is still penetrating the master segment. In $t = 3$, the slave node is in the non-contact area, so no penetration with the master surface takes place. So this contact is removed from the storage.

Additionally, the detailed search phase also checks the contact projection to see if it stays within the contact segment.

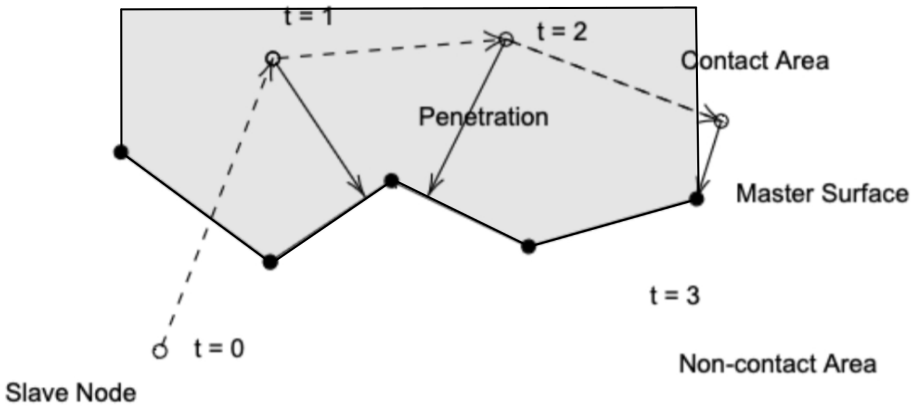


Figure 3.2: Slave Nodes without Orthogonal Projection within the Contact Segment

At $t = 3$, in figure 3.2, the point has no orthogonal projection within the segment, so it maps to the closest segment.

3.2. Elastic Characteristic Based Contact Force Model

MADYMO supports three contact force models: penalty-based contact, adaptive contact, and elastic characteristic based contact. Elastic characteristic based contact is only available for the intersection based contact algorithm.

The elastic characteristic based contact force model also provides three possible contact types:

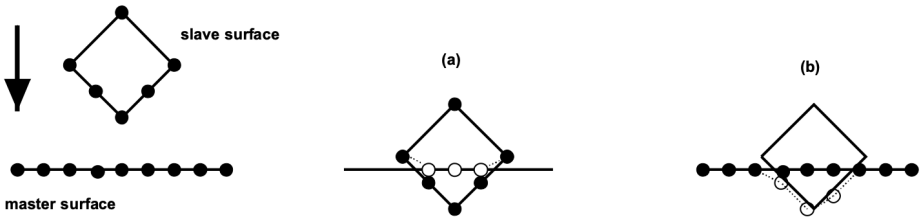


Figure 3.3: Contact type: (a) slave surface deformed, (b) master surface deformed. From MADYMO theory manual.

- **Slave:** the slave surface is treated as 'deformed' and the master surface is 'rigid'. Figure 3.3(a).
- **Master:** the master surface is treated as 'deformed', and the slave surface is assumed to be 'rigid,' Figure 3.3(b).
- **Combined:** the slave and master surface are treated as 'deformed.' An intermediate surface is calculated between the master and slave surface that these penetrations are used for the calculation of the contact forces.

The contact area of the 'deformed' surface is taken. And a constant contact area is assigned to each of the deformed nodes.

To calculate the contact force, only knowing the contact types is not enough. In the elastic characteristic-based contact force model, two methods are available for force calculation: **contact model force** and **contact model stress**. The first type of force calculation method does not work with the third contact type (contact type combined).

3.2.1. Contact Model Force

For the **contact model force** method, we take the force as a function of the penetration depth. The total contact force F only depends on the maximum penetration depth λ_{max} of all contacting nodes n_c. Γ is a function that calculates the total contact force F through the maximum penetration depth λ_{max}, which can be specified by users.

$$F = \Gamma\left(\lambda_{\text{max}}\right) \tag{3.1}$$

where

$$\lambda_{\text{max}} = \text{max}(\lambda_1, \lambda_2, ..., \lambda_{n_c}) \tag{3.2}$$

The process calculates the penetration depth for n_c number of nodes that come into contact with the master surface, then obtain the maximum penetration depth λ_{max} and uses it as an input for the function Γ to get the total contact force F.

Then the contact force per contact node F_i is defined by:

$$F_i = \left(\frac{F}{\sum_{j=1}^{n_c} \lambda_j A_j}\right) \lambda_i A_i \tag{3.3}$$

where λ_j is the penetration depth corresponding to the jth contact node and A_j is the contact area with respect to the jth contact node.

F only depends on the maximum penetration of all contact nodes, which means it is independent of the number of nodes in contact and the total contact area.

3.2.2. Contact Model Stress

In the **contact model stress**, the contact force for each contact node is given by:

$$F_i = A_i \left[\sigma_e \left(\frac{\lambda_i}{t_i} \right) + \left[C_d \frac{\dot{\lambda}_i}{t_i} + \sigma_d \left(\frac{\dot{\lambda}_i}{t_i} \right) \right] f_d \left(\sigma_e \right) \right] \tag{3.4}$$

The elastic contact stress is taken from the stress-penetration/thickness characteristic $\sigma_e \left(\frac{\lambda_i}{t_i} \right)$, with t as the thickness of the master surface.

There are two methods to calculate the master surface thickness t. One is to use the element thickness associated with the penetrated slave node. In another method, the thickness is calculated based on the position of the penetrating slave node. In this project, we don't consider the volumetric case, so we choose constant thickness information for the stress model.

The second term $C_d \frac{\dot{\lambda}_i}{t_i}$ in Eqn. 3.4 represents the damping contact stress. C_d is a constant damping coefficient. σ_d is a damping stress function, which can account for non-linear damping. f_d is the damping amplification factor that is a function of the elastic stress σ_e.

In the stress model, the total contact force $F = \sum_{j=1}^{n_c} F_j$ depends on the contact area. Since the damping term requires thickness information, we don't consider the damping term as well.

4

Methodology

The employed and developed approaches are discussed in this chapter. Section 4.1 begins with an overview of the modeling procedure, followed by an outline of the five main challenges in this problem. In section 4.2, we will go over how to solve the challenges and propose an algorithm with four general steps. Detailed illustrations for the algorithm are provided in sections 4.3, 4.4, and 4.5. A flowchart is used to summarize the algorithm in section 4.6.

4.1. Modeling Overview

This project aims to find a way to utilize the IGA method to deal with an elastic characteristic-based contact problem in order to bypass the 'bottleneck' in the industrial design process (FEA-type shape modification).

Problem setup: Recall the definition of the contact problem. The contact happens when two surfaces 'touch' each other. In chapter 3, solving the contact problem requires two steps: the **search algorithm** and the **force calculation**.

The definition and choice of the primary and secondary surfaces are essential aspects of both steps. So here comes the definition and choice of the primary and secondary surfaces.

- **Master surface (primary surface):** An IGA-based rigid surface. With no initial contact force or velocity assign to it.
- **Slave surface (secondary surface):** A 'deformed' point cloud. With contact force or velocity being assigned to each point.

The fundamental setup is the same as the **contact type slave** in Chapter 3. An illustration of the contact type is given in Figure 4.1:

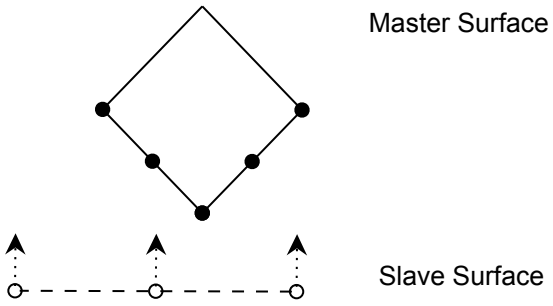


Figure 4.1: Contact Type: Slave Surface Deformed with Velocity or Force Information

As the slave points penetrate the master surface, the search algorithm finds minimum orthogonal projections between the slave node and the master segment shown in Figure 4.2.

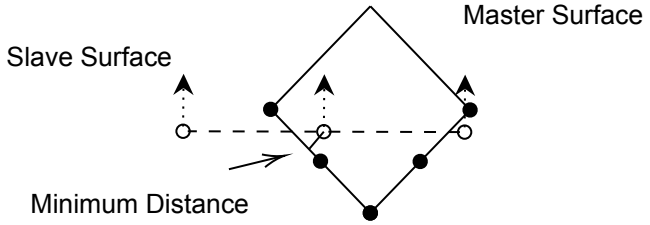


Figure 4.2: Find the Minimum Distance between the Slave Point and Master Surface

Given the minimum orthogonal distance for each of the slave nodes in Figure 4.2, it is easy to obtain the maximum penetration distance. Thus, referring to Eqn 3.1, the total contact force can be computed.

However, if we keep tracking the global minimum distance, the projection might shift to another side of the master surface, which is far from realistic, e.g., in Fig 4.3, the local minimum is not equal to the global minimum. Out of the consideration of accuracy, the algorithm will track the local minimum orthogonal projection instead of the global minimum orthogonal projection.

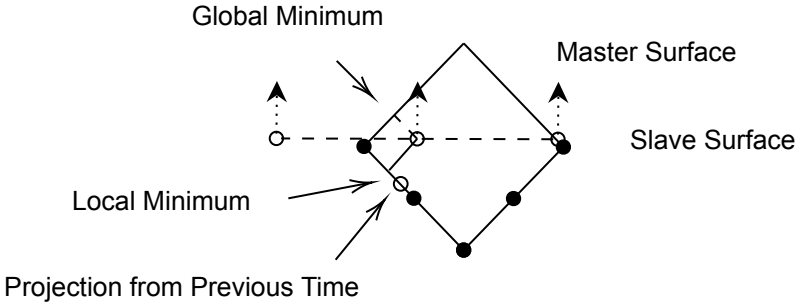


Figure 4.3: Local Minimum and the Global Minimum Orthogonal Projection

Based on the illustration above, when transforming this FE-based rigid body contact method to the IGA method, the following challenges should be solved:

- Generate the NURBS master surface (curves and surfaces).
- Distinguish the contact and not-in-contact side of the master surface, and group slave points into in-contact and not-in-contact group.
- Find the initial guesses for the orthogonal projections.
- Find the local minimum orthogonal projection of penetrated slave points.
- Approximate the contact area for force calculation.

4.2. Algorithm Benchmark

We aim to carry out the FE-based rigid body contact model to the IGA framework and calculate the contact force for each slave node as an equation (Eqn 3.3).

To tackle the challenge stated above, we propose the following approval:

- Master surface generation. Use B-spline and/or NURBS to represent the master surface.
- Specify the in-contact and non-contact sides of the master surface. Determine which side of the master surface the slave points are on. Group the slave points into in-contact and non-contact groups accordingly.
- Perform the detailed search for all in-contact points and remove the point that is no longer in contact with the master surface.
- Search for local minimum orthogonal projection points between in-contact slave points and master surface.
- Get the approximation for the contact area.
- Calculate the contact force for each slave node in different time steps.

Based on these steps, our algorithm consists of four parts: **surface creation**, **search algorithm**, **detailed search phase**, and **force calculation phase**.

Surface creation: We start by building the NURBS shape for the master surface and generating the point cloud as a slave surface. In Chapter 2, a detailed illustration of both the B-spline and NURBS surface creation has been provided. So we skip the surface creation step in this section.

4.3. Search Algorithm

There are two phases for the search algorithm: **slave points grouping** and **point projection**.

4.3.1. Slave Points Grouping

The unit normal vector, **N**, is a vector that points in the direction perpendicular to a surface. In the context of the slave points grouping phase, the unit normal vector is used to determine which side of the master surface the slave points are on. The aim of the **slave points grouping** phase is to check whether the slave point is in contact with the master surface or not. In other words, this phase is to check which side of the master surface that the slave points lie in. In the context of the slave points grouping phase, the unit normal vector is used to specify the **in contact side** and the **non-contact side** of the master surface. The unit normal vector **N** is given by:

$$\mathbf{N} = \frac{C_u \times C_v}{|C_u \times C_v|} \tag{4.1}$$

By default, the unit normal vector points towards to the **non-contact side**. So the slave points that lie in the **non-contact side** are **non-contact points**. Otherwise, they are **in-contact points**. We define C_v and C_u to be the partial derivatives of the master surface C with respect to v and u.

This division of the slave points into two groups can be used to improve the accuracy of the alignment process since it allows the algorithm to focus on the points that are in contact with the surface.

4.3.2. Point Projection

We do the in-contact and non-contact grouping in every time step, then switch to the **point projection** phase. For searching the local minimum projections of the **in-contact** slave nodes P onto the master surface C. So let's we first formulate the minimization problem:

$$\begin{array}{ll} \min_{u_{i,t}} & \left\|u_{i,t} - u_{i,t-1}\right\|^2 \\ \text{s.t.} & \left(C\left(u_{i,t}\right) - P\right) \cdot \left(C\left(u_{i,t}\right) - P\right) = 0 \end{array} \tag{4.2}$$

where $u_{i,t}$ is ith knot value at tth time step.

Rewrite the problem using the Lagrange multiplier μ:

$$L(\mu, u) = \left\|u_{i,t} - u_{i,t-1}\right\|^2 + \mu\left(C\left(u_{i,t}\right) - P\right) \cdot \left(C\left(u_{i,t}\right) - P\right) \tag{4.3}$$

The minimum value attains when

$$\frac{dL(\mu, u)}{du} = 0 \tag{4.4}$$

Instead of minimizing the Euclidean distance on the master surface in the physical space, we choose to minimize the knot value in the parameter space. Then with the knot value $u \in U^{n,p}$ and $v \in V^{m,q}$, we can evaluate the point coordinate S along the master surface C using Algorithm 1.

Algorithm 1 Compute NURBS coordinates

Input: n: number of control points in u-direction -1
m: number of control points in u-direction -1
p, q: degree of the basis function in u and v direction, respectively
U, V: knot vector in u and v directions, respectively
u, v: knot values
C: NURBS surface
Output: S: point coordinate

```
procedure SurfacePoint(n, m, p, q, U, V, u, v, C)
    uspan = FindSpan(n,p,u,U) ▷ Find the knot span of the given knot value u. This algorithm can be found in [21].
    vspan = FindSpan(m,q,v,V)
    Nu = size(u)
    Nv = size(v)
    ComputeBasisPolynomials(uspan, p, u, U, Nu)  ▷ Compute the B-Spline basis polynomials. This algorithm can be found in [21].
    ComputeBasisPolynomials(vspan, q, v, V, Nv)
    for i = 0 → q do
        temp[i] ← 0
        for j = 0 → p do
            temp[i] = temp[j] + Nu[j] * C[uspan - p + j][vspan - q + i]
        end for
    end for
    Sw = 0
    for i = 0 → q do
        Sw = Sw + Nv[i] * temp[i]
    end for
    S = Sw/w
end procedure
```

For the minimization problem, we perform the Newton iteration method until the objective function converges within the tolerance

$$\epsilon = 10^{-3}.$$

The knot value at i-th Newton iteration is denoted by u_i. Then for the knot value at $i+1$th Newton iteration:

$$u_{i+1} = u_i - \frac{f(u_i)}{f'(u_i)} = u_i - \frac{C'(u_i) \cdot (C(u_i) - P)}{C''(u_i) \cdot (C(u_i) - P) + |C'(u_i)|^2} \tag{4.5}$$

We keep proceeding with the Newton iteration until the objective function reaches the tolerance (Eqn 4.5).

For the Newton method, the **initial guesses** should be provided for each iteration. There are two situations:

a. For slave points that are in-contact with the master surface for the first time. We calculate the initial guess based on the movement information.
b. For slave points that are already labeled as in-contact in the previous timestep and are still in-contact for the current timestep. We use the projected value from the previous time step as the initial guess.

Here is a detailed explanation for the type-a initial guess. Suppose the position for a slave point at time step $t = i - 1$ is P_{i-1}, and the position at the current time step i is P_i. Assuming that the contact happens at time step i, we can obtain a B-spline by using P_{i-1} and P_i as control points. Then we calculate the intersection between the connected line and the NURBS segment as the initial guess for the Newton iteration, Fig 4.4.

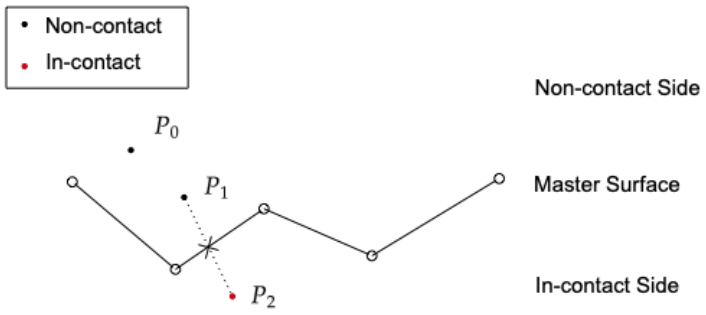


Figure 4.4: Finding the initial guess for type-a in-contact nodes.

For finding the initial guess, let $\hat{C}(u)$ be the B-spline obtained by connecting P_{i-1} and P_i, and calculate the intersection between $\hat{C}(u)$ and $C(u)$, which can be done by solving the minimization problem.

$$\begin{aligned} \min_u \quad & \left\| \hat{C}(u) - C(u) \right\|^2 \\ \text{s.t.} \quad & u \in [0,1] \end{aligned} \tag{4.6}$$

Based on the above minimization problem, we can get the initial guess for the Newton iteration in the 1D case. In the 2D case, we should get the iso-curve for the surface first, then use the iso-curve and $\hat{C}(u)$ as the inputs of this minimization problem.

4.4. Detailed Search

After completing the **search algorithm** phase, we proceed to the **detailed search phase**. We need to search again among all of the in-contact points to make sure they still interact with the master segment.

The contact action can be terminated in two ways: (a) sideway sliding; (b) points go out of the boundary. By checking for these two conditions at each time step, we can determine whether the ”in-contact” points are still interacting with the master segment and whether the contact action should be terminated.

4.4.1. Sideway Sliding

Sideway sliding: If the ”in-contact” points have gone sideways such that the orthogonal projections lie beyond the edge of the master surface.

We check for sideway sliding by determining if the orthogonal projection still exists within the master surface. If not, the points will be removed from the in-contact group.

4.4.2. Out-of-Boundary Check

Points go out of the boundary: If the ’in-contact’ points move outside of the boundary of the surface, the contact action can be terminated. This may occur if the points have no orthogonal projection onto the surface. Or, let $\hat{P}$ be the projection of P onto the master surface and $\mathbf{N}_p$ be the outward normal of the master surface at point $\hat{P}$. The point P is no longer in contact if

$$(\hat{P} - P) \cdot \mathbf{N}_p < 0 \tag{4.7}$$

In both cases, P will be removed from the in-contact group.

4.5. Force Calculation

The force calculation step is the same as the existing algorithm in Chapter 3 except for approximating the contact area.

By running the search algorithm at every time step, we have the projection information for every in-contact slave node. We can obtain μ and get the corresponding total force by finding the maximum distance among the projections.

We produced a more realistic representation of the contact area than MADYMO. In the force computation phase, we define two steps: **contact area approximation** and force model. To begin, we will approximate the overall area of the master surface by tessellating the IGA master surface as a FEM surface.

4.5.1. Contact Area Approximation

In the MADYMO model, the constant area of each control point is used in the force calculation step. We aim to provide a more reasonable approximation of the contact area. To achieve this, we proposed a map of the contact area between the parameter space and the physical space. There are, in general, three steps to approximate the contact area:

- **Tessellate the master surface.** Get the total area of the master surface $\hat{A}_c$ in coordinate space.
- **Cluster the in-contact points.** To get rid of the gaps between groups of in-contact points.
- **Approximate the in-contact clusters using convex hull.** Use the area of each convex hull as the contact area.

Tessellate the master surface. Given a group of m control points $P_1, P_2, ..., P_m$, and the IGA surface C. We define the base surface by connecting P_1 and the P_m.

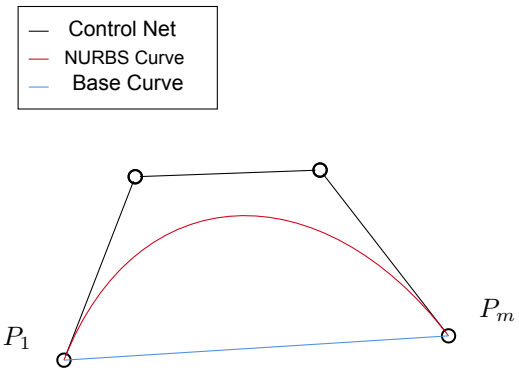


Figure 4.5: Tessellate the IGA surface by using the base surface B.

We were inspired by the fact that the area of the FEM surface is simple to calculate through tessellating the control net Algorithm 2. As a result, we use the base surface to approximate the IGA surface using FEM and define the area of the base surface to be $\hat{A_{\text{base}}}$. And the area of the control net to be $\hat{A_{\text{net}}}$.

At the end of the tessellation process, the algorithm is able to achieve $\hat{A_{\text{net}}} \approx \hat{A_{\text{master}}} \approx \hat{A_{\text{base}}}$ by performing a so-called master surface split process.

There are several different techniques that can be used to split a NURBS or B-spline surface, depending on the application's specific requirements. Some common methods for splitting a NURBS surface include:

- **Using isoparametric curves**: Isoparametric curves are curves on a surface that have a constant value of one of the surface parameters (u or v). By intersecting a NURBS surface with an isoparametric curve, it is possible to split the surface into two separate surfaces [21].
- **Using a trimming curve**: A trimming curve is a curve on a surface that defines a boundary between two regions of the surface. By using a trimming curve, it is possible to split a NURBS surface into two separate surfaces along the curve [18].
- **Using a surface cutting plane**: A surface cutting plane is a plane that intersects a surface at a specific angle. By using a surface cutting plane, it is possible to split a NURBS surface into two separate surfaces along the intersection line [18].

Recall from Chapter 2, that curves and surfaces can be refined without changing the geometric shape using the **knot insertion** approach. One way to split a NURBS surface is by using the knot insertion technique, which is a method for refining the surface by adding knots to the knot vectors. This is similar to the first surface-splitting technique mentioned above, which involves using isoparametric curves to intersect the surface and split it into two separate surfaces.

In the knot insertion technique, knots are added to the knot vectors in a specific pattern based on the desired level of refinement. By adding knots to the knot vectors, we can increase the number of control points, which can result in a more refined, smoother surface.

By using the knot insertion technique, it is possible to split a NURBS surface into multiple separate surfaces, each with its own set of control points and knot vectors.

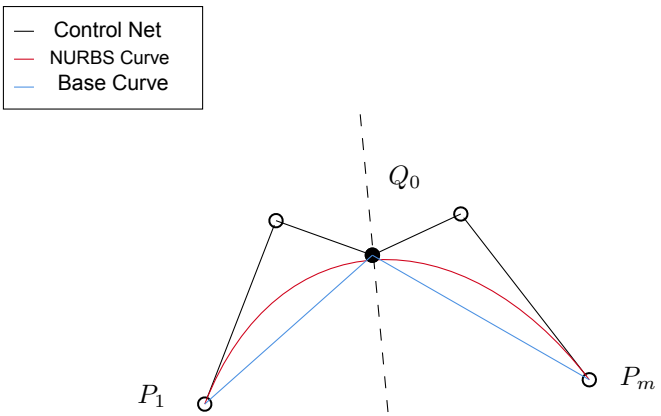


Figure 4.6: Split the IGA surface at the newly inserted control point Q_0.

We determine the area of an IGA surface's control net first, then utilize the base surface to tessellate the IGA surface using the knot insertion technique. And start calculating the area of the IGA surface by tessellating the control point net.

Algorithm 2 Tessellate Control Point Net

Input: n: number of control points in u-direction -1
m: number of control points in u-direction -1
P: control net
Output: FEM mesh for P

```
procedure TessellateCPNet(n, m, P)
    nVertice = (n+1) × (m+1)                ▷ Number of vertices of the final tessellated surface
    nTriangle = n × m × 2                   ▷ Number of triangles of the final tessellated surface
    for j = 0 → m do
        for i= 0 → n do
            verts[v,:] = P[:,i,j]  ▷ Build the vertex matrix by stacking the control net in the u-direction for each column.
            vertsId[i,j] = v
            v+=1
        end for
    end for
    t = 0
    for j = 0 → m do                        ▷ Generate triangle in u-direction for each column.
        for i = 0 → n do
            v1 = vertId[i, j]
            v2 = vertId[i + 1, j]
            v3 = vertId[i + 1, j + 1]
            v4 = vertId[i, j + 1]
            triangles[t, :] = [v1, v2, v3]
            triangles[t + 1, :] = [v1, v3, v4]
            t+=2
        end for
    end for
end procedure
```

With the tessellated control net, we can calculate the control net area and the base area through Algorithm 3 and 4, respectively.

Algorithm 3 Split Master Surface at Specified u-Direction

Input: u: specified knot value
p: degree of the basis function
U: knot vector
P: control net
Output: U1, U2: knot vector
P1, P2: control net
procedure SplitMasterSurface(u, p, U, P)
    n = length(P) - 1 ▷ Number of control points - 1
    k, s = FindSpanMultiplicity(n, p, u, U) ▷ Find multiplicity of a knot.This algorithm can be found in [21].
    if s >= p **then**: ▷ Insert knot p - s times and update knot vector and control points.
        Unew = U
        Pnew = P
    else
        Unew, Pnew = KnotInsertion(n, p, U, P, u, p - s) ▷ Control points and knot vectors.
        U1 = Unew[:, k- s + 1] ▷ Knot vectors after insertion.
        U2 = Unew[k- s:]
        m = n + p + 1
        U1[-1] = u ▷ Split at u
        U1[:-1] = Unew[:k + (p - s) + 1]
        U2[0] = u ▷ Split at u
        U2[1:] = Unew[k - s + 1:]
    end if
end procedure

Algorithm 4 Calculate the Area of Control Point Net and the Base Surface

Input: n: number of control points in u-direction -1
m: number of control points in v-direction -1
P: control net
Output: $\hat{A_{\text{net}}}$, area of the control net; $\hat{A_{\text{base}}}$, area of the base surface
procedure AreaCPNetandBaseNet(n, m, P)
    verts, triangles = TessellateCPNet(n, m, P) ▷ Refer to Algorithm 2
    nTriangles = shape(triangles)
    for j = 0 $\rightarrow$ nTriangles **do**
        v1 = verts[triangles[i,0]]
        v2= verts[triangles[i,1]]
        v3 = verts[triangles[i,2]]
        $\hat{A_{\text{net}}}$ +=0.5 $\times$ norm(cross(v2 - v1, v3 - v1))
    end for
    $\hat{A_{\text{base}}}$ = (0.5 * norm(cross(P[:, n-1, 0] - P[:, 0, 0], P[:, n-1, m-1] - P[:, 0, 0])) + 0.5 * norm(cross(P[:, n-1, m-1] - P[:, 0, 0], P[:, 0, m-1] - P[:, 0, 0])))
end procedure

According to Fig 4.6, we first split the IGA surface at Q_0. Then, calculate $\hat{A_{\text{net}}}$ and $\hat{A_{\text{base}}}$. By recursively inserting the new point Q_i to the surface and doing the surface split at Q_i until:

$$||\hat{A_{\text{net}}} - \hat{A_{\text{base}}}|| < tol, \tag{4.8}$$

where tol is the manually specific tolerance value. By specifying tol, we can get the FEM approximation $\hat{A_{\text{base}}}$ of the IGA surface as a desired level of accuracy.

Get the knot value for in-contact projection. This procedure has already been discussed during the **search algorithm** step in Section 4.3.

Cluster the projected knot value within parameter space. So far, we have the control net area, the base surface area, and the knot value for all of the in-contact points for the contact area approxima-

tion step. One issue remains to be resolved. If there are numerous clusters of in-contact point groups, calculating the total area without accounting for the gaps between each pair of clusters is inaccurate.

We used the k-mean clustering algorithm to divide the in-contact points into different clusters. The k-means clustering algorithm is an iterative method that involves alternating between two steps: (1) assignment step and (2) update step [26].

In the assignment step, each in-contact point is assigned to the closest centroid. In the update step, the centroids are updated to be the mean of all the points assigned to them. This process is repeated until convergence, that is until the centroids do not change anymore.

Once the clusters have been identified, we can calculate the total area by summing up the areas of each cluster, accounting for the gaps between the clusters. This ensures that the contact area is accurately calculated, taking into account the gaps between different clusters of in-contact points.

Suppose there are k clusters in-contact points. Define the k-mean clustering objective function J:

$$min_{u,v} J = \sum_{j=1}^{k} \sum_{i=1}^{n} \left\| C(u_i^j, v_i^j) - c_j \right\|^2, \tag{4.9}$$

where c_j is the centroid for cluster j, which are random selected for k clusters.

Given the parameter space clusters for in-contact points and the area in the parameter space for each cluster $\hat{A}_i$ can be calculated using Jarvis' convex hull algorithm [16].

The k-mean clustering algorithm requires providing the number of clusterings in each timestep, which mean we have to predetermine the number of clusters first. In contrast, the DBSCAN algorithm is a density-based clustering algorithm that works by identifying areas of high density and grouping points within those areas into clusters. It is considered to be a more robust algorithm than k-means clustering, as it does not require the number of clusters to be specified beforehand and is able to handle noise and outliers more effectively [26]. However, it may be less accurate in certain cases, as it relies on the definition of a density threshold to determine which points belong to a cluster. We will discuss this in Chapter 7.

Algorithm 5 Approximate the Contact Area using Convex Hull [16]

Input: S: point projections in parameter space.
Output: P, the set of points that form the convex hull. The final size is i.

```
1:  procedure ConvexHull(S)
2:      PointOnHull = leftmost point in S
3:      P[i] = PointOnHull Endpoint = S[0]        ▷ Initial endpoint for a candidate edge on the hull.
4:      for j = 0 → len(S) and endpoint ≠ P[0] do
5:          if (endpoint == pointOnHull) (S[j] is on left of line from P[i] to endpoint) then
6:              endpoint = S[j]                   ▷ Found greater left turn, update endpoint
7:              i = i + 1
8:              PointOnHull = endpoint            ▷ Wrapped around to first hull point
9:          end if
10:     end for
11: end procedure
```

Mapping the area to the coordinate space.

After the clustering and the convex approximation, we are able to calculate $\hat{A}_i$, the area for every cluster in coordinate space using the $\hat{A_{\text{master}}}$ in parameter space, which equals 1 and the approximated master surface area in coordinate space $\hat{A_{\text{base}}}$. The area of each cluster A_i in parameter space is depicted in Fig 4.7. Therefore, the area of each cluster in coordinate space $\hat{A}_i$ can be expressed as:

$$\hat{A}_i = \frac{A_i}{\hat{A_{\text{master}}}} \times \hat{A_{\text{base}}}, \text{ with } i \in (0, k] \tag{4.10}$$

Here is an illustration for mapping the area from the parameter space to the coordinate space.

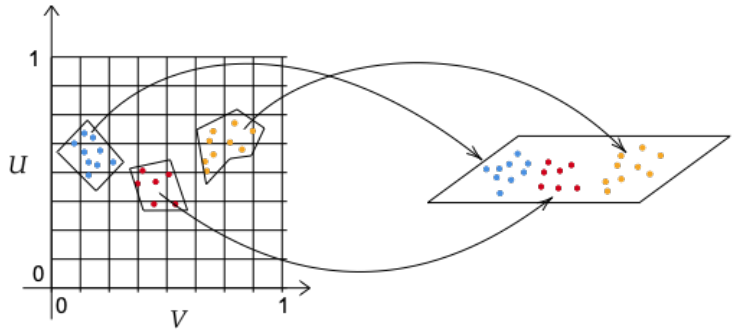


Figure 4.7: Clustering and mapping the area from parameter space to the coordinate space.

4.5.2. Contact Model

There are still two types of contact models in the IGA-based contact algorithm: **contact model force** and **contact model stress**. The contact force and stress computations are the same as in MADYMO, which can be found in Chapter 3, and will not be repeated here.

4.6. Summary

To summarize, the IGA-based contact method contains four phases: the surface creation phase, the search algorithm phase, the detailed search algorithm phase, and the force calculation phase. To begin with, it takes as input the rigid IGA master surface and a point cloud describing the slave surface, then splits the slave points into an in-contact and a non-contact group by calculating the normal vector of the master surface.

Then for the search algorithm phase, the local minimum orthogonal projections are computed for points in contact with the master surface. The out-of-boundary check and the sideway sliding check are applied to each slave node to ensure that it is still in contact with the master surface, which is the detailed search phase. For the force calculation phase, the area of the master surface is then calculated by approximating the IGA surface with a tessellated FEM surface. It searches the parameter space for numerous clusters of contact point groups in order to close the gaps between each pair of clusters. The contact area for each cluster in the coordinate space is then determined by multiplying the assumed base surface area. Finally, it sends all distance and area data to the stress-based model or maximum penetration and area data to the force-based model.

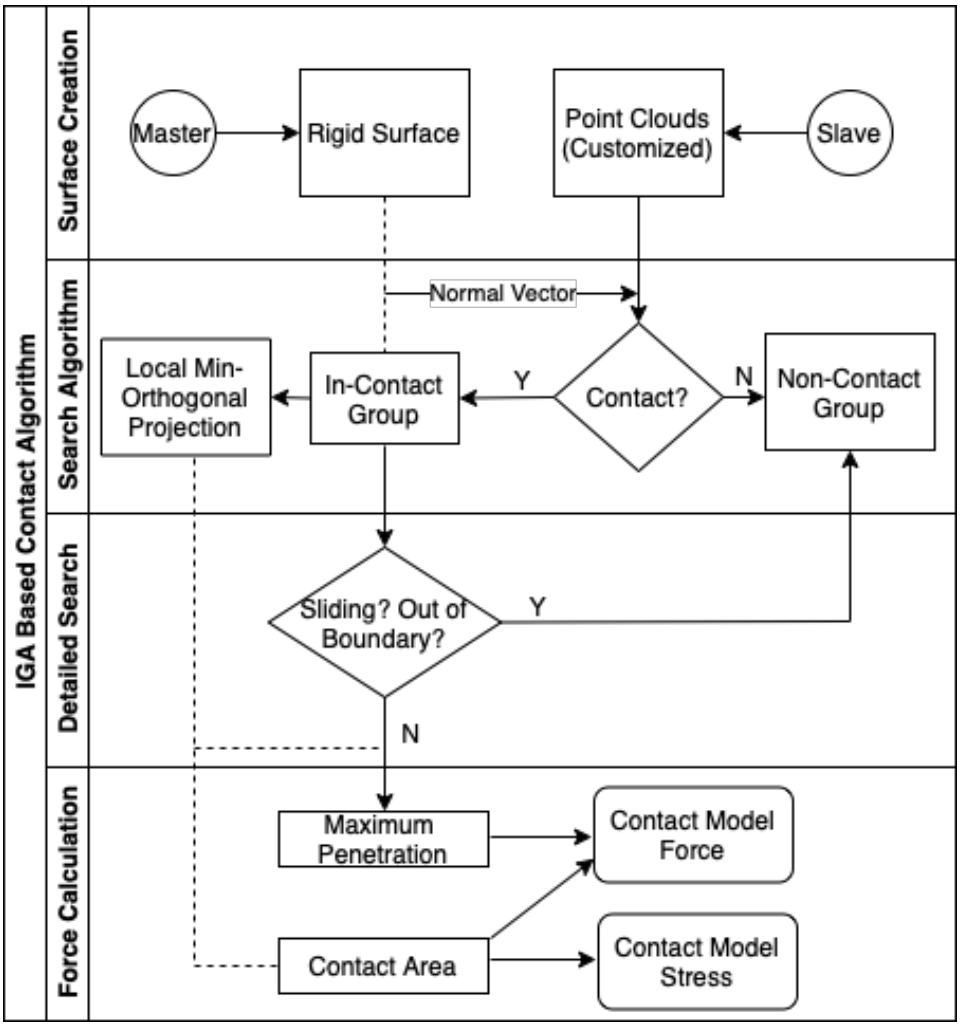


Figure 4.8: Flowchart for the IGA-based Contact Algorithm

5 Results

In the methodology chapter, we illustrated that the algorithm needs four general steps: surface creation, search algorithm, detailed search, and force calculation. In the result chapter, we aim to illustrate the results following the order of these four steps.

5.1. Surface Creation

There are two surfaces required within the algorithm, the rigid master surface and the slave point clouds. For the master surface, two types of surface definition are allowed in the algorithm, namely, B-splines and NURBS.

5.1.1. Surface Creation: B-Splines Implementation

Using B-splines as the surface definition for the master surface, one needs to specify a set of control points and basis functions that define the shape of the surface. The control points are points in space that are used to define the overall shape of the surface, and the basis functions are used to determine how the curve should be shaped between the control points.

B-Spline Surface 1D (Curve)

We start by introducing the B-spline implementation. In a one-dimensional case, it would be simply the curve.

Figure 5.3 is an illustration of a quadratic B-Spline curve with ten control points. The control points are shown in figure 5.1.

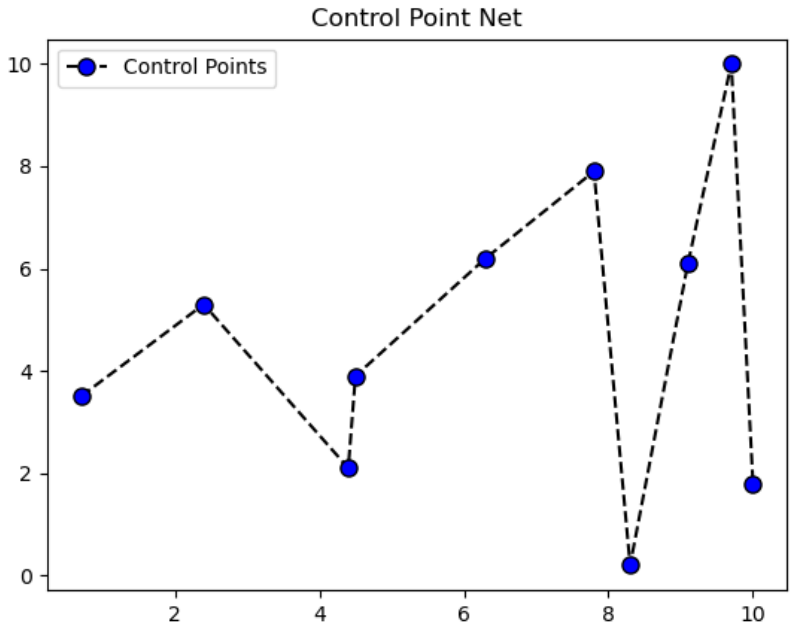


Figure 5.1: Control point net, including ten control points.

The quadratic curve is generated through the uniform open knot vector

$$[0, 0, 0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1, 1, 1].$$

With the help of the basis function (Fig 5.2) in the parameter domain Ω' together with the control net P (Fig 5.1), taking the linear combinations of the basis and the control net, results in the B-spline curve Fig 5.3.

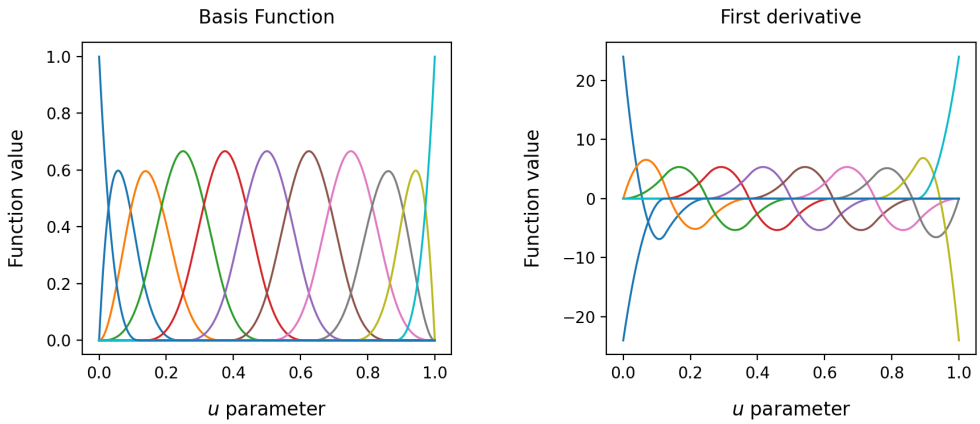


Figure 5.2: Basis function and its first and second derivatives.

In Fig 5.2, an implementation for the cubic basis function with maximum index 9, counting from 0. Since the degree of the polynomial is three, the first-order derivative shows a smooth transition between segments.

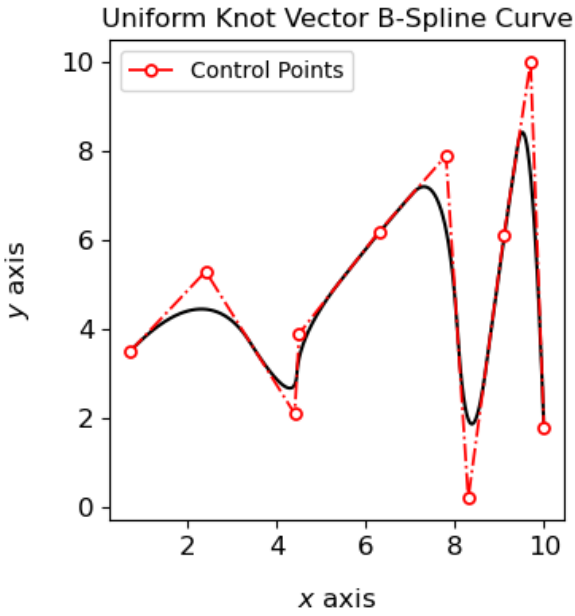


Figure 5.3: Example of B-Spline curve with control points and control polygon.

In figure 5.3, the red points are the control points. The control points at the two ends of the curve are interpolatory due to a p-fold knot repetition. A p-fold knot repetition refers to the fact that the knots (which define the locations of the basis functions) are repeated p+1 times at the ends of the curve. This can be used to ensure that the curve passes through the endpoints of the curve, making the endpoints interpolatory (i.e., the curve passes through the control points at the ends) [13]. The red dashed control polygon results from linear interpolation between the control points and tangential orientation at the polygon endpoints. The resulting B-spline curve is self-contained within the convex hull of its local control polygon, as required by the strong convex hull property.

B-Spline Surface 2D

A B-spline surface is a surface defined in a similar way as the B-spline curve. Recall that the construction of the B-Spline surface requires two knot vectors to span the surface.

$$U = [0, 0, 0, 0, 0.5, 1, 1, 1, 1],$$

$$V = [0, 0, 0, 0, 1, 1, 1, 1].$$

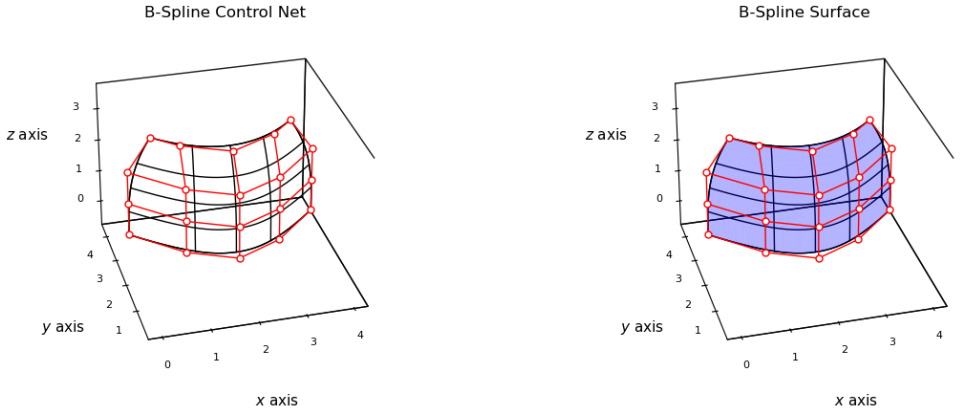
The following plot shows a B-Spline surface and its control net. It takes 20 points as the control points. The coefficient of the control point is the product of two one-dimensional B-spline basis functions, one in the u-direction and another in the v-direction.

5.1.2. Surface Creation: NURBS Implementation

In this section, we use NURBS as surface definition for the master surface. To have a NURBS-based master surface, it requires a set of control points and basis functions.

One key difference between B-splines and NURBS is that NURBS curves are rational, which means that they can represent more complex shapes than B-splines, since they can take into account the weight of the control points as well as their position. This allows NURBS to more accurately represent shapes that have variable thicknesses, such as tubes or bottles.

NURBS Curve



(a) Control Polygon **(b)** B-Spline Surface

Figure 5.4: Control Point Net and B-Spline Surface

The same as B-spline, one dimensional NURBS surface is the NURBS curve. Here is an example for the NURBS curve with a non-uniform knot vector:

$$u = [0, 0, 0, 0.125, 0.3, 0.375, 0.5, 0.6, 0.7, 0.875, 1, 1, 1]$$

Then the following Fig 5.5 illustrates the basis function and its first order derivative.

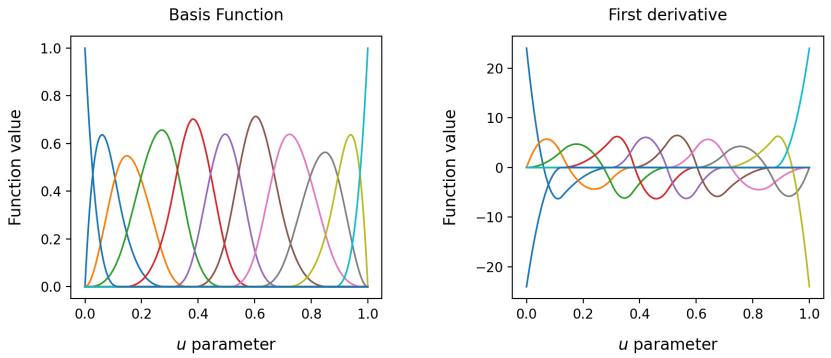
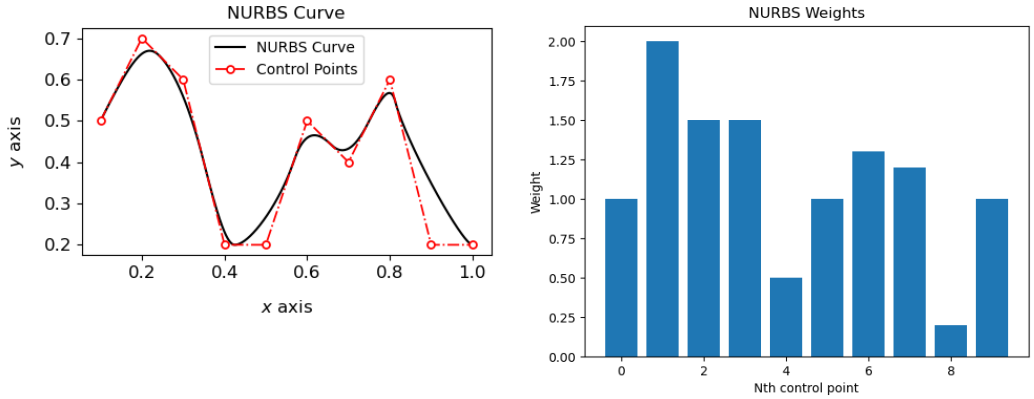


Figure 5.5: Basis function and its first order derivative for NURBS curve.

The construction of the NURBS curve requires the definition of the weights w. Fig 5.6 and Fig 5.7 give the weighted control polygon. It is important to note that the simplest version of a NURBS curve is a B-spline curve with all weights equal to 1. This means that a NURBS curve with all weights equal to 1 will have the same shape as a B-spline curve with the same control points.

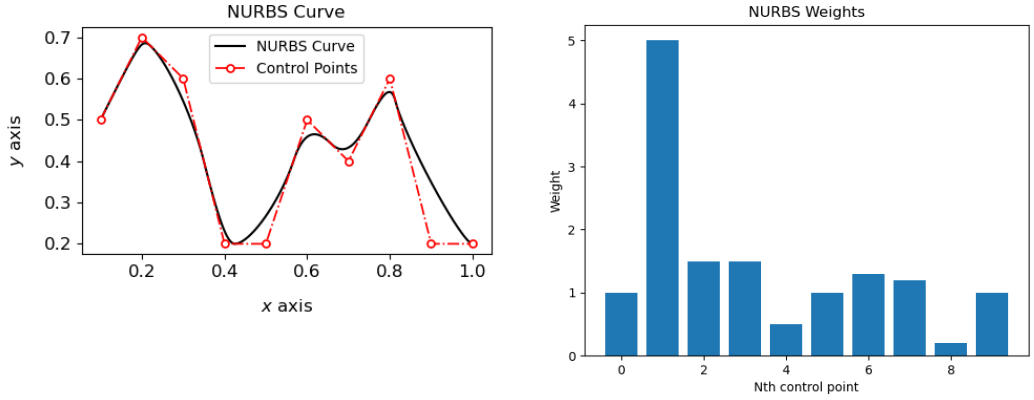
The weights can be used to make the curve more flexible, allowing it to represent more complex shapes than a B-spline curve with the same control points. Increasing and decreasing the weights w_i will increase and decrease the value of $N_{i,p}(u)$, respectively. In Fig 5.6, the three higher weights (2, 1.5, 1.5) in the right plot pull the curve toward the corresponding control points.



(a) NURBS Curve **(b)** Weight Vector: $[1, 2, 1.5, 1.5, 0.5, 1, 1.3, 1.2, 0.2, 1]$

Figure 5.6: NURBS curve and its weight vector.

As we adjust the weight for the second control point from 2 (Fig 5.6b) to 5 (Fig 5.7b), we can see that the weights of the control points have a strong influence on the shape of the curve, and the curve has been ”pulled” towards that control point by comparing the second point in Fig 5.7a with the second point in Fig 5.6a.



(a) NURBS Curve **(b)** Weight Vector: $[1, 5, 1.5, 1.5, 0.5, 1, 1.3, 1.2, 0.2, 1]$

Figure 5.7: NURBS curve and its weight vector.

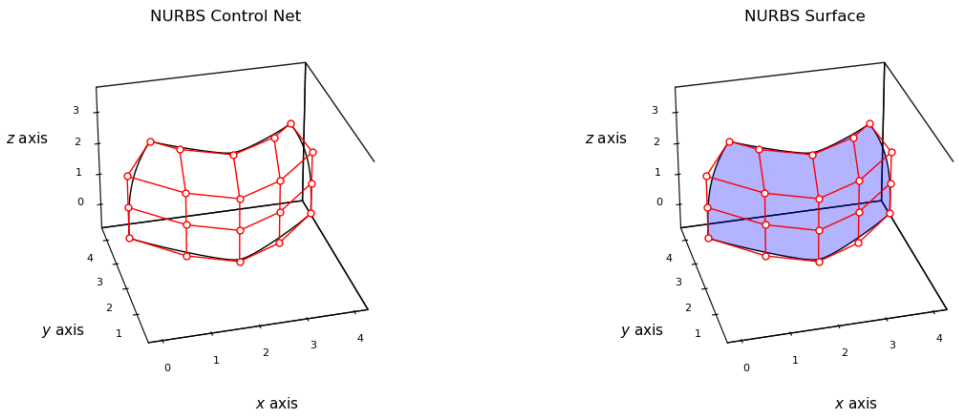
Therefore, Fig 5.6 and Fig 5.7 give an illustration of how the NURBS allows for a great deal of flexibility in curve and surface design.

NURBS Surface

It is natural to come up with constructing the NURBS surface. A NURBS surface is defined using a set of control points and two knot vectors, one for the u-direction and one for the v-direction. In the following example, we used the same knot vectors as the B-spline example:

$$V = [0, 0, 0, 0, 1, 1, 1, 1],$$
$$U = [0, 0, 0, 0, 0.5, 1, 1, 1, 1].$$

The dot in the control net indicates the control points, and the blue surface in the right plot is the NURBS surface.



(a) Control Net

(b) Generated NURBS Surface

Figure 5.8: Control Points Net and NURBS Surface

5.1.3. Surface Creation: Slave Surface

Recall the slave surface is a deformed point cloud, it can be generated from the NURBS and/or B-spline shapes. An easy way to generate the slave point cloud with a custerimized shape could be sampling a group of points along the IGA surface.

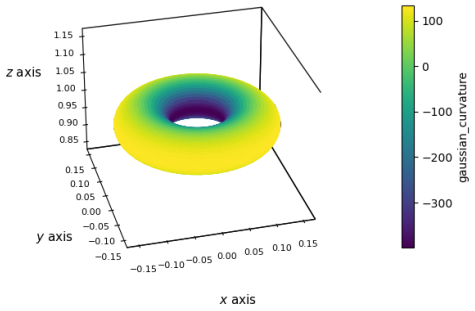


Figure 5.9: NURBS Torus

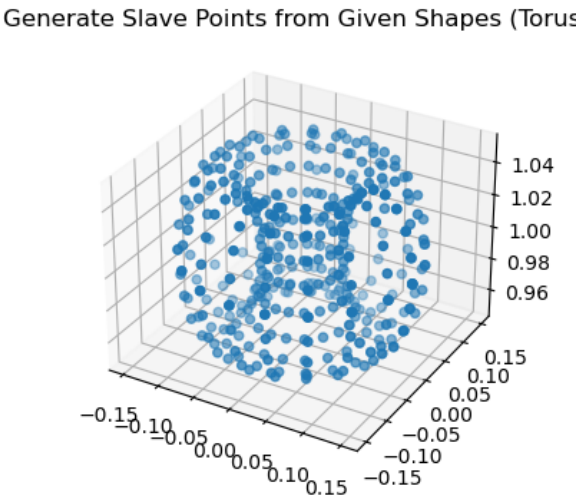


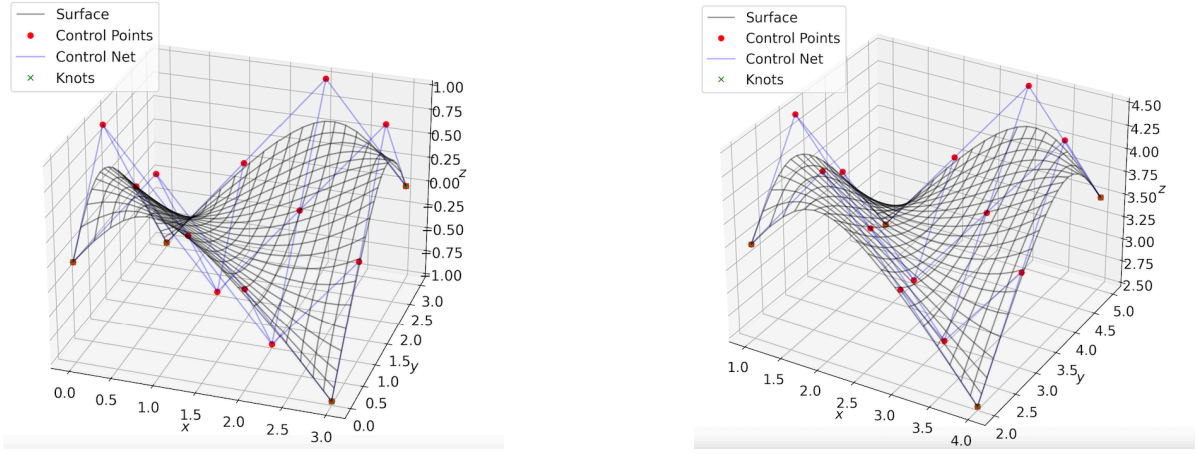
Figure 5.10: Slave Points Sampled from NURBS Torus Surface

Fig 5.9 gives a torus shape created through the NURBS surface. Sampling 10 knots in the u-direction and 10 knots in the v-direction and taking a tensor product between these two knot vectors gives the slave point cloud with a torus shape (Fig 5.10).

5.1.4. Common Geometric Transformations

Both B-spline and NURBS curves and surfaces are invariant under common geometric transformations, such as translation, rotation, and scaling. This means that if we apply one of these transformations to

a B-spline curve or NURBS surface, the shape of the curve or surface will not change.

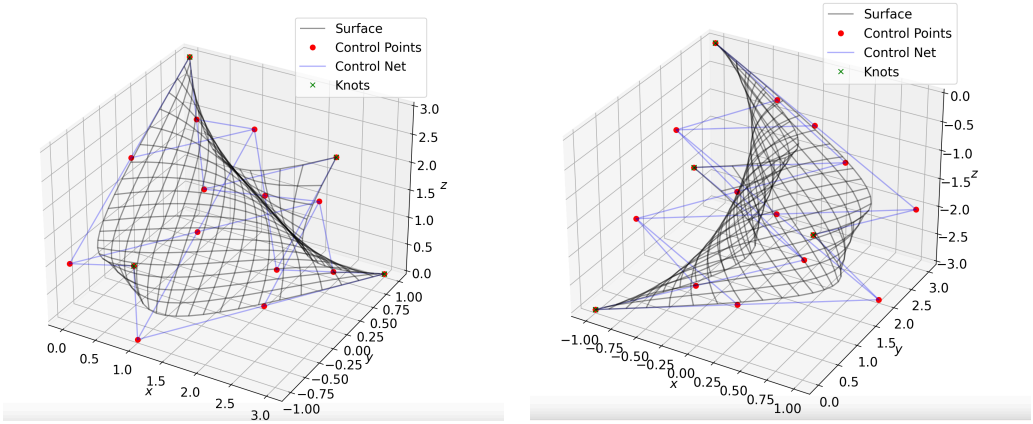


(a) Original NURBS Surface

(b) Translate around y-axis

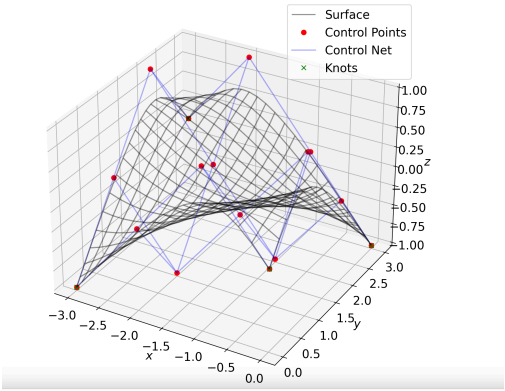
Figure 5.11: Translation Operation

Similarly, if you rotate a B-spline curve or NURBS surface by a certain angle, the curve or surface will be rotated by the same angle, but the shape of the curve or surface will remain unchanged. The following three figures show the surface rotation along the x-axis, y-axis, and z-axis.



(c) Rotate around x-axis

(d) Rotate around y-axis



(e) Rotate around z-axis

Figure 5.11: Rotation Operation

5.2. Search Algorithm

The second phase in the algorithm is the search algorithm phase. As previously stated in Chapter 4.2, there are two substeps within the search algorithm phase, the slave points grouping, and the point projection.

5.2.1. Search Algorithm: Slave Points Grouping

The **slave points grouping** step is to calculate the normal vector for the surface, find the **in-contact** part and the **non-contact** part of the surface, and label slave points to **in-contact points** and **non-contact points**. Clearly, in the figure below, the slave points are grouped into in-contact points and non-contact points.

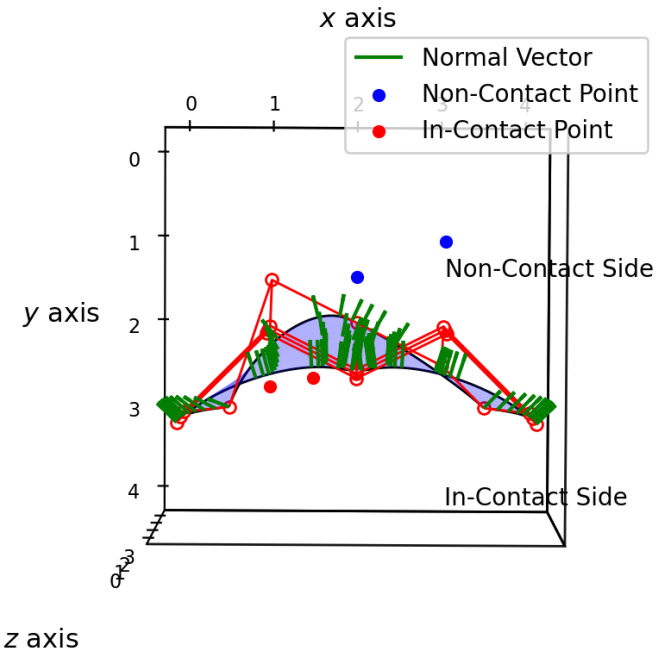


Figure 5.12: Normal Vector and Tangent Vector for NURBS Surface with Slave Point Specified

Since the blue slave points in Fig 5.12 are lying on the non-contact side of the master surface, so they are non-contact points.

5.2.2. Search Algorithm: Point Projection

For the point projection substep, we want to estimate the local minimum orthogonal projection for the in-contact nodes.

To use the Newton iteration method for orthogonal projection search, the first step is to choose a suitable initial guess for the Newton iteration. We know the time history of a point movement given a group of slave points. After categorizing slave nodes into in-contact and non-contact groups, the initial guess for the in-contact group can be computed.

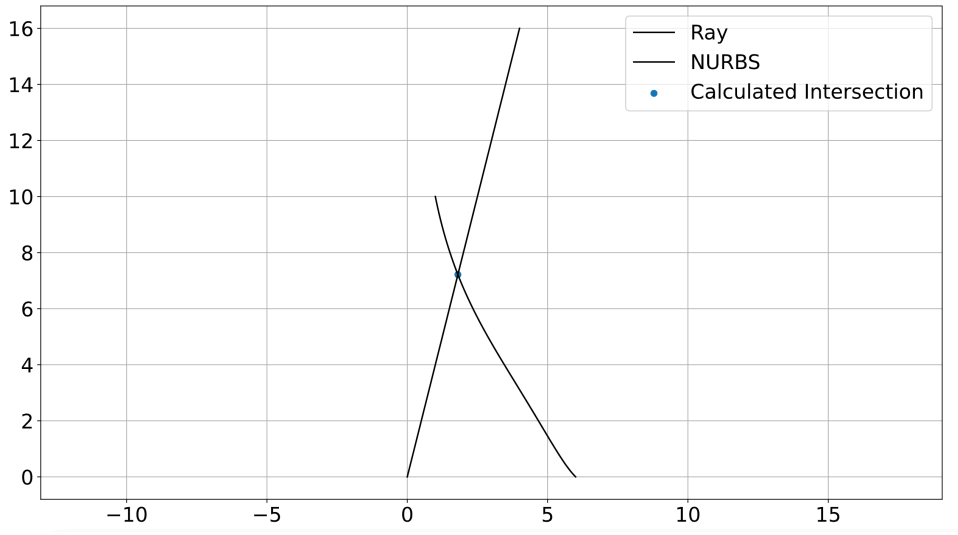
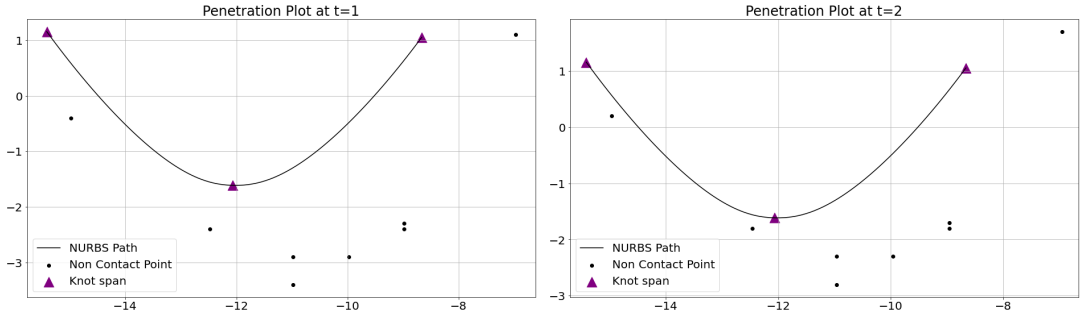


Figure 5.13: An Example for Finding the Intersection Point between A NURBS Curve and A Line

The above figure gives an example of connecting the point position P_{i-1} and P_i and finding the intersection between the connected line and the NURBS curve.

The shape of the master surface influences the search for orthogonal projection points. In the force calculation stage, rather than searching for the global minimum orthogonal projection, we search for the local minimum projection with the smallest jump.

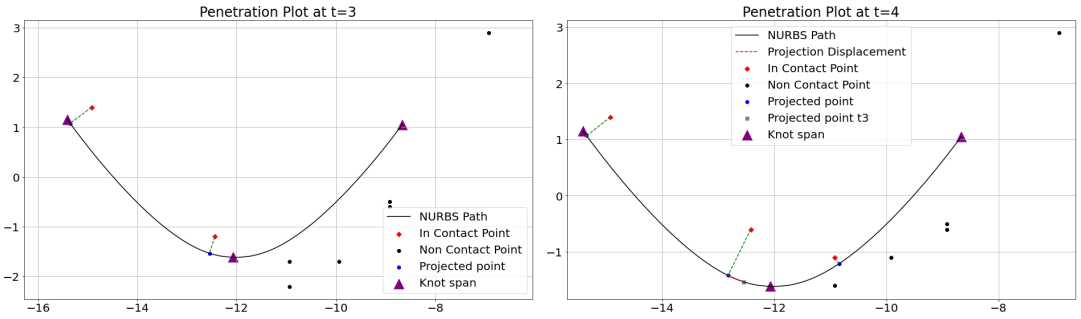


(a) $t = 1$ **(b)** $t = 2$

Figure 5.14: 1D Searching Algorithm Implementation

There is no in-contact point detected in the first two time steps. All of the slave nodes belong to the non-contact group. Then the search algorithm finds two slave nodes (Fig 5.15a) that in-contact with the master segment, in other words, penetrates the master segment. We used the type-a initial guess for the Newton iteration method.

As shown in Fig 5.15b, the search algorithm finds another in-contact slave node. For the previous existing slave node, it uses the projected points in t_3 to feed the Newton iteration as the initial guess.



(a) $t = 3$ **(b)** $t = 4$

Figure 5.15: 1D Searching Algorithm Implementation

Recall the objective function (Eqn 4.2), and it aims to minimize the knot value difference between the projected point in the previous time step and the projected point in the current time step. Figure 5.16 demonstrates the case that the global minimum and the local minimum are not the same. If we select the global minimum, the penetration distance calculation will increase dramatically. The total force plot may exhibit an erratic leap upon entering the penetration distance into the force calculation function. The algorithm chose the local minimum point as the projected point by comparing the two projection shifting distances.

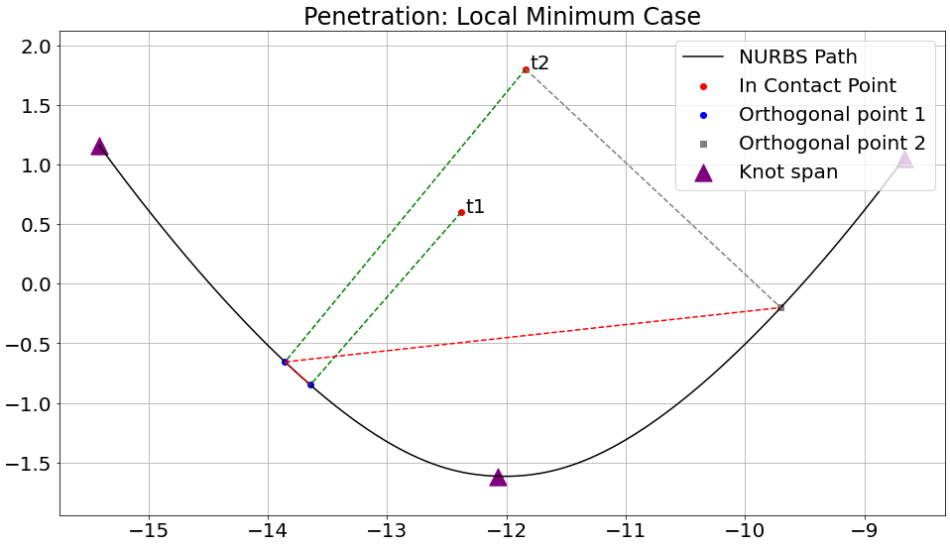


Figure 5.16: Searching Algorithm that Finds the Local Minimum

The group of slave points starts going down at $t = 7$. Some slave points are labeled as non-contact because they are departing the contact side. As the global minimum meets the local minimum at this time step, the algorithm does not need to pick between two projections.

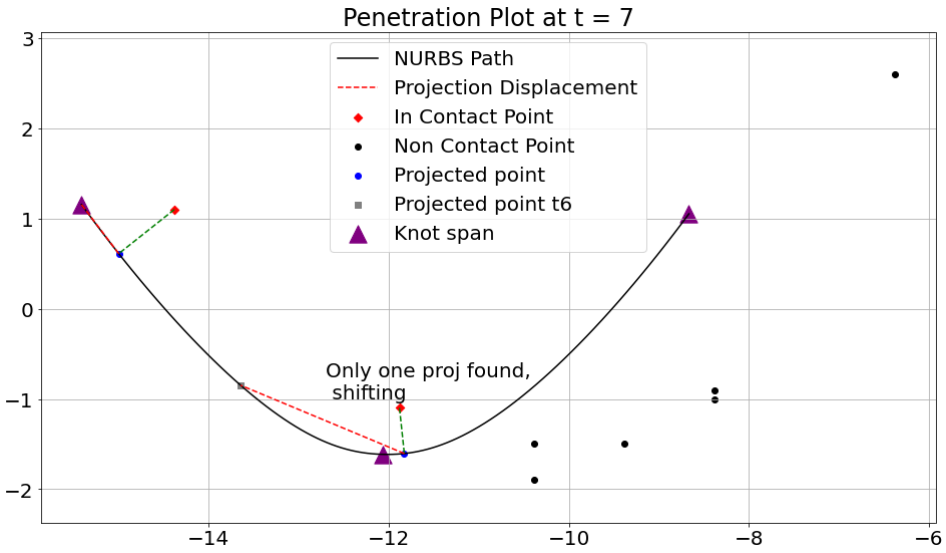


Figure 5.17: Searching Algorithm at $t = 7$

5.2.3. Search Algorithm: 2D Implementation

This method was also utilized to find the local orthogonal projection on the 2D-NURBS surface. A single point is shown in Fig 5.18 going upward and penetrating the master surface. The yellow spot represents the expected position from the previous time step. When the projection shifts in the red dash line and the black dash line are compared, the algorithm determines that the red dash line is shorter. As a result, it selects the local minimum rather than the global minimum. By selecting the local minimum distance rather than the global minimum distance, the algorithm is able to find the point on the surface that is closest to the given point and is orthogonal to the surface at that point.

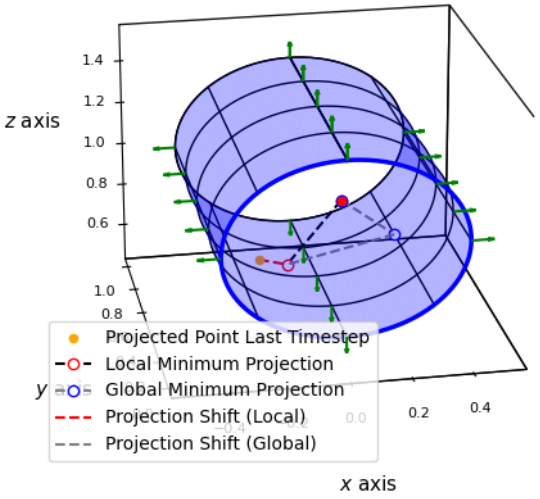
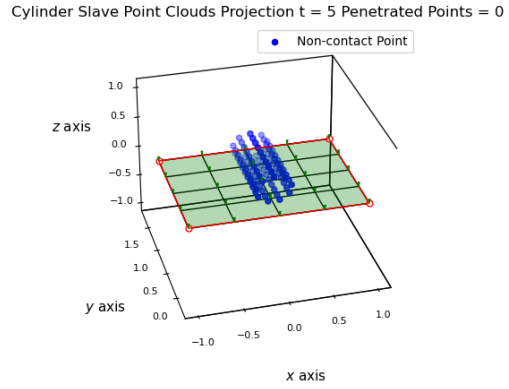


Figure 5.18: Local Minimum Finding in 2D

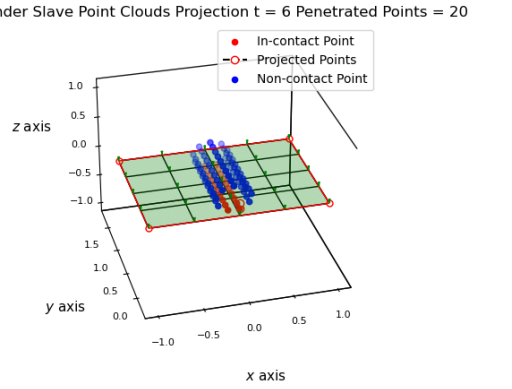
It is indeed possible that the search algorithm may behave differently in a 2D scenario compared to a 1D scenario due to the added complexity of the problem. In a 2D scenario, there may be more local minima to consider, and it may be more difficult to determine which local minima we want to use.

To validate the outcome of the search algorithm in a 2D scenario, it can be helpful to test it using different types of NURBS master surfaces. By using two different types of surfaces, we can ensure that the algorithm is working correctly for a variety of different shapes and configurations. This can help to identify any potential issues or biases in the algorithm and allow you to make any necessary adjustments to improve its performance.

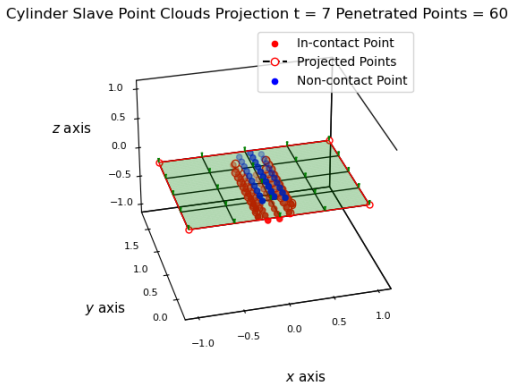
We first validated the algorithm by using a flat master surface and a cylinder-shaped slave point cloud. Using a flat master surface and a cylinder-shaped slave point cloud is a good way to validate the algorithm since it allows us to test the performance of the algorithm for a simple, well-defined shape. A flat surface should be relatively easy to match to a cylinder-shaped point cloud since the point cloud has a well-defined curvature and the surface is flat.



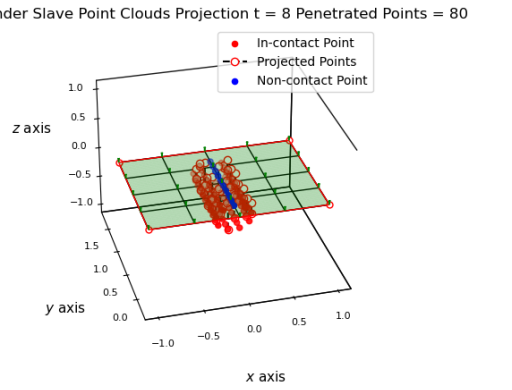
(a) $t = 5$



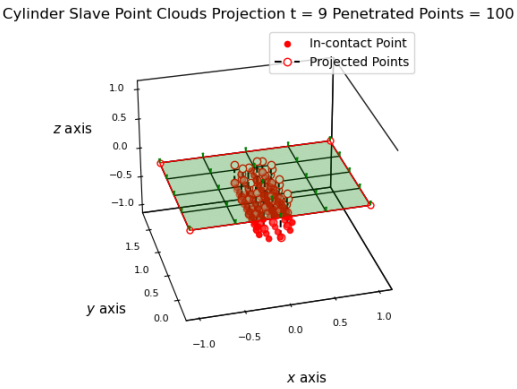
(b) $t = 6$



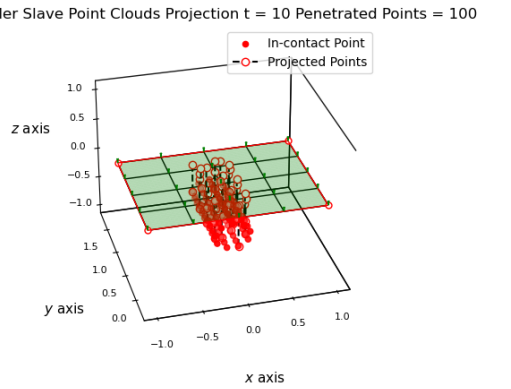
(c) $t = 7$



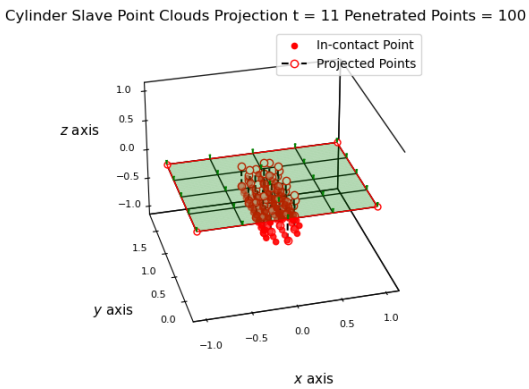
(d) $t = 8$



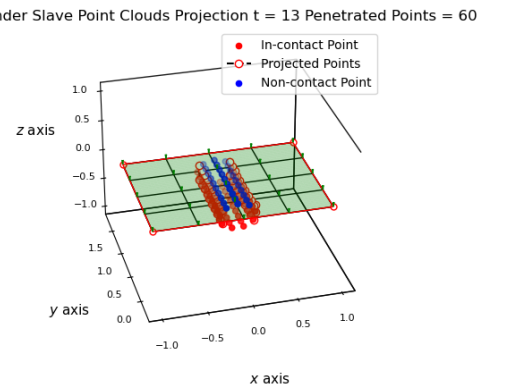
(e) $t = 9$



(f) $t = 10$

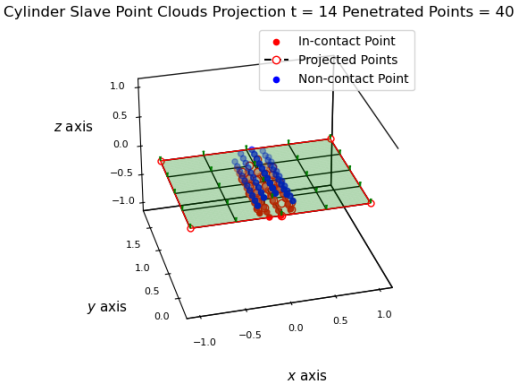


(g) $t = 11$



(h) $t = 13$

Figure 5.19: 2D Implementation: Cylinder Penetrates Flat Master Surface



(i) $t = 14$

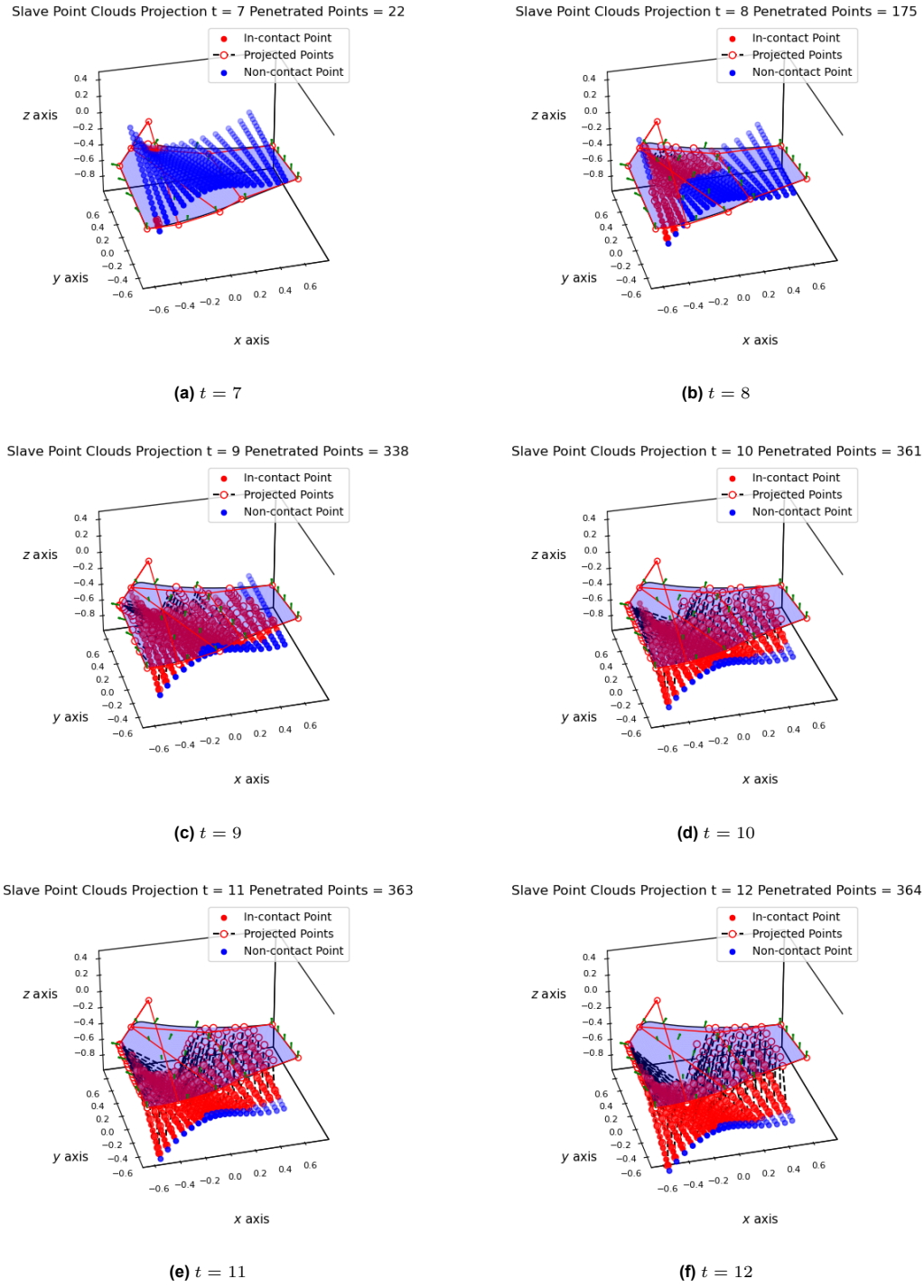
Figure 5.19: 2D Implementation: Cylinder Penetrates Flat Master Surface

Indeed, it finds the local minimum orthogonal projection at each time step and correctly groups the in-contact and non-contact points. However, since the surface is flat in this case, the local minimum may always equal the global minimum, which means that the algorithm may not be able to handle more complex shapes. As a result, we evaluated this approach on a non-flat master surface to check if it still worked.

5.3. Detailed Search: Non-flat Master Surface and Out of Boundary Check

To see whether the search algorithm still works on a more complex surface, we added some uneven features to the master surface, as illustrated in Fig 5.20.

Recall that there are two possible scenarios that the in-contact points can become non-contact again at a later time step.

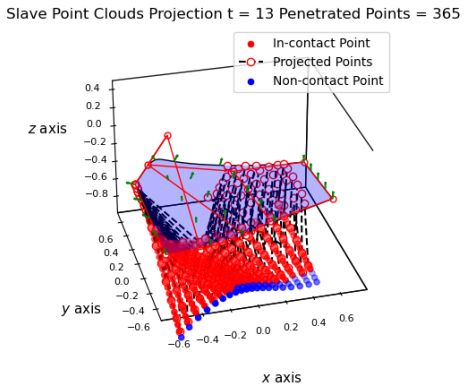


(a) $t = 7$ **(b)** $t = 8$

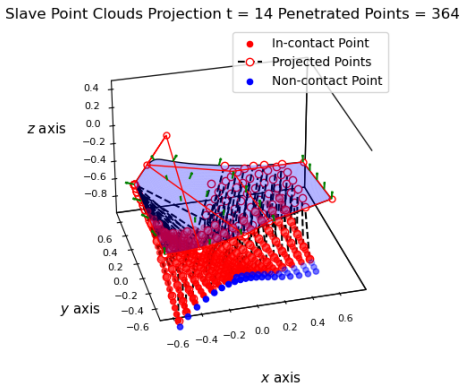
(c) $t = 9$ **(d)** $t = 10$

(e) $t = 11$ **(f)** $t = 12$

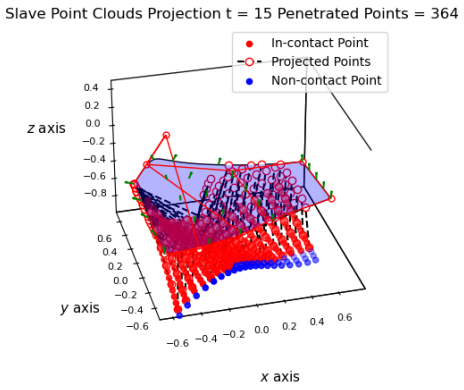
Figure 5.20: 2D Implementation: Non-flat Master Surface



(g) $t = 13$



(h) $t = 14$



(i) $t = 15$

Figure 5.20: 2D Implementation: Non-flat Master Surface

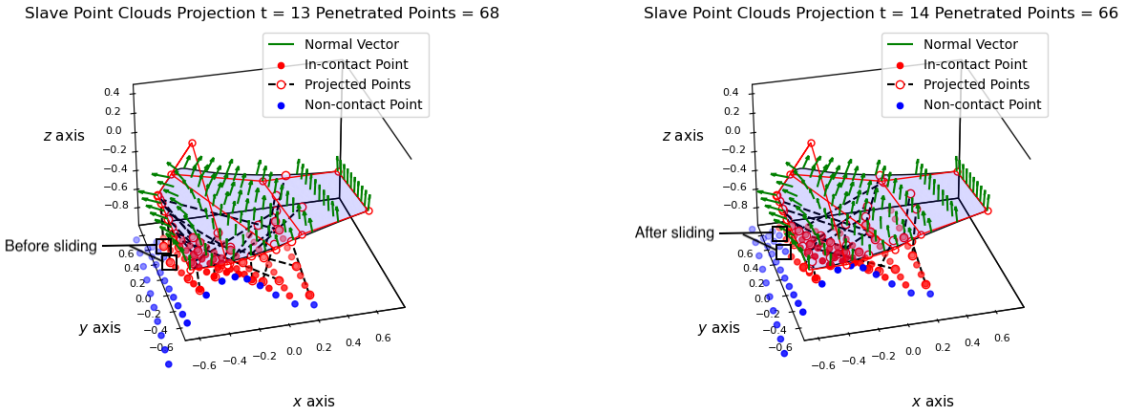
From $t = 10$, there is a clear separation characteristic of the slave point projections, and there is no shifting of the projection from the left master surface to the right master surface, indicating the algorithm's success in the uneven example.

Meanwhile, some of the points are outside the boundaries. There is no orthogonal projection for these points. The algorithm identifies them as being outside of the boundary points. From $t = 9$, slave points are always classified as non-contact points, indicating that they are outside of boundary points.

There is another possible situation where points are recognized as out-of-boundary points, see Eqn 4.7. As the in-contact points move upward, the dot product between $\hat{P} - P$ and N_p is negative. Then they are out-of-boundary points. This situation has been illustrated in the flat-surface example, Fig 5.19(h) and Fig 5.19 (i).

5.3.1. Detailed Search: Sideway Sliding

The second situation in that we need to delete points from the in-contact group is the sideway sliding condition. In this case, the points may no longer be considered ”in-contact” with the surface, as they are not interacting with it in the same way as points that have orthogonal projections onto the surface. In Fig.5.21a, two points labeled inside of the black square belong to the ’in-contact’ group. And the algorithm finds the orthogonal projection for these two points. There are 68 points in total in-contact with the master surface.



(a) Algorithm Finds Orthogonal Projections

(b) Remove the Sliding Points from In-contact Group

Figure 5.21: Sideway Sliding Detection

All of the slave points are moving north-west side, so after $t = 13$ the algorithm processes the detailed search step at $t = 14$. As there is no orthogonal projection for these two labeled points within the master surface, they are no longer recognized as the in-contact points, and we add them to the non-contact point group. Therefore, there are 66 points in the in-contact group at $t = 14$.

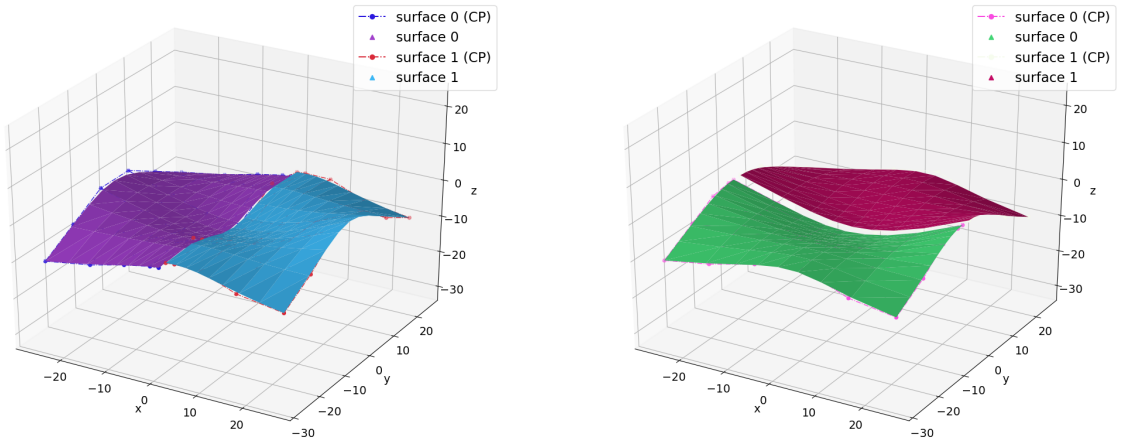
5.4. Force Calculation

Recall in Chapter 4, there are two sub-steps in the force calculation phase: contact area calculation and the contact model. The contact area calculation step involves determining the area of the surface that is in contact with the slave points. The contact model step involves using the MADYMO force model to calculate the forces acting on the surface based on the contact area and other factors, such as the material properties of the surface and the slave points.

5.4.1. Contact Area Calculation

In this chapter, we will only present the NURBS split step for the tessellation of the control net and base surface. Since in Chapter 6, we will tessellate the master surface to a desired order of accuracy, a detailed implementation will be presented later.

As we stated in Chapter 4, it is possible to split the NURBS surface by inserting an additional control point. The following figure (Fig 5.22) compares splitting at $u = 0.5$ and $v = 0.5$.



(a) Split surface at $u = 0.5$.

(b) Split surface at $v = 0.5$.

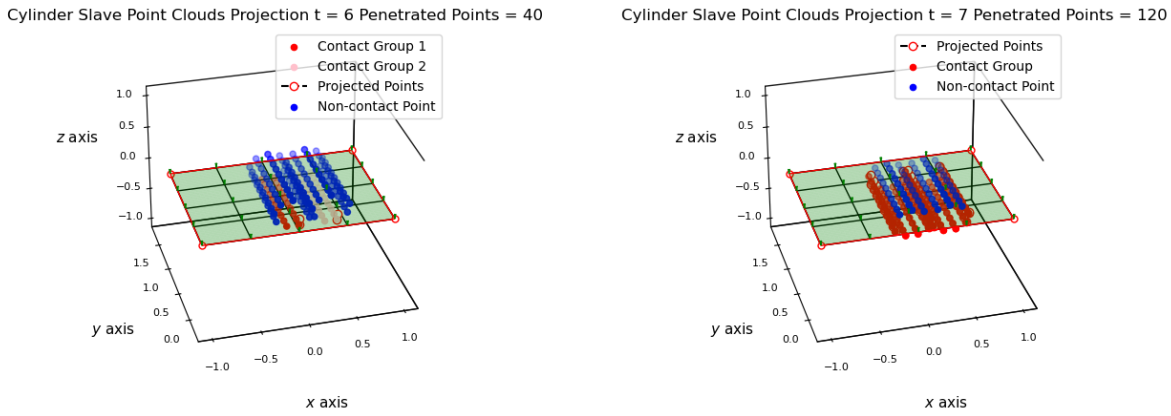
Figure 5.22: Master surface split along u and v directions.

Then it comes to the k-mean clustering step. Our algorithm succeeds in grouping in-contact points with a predicted choice of clusters.

Let us continue with the cylinder penetrating a flat master example. We now do the penetration by placing two cylinders side by side. If the gap between the contacted group is bigger than the normal gap between slave points, we assume that the number of clusters is $k = 2$. Otherwise, k equals 1.

As we can see in Fig 5.23a, the gap between the left cylinder and right cylinder is larger than the normal gap between slave points. Therefore, we define the number of clusters to be $k = 2$, and as clearly stated in the picture, the algorithm detects two clusters of slave points.

However, as the cylinder moves further downwards, the contact gap decreases to 0, see Fig 5.23b. So the algorithm re-defines the number of clusters to $k = 1$.



(a) Two clusters.

(b) One cluster.

Figure 5.23: Side-by-side cylinder penetration. Two clusters of points have been detected.

It is easy to calculate the analytical contact area for each time step by hand for the one cylinder model. We approximated the contact area for the cylinder model (Fig 5.19) and compared the approximated area with the analytical solution.

N (points)	20	60	80	100	100	100	100	60	40
Timestep	6	7	8	9	10	11	12	13	14
A (Approx)	0.133	0.438	0.394	0.394	0.394	0.394	0.394	0.394	0.263
A (Analytical)	0.08	0.40	0.40	0.40	0.40	0.40	0.40	0.40	0.24

Table 5.1: Analytical Total Contact Area v.s Approximated Total Contact Area

Both the estimated contact area and the analytical contact area exhibit the same shifting trend (Tab 5.1). Due to the initial guesses, there is a slightly larger discrepancy at the beginning of the computation. For the subsequent timestep, the Newton iteration uses the previous timestep projections as the initial guess, so it searches the neighborhood around the initial guess to get the projection within the tolerance. Then we calculate the contact area based on the point projections in parameter space. When there is no point that is newly entering the in-contact group, the algorithm uses the projections from the last time step as the initial guess and searches the projections for the current time step around the initial guesses. This resulted in a decrease in the difference between the approximated area and the analytical contact area.

5.4.2. Contact Model

So far, we have presented another choice of contact area compared to MADYMO. The MADYMO calculates the contact force by choosing a constant contact area.

Contact Model Force: For the contact model force, we chose the elastic coefficient $\epsilon = 0.5$. Since the total contact force is only a function of the penetration distance, we have the same plot for the analytical contact area, approximated contact area, and MADYMO.

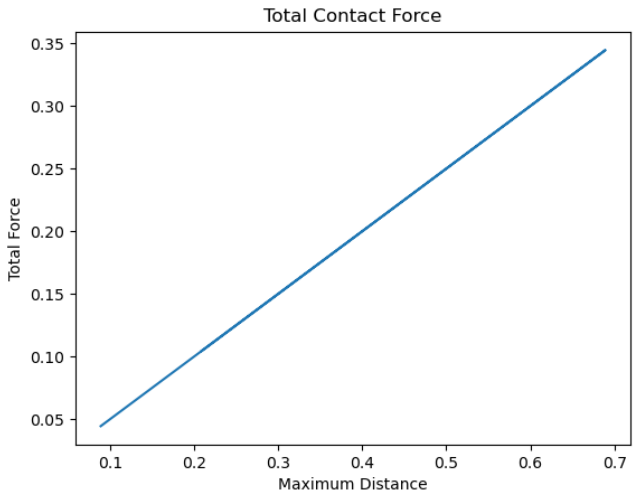
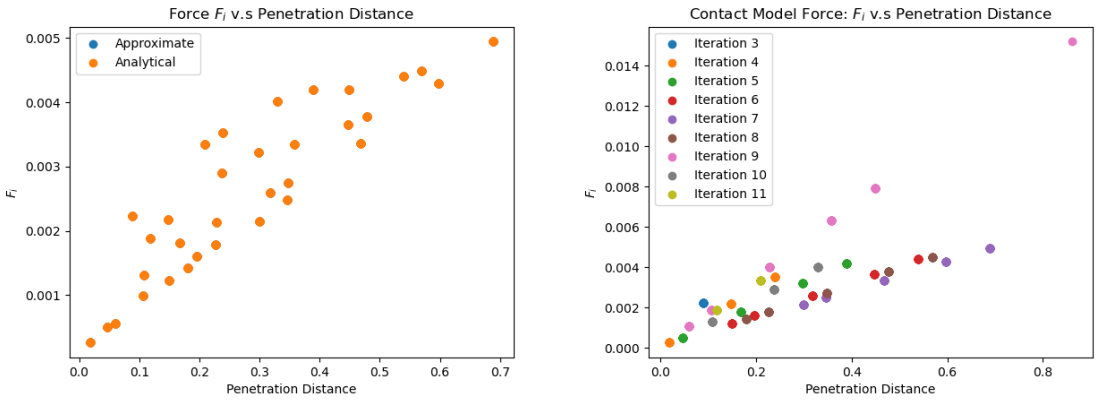


Figure 5.24: Contact Model Force: Total Contact Force F_{total}

The plot above shows a clear linear relationship between the maximum penetration and the contact force, which meets our expectations.

The F_i v.s penetration plots for the IGA-based algorithm, analytical solution, and MADYMO are shown in Fig 5.27a and Fig 5.27b, respectively. The analytical contact areas are calculated by hand since the shape of the contact part for the cylinder is always a rectangle.

Clearly, the result for the approximated area shows exactly the same pattern as the analytical result, which is far more accurate than the MADYMO model. We chose the area for each node $A_i = 0.1$.



(a) Contact Model Force: Analytical Area v.s Approximated Area.

(b) MADYMO

Figure 5.25: Contact Model Force: Approximated Contact Area and MADYMO (constant area)

Here is the difference between the approximated result and the analytical result (Fig 5.26). Therefore, this approximated area gives an error of $O(10^{-12})$, which means that it is sufficiently small enough for a relatively flat master surface contact force model simulation.

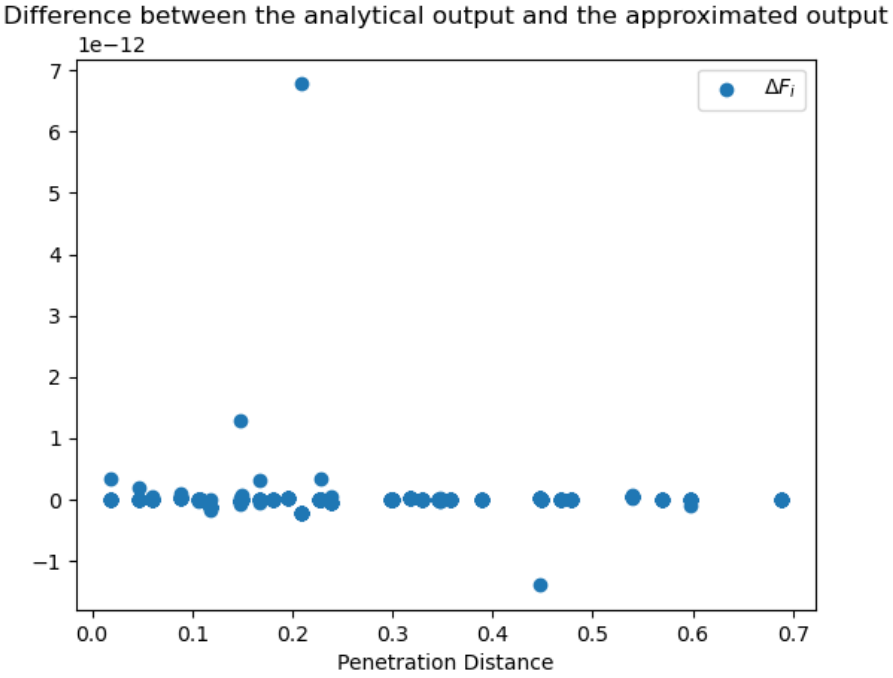
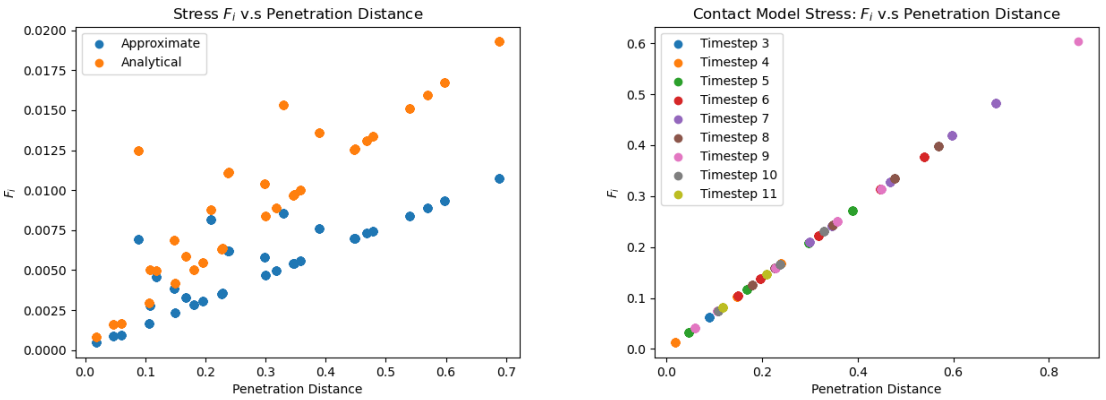


Figure 5.26: ΔF_i for the contact force model.

To obtain the contact force per contact node F_i, the maximum penetration distance at each time step must be determined. In what follows, we present simulation results that display the maximum penetration for each time step. The maximum penetration distance is also indicated in the figure.

Contact Model Stress: We chose the damping coefficient $C_d = 0.2$, and the master surface thickness $t = 0.1$. And we obtained the F_i v.s penetration plot. The left plot represents the F_i simulation result from the contact model stress by using the analytical area and the approximated area from Tab 5.1. The right plot is the simulation result from the MADYMO model.



(a) Contact Model Stress: Analytical Area v.s Approximated Area.

(b) MADYMO

Figure 5.27: Contact Model Stress: Approximated Contact Area and MADYMO (Constant Area)

Obviously, for the contact model stress, the approximated area yields a similar pattern and value as the analytical area, indicating that the approximated area model is superior to the constant area model for each node. The contact stress model is more affected by the approximation of the contact area than the contact force model, and there is a noticeable difference between the analytical result and the approximated result.

6

Performance Comparison: IGA and FEM

In the preceding chapter, the outcomes of each step were given according to the methodology. And we demonstrated the superiority of the new proposed contact area approach over the constant contact area method. Naturally, two questions come into our minds:

- Will the IGA-based contact model outperform the FEM-based approach?
- How comparable is the IGA mesh to the FEM mesh?

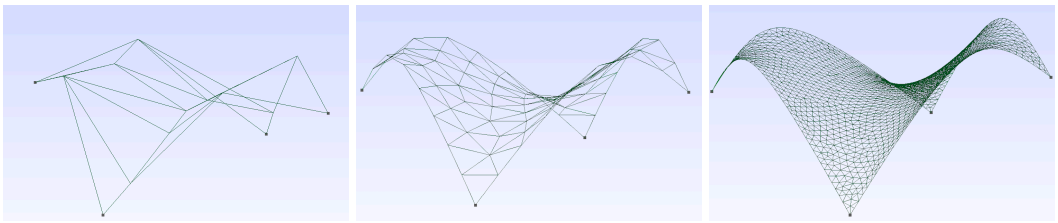
It is difficult to compare the outcomes for the IGA-based contact method and the FEM-based contact method to a totally equal measure. We, therefore, designed the as fair as possible test cases for the performance comparison. Here is the data used in the comparison process:

a. True result: IGA-based master surface.
b. Approximated result: FEM-based master surface.

Recall we mentioned in Chapter 4, the tessellation method is used to estimate the contact area for the IGA surface. We are able to specify the tolerance as shown in Eqn 4.8, which allows us to approximate the IGA surface with a manually chosen precision. We want to compare these two methods in terms of computation time and accuracy.

6.1. Generate the FEM Mesh

Three FEM meshes are generated through the tessellation algorithm (Algorithm 2), with tolerance $tol = 0.5$, $tol = 0.1$, and $tol = 0.01$, respectively. We visualized the meshes through Gmsh (Fig 6.1).



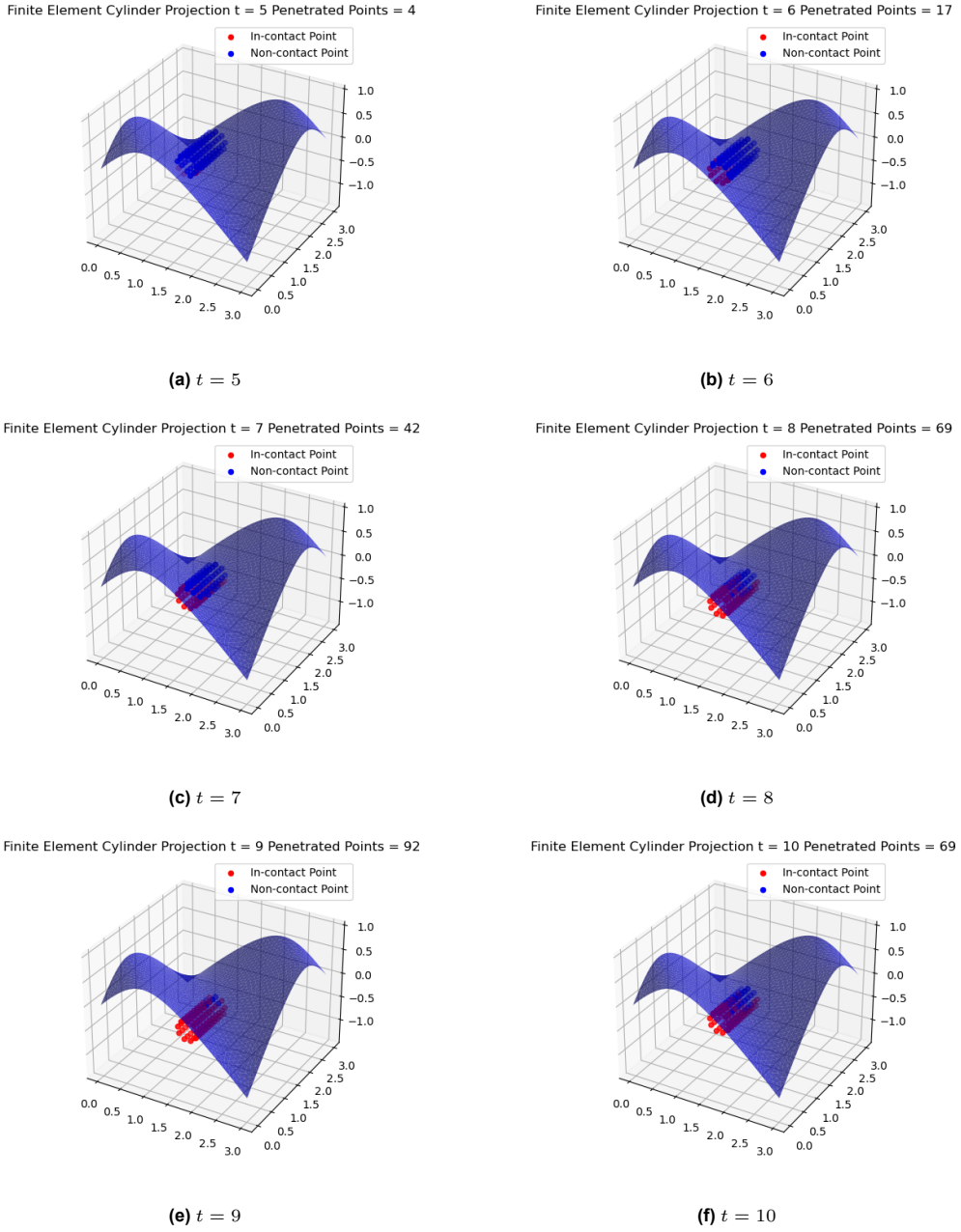
(a) tol = 0.5 **(b)** tol = 0.1 **(c)** tol = 0.01

Figure 6.1: Visualize the FEM mesh through Gmsh.

6.2. FEM Model Implementation

To compare these two different methods, the IGA-based and the FEM-based approaches, we must modify our algorithm to work with the FEM mesh. We did not transfer the contact area implementation to the FEM-based model because we only cared about the influence of the mesh itself. The contact area is constant as in MADYMO.

In what follows, we present the numerical results that illustrate the performance of the search algorithm on the FEM mesh ($tol = 0.01$), with the cylinder slave surface moving downward. All of the implementations have similar results on these three meshes.



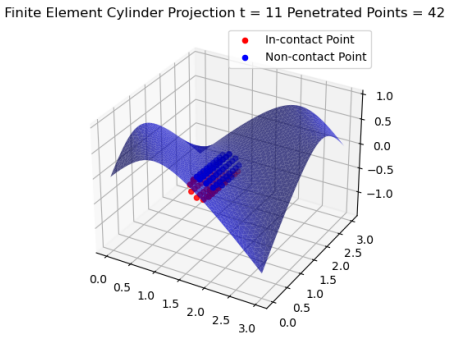
(a) $t = 5$ **(b)** $t = 6$

(c) $t = 7$ **(d)** $t = 8$

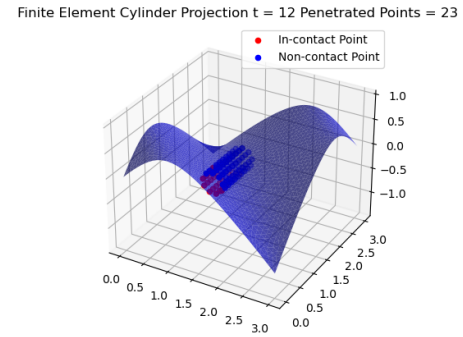
(e) $t = 9$ **(f)** $t = 10$

Figure 6.2: Searching Algorithm on FE-Mesh (tol = 0.01)

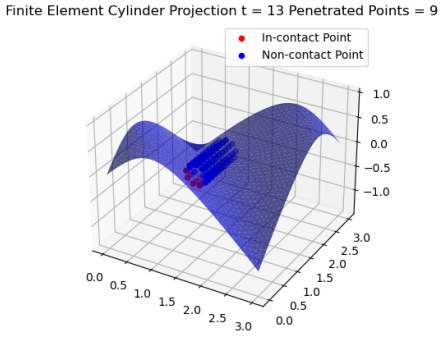
The search algorithm also applies to the FEM master surface, which detects in-contact and non-contact points by calculating the master surface's normal vector. It also finds the local minimal orthogonal projection. Meanwhile, the out-of-boundary and sideway sliding checks are irrelevant in this comparison. As a result, they are not included in the FEM-based contact search method.



(g) $t = 11$



(h) $t = 12$



(i) $t = 13$

Figure 6.2: Searching Algorithm on FE-Mesh (tol = 0.01)

6.3. Performance Comparision

First, we want to know how long each method takes to execute. As illustrated in Table 6.1, utilizing the FE mesh to achieve the order $O(10^{-2})$ precision as the IGA model requires two to three times the execution cost of the IGA approach, as we can see by using the FEM mesh with $tol = 0.01$, it takes the execution time between 17.369s to 33.621s. If we don't require that high-level accuracy, a FE mesh can save much time in computation. Since the computation time for the IGA-based approach remains constant at around 9.5s per time step, however, by higher the tolerance of the FEM mesh, we can reduce the computation time to around 0.18s-0.30s per time step.

Timestep	3	4	5	6	7	8	9	10	11	12	13	14
IGA (20 points)	9.391	9.303	9.531	9.563	9.584	9.719	9.832	9.921	9.443	9.316	9.350	9.377
FEM (tol = 0.5)	0.179	0.211	0.211	0.199	0.220	0.245	0.243	0.295	0.223	0.216	0.182	0.181
FEM (tol = 0.1)	4.953	4.962	7.134	6.922	7.099	7.367	7.352	7.423	7.103	6.883	4.893	4.862
FEM (tol = 0.01)	21.642	19.675	22.658	22.137	27.905	29.647	33.621	32.823	25.767	22.736	17.369	18.724

Table 6.1: Execution Time (s) for IGA and FE Non-Flat Mesh

For the comparison of the maximum distance, through observing the moving history, the contact only happens through $t = 5$ to $t = 13$.

Timestep	5	6	7	8	9	10	11	12	13
IGA (20 points)	0.066	0.147	0.230	0.314	0.398	0.484	0.147	0.0823	0.0173
FEM (tol = 0.5)	0	0.083	0.197	0.273	0.368	0.432	0.081	0.0649	0
FEM (tol = 0.1)	0	0.084	0.187	0.267	0.375	0.456	0.099	0.0769	0.0194
FEM (tol = 0.01)	0.066	0.148	0.223	0.310	0.389	0.489	0.149	0.0835	0.0172

Table 6.2: Maximum Penetration Distance for IGA and FE Non-Flat Mesh

We recorded the maximum distance for the IGA-based and the FEM-based approaches. In Table 6.2, only the FE mesh with tolerance 0.01 recognized the same number of in-contact points for each time step. The tolerance 0.5 FE mesh failed to find the in-contact points at both $t = 5$ and $t = 13$, and the tolerance 0.1 FE mesh failed to find the in-contact points at $t = 5$. Other than that, the penetration difference between IGA and FEM ($tol = 0.5$ and $tol = 0.1$) was quite large. Only the FEM ($tol = 0.01$) found a similar maximum distance as the IGA mesh.

To conclude, in order to achieve the same level of in-contact point detection result as IGA, a tolerance order $O(10^{-2})$ FE mesh must be used. However, it results in a large execution time. The IGA-based master surface can have a better order of accuracy as the FEM ($tol = 0.01$), but the same order of execution time as FEM ($tol = 0.1$).

7

Discussion: Clustering Algorithm for Contact Area Approximation

The contact area approximation step, as mentioned in Chapter 4, is one of two steps in the force calculation stage. To eliminate gaps between different groups of in-contact points, we used the k-mean clustering algorithm. Chapter 5 shows the k-mean clustering algorithm works for this purpose.

However, we already stated the limitation of the k-mean clustering algorithm in Chapter 4. For each timestep, the number of clusters has to be provided manually. Therefore, we tried another density-based clustering algorithm, DBSCAN (Density-based spatial clustering of applications with noise) [26].

7.1. Density Based Clustering Algorithm

This intuitive concept of ”clusters” and ”noise” underpins the DBSCAN algorithm. The key idea is that the neighborhood of a given radius must contain at least a certain number of points for each point of a cluster.

The DBSCAN algorithm has two parameters: ϵ and P_{min}. ϵ determines the radius of the neighborhood around each point, and P_{min} is the minimum number of points that must be in the neighborhood of a point for it to be considered a core point. A point is considered a core point if it has at least P_{min} points within a distance ϵ. Points that are not core points but are within the distance ϵ of a core point are considered border points. Points that are neither core nor border points are considered noise points.

One of the advantages of the DBSCAN algorithm is that it does not require the user to specify the number of clusters in the dataset beforehand.

7.2. Problem with the DBSCAN Algorithm

The motivation for using the DBSCAN algorithm is to reduce the need to manually enter the number of clusters. However, it can be sensitive to the choice of the ϵ and P_{min} parameters and may not perform well on datasets with varying densities or with clusters of different sizes.

Surprisingly, k-means performs better than DBSCAN in our implementation. The following is a picture of the penetration process with in-contact and non-contact points specified before determining the contact area.

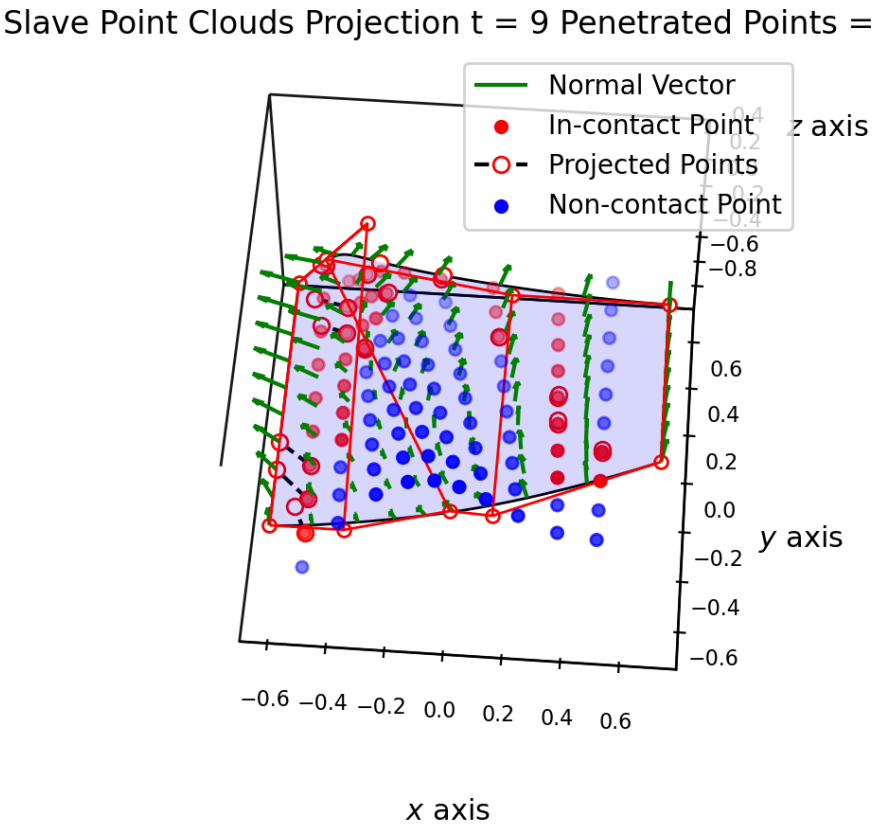


Figure 7.1: Testing Case before Performing Force Calculation Stage

Through observation, it is easier to determine a suitable number of clusters. In this case, we consider the number of clusters $n = 2$.

By providing this number to the k-mean algorithm, it gave the result in parameter space as follows:

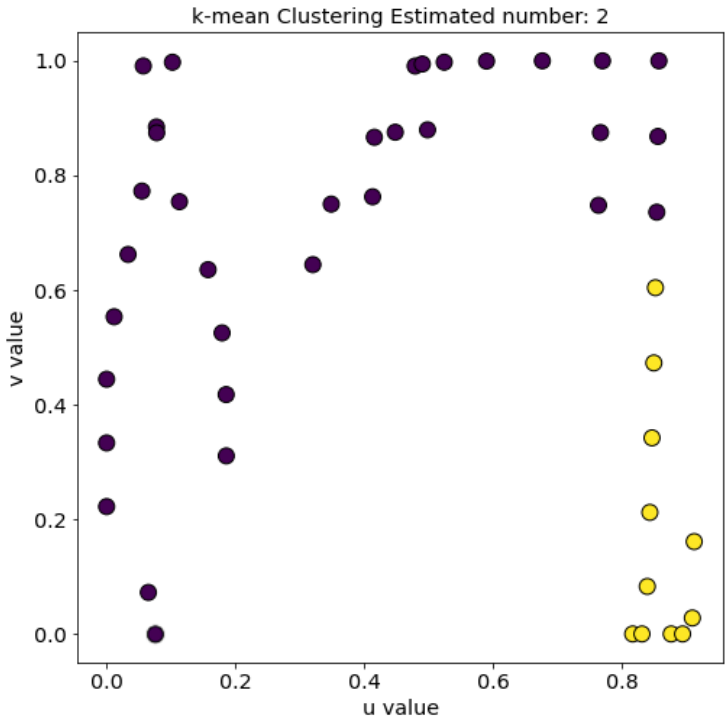
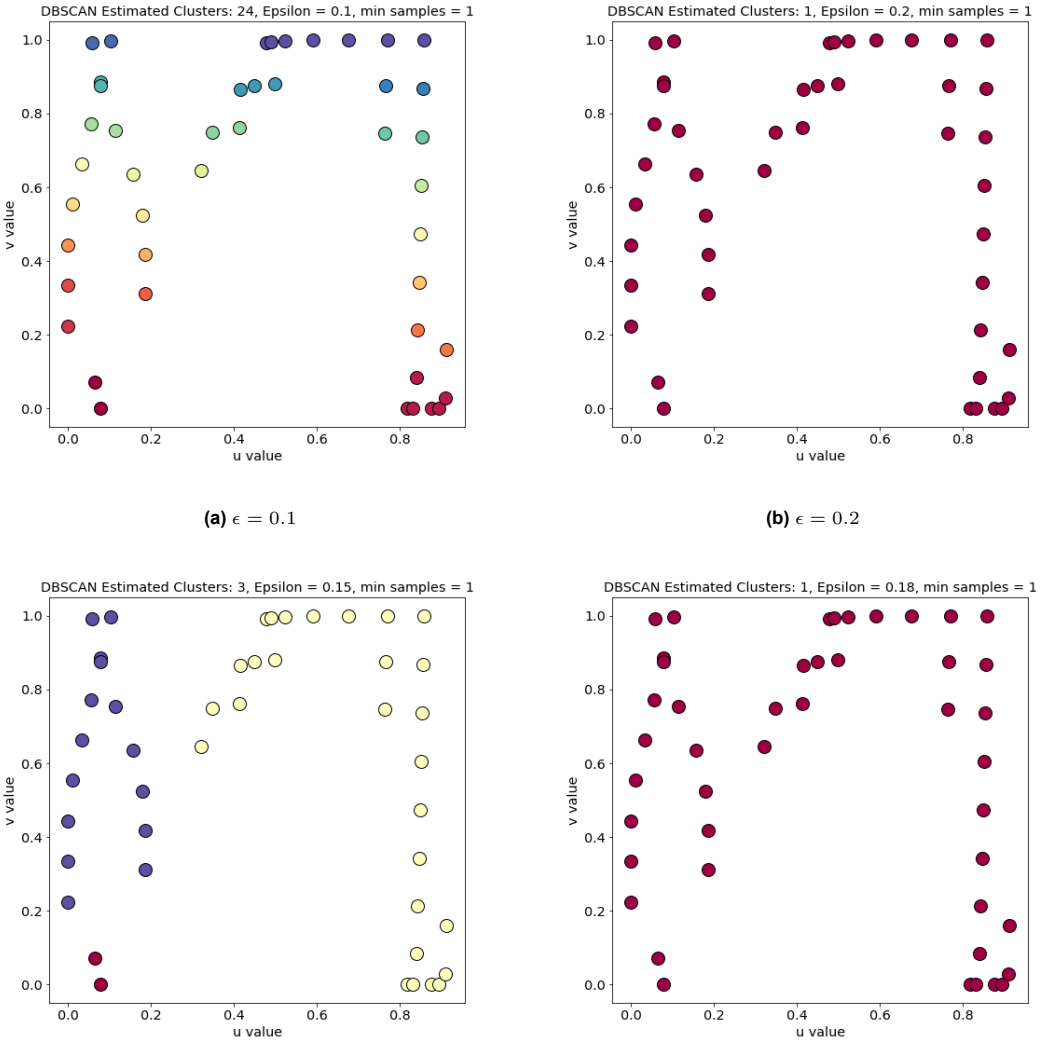


Figure 7.2: K-mean Clustering Algorithm Finds $n = 2$

Through observing the slave point cloud (Fig 7.1), it is difficult to say what the suitable ϵ value is and what the minimum number of points in the neighborhood is.

Even through a very careful modification of the ϵ value, it is still difficult to get the desired number of clusters.



(a) $\epsilon = 0.1$

(b) $\epsilon = 0.2$

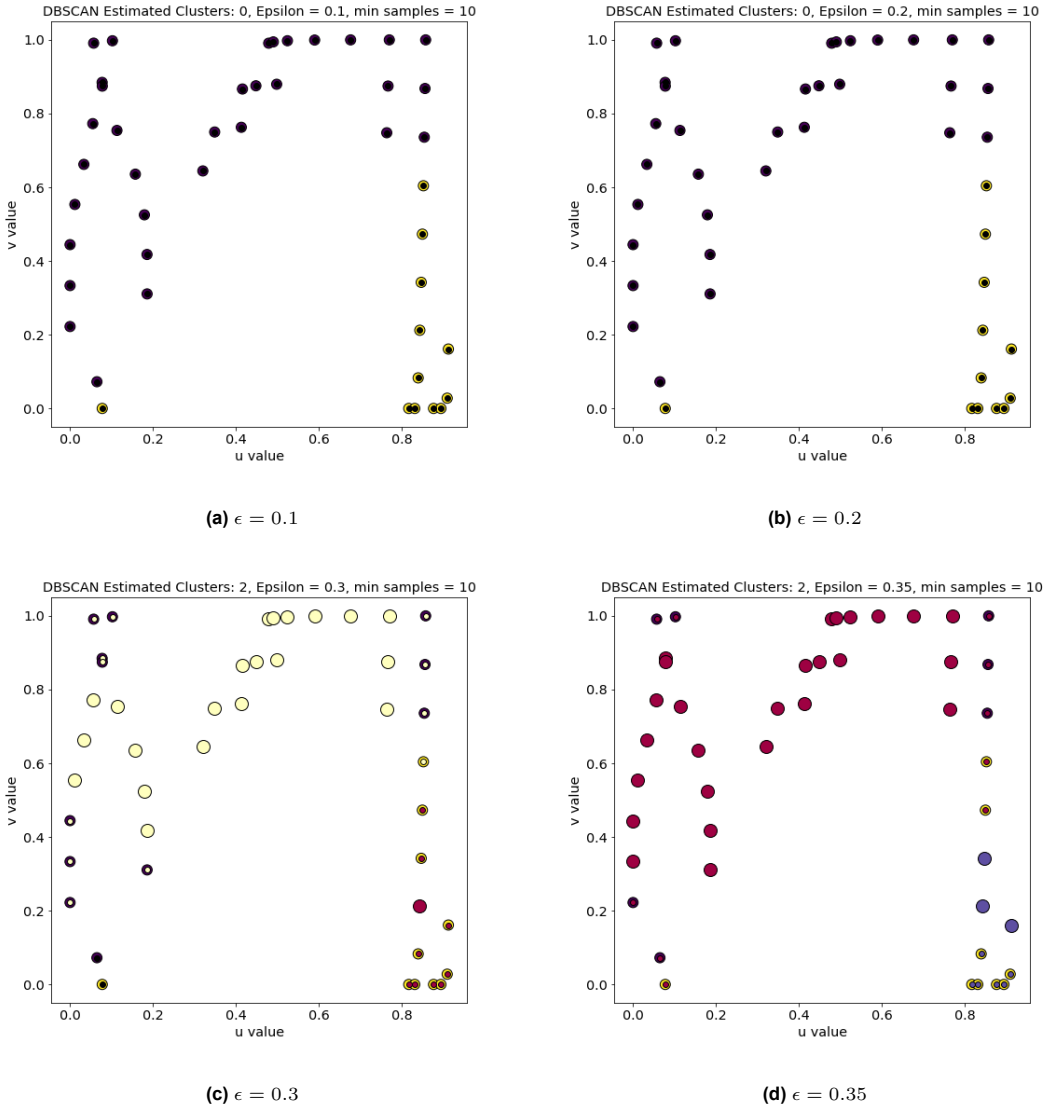
(c) $\epsilon = 0.15$

(d) $\epsilon = 0.18$

Figure 7.3: DBSCAN Implementation: $P_{\text{min}} = 1$

Since we allow $P_{\text{min}} = 1$, we can have very small clusters by using the DBSCAN algorithm. And the clustering process is very sensitive to ϵ as we can see from Fig. 7.3 the number of clusters can vary from 24 to 1 with only a 0.1 difference in ϵ.

Naturally, we started to think about if the choice of P_{min} is too small. So the following example gives the implementation result when $P_{\text{min}} = 10$.



(a) $\epsilon = 0.1$

(b) $\epsilon = 0.2$

(c) $\epsilon = 0.3$

(d) $\epsilon = 0.35$

Figure 7.4: DBSCAN Implementation: $P_{\text{min}} = 10$

Although the algorithm is not too sensitive to ϵ in this case with $P_{\text{min}} = 10$, we did not get the desired clustering result as in the k-mean implementation.

7.3. Discussion

K-means is a centroid-based algorithm, or a distance-based algorithm, where we try to minimize the distance between points within a cluster and the centroid of that cluster. On the other hand, DBSCAN is a density-based algorithm where we try to identify clusters based on the density of points in a region. In cases where the clusters have a clear distance-based structure, k-means would perform better. In cases where the clusters are based on density and have a more complex or irregular shape, DBSCAN would perform better.

8

Conclusion and Future Work

The main goals of this thesis were to transform the existing Finite Element based contact method in MADYMO to an IsoGeometric Analysis-based contact method and compare the new contact method with MADYMO's algorithm.

8.1. Conclusion

To achieve the first main goal, this thesis takes advantage of IGA and builds a new contact technique that transforms the FE-based contact method into an IGA-based method. The new contact methods consist of four general phases: the surface creation phase, the search algorithm phase, the detailed search phase, and the force calculation phase.

The algorithm takes the rigid IGA master surface and the point cloud slave surface as input which is the same as MADYMO. Users can generate B-spline and NURBS surfaces in 1D or 2D during the surface creation process. With the master surface, it is also possible to utilize the refinement methods such as knot insertion to modify the design without changing the whole geometry. For the slave surface, users can generate a random point cloud or generate points along an IGA surface. The new method also provides an easier approach to getting IGA surfaces like bi-linear NURBS surfaces, excluded NURBS surfaces, and ruled NURBS surfaces.

During the search algorithm phase, we employed the normal vector to designate the in-contact and non-contact sides of the master surface before performing slave point grouping. Instead of resolving the minimization problem in coordinate space, the search algorithm exploits the availability of underlying parameter space. This allows it to locate the local orthogonal minimum rather than the global minimum in the parameter space. Because the domain of the parameter space is always enlarged by the tensor product of $[0, 1]$, users do not need to worry about how to adjust the constraints and boundaries for the minimization problem when using different geometries. This is because the domain is always expanded by taking the tensor product of $[0, 1]$.

However, there are two situations that require a more detailed search. The first situation is that in-contact points go sideways, and their orthogonal projections are out of the master surface. The second situation is the in-contact points go out of the boundary of the master surface. So we formulate the detailed search phase by determining the sign of the dot product between $(P - \hat{P})$ and N_p, and checking if the projections still stay within the master surface. And remove points that do not satisfy these two criteria from the in-contact group.

Because an algorithm's accuracy order is, in general, equal to the lowest order within the algorithm, we noticed that if we continued to use a constant contact area like MADYMO, we would lose the accuracy benefit from IGA. So the contact area step has been added to the force calculation phase. The tessellation technique gives the advantage of manually choosing the level of accuracy of the contact area. Furthermore, the procedure of clustering contact points reduces the inaccuracy of including the gap between two clusters of in-contact points.

Then we can answer the second question, how the new algorithm compares to the FE-based contact method in MADYMO. The answer is: it depends on the expectation of the users. If users want a fast simulation and just want to know what the contact will be like, we suggest to switch to the FE-based

contact method. For the simplest FE mesh in Chapter 6, it only takes 0.2 seconds for each time step, and the contact search result is somewhat adequate. However, if users want an exact and robust simulation, then IGA will be the best choice. When compared to the FE approach, IGA saves 50% to 60% of the execution time for the same level of accuracy.

8.2. Future Work

There is still a lot of work to be done in the future, and this thesis is only the starting point for equipping MADYMO with the IGA technique.

There are two types of future work, the first type is a suggestion about the project itself, and the second type is a suggestion about work after this project.

In the clustering step, we have to manually feed the number of clusters to the algorithm, which means we have to run the algorithm back and forth to adjust the clustering number. This project also tested one of the density-based clustering algorithms, which did not perform as well as the k-means algorithm. One possible direction is to try out Mean-Shift Clustering, but this algorithm is more time-consuming than the k-means algorithm.

In the contact area mapping step, which is mapping from the parameter space to the coordinate space, we applied the linear map to this problem. However, this mapping is definitely not accurate. Therefore, we could only use the flat master surface to test the area approximation. If there are more textures in the master surface, it is necessary to find a suitable mapping from the parameter space to the coordinate space.

For a large scaled problem, it is possible to have multi-patch geometries. How to adapt this algorithm to the multi-patch environment? And if we have a deformable master surface, how to perform the deformation?

Finally, let's finish this thesis by looking at the MADYMO example again. In this project, we only used the 'IsoGeometric' part of the IGA, and we haven't used the 'Analysis' part of the IGA. It is possible to utilize the analysis features in the future for different types of simulations, for example, simulating the contact activity for the safety belt.

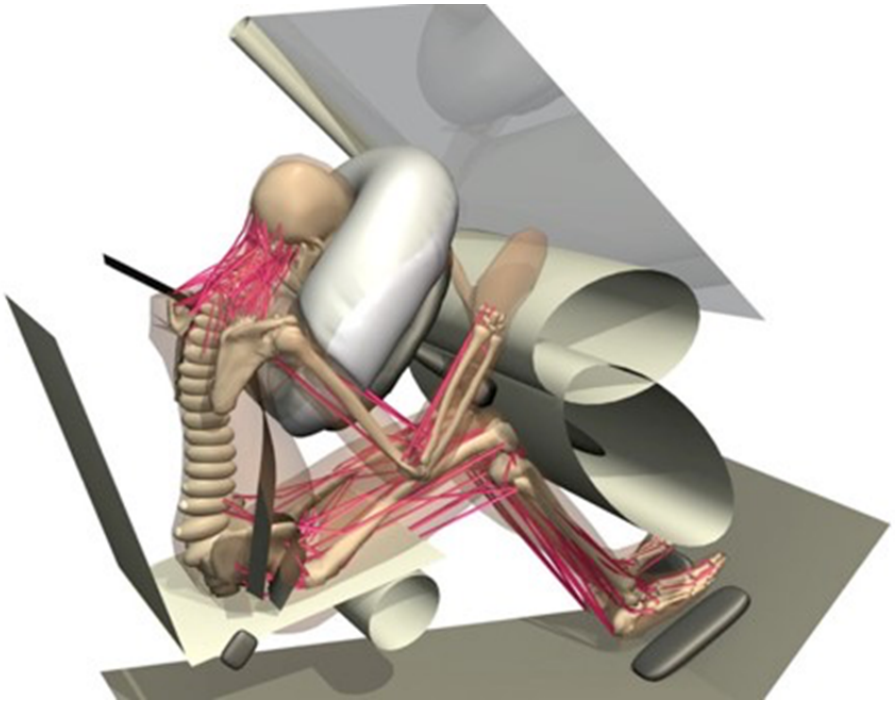


Figure 8.1: MADYMO simulation result [2].

In my opinion, this project is merely the start of the IGA-based contact algorithm: a toy model. The real scenario is far more complicated, but this project suggests the possibility of having a more elegant and accurate contact simulation by using IGA.

References

[1] Y. Bazilevs and T. J. Hughes. “NURBS-based isogeometric analysis for the computation of flows about rotating components”. In: *Computational Mechanics* 43.1 (2008), pp. 143–150. doi: 10.1007/s00466-008-0277-z.

[2] CAEWISSEN. *Simcenter Madymo Active Human Model*. url: https://www.caewissen.com/object/B11/B11.6tt734848q4n3b67u8k476997zwsy563490828499/safetywissen?prev=%2Fobject%2FB02%2FB02.vlc7376771532t9tiso38983tz82na63735245383%2Fsafetywissen%3F_k%3Dxte%26desc%3Dtrue%26field%3Df036t_Gueltigkeitsdatum_ISO%26prev%3D%252Fnews%252FCAENEWS%252F%253Fquery%26type%3DA11.

[3] L. De Lorenzis, P. Wriggers, and G. Zavarise. “A mortar formulation for 3D large deformation contact using NURBS-based isogeometric analysis and the augmented lagrangian method”. In: *Computational Mechanics* 49.1 (2011), pp. 1–20. doi: 10.1007/s00466-011-0623-4.

[4] L. De Lorenzis et al. “A large deformation frictional contact formulation using NURBS-based isogeometric analysis”. In: *International Journal for Numerical Methods in Engineering* 87.13 (2011), pp. 1278–1300. doi: 10.1002/nme.3159.

[5] Ronald A. DeVore and George G. Lorentz. “Bernstein polynomials”. In: *Grundlehren der mathematischen Wissenschaften* (1993), pp. 303–332. doi: 10.1007/978-3-662-02888-9_10.

[6] R. Dimitri et al. “Isogeometric large deformation frictionless contact using T-Splines”. In: *Computer Methods in Applied Mechanics and Engineering* 269 (2014), pp. 394–414. doi: 10.1016/j.cma.2013.11.002.

[7] T. Elguedj et al. “B and F projection methods for nearly incompressible linear and nonlinear elasticity and plasticity using higher-order Nurbs Elements”. In: (2007). doi: 10.21236/ada478310.

[8] Gerald Farin. “A history of curves and surfaces in cagd”. In: *Handbook of Computer Aided Geometric Design* (2002), pp. 1–21. doi: 10.1016/b978-044451104-1/50002-2.

[9] Eric Guilbert and Hui Lin. “B-spline curve smoothing under position constraints for line generalisation”. In: *Proceedings of the 14th annual ACM international symposium on Advances in geographic information systems - GIS ’06* (2006). doi: 10.1145/1183471.1183474.

[10] Ming-Chen Hsu, Ido Akkerman, and Yuri Bazilevs. “High-performance computing of wind turbine aerodynamics using isogeometric analysis”. In: *Computers amp; Fluids* 49.1 (2011), pp. 93–100. doi: 10.1016/j.compfluid.2011.05.002.

[11] Ming-Chen Hsu et al. “Fluid–structure interaction analysis of bioprosthetic heart valves: Significance of arterial wall deformation”. In: *Computational Mechanics* 54.4 (2014), pp. 1055–1071. doi: 10.1007/s00466-014-1059-4.

[12] T.J.R. Hughes, J.A. Cottrell, and Y. Bazilevs. “ISOGEOMETRIC analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement”. In: *Computer Methods in Applied Mechanics and Engineering* 194.39-41 (2005), pp. 4135–4195. doi: 10.1016/j.cma.2004.10.008.

[13] Thomas J. R. Hughes and John A. Evans. “Isogeometric Analysis”. In: 2010.

[14] J. Kiendl et al. “Isogeometric shell analysis with Kirchhoff–Love Elements”. In: *Computer Methods in Applied Mechanics and Engineering* 198.49-52 (2009), pp. 3902–3914. doi: 10.1016/j.cma.2009.08.013.

[15] Georgia Kikis, Wolfgang Dornisch, and Sven Klinkel. “Isogeometric Reissner-Mindlin shell analysis - employing different control meshes for displacements and rotations”. In: *PAMM* 16.1 (2016), pp. 209–210. doi: 10.1002/pamm.201610093.

[16] Vinod Kumar et al. “Convex hull: Applications and dynamic convex hull”. In: *Ambient Communications and Computer Systems* (2022), pp. 371–381. doi: 10.1007/978-981-16-7952-0_34.

[17] Michael Lachance. “An introduction to splines for use in computer graphics and geometric modeling”. In: *Computer Vision, Graphics, and Image Processing* 50.1 (1990), pp. 125–126. doi: 10.1016/0734-189x(90)90071-3.

[18] Xin Li and Falai Chen. “Exact and approximate representations of trimmed surfaces with NURBS and BéZier Surfaces”. In: *2009 11th IEEE International Conference on Computer-Aided Design and Computer Graphics* (2009). doi: 10.1109/cadcg.2009.5246888.

[19] Simcenter Madymo. “9”. In: *Madymo Theory Manual*. Siemens, 2022, pp. 183–220.

[20] M.E. Matzen, T. Cichosz, and M. Bischoff. “A point to segment contact formulation for isogeometric, NURBS based finite elements”. In: *Computer Methods in Applied Mechanics and Engineering* 255 (2013), pp. 27–39. doi: 10.1016/j.cma.2012.11.011.

[21] Lies A. Piegl and Wayne Tiller. *The NURBS book*. Springer, 1997.

[22] Denbigh Starkey. *Cubic spline curves and surfaces aui course - Montana State University*. url: https://www.cs.montana.edu/courses/spring2009/425/dslectures/splines.pdf.

[23] İ. Temizer, P. Wriggers, and T.J.R. Hughes. “Contact treatment in isogeometric analysis with NURBS”. In: *Computer Methods in Applied Mechanics and Engineering* 200.9-12 (2011), pp. 1100–1112. doi: 10.1016/j.cma.2010.11.020.

[24] İ. Temizer, P. Wriggers, and T.J.R. Hughes. “Three-dimensional mortar-based frictional contact treatment in isogeometric analysis with NURBS”. In: *Computer Methods in Applied Mechanics and Engineering* 209-212 (2012), pp. 115–128. doi: 10.1016/j.cma.2011.10.014.

[25] Clemens V. Verhoosel et al. “An isogeometric analysis approach to gradient damage models”. In: *International Journal for Numerical Methods in Engineering* 86.1 (2011), pp. 115–134. doi: 10.1002/nme.3150.

[26] Xiaocui Yuan, Huawei Chen, and Baoling Liu. “Point cloud clustering and outlier detection based on spatial neighbor connected region labeling”. In: *Measurement and Control* 54.5-6 (2020), pp. 835–844. doi: 10.1177/0020294020919869.