FACULTY OF APPLIED SCIENCES
UNIVERSITY
OF WEST BOHEMIA

DEPARTMENT OF
COMPUTER SCIENCE
AND ENGINEERING

Bachelor's Thesis

Automatic Generation of Internal Fibres of Skeletal Muscles

Duc Long Hoang

PILSEN, CZECH REPUBLIC

2023

FACULTY OF APPLIED SCIENCES
UNIVERSITY
OF WEST BOHEMIA

DEPARTMENT OF
COMPUTER SCIENCE
AND ENGINEERING

Bachelor's Thesis

Automatic Generation of Internal Fibres of Skeletal Muscles

Duc Long Hoang

Thesis advisor
doc. Ing. Josef Kohout, Ph.D.

PILSEN, CZECH REPUBLIC 2023

Citation in the bibliography/reference list:
HOANG, Duc Long. *Automatic Generation of Internal Fibres of Skeletal Muscles*. Pilsen, Czech Republic, 2023. Bachelor's Thesis. University of West Bohemia, Faculty of Applied Sciences, Department of Computer Science and Engineering. Thesis advisor doc. Ing. Josef Kohout, Ph.D.

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Duc Long HOANG**
Osobní číslo:	**A19B0054P**
Studijní program:	**B0613A140015 Informatika a výpočetní technika**
Specializace:	**Informatika**
Téma práce:	**Automatické generování vnitřních vláken kosterních svalů**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s projektem Muscle Wrapping 2.0
2. Popište a porovnejte přístupy metod Kukačka a VIPER pro automatické generování vnitřních vláken kosterních svalů
3. Prozkoumejte možnosti generování bodů uvnitř nekonvexního polygonu
4. Navrhněte a naimplementujte algoritmus pro generování bodů rovnoměrně rozmístěných uvnitř obecně neplanárního polygonu reprezentující řez kosterním svalem v metodě Kukačka
5. Na základě metody VIPER navrhněte a naimplementujte algoritmus pro optimální přiřazení bodů na jednotlivých řezech svalu za účelem generování vnitřních vláken
6. Porovnejte vlákna generovaná novou metodou s těmi generovanými metodou Kukačka

Rozsah bakalářské práce: **doporuč. 30 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování bakalářské práce: **tištěná/elektronická**
Jazyk zpracování: **Angličtina**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce

Vedoucí bakalářské práce: **Doc. Ing. Josef Kohout, Ph.D.**
Nové technologie pro informační společnost

Datum zadání bakalářské práce: **3. října 2022**
Termín odevzdání bakalářské práce: **4. května 2023**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 25. října 2022

Declaration

I hereby declare that this Bachelor's Thesis is completely my own work and that I used only the cited sources, literature, and other resources. This thesis has not been used to obtain another or the same academic degree.

I acknowledge that my thesis is subject to the rights and obligations arising from Act No. 121/2000 Coll., the Copyright Act as amended, in particular the fact that the University of West Bohemia has the right to conclude a licence agreement for the use of this thesis as a school work pursuant to Section 60(1) of the Copyright Act.

In Pilsen, on 20 April 2023

. . . . . .

Duc Long Hoang

Abstract

The main focus of this bachelor's thesis is the decomposition of skeletal muscles into mechanical fibers, that approximate real muscle fibers and tendons. The work examines the existing methods of muscle decomposition, namely Kukačka and VIPER, and aims to introduce a new one. The thesis also deals with the problem of the even distribution of points inside a polygon and the problem of optimal assignment. Part of the thesis was also a small survey, where participants compared the visual aspect of the fibers created by the new method and Kukačka.

Abstrakt

Hlavním cílem bakalářské práce je rozložení kosterních svalů do mechanických vláken, které aproximují reálná svalová vlákna a šlachy. Tato práce pojednává o existujících metodách dekompozice svalů, jmenovitě Kukačka a VIPER, a má za cíl navrhnout novou metodu. V této práci jsou, mimo jiné, řešeny problémy rovnoměrného rozložení bodů v mnohoúhelníku a problém optimálního přiřazení. Součástí byl také krátký dotazník, kde účastníci porovnávali vizuální stránku vláken vytvořených novou metodou a metodou Kukačka.

Keywords

health informatics • muscle fibers • even distribution • polygons • optimal assignment • Voronoi • Munkres • Kukačka • VIPER

Acknowledgement

I would like to first and foremost thank my thesis advisor, doc. Ing. Josef Kohout, Ph.D., for all his thoughtful inputs and suggestions during the journey of this bachelor's thesis. My thanks also go to my family, friends and loved ones for supporting and encouraging me throughout these last few months. And namely, my classmate Lukáš Varga, who motivated me the most with his focus on his own thesis.

My appreciations also go to Ing. Kamil Ekštein, Ph.D. for creating the LaTeX template that this bachelor's thesis uses for stylistic typography and to Ing. Martin Úbl for teaching me everything I know about the programming language C++.

Contents

Introduction 1

Modern medicine uses computational models of musculoskeletal systems for a wide range of applications. These include calculations to estimate the contact force on lower limb joints and simulations of musculotendon mechanisms in individuals, both healthy and pathological. Although these models are extensively used, there are only a few practical methods for depicting muscle anatomy. Another issue is that these models were created from a dataset of a small number of autopsies. This limits the use of these models in personalized medicine because the musculotendon paths of specific subjects are created by mapping the established muscular system to the individual's bone structure.

While representing skeletal muscles, two key design components should be considered. These are: (a) the number of muscle fibers created to approximate the muscles (muscle discretization levels), and (b) the number of straight line segments that each fiber is divided into.

The aim of this thesis is to examine the existing methods for muscle decomposition and to design a new one.

Muscle Wrapping 2.0

2

Muscle Wrapping 2.0 is a project co-developed by the Department of Computer Science and Engineering, University of West Bohemia and the Department of Civil and Environmental Engineering, Imperial College London. The aims of this project are (a) to wrap muscles (represented by a triangular surface mesh) around moving bones and (b) to automatically decompose said muscles into mechanical lines of action for further analysis and calculations. These objectives are designed to be compatible with **OpenSim**, which is an open-source software system used for biomechanical modeling, simulation, and analysis.

The project architecture, as seen from Figure 2.1, consists of 2 main applications, the first one being *OsimMuscleGenerationTool* and the latter being *AttachmentEstimation*. The central focus of this thesis is the former of these two applications. The *OsimMuscleGenerationTool* is a plug-in for **OpenSim** that handles the decomposition of skeletal muscles into lines of action (see Chapter 3) and it will be further expanded.

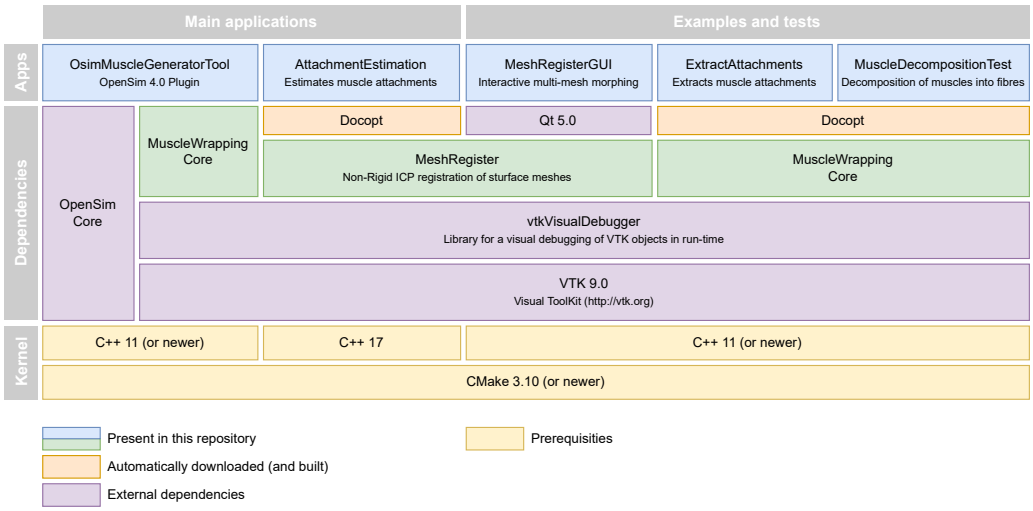


Figure 2.1: *Architecture of Muscle Wrapping* 2.0

Muscle decomposition

3

In a clinical setting, muscles are often represented by one or several poly-lines, also known as *lines of action* (see Figure 3.1) which connect the origin and insertion points of the muscle. The origin and insertion points are locations where the muscle is attached to the bone structure via tendons. These poly-lines can be configured to cross several points of interest called *via points*, to be anchored to an underlying bone or they can be made to automatically wrap around other parametric objects. Essentially, *lines of action* are simplified representations of both muscle fibers and tendons, this allows for faster processing speeds.

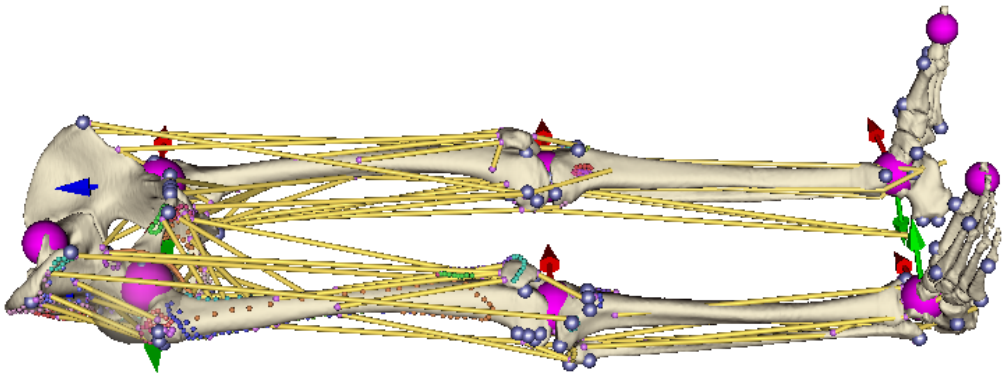


Figure 3.1: *Lines of action in yellow. Image from* [*KK*14]

However, Kohout et al. [KK14] argue that representing a muscle in this way may result in a loss of musculotendon mechanics, leading to generally less accurate predictions. Nonetheless, this drawback can be overcome by defining more lines of action per muscle although not more than two are usually specified in practice.

The following sections will discuss different (yet similar) approaches to muscle decomposition. Unless specified otherwise, all the images in this chapter are my own.

3.1 Kukačka method

The Kukačka method [KK14] provides a means of decomposing any muscle given that (a) internal structure of the muscle is known (see Figure 3.2) and (b) the muscle's origin and insertion points (attachment areas) to the bone are identified. The attachment areas (see Figure 3.3) are defined by sets of expert-specified landmarks, known as *points of interest*. These *points of interest* are fixed to the underlying bone and move with it accordingly.

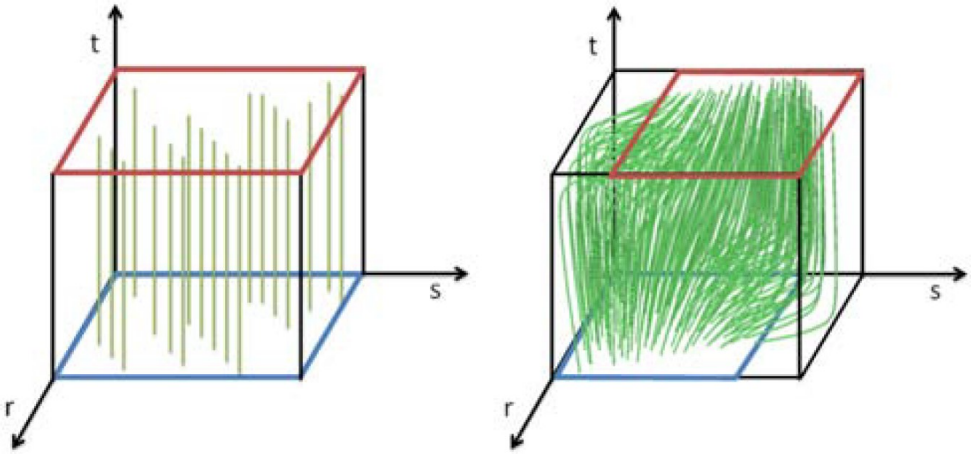
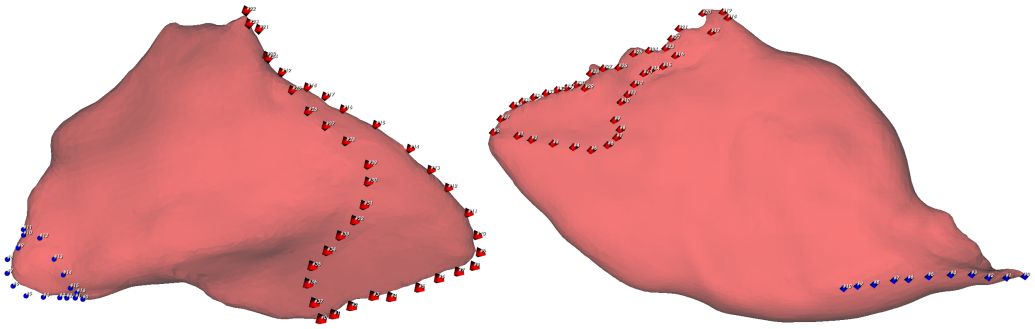


Figure 3.2: *Types of predefined fiber templates. Each template represents a unit space with two emphasized attachment areas on its bounds (red and blue rectangles). These attachment areas are connected via an arbitrary number of muscle fibers that are described by composite Bézier curves of orders ranging from 2 to 4. The four types of fiber templates are: parallel (left), pennate (right), fanned and curved. Image from [KK14]*



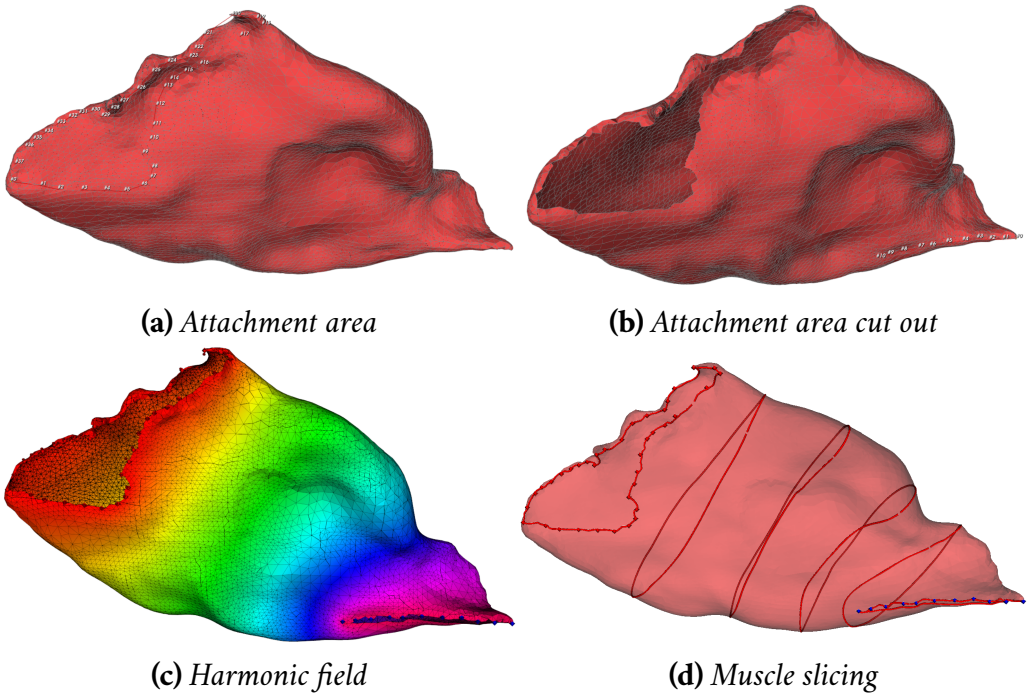
(a) *Gluteus medius* **(b)** *Gluteus maximus*

Figure 3.3: *Gluteus maximus and gluteus medius. The red and blue points are the expert-specified landmarks which denote an attachment area. Muscle sizes are not to scale*

The Kukačka method consists of two parallel processes that are combined to form the resulting muscle decomposition.

3.1.1 Process A

First, the attachment areas are projected onto the muscle surface mesh (Figure 3.4a). Then the projection-covered part of the mesh is removed, leaving only the mesh and two boundaries (Figure 3.4b). Next, a harmonic scalar field is calculated for the surface mesh (Figure 3.4c) and finally, the muscle is sliced into `N` number of slices (Figure 3.4d) where `N` is the user-specified number of line segments that each muscle fiber must be divided into. The slices are chosen as isolines of the scalar field.



(a) *Attachment area* **(b)** *Attachment area cut out*

(c) *Harmonic field* **(d)** *Muscle slicing*

Figure 3.4: *Figures that depict the muscle slicing process of the Kukačka method*

3.1.2 Process B

Second, using the knowledge of the internal structure of the muscle, a specific fiber template is chosen. This template is then sliced into `N` number of parallel planes, where `N` is the same as in Section 3.1.1. Figure 3.5 depicts how a fiber template is fitted to the dimensions of the muscle, creating a bounding box.

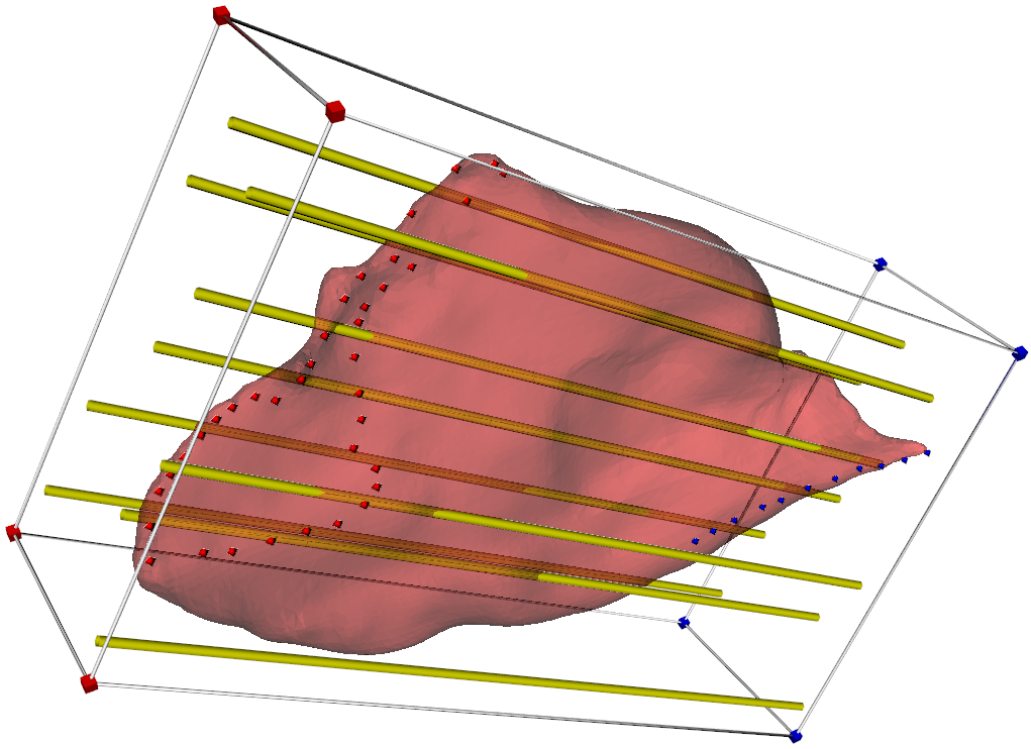
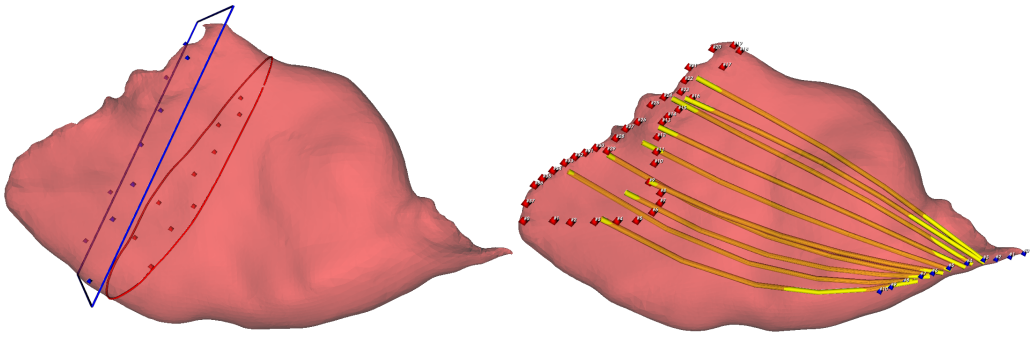


Figure 3.5: *Gluteus maximus inside a fiber template of type parallel*

3.1.3 Merging the processes

Finally, to complete the muscle decomposition process, the fiber template is mapped slice-by-slice (Figure 3.6a) into the interior of the muscle by exploiting the MVC coordinates. MVC coordinates (most valuable control coordinates) help with optimizing the deformation of a source shape into a target shape, for details - see [KK14].



(a) *Mapping the fiber points* **(b)** *The created fibers*

Figure 3.6: *The final stages of the muscle decomposition using the Kukačka method (a) and the result (b)*

Figure 3.7 shows a brief summarization of the decomposition process.

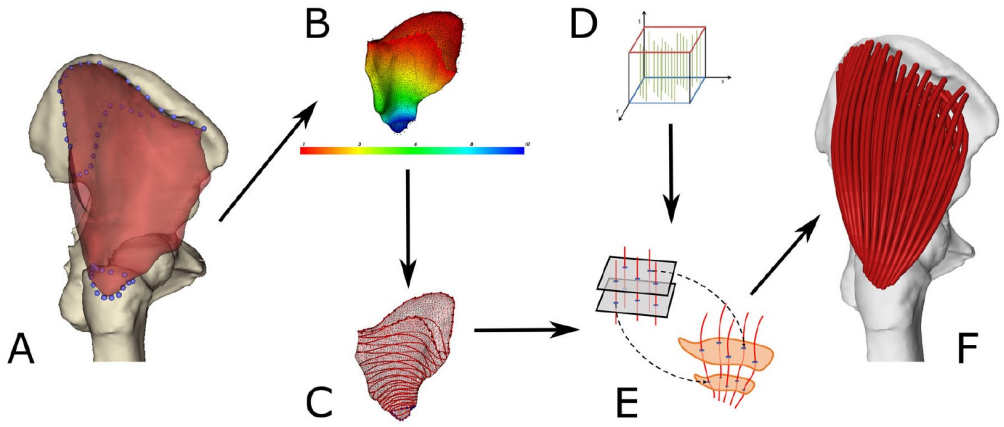


Figure 3.7: *Kukačka method for muscle decomposition. A - program input, B - harmonic scalar field for the muscle surface, C - extraction of iso-lines, D - fiber template selection, E - merge between the sliced muscle and template, F - program output. Image taken from* [*MK*20]

3.2 VIPER method

Another approach to decomposing muscles into fibers is the Volume Invariant Position-based Elastic Rods method, also known as VIPER [Ang+19]. The authors of this method, named the decomposition process *viperization*. VIPER is based on the Coserrat rods, but unlike them, the deformation of the rod does not change its volume, as seen in Figure 3.8. This modified rod is called a VIPER rod. Because this method utilizes a position-based dynamics approach to fiber simulation, it is well suited for modelling deformations e.g. in biomechanics.

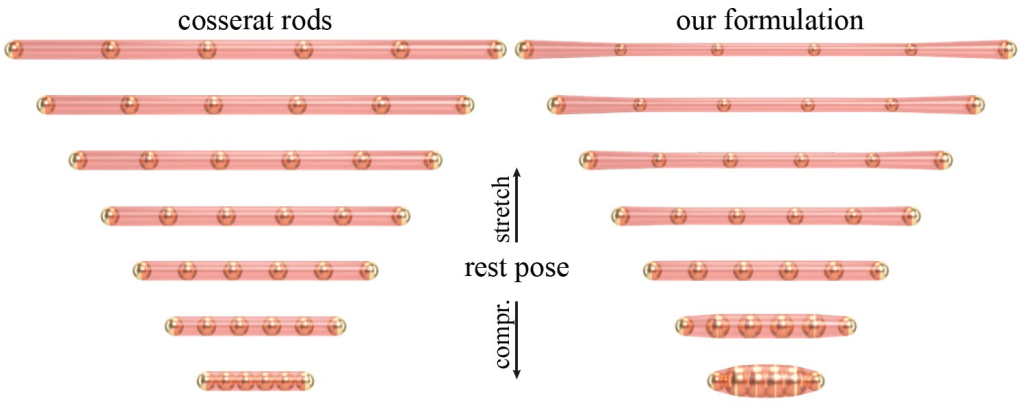


Figure 3.8: *Comparison between Cosserat rods (left) and VIPER rods (right). Image from* [*Ang+*19]

The VIPER method takes a muscle model as input, which is followed by a volumetric discretization of the muscle surface. The source and the sink of the muscle are marked[1] and a harmonic function is applied to the volume of the muscle, similar to the approaches of [CB13] and [KK14]. Next, a `K` number of iso-levels is extracted, each containing `N` points, which corresponds to the number of fibers to be created. These points are uniformly distributed within each iso-level using a *restricted centroidal Voronoi diagram* (CVD), and each of them corresponds to a VIPER rod vertex respectively. Lastly, an optimization algorithm is used to connect vertices between iso-levels in order to create the desired VIPER rods, while minimizing the sum length of the rods.

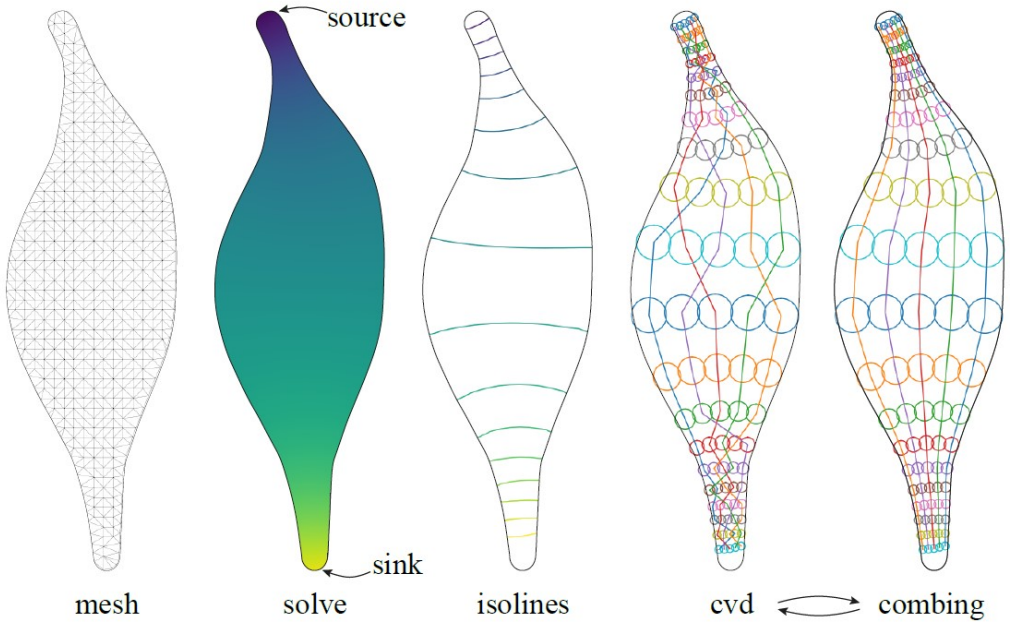


Figure 3.9: *Viperization process*

3.3 Comparing both methods

Table 3.1 outlines the key differences in input and fiber creation method between the two approaches.

While both Kukačka and VIPER use a harmonic scalar field to draw iso-lines, they differ in their initial input. Kukačka takes a 2D mesh as input, with the harmonic solve being calculated for the muscle surface while, VIPER takes in a 3D mesh as input and calculates the harmonic field for the muscle volume. This significant difference also affects the resulting slices. The ones produced by Kukačka are closed-space curves, while VIPER slices are bounded triangle meshes.

[1] Please notice the parallel with the Kukačka method and the origin and insertion points

Additionally, the methods diverge in their fiber creation process. Kukačka uses fiber templates to map the fibers onto the slices, while VIPER partitions each slice into evenly distributed areas using a restricted CVD and uses the resulting centroids as fiber points. These points are then connected across slices to create a fiber.

	Kukačka	**VIPER**
Input	2D mesh	3D mesh
Fiber creation	fiber templates	restricted CVD

Table 3.1: *The main differences between the Kukačka and VIPER methods*

3.4 New proposed method

This bachelor thesis proposes a novel method for muscle decomposition that combines the key elements of the two previously described approaches. Similar to Kukačka, the new method calculates a harmonic scalar field of the muscle surface and slices the muscle by the specified iso-lines. However, unlike Kukačka, the new method does not use fiber templates and instead takes inspiration from VIPER and its point distribution[2] done by a *centroidal Voronoi diagram*, see Table 3.2. Once the points have been generated on all slices, a matching algorithm will be used to connect the points across each slice, creating the desired fibers.

	Kukačka	**VIPER**	**new method**
Input	2D mesh	3D mesh	2D mesh
Fiber creation	fiber templates	point distribution	point distribution

Table 3.2: *The new method using key principles from Kukačka and VIPER*

It is important to note, that VIPER distributes the points based on the volume of the slice, and Kukačka slices are closed-space curves that can be described by non-planar non-convex polygons. Therefore one of the main challenges for the new method is the distribution of points within these non-planar non-convex polygons. This issue will be further explored in the following chapters.

[2] From now on, *point distribution* and *point generation* will be used interchangeably

Generating points in non-convex polygons

4

To address the challenge of distributing points inside non-planar non-convex polygons, we can first consider the problem of point generation inside planar non-convex polygons. This is, because the slices produced by Kukačka are not too dissimilar from planar non-convex polygons. The case for non-planar non-convex polygons will be discussed in the following chapter.

For now, different methods of point distribution inside planar non-convex polygons[1] will be explored.

4.1 Stochastic method

The stochastic method is the most basic approach for generating points inside a polygon, because it is done randomly. This makes the resulting distribution highly irregular, especially for a small number of points (< 1000). To minimize this drawback, a quasi-random sampling, such as the Sobol sequence, can be applied.

Kukačka uses the Sobol sequence during Process B (see Figure 3.5) to evenly distribute the fibers inside the fiber template. That ensures an even distribution of fibers inside the muscle. For more information, please refer to [KK14].

4.2 Poisson disk sampling

The Poisson disk sampling algorithm [Coo86] is used to randomly distribute points inside a plane but ensures that no two points are too close to each other (as seen in Figure 4.1). It does so by using a parameter `r` which is the minimal distance between any two points. Various implementations of the Poisson disk sampling are showcased in the following sections.

[1] For the rest of this chapter, planar non-convex polygons will be referred to as polygons

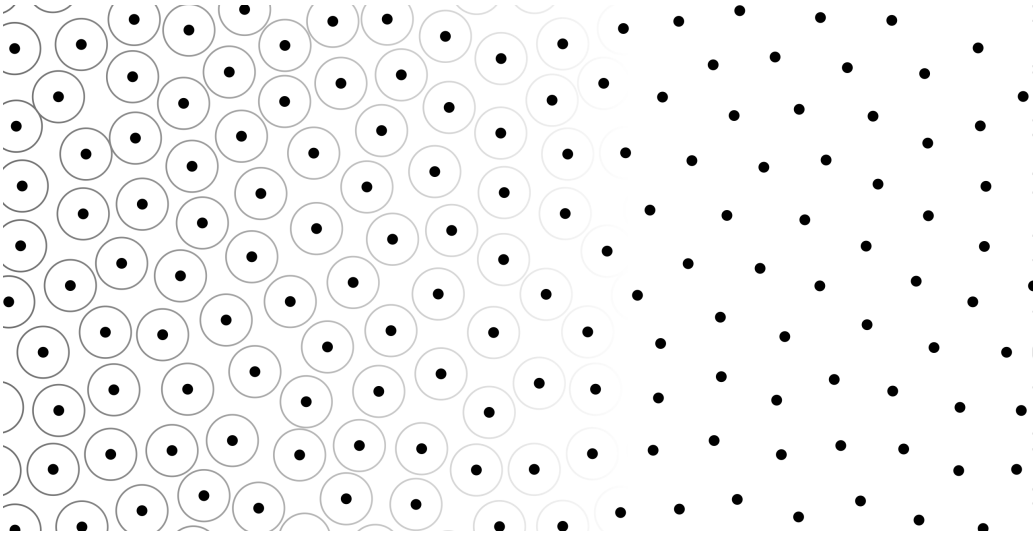


Figure 4.1: *Points distributed via the Poisson disk sampling*[2]

4.2.1 Naive implementation

The naive implementation for point distribution involves randomly placing points on a plane, but with a check to ensure that every new point P_{x+1} is not too close to any existing point $\in \{P_0, P_1, \dots, P_{x-1}, P_x\}$. If a conflict occurs, the point P_{x+1} is discarded and a new one is generated instead. This process is repeated until the desired number of points is generated or until parameter `k` is exceeded. Parameter `k` is the maximum number of attempts for generating a point. Without this parameter, the algorithm could run indefinitely in an endless loop.

4.2.2 Bridson's implementation

A much more efficient algorithm, called *Fast Poisson disk sampling*, has been proposed by [Bri07]. The idea is that instead of checking every point for conflicts, it is sufficient to check only the closest points. This is achieved by creating a background grid with a cell size of $\frac{r}{\sqrt{2}}$, see Figure 4.2. The newly generated points are assigned to a cell in the grid and are only checked against points in nearby cells.

This algorithm starts with an initial point `P` and generates points around it within a distance range of $\langle r, 2r \rangle$. A maximum of `k` attempts is made to generate these points. After `k` attempts, one of the newly generated points is selected and the process is repeated. It is worth noting the difference in the role of `k` between Bridson's implementation and the naive one. For more details see Algorithm 1.

[2] Source: `https://en.wikipedia.org/wiki/Supersampling#Poisson_disk`

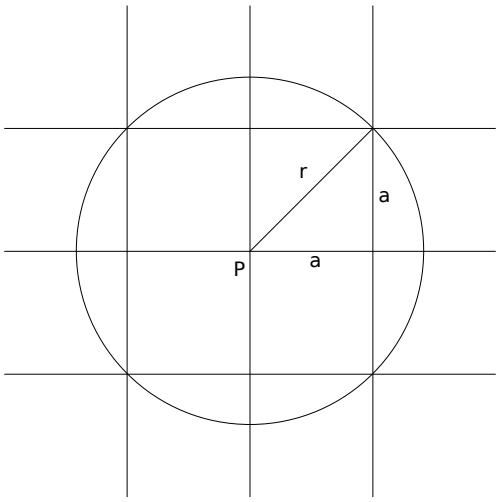


Figure 4.2: *Size of a grid cell* ***a*** *in relation to parameter* ***r*** *- the minimal distance between points. If a new point was to be added into this picture, it would need to be outside the ring that is around point* ***P****. Keep in mind that rings of two different points can intersect each other*

Algorithm 1: Fast Poisson disk sampling

Input: n – number of points to generate
Input: r – minimum distance between points
Input: k – number of attempts to generate points around a point
Output: List of generated points

Create an empty 2D grid with cell size of $\frac{r}{\sqrt{2}}$
Create two lists, an *active* list and *result* list
Randomly create an initial point and put it into both lists
while *active list* $\neq$ *empty* $\wedge$ *size(result)* $<$ ***n*** **do**
    Randomly choose a point P from the *active* list
    Randomly generate points around P with distance $\in \langle \mathbf{r}, \mathbf{2r} \rangle$
    forall *generated points around P* **do**
        Check in the nearby cells for conflict
        if *conflict = false* **then**
            Add the generated point to the *active* list and *result* list
        end
    end
    if *after* ***k*** *attempts no point has been successfully generated* **then**
        Remove point P from the *active* list
    end
end
return *result list*

4.2.3 Polygon implementation

In order to generate points inside a polygon, Bridson's algorithm requires slight modifications. Specifically, the procedure needs to take the polygon as an additional input and check whether each generated point is inside the polygon or not. This problem is commonly known as *point-in-polygon*, and it is typically solved by using a ray casting algorithm[3]. The modified pseudocode is seen in Algorithm 2.

Algorithm 2: Fast Poisson disk sampling inside a polygon

Input: n – number of points to generate
Input: r – minimum distance between points
Input: k – number of attempts to generate points around a point
Input: p – polygon to generate the points into
Output: List of generated points

```
Create an empty 2D grid with cell size of r/√2
Create two lists, an active list and result list
Randomly create an initial point and put it into both lists
while active list ≠ empty ∧ size(result) < n do
    Randomly choose a point P from the active list
    Randomly generate points around P with distance ∈ ⟨r, 2r⟩
    forall generated points around P do
        if generated point ∉ polygon then
            Continue
        end
        Check in the nearby cells for conflict
        if conflict = false then
            Add the generated point to the active list and result list
        end
    end
    if after k attempts no point has been successfully generated then
        Remove point P from the active list
    end
end
return result list
```

[3] For more details see: `https://en.wikipedia.org/wiki/Point_in_polygon`

4.3 Centroidal Voronoi diagram

A Voronoi diagram[4] is a partitioning of a plane into regions, where each region is associated with a point called a *Voronoi seed*. In any given region, every point is closer to its seed than to any other seed in the whole plane[5]. These regions are referred to as *Voronoi cells* and each cell contains one *Voronoi seed*. A *centroidal Voronoi diagram* is a special type of Voronoi diagram, where the Voronoi seed is also the centroid (i.e., center of mass) of its corresponding Voronoi cell[6]. Please see Figure 4.3 for a visual representation of the differences between Voronoi and centroidal Voronoi diagrams.

(a) *Generic Voronoi diagram* **(b)** *Centroidal Voronoi diagram*

Figure 4.3: *Voronoi diagrams with 5 generating points*[7]

To achieve an even distribution of points inside a polygon, one approach is to utilize the *Voronoi seeds* themselves, a technique employed by VIPER to get an even distribution of VIPER rod vertices inside a slice (see 3.2 for details). Two commonly used methods for achieving a *centroidal Voronoi tesselation* are:

1. Lloyd's algorithm for K-means clustering, as described in Algorithm 3
2. McQueen's algorithm, involves random sampling and averaging

[4] In some sources also called a Voronoi tesselation
[5] Source: `https://mathworld.wolfram.com/VoronoiDiagram.html`
[6] Source: `https://w.wiki/6eL5`
[7] Created at: `https://alexbeutel.com/webgl/voronoi.html`

Algorithm 3: K-means clustering inside a polygon

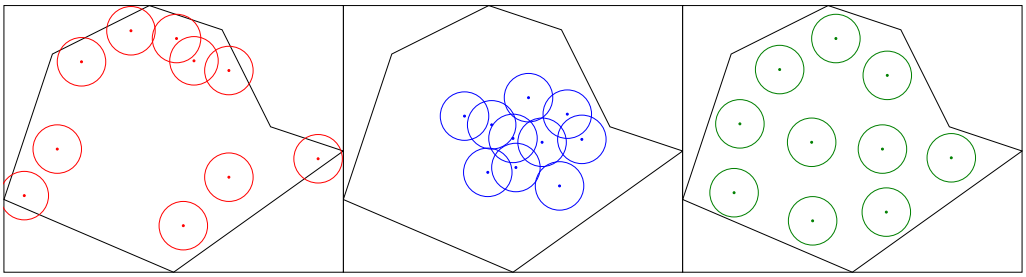
Input: n – number of points to generate
Input: p – polygon to generate the points into
Output: List of generated points

```
1  Randomly populate the polygon p with a large number of points P_{0→i}
2  Randomly select n points as initial cluster centers
3  while true do
4      forall points P_{0→i} do
5          Assign point to the nearest cluster center
6      end
7      forall cluster centers do
8          Calculate the new position as the mean of the cluster's points
9      end
10     if cluster centers moved positions ≠ true then
11         Terminate loop
12     end
13 end
14 return cluster centers
```

After running the K-means clustering algorithm, a set of cluster centers is returned. These cluster centers correspond to the evenly distributed points inside a polygon. Please note that the algorithm pre-populates the polygon with points, leveraging the *Law of large numbers* to obtain a uniform distribution. These points are then used to form the clusters.

4.4 Test implementation

To evaluate the real-world performance of the three methods mentioned before, I implemented them in a program, which is available at my GitHub repository[8]. All three methods take a polygon and the number of points to generate as input. The initial comparison can be seen in Figure 4.4.

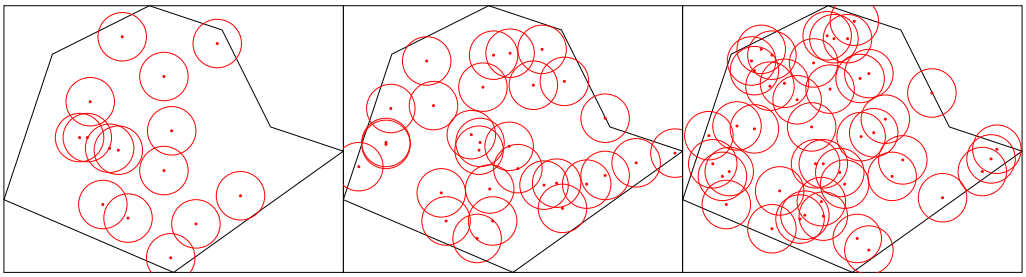


(a) *Random* **(b)** *Poisson*[9] **(c)** *Voronoi*

Figure 4.4: *Generating* 10 *points with different distribution methods. For better visualization, circles were drawn around the points, although in the Poisson disk sampling method, the circle radius determines the minimal distance between two points*

As seen in Figure 4.4, the random sampling method has the worst result, with several points overlapping and being clustered in some locations of the polygon. The Poisson disk sampling does not have a problem with overlapping points but due to its random nature, the polygon is populated non-uniformly. The k-means method yields the best results with all points being distributed evenly across the whole polygon.

Figures 4.5 to 4.8 showcase all three methods and the influence of parameters on the results.

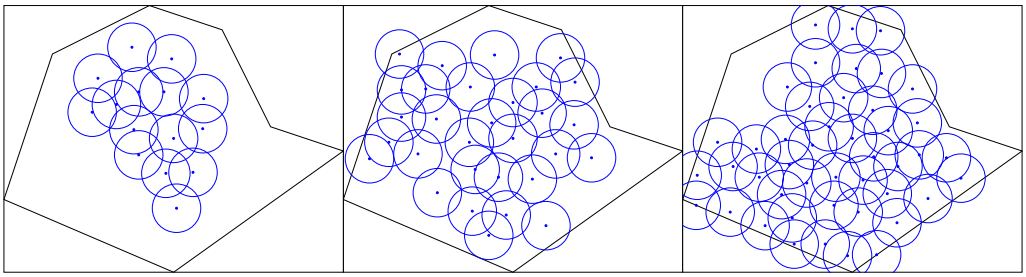


(a) 15 *points* **(b)** 30 *points* **(c)** 45 *points*

Figure 4.5: *Generating points with the stochastic method. It is evident that increasing the number of generated points improves the result as the points are more evenly distributed*

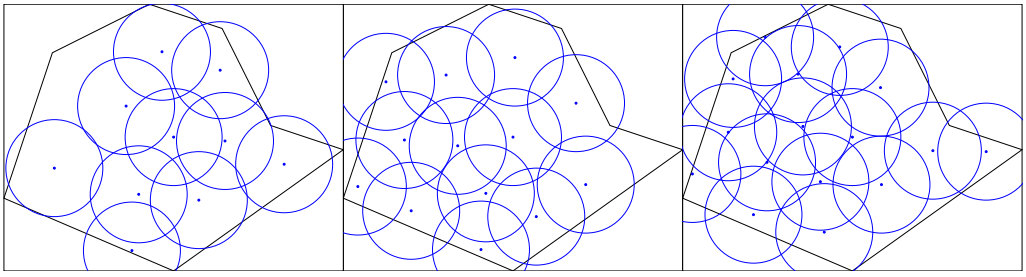
[8] Source codes: `https://github.com/DucLongHoang/PointGeneration.git`

[9] Implementation details from: `https://youtu.be/7WcmyxyFO7o` and `https://youtu.be/flQgnCUxHlw`



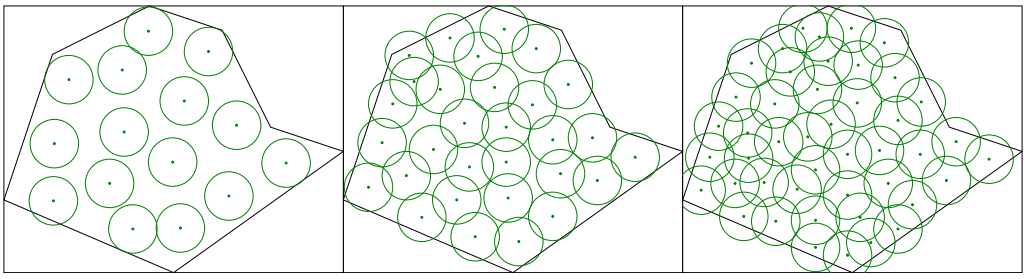
(a) 15 *points* **(b)** 30 *points* **(c)** 45 *points*

Figure 4.6: *Generating points with the Poison disk sampling method with a minimal distance of* 25 *units. The results are promising, and they could be improved if the minimal distance between two points could be calculated automatically based on the polygon's surface area*



(a) 10 *points* **(b)** 15 *points* **(c)** 20 *points*

Figure 4.7: *Generating points with the Poison disk sampling method with a minimal distance of* 50 *units. Please note, that there were only* 13 *and* 16 *points generated in figures b and c respectively. That is due to the method's rejection parameter, that prematurely terminates the process*



(a) 15 *points* **(b)** 30 *points* **(c)** 45 *points*

Figure 4.8: *Generating points with the centroidal Voronoi diagram. This method gives the best results for any number of points and, unlike the Poisson disk sampling method, it is not dependent on any additional parameters*

This brief test implementation which served as a *proof of concept* concludes this chapter.

Generating points in non-planar non-convex polygons

5

In the previous chapter, we explored different methods for generating points inside planar non-convex polygons that represented muscle slices. However, in this chapter we will consider the more realistic scenario. That is, muscle slices are non-planar non-convex polygons and we will discuss how to distribute points inside them.

This issue can be easily overcome by first transforming the muscle slice into a planar polygon. The polygon is then populated with the desired number of points by using one of the methods from the previous chapter. Finally, the polygon is transformed back, mapping the distributed points accordingly.

Therefore the main obstacle in this chapter is the transformation of 3D objects into 2D, et vice versa, which will be the focus of the following sections.

5.1 Best fitting plane

In the same way, it is possible to find a best-fitting line for a set of XY points using linear regression, we can likewise do so for points with XYZ coordinates by utilizing the principal component analysis (PCA). This is a statistical method, whose goal is to reduce the dimensionality of a dataset, while retaining as much of the original information as possible.

It works, by finding the directions of maximum variance in the initial dataset, these are called the *principal components*. The first principal component is the direction in which the data varies the most, followed by the second principal component, and so on. Please see Figure 5.1.

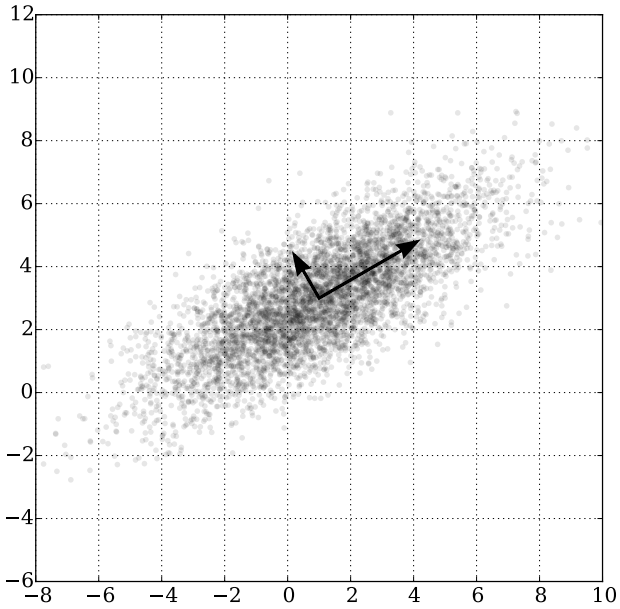


Figure 5.1: *PCA with the first and second principal components*[1]

Coincidentally, the direction of the first principal component is the same as the normal vector of the best fitting plane of the dataset. Therefore, by applying the PCA on a set of 3D points we can find the plane of best fit that corresponds to these points (Figure 5.2). This information can be used to find the best fitting plane of a given muscle slice, which is described by a non-planar polygon. This non-planar polygon is then projected onto the plane of best fit to obtain a planar polygon.

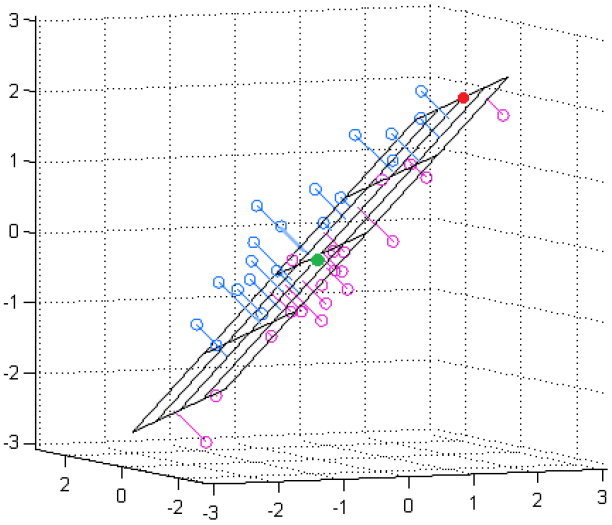


Figure 5.2: *Projection of points onto a plane of best fit*[2]

[1] Source: `https://w.wiki/3kQ3`
[2] Source: `https://math.stackexchange.com/q/3501135`

However, this method has a drawback. Once the planar polygon is populated with the points, transforming the polygon back with the points projected correctly is a complex task. We might know the direction in which to projected the point, but not the projection distance. That is due to the missing information about the surface of the slice, as we only know its contour (iso-line).

5.2 Thin plate membrane

One way to overcome the drawback of the previous method, is to reconstruct the surface of the muscle slice by using a hole-filling algorithm. One example of such an algorithm is the *thin plate membrane* method which tries to minimize the deformation energy of the reconstructed surface. Imagine trying to wrap a bowl of food with plastic wrap. The edge of the bowl is our muscle slice contour and the plastic wrap on top of the bowl is the reconstructed surface. If there is a lot of food in the bowl then the plastic wrap has to wrap around the food as well, creating a small peak - that is the deformation energy that we want to minimize. The same analogy can be used for the bowl having uneven edges. Ideally, the plastic wrap (as well as the reconstructed surface) should look like a membrane, hence the name of the method.

After the surface has been reconstructed, it will be possible to continue like in the previous method with the difference that this time we will know the projection distance of the fiber points - it is the intersection between the projection (defined as a line) and the membrane (represented by a triangle mesh). This can be effectively solved using the barycentric coordinates of a triangle.

Though an improvement over the previous method, the thin plate membrane still has a drawback. Projections in general distort space, and so the triangles of the triangle mesh will not preserve their surface area and the placement of the generated points could be irregular.

5.3 Mesh unfolding

The last approach, similar to the previous one, also reconstructs the surface of the slice but differs in the following steps. Once a triangle mesh has been created, it is then unfolded creating a planar polygon, which will be populated by the fiber points using a method from the previous chapter. This method ensures that the points are distributed evenly across the whole mesh.

Mesh unfolding is a very complex process - the unfolding could produce overlapping folds and we are not guaranteed to obtain a satisfactory result. There also exist many different ways how to unfold a mesh, Figure 5.3 shows an implementation by Korpitsch et al [Kor+20] - this specific approach will not necessarily be used in the final implementation.

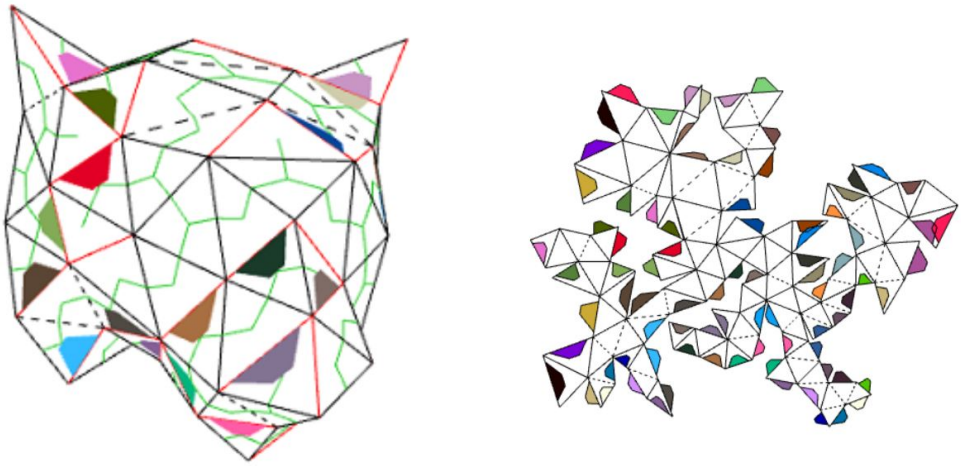


Figure 5.3: *3D mesh (left), unfolded mesh (right). Image from [Kor+20]*

5.4 Comparison

In Table 5.1 we can see the main differences among the three approaches described in this chapter. The best fitting plane method uses PCA to get a plane of best fit, projects the polygon onto the plane, generates points into the projected polygon, and projects it all back onto the initial slice. The thin plate membrane approach also uses the PCA and projects the points onto a reconstructed surface of the muscle slice. The mesh unfolding method also reconstructs the surface but unfolds the obtained mesh into a planar polygon on which it generates the points.

methods used	best fit plane	thin plate membrane	mesh unfolding
PCA	✓	✓	✗
projection	✓	✓	✗
surface rec.	✗	✓	✓
unfolding	✗	✗	✓

Table 5.1: *Quick comparison of the best fitting plane, thin plate membrane, and mesh unfolding methods*

Generating points in a muscle slice — 6

As described in Chapter 5, one of the methods to transform a non-planar polygon into a planar polygon, is to find its *best-fitting plane* (BFP) and project every point of the polygon[1] onto the plane. The result of such a projection can be seen in Figure 6.1. Unless stated otherwise, all figures in this chapter are my own.

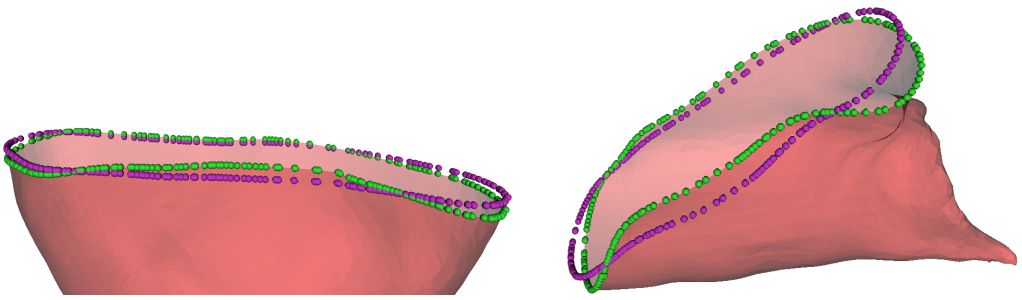
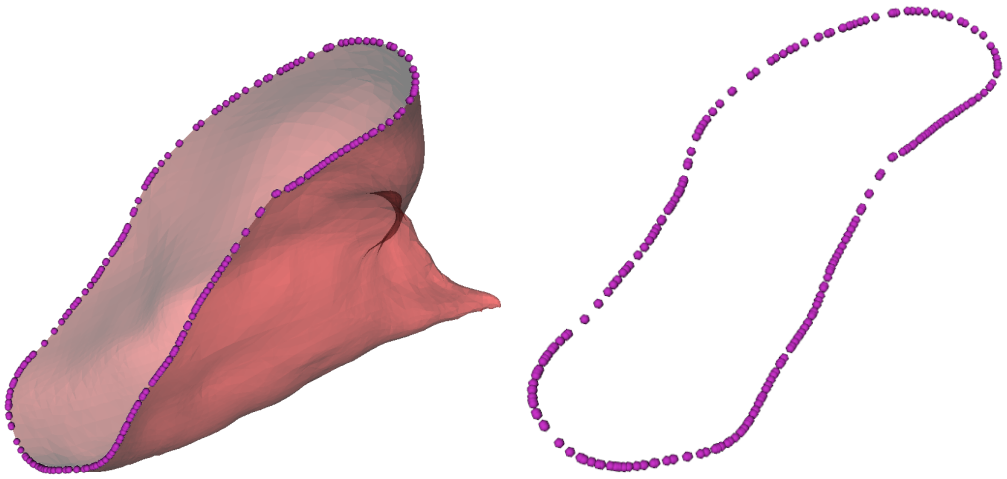


Figure 6.1: *Two instances of a BFP projection (purple) of a slice (green)*

At this point, it is still not possible to generate points using methods from Chapter 4 inside these polygons. Although they are planar, they have more than 2 non-zero coordinates in Euclidean space so it is essential to transform these polygons down to 2 dimensions. One solution is to extract the current frame of reference and the relative coordinates of a point and represent it in a different coordinate system - for details see [SE03] page 128, Chapter 4.6. An example of such a transformation can be seen in Figure 6.2, the implementation was provided to me by my thesis advisor.

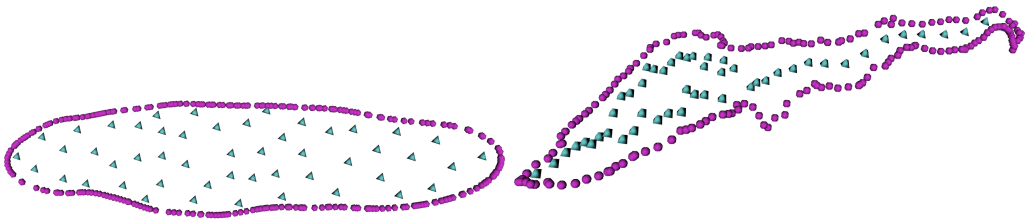
[1] From now on, a *polygon* and a *set of points that define a polygon* will be used interchangeably, to refer to the same concept



(a) *Muscle slice* **(b)** *2D polygon*

Figure 6.2: *Initial muscle slice contour being transformed to 2D*

Now that the planar polygons are obtained, it is possible to generate points inside them. For doing so, the *Centroidal Voronoi diagram* (CVD) method described in Section 4.3 has been chosen, and as we can see in Figure 6.3a, this results in an even distribution of the points inside the whole polygon.

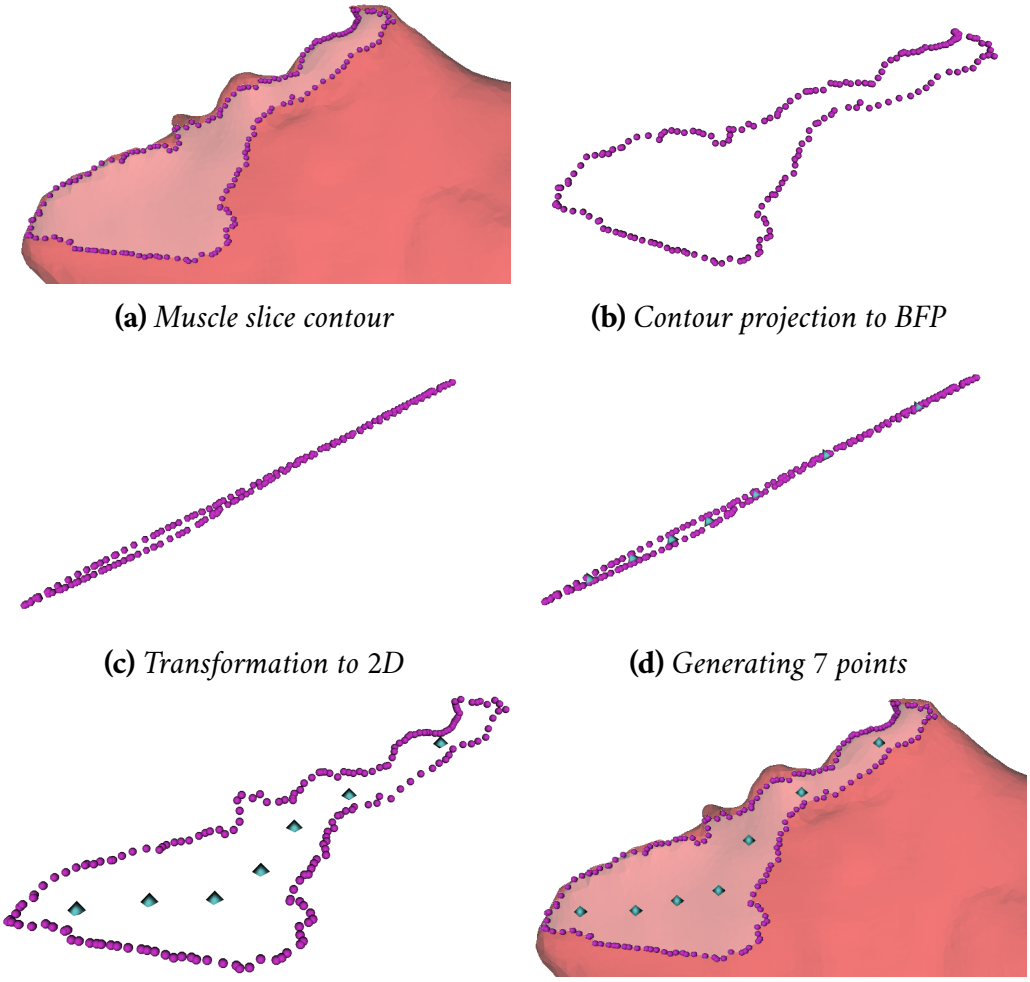


(a) *Points in planar polygon* **(b)** *Mapping the result back*

Figure 6.3: *Generating points inside planar polygons (a), mapping the polygon and generated points back onto the initial contour (b). These two polygons do not necessarily correlate to each other*

With the points now generated inside the planar polygons we can map them back to the non-planar polygon using an inverse transformation. By doing so, all our generated points are mapped as well - see Figure 6.3b

The whole process of generating 7 points on the same muscle slice can be seen in Figure 6.4.



(a) *Muscle slice contour* **(b)** *Contour projection to BFP*

(c) *Transformation to 2D* **(d)** *Generating 7 points*

(e) *Initial muscle slice with generated points* **(f)** *Visualization with the muscle*

Figure 6.4: *The procedure starts by slicing the muscle (for details see Section 3.1 and [KK14]) and leaves us with a muscle contour (a). Later a plane of best fit is calculated and the contour is projected onto it (b). This is followed by a transformation of the polygon into 2D (c) - notice the almost line-like shape. Later, points are generated inside the planar polygon (d), which is then transformed back into the original contour (e). Picture (f) visualizes the final state with the generated points being inside the muscle. Please observe that once transformed back, the points are no longer evenly distributed due to the point generation happening on a very thin polygon*

Once the process has been repeated for every muscle slice we should get a result similar to the one in Figure 6.5.

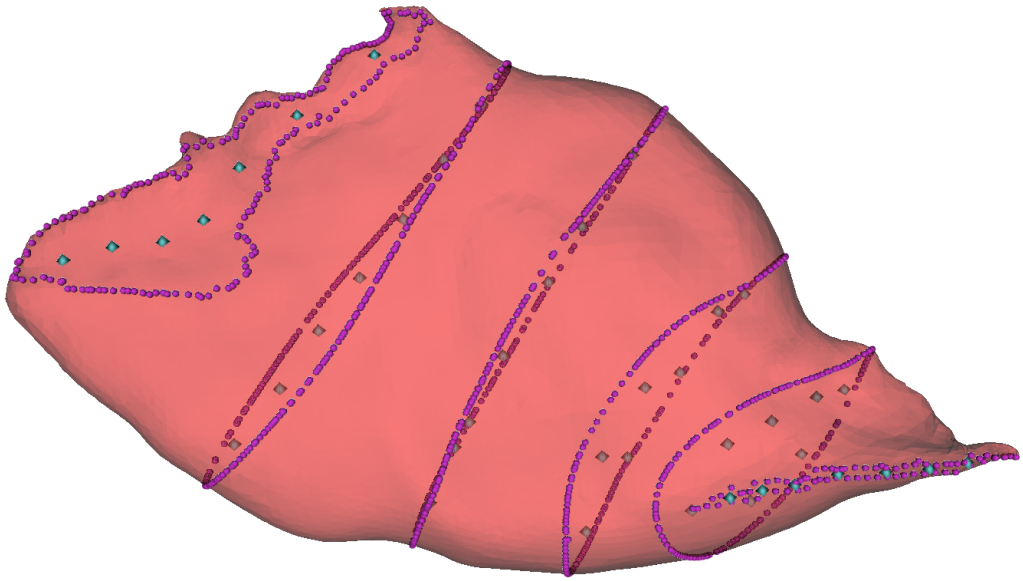


Figure 6.5: *Points generated on all slices*

The next task is to connect points between every two slices to receive muscle fibers. Optimizing the result means correctly assigning the points so that the sum of their lengths is minimized. This task belongs to the class of optimal assignment problems and will be further explored in the following chapter.

Connecting points between neighboring slices

7

Now that all the points have been generated on all the slices, they need to be correctly assigned to each other to create the resulting fibers. The Kuhn-Munkres algorithm, also known as the Hungarian Marriage algorithm or Munkres algorithm, has been chosen to reach the result. The following sections will shortly explain the algorithm as well as how it can be used for the problem of assigning fiber points to each other.

7.1 Kuhn-Munkres algorithm

This combinatorial optimization algorithm [Kuh55] was based on the works of two Hungarian mathematicians, Dénes Kőnig and Jenő Egerváry, hence the alias of *Hungarian marriage*. The purpose of this algorithm and the reason it was created is best explained in an example.

Let us assume we have `N` number of workers and `N` different tasks that each worker is capable of doing for a different amount of money. The desired outcome is to utilize all the workers and accomplish all the tasks while minimizing the cost. This situation is visualized in Table 7.1.

Solving this problem using brute force by calculating all the possible solutions and choosing the best has a time complexity of $n!$ while the Munkres algorithm arrives at the same solution with a algorithmic complexity of n^3.

	Alice	**Bob**	**Rick**
Maintenance	$10	$20	$16
Rolling	$12	$15	$15
Cooking	$19	$14	$17

Table 7.1: *Cost for each worker to do a task*

From this table it is possible to extract an `N` x `N` cost matrix as can be seen in Equation 7.1.

$$\begin{pmatrix} 10 & 20 & 16 \\ 12 & 15 & 15 \\ 19 & 14 & 17 \end{pmatrix} \tag{7.1}$$

After applying the Kuhn-Munkres algorithm[1] to the matrix in question, we will obtain a new matrix that represents the optimal assignments (see Equation 7.2). In this matrix, each cell that corresponds to the assignment contains a 0, and only one 0 appears per row and column, indicating the chosen worker for each task (optimal assignments are highlighted in yellow).

$$\begin{pmatrix} 0 & 10 & 3 \\ 0 & 3 & 0 \\ 5 & 0 & 0 \end{pmatrix} \tag{7.2}$$

Transferring the highlighted locations from the result matrix to the initial table will result in Table 7.2. We can see that the solution is to have Alice do the Maintenance, Bob do the Cooking and assign Rick to Rolling , this permutation of worker-task pairs has minimized the total cost to $39.

	Alice	**Bob**	**Rick**
Maintenance	$10	$20	$16
Rolling	$12	$15	$15
Cooking	$19	$14	$17

Table 7.2: *Table with optimal assignments highlighted in yellow*

Solving the assignment problem using brute force (see Equation 7.3), we can see that the minimal cost is indeed $39.

$$\begin{aligned} 10 + 15 + 17 &= 42 \\ 10 + 14 + 15 &= 39 \\ 12 + 20 + 17 &= 49 \\ 12 + 14 + 16 &= 42 \\ 19 + 20 + 15 &= 54 \\ 19 + 15 + 16 &= 50 \end{aligned} \tag{7.3}$$

7.2 Algorithm application

Now that we know what the Munkres algorithm is and what it solves, we can apply it to the problem of fiber points assignment. Let us consider two neighboring slices,

[1] Implementation details at: `https://brilliant.org/wiki/hungarian-matching/`

slice A and slice B for which we want to find the correct assignment of the fiber points. We can imagine that the fiber points of slice A are the *workers* and that the fiber points of slice B are the *tasks*, see Table 7.3. By executing the Hungarian algorithm we will obtain a matrix where each fiber point from slice A is assigned to a fiber point in slice B. The values inside the matrix are the cost of assigning the points to each other - it could be something as simple as the Euclidean distance between two points or the cosine distance or maybe even a combination of both.

		Slice A		
		point_{A1}	point_{A2}	point_{A3}
Slice B	point_{B1}	$w(p_{A1}, p_{B1})$	$w(p_{A2}, p_{B1})$	$w(p_{A3}, p_{B1})$
	point_{B2}	$w(p_{A1}, p_{B2})$	$w(p_{A2}, p_{B2})$	$w(p_{A3}, p_{B2})$
	point_{B3}	$w(p_{A1}, p_{B3})$	$w(p_{A2}, p_{B3})$	$w(p_{A3}, p_{B3})$

Table 7.3: *Using the Munkres algorithm to solve the fiber point assignments for slice A and slice B*

After solving the first matrix, the Hungarian marriage algorithm is then again applied for slices B and C, see Table 7.4. This process of solving the matrix is repeated $k - 1$ times where k is the number of slices. Given the knowledge of the Munkres algorithm's computational complexity of n^3 where n is the number of fibers, it can be assumed that the time complexity of assigning all fiber points to each other across all slices will be $O(n) = n^3 * (k - 1)$.

		Slice B		
		point_{B1}	point_{B2}	point_{B3}
Slice C	point_{C1}	$w(p_{B1}, p_{C1})$	$w(p_{B2}, p_{C1})$	$w(p_{B3}, p_{C1})$
	point_{C2}	$w(p_{B1}, p_{C2})$	$w(p_{B2}, p_{C2})$	$w(p_{B3}, p_{C2})$
	point_{C3}	$w(p_{B1}, p_{C3})$	$w(p_{B2}, p_{C3})$	$w(p_{B3}, p_{C3})$

Table 7.4: *Using the Munkres algorithm to solve the fiber point assignments for slice B and slice C*

For this bachelor's thesis, the cost of assigning two fiber points of neighboring slices to each other has been a linear combination of the points' normalized Euclid distance in 3D (see Equation 7.4) and cosine distance (see Equation 7.5).

$$w_e(p_{Ai}, p_{Bj}) = d_e(p_{Ai}, p_{Bj}) = \frac{\sqrt{\sum_{n=1}^{3}(p_{Bj_n} - p_{Ai_n})^2}}{w_{e_{max}}} \quad \text{where}$$

$$p_{Ai} \text{ is the } i\text{-th point of Slice A} \tag{7.4}$$

$$p_{Bj} \text{ is the } j\text{-th point of Slice B}$$

$$w_{e_{max}} \text{ is the max weight of any two points } p_{Ai} \text{ and } p_{Bj}$$

$$w_{cos}(p_{Bi}, p_{Cj}) = d_{cos}(\overrightarrow{p_{A^*}p_{Bi}}, \overrightarrow{p_{Bi}p_{Cj}}) = \cos(\theta) = \frac{\overrightarrow{p_{A^*}p_{Bi}} \cdot \overrightarrow{p_{Bi}p_{Cj}}}{|\overrightarrow{p_{A^*}p_{Bi}}| \cdot |\overrightarrow{p_{Bi}p_{Cj}}|} \quad \text{where}$$
$$p_{Bi} \text{ is the } i\text{-th point of Slice B}$$
$$p_{Cj} \text{ is the } j\text{-th point of Slice C} \tag{7.5}$$
$$p_{A^*} \text{ is a point from Slice A that } p_{Bi} \text{ is assigned to}$$
$$\theta \text{ is the angle between vectors } \overrightarrow{p_{A^*}p_{Bi}} \text{ and } \overrightarrow{p_{Bi}p_{Cj}}$$

The linear combination of these two metrics can be defined as seen in Equation 7.6 with α and β being parameters.

$$w_{total} = \alpha * w_{cos}(p_{Ai}, p_{Bj}) + \beta * w_e(p_{Ai}, p_{Bj}) \quad \text{where}$$
$$\beta = 1 - \alpha \tag{7.6}$$
$$\alpha \geq 0 \quad \text{and} \quad \beta \geq 0$$

Please note that some slight modifications need to be done as the normalized Euclid distance $\in \langle 0, 1 \rangle$ and the cosine distance $\in \langle -1, 1 \rangle$ where 1 indicates identical vectors and -1 indicates opposite vectors. Therefore it is needed to change the range of the cosine distance to $\langle 0, 1 \rangle$ which can be done in 2 ways:

1. $d_{cos} = \begin{cases} d_{cos} & \text{if } d_{cos} \in \langle 0, 1 \rangle \\ 0 & \text{otherwise} \end{cases}$
   because a value of 0 indicates right-angle vectors and anything greater would mean a fiber bending backwards

2. $d_{cos} = \frac{d_{cos}+1}{2}$
   which transforms the interval $\langle -1, 1 \rangle$ to $\langle 0, 1 \rangle$

We also cannot omit inverting the values, as cosine distance of 1 is our desired value but the Munkres algorithm is operating based on minimal costs.

With these modifications, the cost of connecting two points should look like in Equation 7.7.

$$w_{total} = \alpha * (1 - w_{cos}(p_{Ai}, p_{Bj})) + \beta * w_e(p_{Ai}, p_{Bj}) \tag{7.7}$$

For better understanding of the costs, please see Figure 7.1.

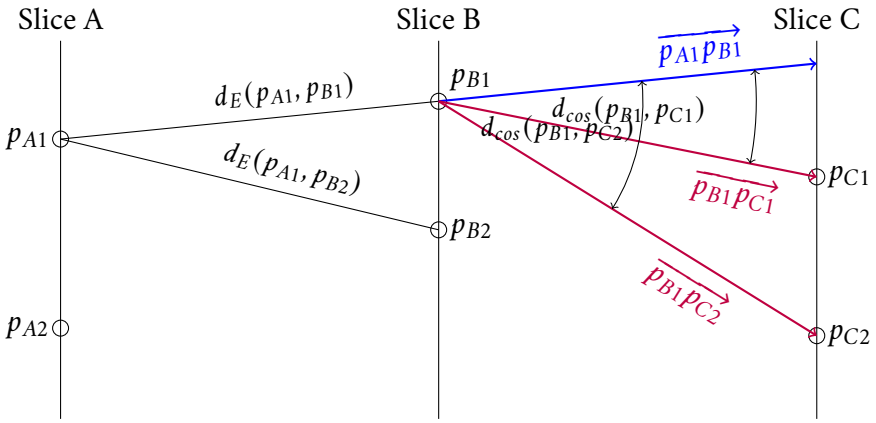
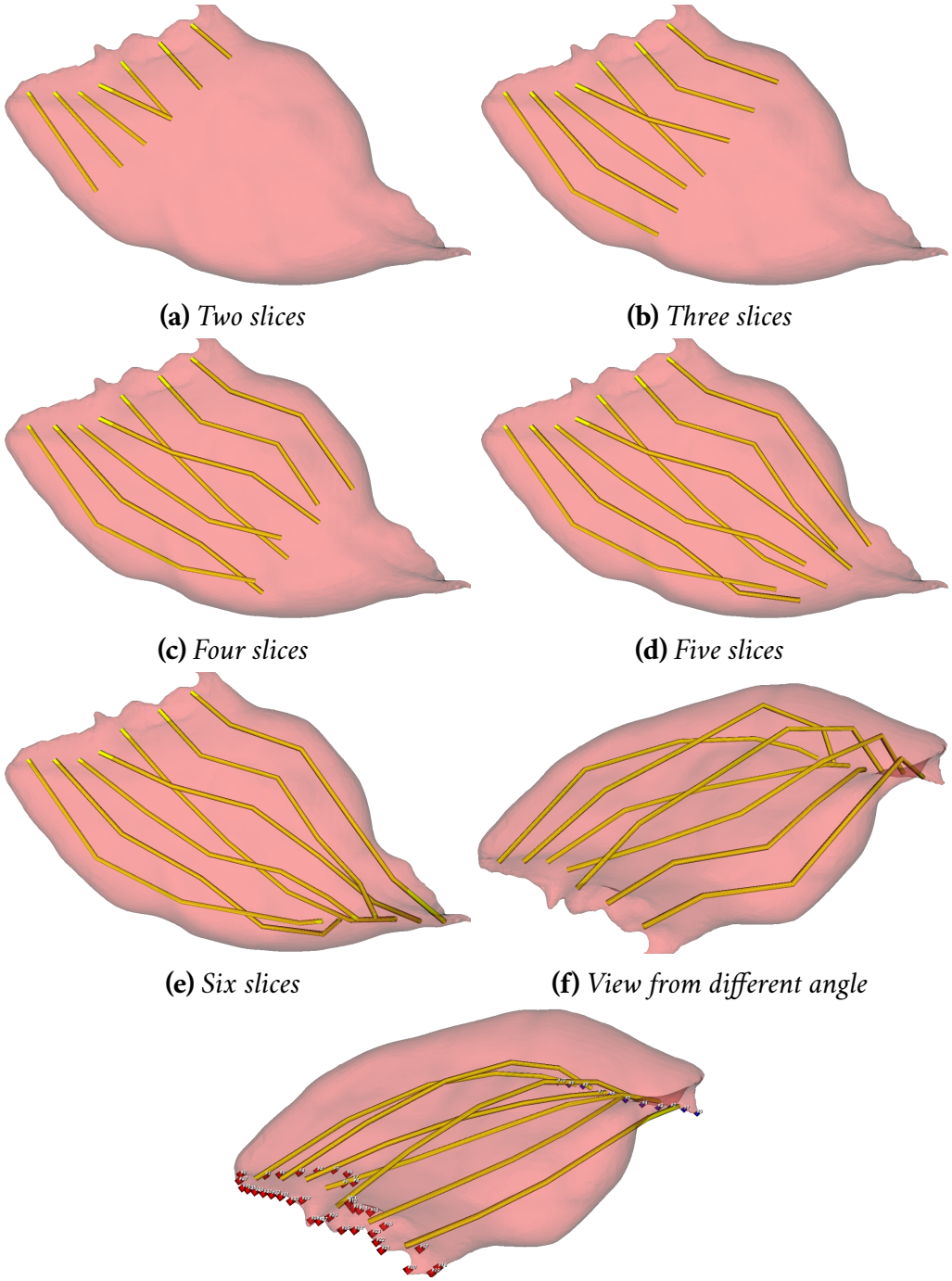


Figure 7.1: *The assignment of points between slices (visualized in 2D) and the assignment cost using the Euclid and cosine distance. The fiber points of slice A and slice B are assigned using Euclidian distance only. All subsequent pairs are using a linear combination of the Euclidian and cosine distance. The cost of assigning point p_{B1} to point p_{C1} is calculated as the cosine distance between vector $\overrightarrow{p_{A1}p_{B1}}$ (blue) which consists of the points p_{B1} and p_{A1} (a point from the previous slice it was assigned to) and the vector $\overrightarrow{p_{B1}p_{C1}}$ (red)*

Chapter 6 outlined the process of generating points inside muscle slices and the result was shown in Figure 6.5. Adding to that, all the theoretical concepts presented in this chapter about matching the points between two slices, we can visualize a process similar to Figure 7.2.



(a) *Two slices*

(b) *Three slices*

(c) *Four slices*

(d) *Five slices*

(e) *Six slices*

(f) *View from different angle*

(g) *Result with line smoothing turned on*

Figure 7.2: *The process of connecting fiber points of 7 fibers across 6 slices. Although fibers are crossing each other starting from subfigure B, it is due to the 2D view and if seen from different angles (subfigure F) the fibers behave correctly. From subfigure G we can observe the impact of smoothing on the fibers. The fibers are less jagged, but they do not fill out the muscle as well as before*

Implementation 8

In this Chapter, we will discuss the key implementation parts of this bachelor's thesis, how they are dependent on each other and how together they expanded the Muscle Wrapping 2.0 application with new features.

8.1 Overview of the Code

The most important `.h` and `.cpp` files added during the research for the bachelor's thesis are:

- `Geometry` - contains the definition for the template class `Point`, `Rectangle` and `Polygon`
- `ISampling` - defines the abstract class `ISampling` with an abstract method of generating points inside a 2D polygon. This class is extended by `RandomSampling`, `PoissonSampling` and `VoronoiSampling` which each defines its own logic for the point generation
- `Munkres` - this header file contains the classes `Matrix` and `Munkres` which solves an instance of the `Matrix` class
- `PointMapper` - a static class that defines helper methods for converting points between different representations/formats
- `vtkMAFMuscleDecompositionHoang` - the core of this bachelor's thesis. Since the Hoang method heavily relies on the muscle slicing of the Kukačka method, this class has been implemented as the child class of `vtkMAFMuscleDecompositionKukacka` to inherit the muscle slicing but diverges from it afterward

The dependencies and relationships between the classes are best seen in Figure 8.1.

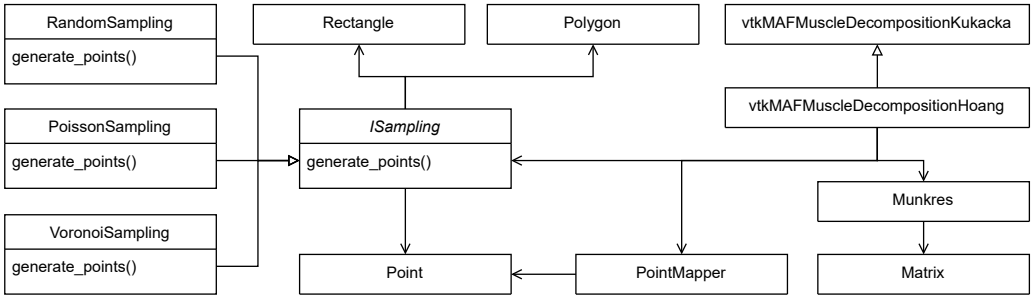


Figure 8.1: *UML diagram depicting the relationships between all classes*

All code is written in modern C++, oftentimes even using the latest features of C++20 like the <ranges> library. A great deal of effort has been put into the utilization of the latest capabilities of the programming language and whenever possible, old code was rewritten using more recent syntax to reduce the number of lines and increase readability[1] and effectiveness. An example of such a refactored code is shown in Listing 8.1.

Source code 8.1: *Example of code refactoring by using modern C++*

```
/* creating an array of booleans */
// old code
RowMask = new bool[Matrix->GetNumberOfRows()];
for ( int i = 0 ; i < Matrix->GetNumberOfRows() ; i++ ) {
    RowMask[i] = false;
}

// refactored code
mRowMask = std::vector<bool>(mMatrix.get_row_num(), false);
```

8.2 Code in more detail

This section will go through the most important classes and highlight the code snippets and methods deemed relevant.

8.2.1 Point

The `Point` class is probably the most important of all because it is used in every other class. It is defined as a template class with the template parameter `N` being a non-negative integer which denotes the dimensionality of the `Point` and internally it is represented as an array of `N` floating point values (see Listing 8.2).

[1] Readability is not always increased especially if the reader isn't well-versed in the latest header files and syntax

Source code 8.2: *Snippet of the* `Point` *class*

```
template<size_t N>
class Point {
    std::array<double, N> mCoordinates;

    template<typename ... Args>
    Point(const Args& ... args) noexcept :
        mCoordinates({ args... }) {};

    double distance(const Point& other) const {
        std::array<double, N> deltas{};
        for (size_t i = 0; i < N; ++i)
            deltas[i] = other[i] - (*this)[i];

        auto square = [](double num) { return num * num; };
        auto squaredDeltas = deltas
            | std::views::transform(square);

        return sqrt(std::accumulate(
            squaredDeltas.begin(), squaredDeltas.end(), 0.0)
        );
    }
}
```

This class defines some basic methods like translating the point, accessing the coordinates as well as calculating the distance to another point and is used by the `Rectangle` and `Polygon` classes. The latter were heavily inspired by the `java.awt.geom` Java package.

8.2.2 ISampling

This abstract class is the basis for all other types of Samplings which have the sole task of generating points inside a polygon. The `ISampling` is defined as seen in Listing 8.3.

Source code 8.3: *Snippet of the* `ISampling` *class*

```
// type aliasing the template Point class
using Point2D = Point<2>;
using Point2DList = std::vector<Point2D>;

class ISampling {
    virtual Point2DList generate_points(
        const Polygon& polygon, size_t k) = 0;
};
```

The specific implementations of `RandomSampling`, `PoissonSampling` and `VoronoiSampling` are almost a direct translation of their respective descriptions in Chapter 4.

8.2.3 Munkres

This `Munkres` class needs a matrix structure to work properly therefore one has been implemented within the same header file. The `Matrix` class is internally represented as a vector of vectors and to provide access to individual elements a hack as seen in 8.4 has been done[2].

Source code 8.4: *Snippet of the `Matrix` class*

```
// type aliases
using _Row = std::vector<double>;
using _Matrix = std::vector<_Row>;

class Matrix {
    // internal representation of the Matrix
    _Matrix mMatrix;

    // proxy class for [][] indexing
    class Row {
        friend class Matrix;
        _Matrix& mParent;
        size_t mRow;

        Row(_Matrix& parent, size_t row) :
            mParent(parent), mRow(row) {}

        double& operator[](size_t col) {
            return mParent[mRow][col];
        }
    };

    // used for [][] access of Matrix
    Row operator[](size_t row) {
        return Row(mMatrix, row);
    }
};
```

The `Munkres` class has only one public method, `solve()`, which takes in a `Matrix` object as a parameter and returns a new one with the calculated matches. The code for this class was provided to me by my thesis supervisor, although minor modifications have been made.

[2] For details about proxy class, see: `https://tinyurl.com/mtd869yf`

8.2.4 PointMapper

This utility class consists solely of `public static` methods that convert a point among different representations, which are `Point`, `VCoord`, `vtkPoints` (see Listing 8.5).

Source code 8.5: *Point representations that are converted between each other*

```
typedef double VCoord[3];
using Point2D = Point<2>;
using Point3D = Point<3>;
class vtkPoints {};
```

8.2.5 vtkMAFMuscleDecompositionHoang

This class combines all previously mentioned modules and is the main focus of this bachelor's thesis, where most of the work is done. The class heavily exploits the `<execution>` library to increase the effectiveness of numerous procedures by using methods for concurrent operations, e.g. the points are generated on all muscle contours at once and not one by one. Please notice that the piece of code in Listing 8.6 is an implementation of the theory from Figure 7.1.

Source code 8.6: *Snippet of the `vtkMAFMuscleDecompositionHoang` class*

```
// populating the cost matrix non-sequentially
std::for_each(std::execution::par_unseq, pB_idxs.begin(),
                pB_idxs.end(), [&](const auto& pB_idx) {
    // already connected points -> get Line
    Point3D pB = slices[currSlice][pB_idx];
    Point3D pA = get_previous_point(pB_idx,
                    prevMatrix, slices[currSlice - 1]);
    Line3D line = std::make_pair(pA, pB);

    // use line to calculate the cost using cosine distance
    std::for_each(std::execution::par_unseq, pC_idxs.begin(),
                    pC_idxs.end(), [&](const auto& pC_idx) {
        Point3D pC = slices[currSlice + 1][pC_idx];
        double cos = (pC.cos_distance(line) + 1) * 0.5;
        costMatrix[pC_idx][pB_idx] = alpha * (1 - cos) +
            beta * (pC.distance(pB) * invMaxEuclidDistance);
        });
    });
```

The code above is equivalent to classic nested `for` loops that traverse a matrix. The modifications using `std::for_each` and `std::execution::par_unseq` allow for parallel and non-sequential traversal while setting the value of each individual cell.

Testing and comparison with Kukačka

9

This Chapter is dedicated to the analysis and testing of the new method (which will be called the Hoang method from now on) of generating muscle fibers inside a muscle and how it behaves with different parameter configurations. The advantages and disadvantages, as well as a detailed comparison against the Kukačka method, will be thoroughly discussed to determine the effectiveness of this new method and whether it could replace the established Kukačka method.

9.1 Testing machine

The machine used for development and testing was the laptop HP ProBook 455 G6 6MR45ES[1] with some slight modifications. The laptop has expanded storage from 512GB to 1TB[2] and increased memory from 16GB to 32GB[3]. The operating system used was Windows 10 Pro and the program was developed in Visual Studio 2022 using C++20. A small program to measure the fiber lengths was made while using JetBrains IntelliJ IDEA 2023.1 and the programming language Kotlin 1.8.

9.2 Testing methods

The following measurements were done with the fiber points generated by the *Centroidal Voronoi Diagram* see Section 4.3. To be specific, Lloyd's algorithm for k-means clustering was used for each muscle slice. The muscle slices were projected onto a *plane of best fit* and transformed to 2D. Later the Munkres algorithm was applied to every pair of slices to get the assignments in the form of a matrix of matches. Once all the matches were made, the fibers were created. The weight of

[1] Details at: `https://www.hpmarket.cz/productOpt.asp?konfId=6MR45ES`
[2] SSD details: `https://tinyurl.com/3a7pfkut`
[3] RAM details at: `https://tinyurl.com/yvc3zk46`

assigning points to each other was a linear combination of the cosine and Euclid distance with parameters $\alpha = 0.2$ and $\beta = 0.8$ respectively, please see Equation 7.7. The parameters were chosen based on the author's discretion with no experimental data backing up this specific configuration. The author determined these values were an adequate starting point and changed them appropriately to obtain the best results.

In the following sections we will compare the fibers generated by both the Kukačka method and Hoang method proposed in this bachelor's thesis. For that, we use two methods of comparison:

1. Length comparison: the fibers will be compared by length, to be precise, the distribution of the fiber lengths will be evaluated
2. Aesthetic comparison: the fibers will be compared from an aesthetic point of view and since everybody has a different preference regarding aesthetics, a sample of around 40 people from the author's social circle has been asked
3. Time complexity comparison: the Hoang and Kukačka method will be compared by their runtimes

Additionally, there will be two distinct configurations for the fiber length comparison, namely:

1. Coarse fibers: the fibers are compared exactly as they were created by each method
2. Smooth fibers: the fibers are compared after a post-process smoothing

Therefore, a total of four comparisons will be conducted. Further information regarding each comparison method and its respective configuration will be explained in the following sections of this chapter.

9.3 Input and output data

During the slicing, it is possible to extract the points of the cut, which coincidentally define a non-planar polygon[4]. The output of the Hoang method is a *txt* file containing all the points and lines that are then processed externally in a Kotlin program. That program calculates the length of each fiber given the dataset of points and prints it out in an easy-to-manipulate format which is then used as an input to an online chart-making tool.

[4] From now on, points that define a polygon will be referred to as a "polygon"

9.4 Influence of user parameters

This section goes briefly over the influence of the user-defined parameters on the final result and the runtime of the Hoang method.

As previously mentioned in Section 7.2 the runtime complexity of the Hoang method is

$$O(n) = n^3 * (k - 1), \text{ where } n = \text{number of fibers}, k = \text{number of slices}$$

Both the n and k parameters are user-defined albeit k is the number of slices whereas the user is defining parameter l instead which is the number of line segments that each fiber consists of, therefore $l = k - 1$. And the time complexity, dependent on the user parameters, is

$$O(n) = n^3 * l$$

The user is also able to define α and β parameters (see Equation 7.7) which has an impact on the way the fiber points are connected to each other.

Another important factor is the influence of fiber smoothing on the created fibers, which in some cases drastically changes the distribution of fiber lengths, and in all cases, it decreases the average fiber length. The smoothing algorithm used was the same one that the Kukačka method uses, which is a simple method of weighted averages where a point is given a weight and is averaged with the points in its nearest neighborhood.

9.5 Comparing lengths of fibers

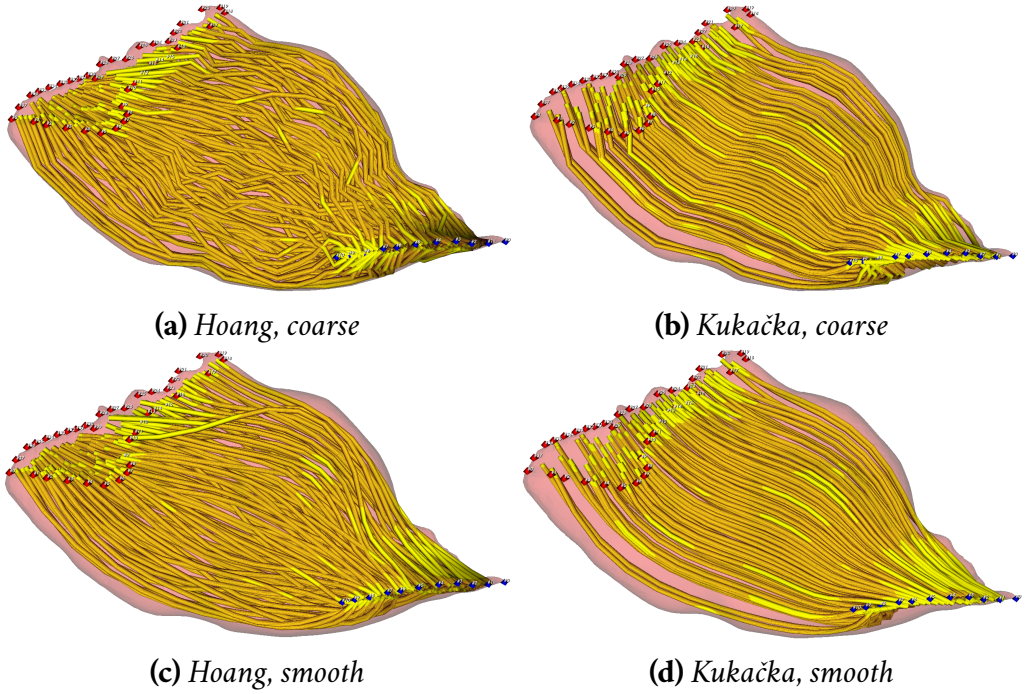
According to [MK20], to get a statistically satisfying sample size, it is essential to create fibers with at least 15 line segments. That publication also used muscles with 100 fibers throughout the study, therefore the author of this thesis, deemed that amount of fibers to be sufficient for a fiber length distribution analysis. The testing was done mainly on the muscles around the hip area, namely the gluteus maximus, gluteus medius, iliacus, and adductor brevis.

The representation of the muscle fiber lengths is done using a violin plot[5] for simple visual analysis and comparison of both Hoang and Kukačka methods with or without fiber smoothing.

[5] Violin plots were created at: `https://tinyurl.com/4uteypbh`

9.5.1 Gluteus maximus

The fibers generated for gluteus maximus are seen in Figure 9.1. From it, we can observe that the Hoang method works quite well but in some cases, the fibers created did not connect sensibly and the sum length of fibers is not minimized. In this particular instance setting the $\beta = 1$ would have probably yielded better results.



(a) *Hoang, coarse* **(b)** *Kukačka, coarse*

(c) *Hoang, smooth* **(d)** *Kukačka, smooth*

Figure 9.1: *Gluteus maximus with* 100 *generated fibers*

From Figure 9.2, it is evident that the distribution of fiber lengths is comparable in both smooth settings but the median fiber length of the Hoang method is slightly higher. With smoothing applied, both methods produced fibers with nearly Gaussian length distribution, with the Hoang method having a higher standard deviation.

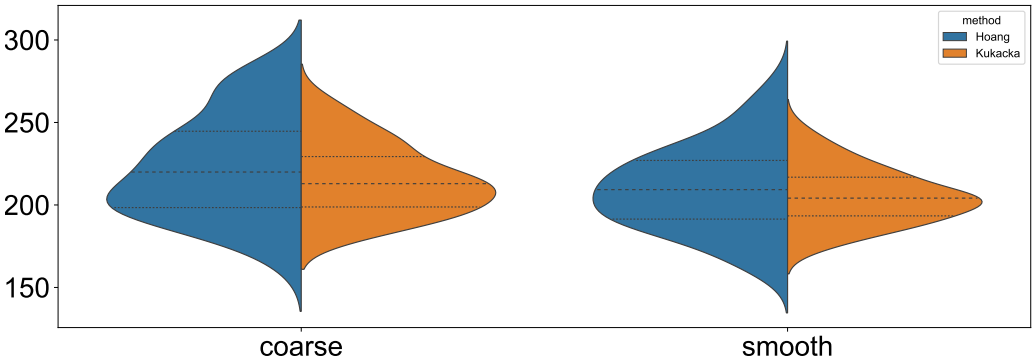
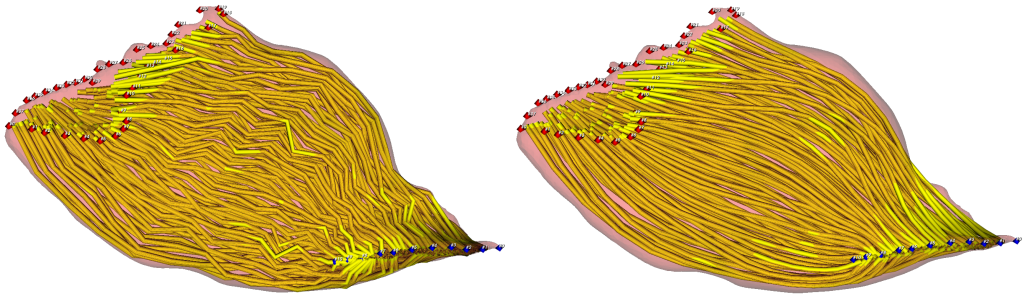


Figure 9.2: *Violin plot for gluteus maximus. Hoang method (blue), Kukačka method (orange), the Y-axis denotes the fiber lengths in absolute units*

The experiment was repeated for $\beta = 1$ with a much better result as seen in Figure 9.3. The fibers don't intersect as much as before and no fiber deviates.



(a) *Coarse* **(b)** *Smooth*

Figure 9.3: *Fibers generated via the Hoang method and with* $\beta = 1$

Comparing the fibers of the Hoang method in Figure 9.4 it can be concluded that surprisingly the median fiber length has increased but after smoothing it is equal and the average fiber length is also lower. From Figure 9.5, we can see that the fiber length distribution of the Hoang method is very similar to the Kukačka method, albeit with a higher median and greater standard deviation in both cases.

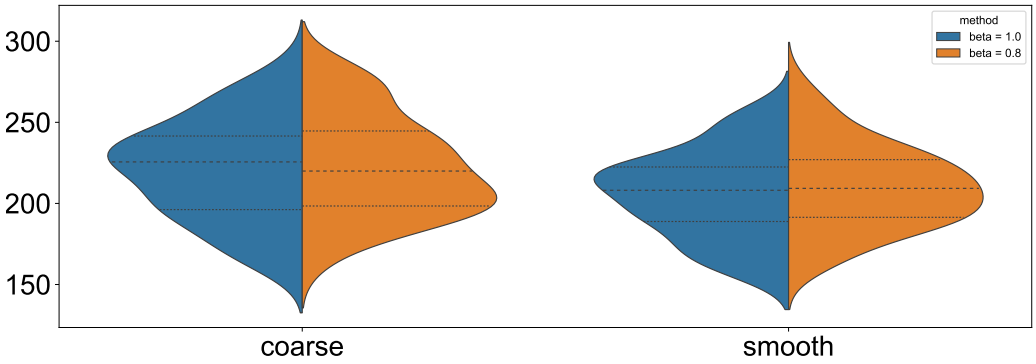


Figure 9.4: *Hoang method with* $\beta = 1$ *(blue) and* $\beta = 0.8$ *(orange)*

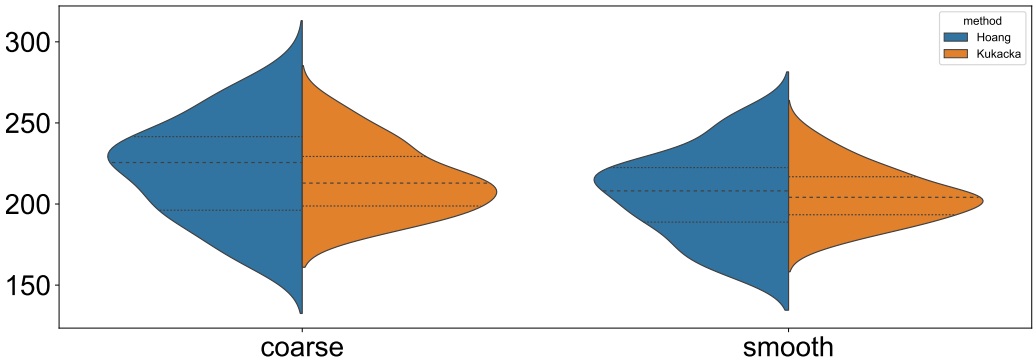
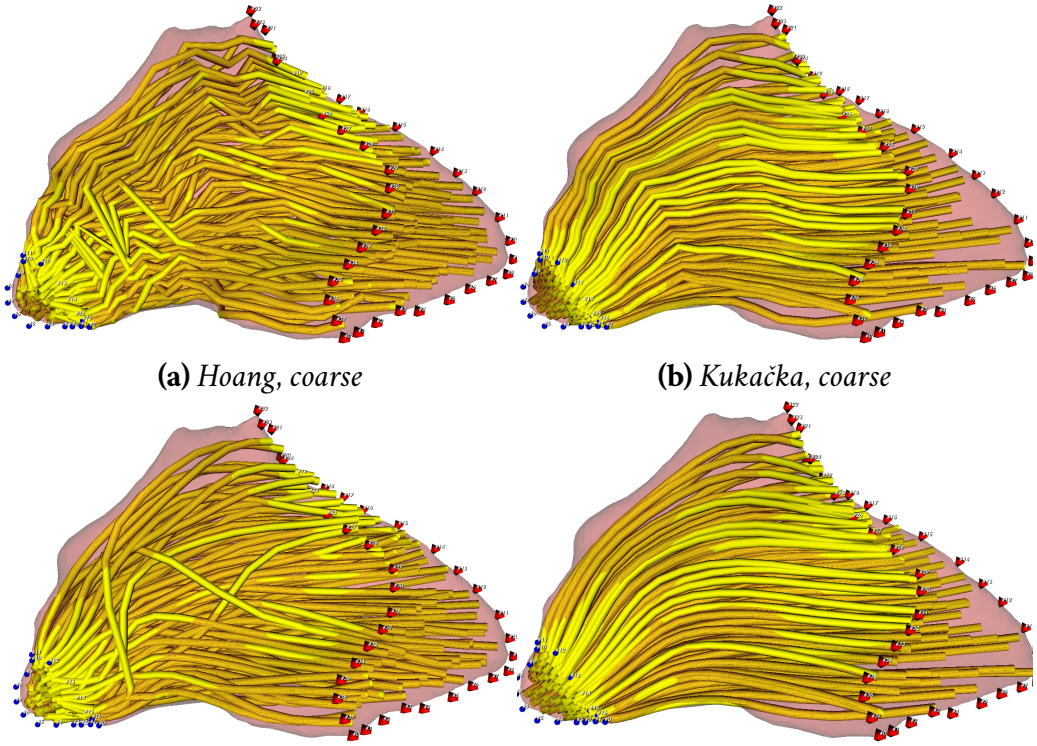


Figure 9.5: *Hoang method with* $\beta = 1$ *(blue) and Kukačka (orange)*

9.5.2 Gluteus medius

In Figure 9.6 we can see that multiple fibers created by the Hoang method are crossing across the whole muscle, with some even protruding. Probably a different β or a more complex cost-assigning function could have improved this result.



(a) *Hoang, coarse* **(b)** *Kukačka, coarse*

(c) *Hoang, smooth* **(d)** *Kukačka, smooth*

Figure 9.6: *Gluteus medius with* 100 *generated fibers*

By referring to the violin plot presented in Figure 9.7, it can be observed that both methods produced fibers a similar distribution and again the Hoang method has a higher median fiber length.

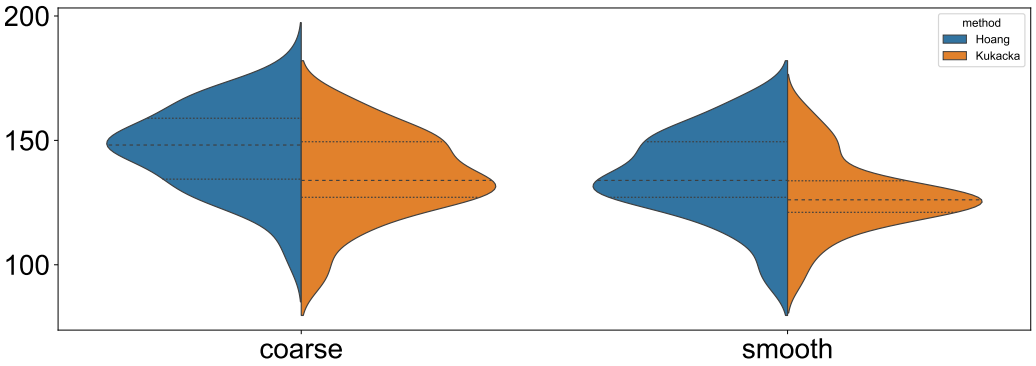
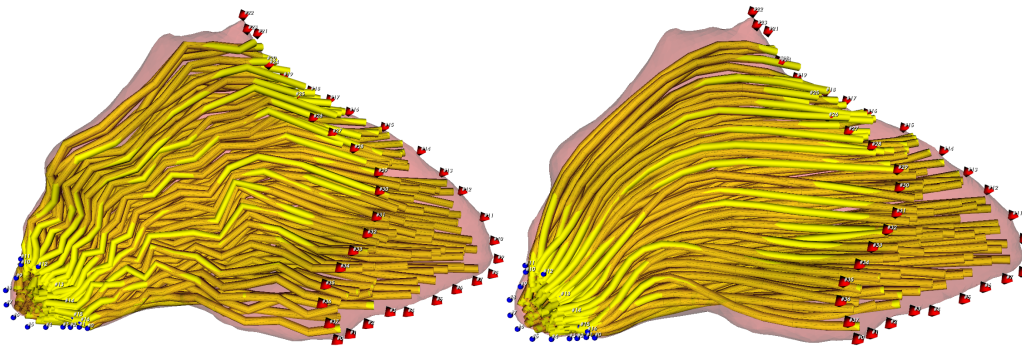


Figure 9.7: *Violin plot for gluteus medius*

Setting the parameter $\beta = 1$ resulted in visually much better fibers, see Figure 9.8. This time no fiber is outside the muscle and they area also not crossing and the quality of the output is comparable to the Kukačka method.



(a) *Coarse* **(b)** *Smooth*

Figure 9.8: *Fibers generated via the Hoang method and with* $\beta = 1$

Although the median fiber length and average fiber length are higher with $\beta = 1$, please see Figure 9.9, the length distribution of the fibers is almost identical and most probably, if given a higher sample size of the fibers, the discrepancies would diminish.

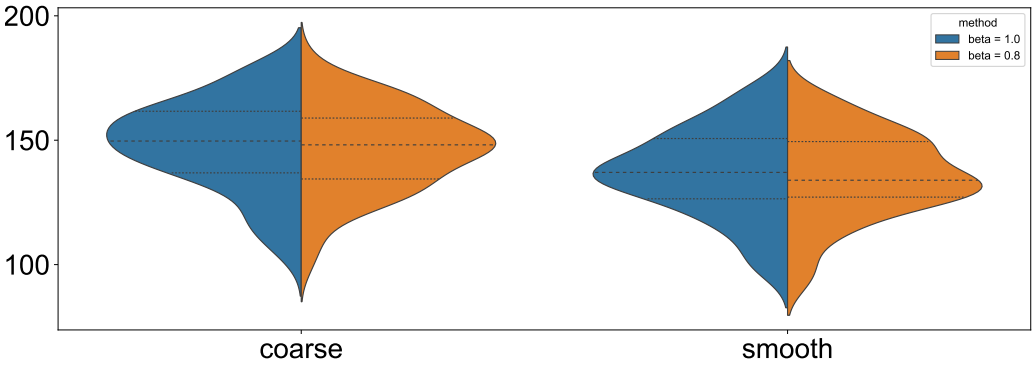
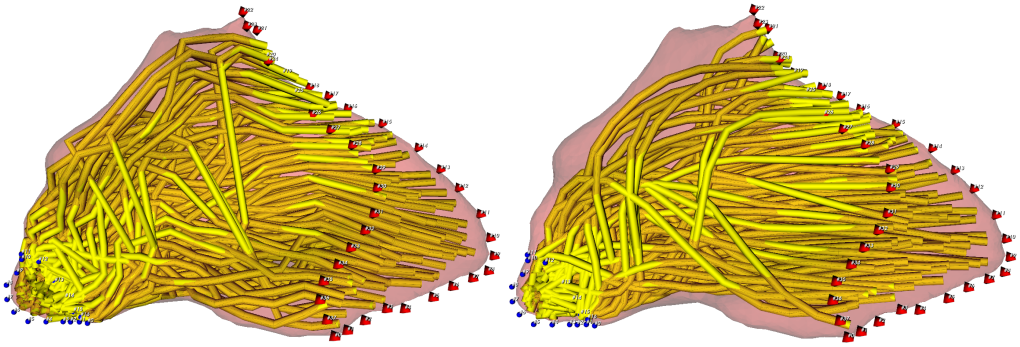


Figure 9.9: *Hoang method with* $\beta = 1$ *(blue) and* $\beta = 0.8$ *(orange)*

Out of curiosity, a configuration with $\alpha = 0.5$ and $\beta = 0.5$ has been made, the result of which can be seen in Figure 9.10. From that it can extrapolated that increasing the parameter α would yield progressively inferior fiber quality and more fibers would be outside the bounds of the muscle. We can also see that the fibers are not distributed evenly within the muscle itself.



(a) *Coarse* **(b)** *Smooth*

Figure 9.10: *Fibers generated via the Hoang method and with* $\alpha = 0.5$ *and* $\beta = 0.5$

Given the plot in Figure 9.11 it can stated that the average fiber length has increased for $\beta = 0.5$, and some very high outlier values were produced. With smoothing applied, the results did not show any significant improvement.

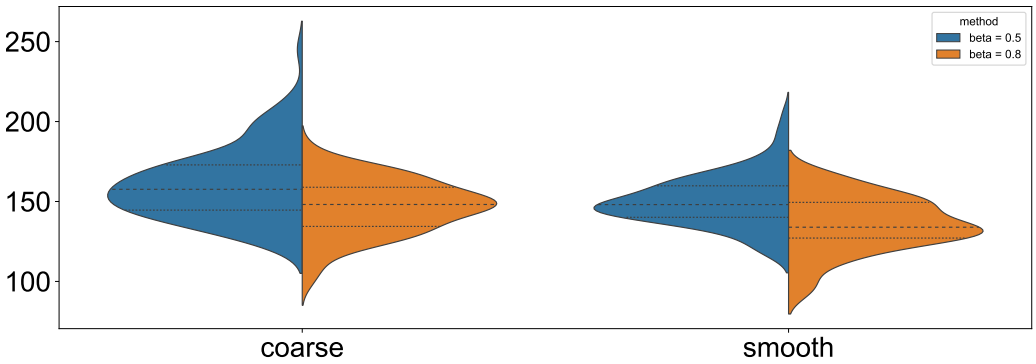
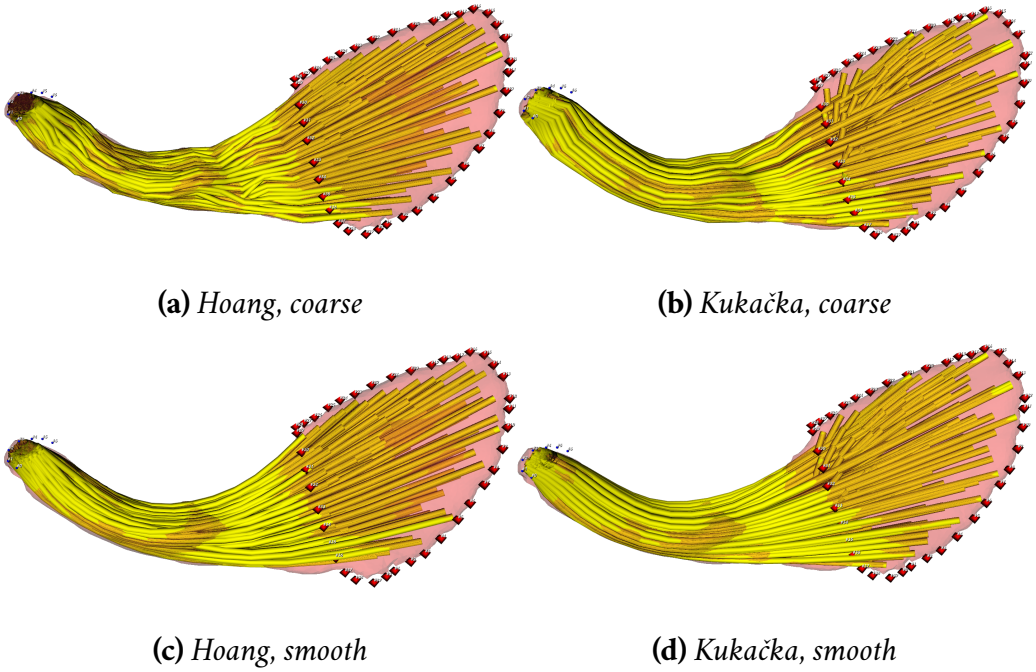


Figure 9.11: *Hoang method with* $\beta = 0.5$ *(blue) and* $\beta = 0.8$ *(orange)*

9.5.3 Iliacus

Iliacus is the first muscle where we can see the Hoang method improving upon the Kukačka method. As seen in Figure 9.12, it better fills out the muscle while also not having fibers that intersect each other.



(a) *Hoang, coarse* **(b)** *Kukačka, coarse*

(c) *Hoang, smooth* **(d)** *Kukačka, smooth*

Figure 9.12: *Iliacus with* 100 *generated fibers*

From the plot in Figure 9.13, it can be concluded that the smoothing process did not change the fiber length distribution excessively. Notice that while the median fiber length of the Hoang method remained slightly higher, the first quartile is noticeably lower than in the Kukačka method. The length distribution interval is also much wider in the former.

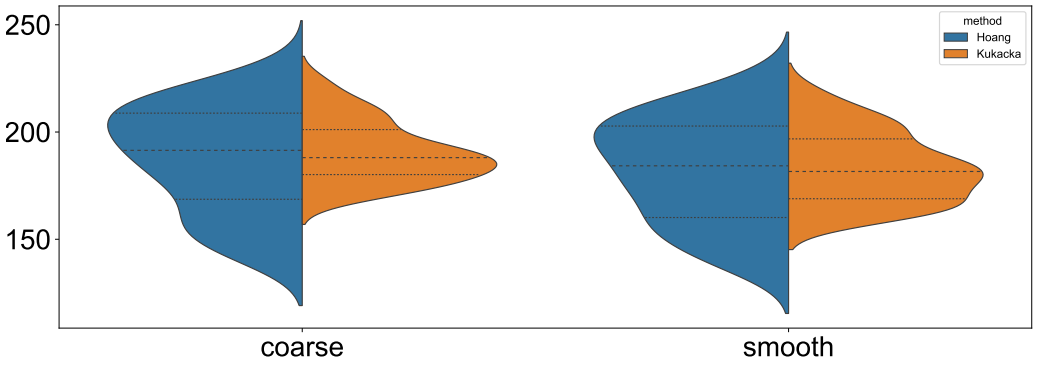
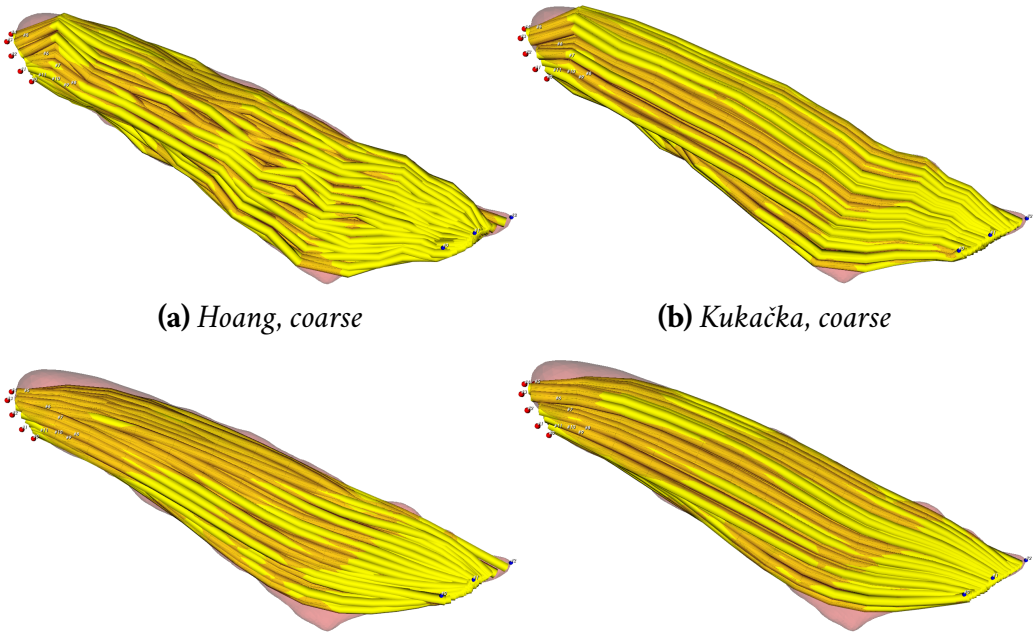


Figure 9.13: *Violin plot for iliacus*

9.5.4 Adductor brevis

Observing the multiple adductores breves[6] from Figure 9.14, it can be concluded that both methods output similar fibers, and after the smooth post-processing, they are nearly indistinguishable from each other.



(a) *Hoang, coarse* **(b)** *Kukačka, coarse*

(c) *Hoang, smooth* **(d)** *Kukačka, smooth*

Figure 9.14: *Adductor brevis with* 100 *generated fibers*

Figure 9.15 shows a big difference in the median fiber length of both methods (Hoang being higher) and how the smoothing has a greater impact on the Hoang method (Kukačka was changed slightly). The difference in the medians has also decreased due to the reduced standard deviation.

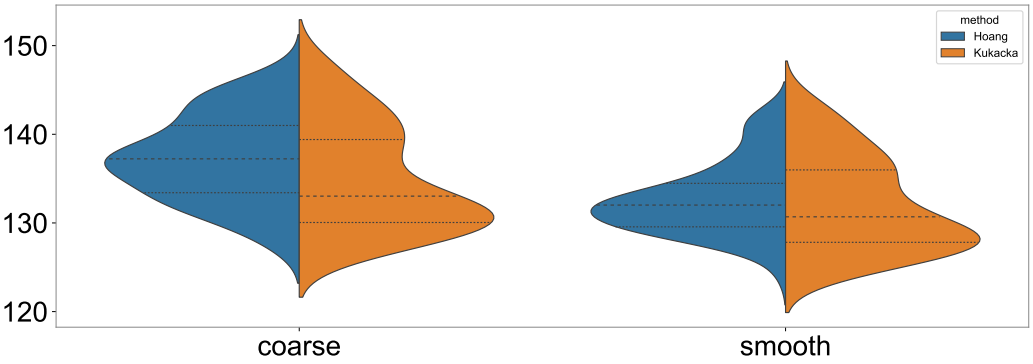


Figure 9.15: *Violin plot for adductor brevis*

[6] Plural form of 'adductor brevis' in Latin

9.6 Comparing the aesthetic of the fibers

As part of the testing and comparison, a small survey[7] was conducted with a total of 44 participants. Each participant was presented with same set of twelve questions, where each question consisted of two images, depicting the same muscle, one with fibers generated by the Hoang method and the other by the Kukačka method.

Figure 9.16 shows a screenshot of the poll. The aim of the survey was to determine which of the two methods created more visually appealing muscle fibers. For this, the unbiased opinions of the participants was needed.

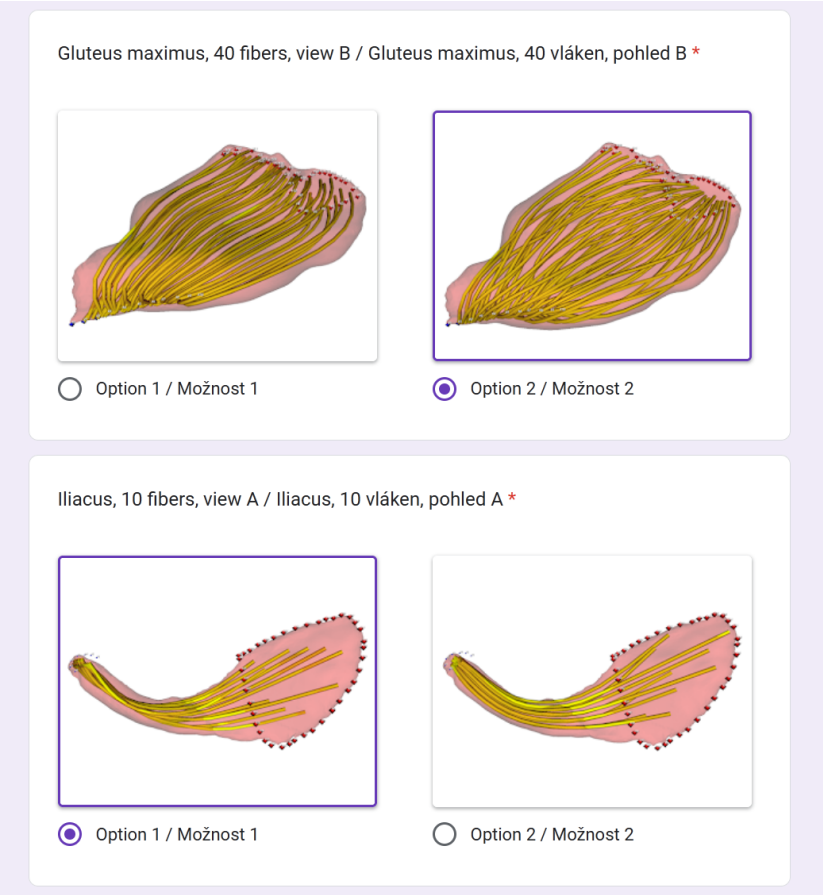


Figure 9.16: *A screenshot from the poll*

For the survey, gluteus maximus and iliacus were selected and a total of 24 images were created. Each shows one of the two muscles in two different views. There were three configurations for the number of fibers (10, 20, and 40) and each fiber had 15 line segments. Lastly, the Hoang method was used with parameter $\beta = 1$, and the fibers of both methods were post-processed by smoothing.

[7] The form is available at: `https://forms.gle/v4UgDdYtKywaCG3p8`

At the end of the survey, the participants were asked, by which metric they chose the more appealing fibers. The three options were (a) 'better-looking fibers', (b) 'better filled out muscle', and (c) 'both'. The number of participants in each category can be seen in the pie in Figure 9.17.

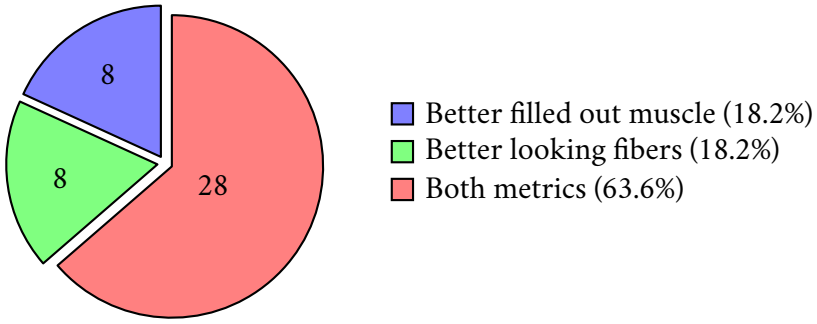
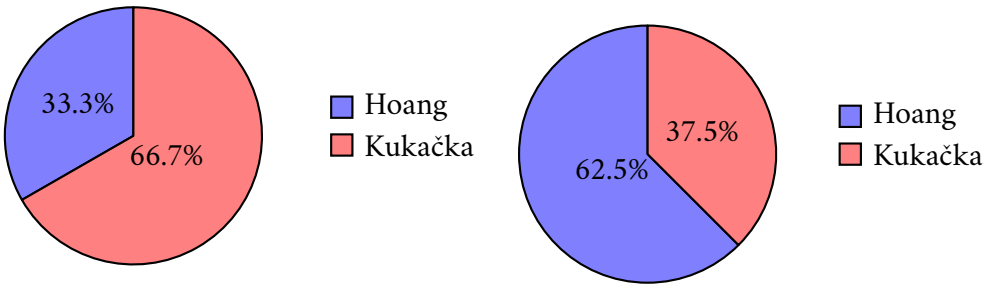


Figure 9.17: *Pie chart depicting the distribution of the participants' chosen method of comparison*

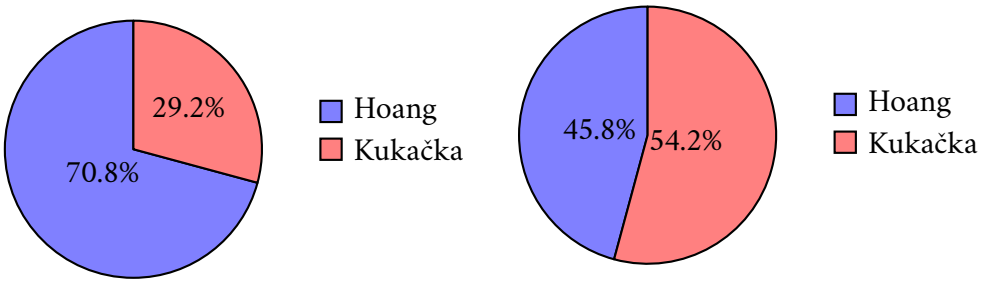
Figure 9.18 shows the preferred method for fiber generation of the 8 participants that chose the metric 'better looking'. We can see that the participants preferred Kukačka for gluteus maximus, while for iliacus, the Hoang method was the more favorable option. This came as a surprise to the author, as he himself would have chosen the Kukačka method for both cases.



(a) *Gluteus maximus* **(b)** *Iliacus*

Figure 9.18: *Pie charts showing the preferred method of fiber generation for participants that chose the 'better looking' metric*

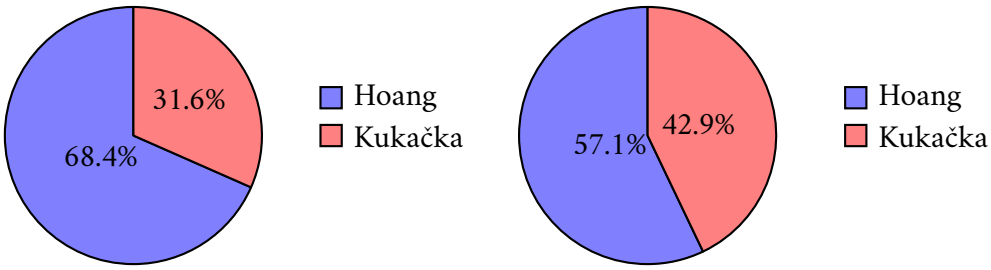
We can conclude from Figure 9.19, that the 8 participants, who compared the fibers by how well they filled out the muscle, preferred the Hoang method for gluteus maximus and favored Kukačka for iliacus.



(a) *Gluteus maximus* **(b)** *Iliacus*

Figure 9.19: *Pie charts showing the preferred method of fiber generation for participants that chose the 'better filled out' metric*

It is evident from Figure 9.20, that the 28 participants, who compared the muscle fibers by both metrics, preferred the Hoang method.



(a) *Gluteus maximus* **(b)** *Iliacus*

Figure 9.20: *Pie charts showing the preferred method of fiber generation for participants that chose the 'both' metric*

From the responses of all 44 participants, it can be concluded that the Hoang method was the preferred method for fiber generation, as can be seen in Figure 9.21.

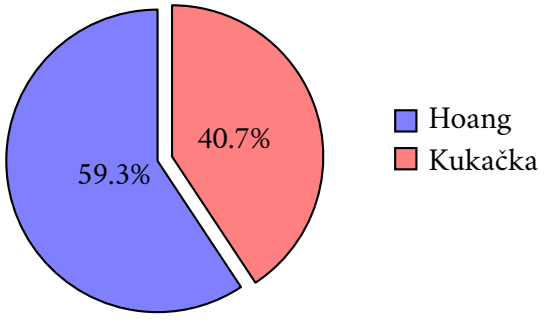


Figure 9.21: *Pie chart showing the preferred method of fiber generation of all participants*

Please keep in mind that the muscle fibers were presented in image format and using a video format could yield different results. Additionally, different muscles and configurations of both methods may have affected the survey outcome.

9.7 Comparing time complexities

This section provides a brief overview of the runtime complexities of both methods, with the Hoang method ($\beta = 1$) being measured with and without optimizations of the `<execution>` library. The measurements were limited to the decomposition itself and common factors, e.g. fiber smoothing and pre-processing were not included. The experiment was tested varying numbers of fibers (5, 25, 50, 100, and 200) using different numbers of line segments (5, 10, and 15). Time is measured in milliseconds [*ms*] and each data point shown in the following graphs represents the average of five independent measurements.

Figure 9.22 illustrates that Kukačka is the fastest of all the decomposition methods, with execution time remaining constant as the number of fibers increases. Although the runtime of the unoptimized Hoang method experiences rapid growth, optimizing the decomposition with concurrent techniques can aid in mitigating this growth, resulting in up to 4 times increase in speed.

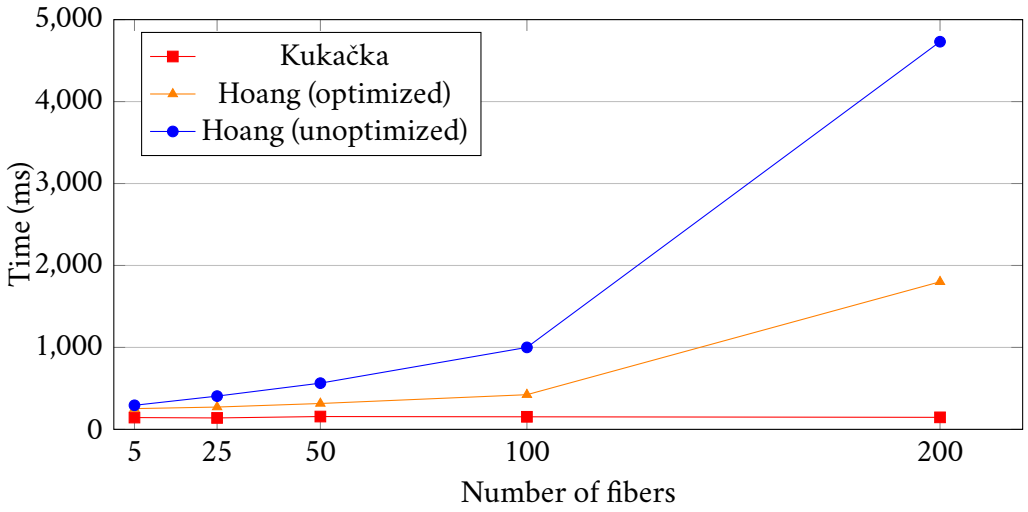


Figure 9.22: *Number of line segments = 5*

The same trend continues with increasing numbers of line segments as can be seen in Figures 9.23 and 9.24. It is worth noting, that Kukačka still executes in constant time while the runtimes of both configurations of the Hoang method show an increase as the number of line segments increases.

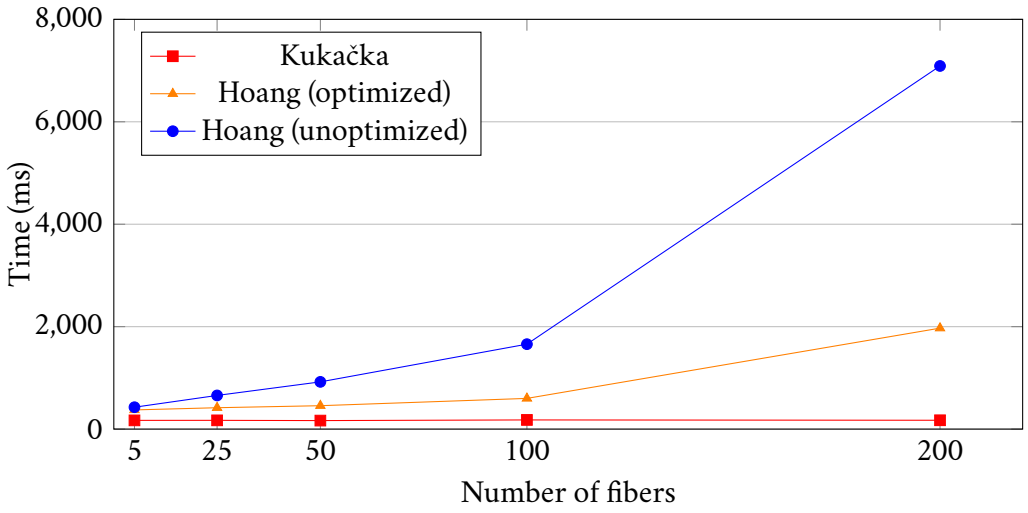


Figure 9.23: *Number of line segments* = 10

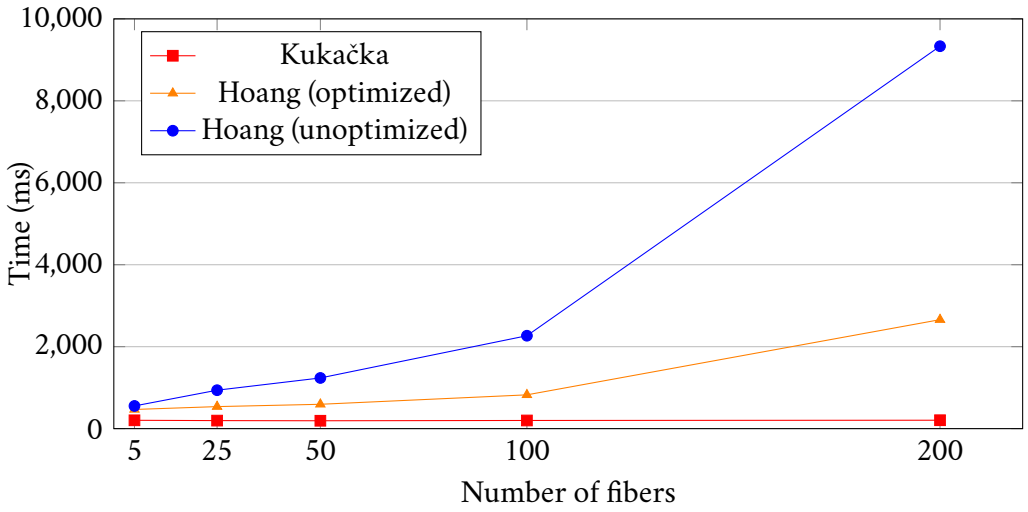


Figure 9.24: *Number of line segments* = 15

9.8 Summary of testing results

In summary, the testing results presented in this chapter show that the new Hoang method can create fibers of comparable quality (i.e. the fibers do not intersect and fill out the muscle evenly) to the ones produced by Kukačka as evident from Figure 9.3, and in some cases, it even produces better results (see Figure 9.12). For gluteus maximus and gluteus medius, the distribution of fiber lengths was almost identical for both methods, while for iliacus, the length distributions were very different.

From the aesthetic point of view, the fibers created by the Hoang method were considered to be more visually appealing as the conducted survey has shown (Figure 9.21). However, it is important to note that the Hoang method has one significant drawback. That is, the runtime of the fiber generation has a much worse computational complexity as the time increases rapidly with the number of fibers and the number of line segments. While Kukačka remains constant, the Hoang method is 4 times slower for 100 fibers and around 12 times slower for 200 fibers.

Conclusion 10

The main goal of this thesis was to propose a new method of muscle decomposition and compare the novel method with Kukačka in multiple aspects. The goals were fully reached and it can be concluded that the fibers generated by the Hoang method are comparable to those created by Kukačka and sometimes even surpass it.

Part of the thesis was also the problem of even distribution of points inside a polygon which was solved by using the centroids of a *centroidal Voronoi diagram* (CVD), implemented using Lloyd's algorithm for k-means clustering. In future works, it would be interesting to explore a different implementations of CVD, e.g. McQueen's algorithm or a completely different approach to point distribution.

The transformation of non-planar polygons into planar-polygons was resolved using a projection onto a *best fit plane* with a subsequent transformation of the reference frame. This way every point of the polygon was described by two non-zero coordinates. The implementation for transforming the coordinate system most probably contains an error, as can be seen in Figure 6.4 where the planar polygon is transformed into a polygon with a line-like shape. This rare error occurs only on some muscle slices and disappears when more fibers are created. Due to time constraints, the author could not investigate the cause of the error and find a solution. However, this issue could have been completely circumvented if the planarity of the polygon had been solved by using the *mesh unfolding* method, see Figure 5.3.

To connect the points between slices, the Munkres algorithm has been employed, which yielded satisfying and accurate results. In the future, a different implementation of the Munkres algorithm or a different cost function could be used for better time complexity and results.

Basic user manual

A.1 Quick start

For the purposes of a quick demonstration, a test run has been configured (for Windows 10 only), which can be found inside the submitted zip file. The full path to the file is as follows `./Application_and_libraries/bin/run.cmd`.

To execute this file, open a command line and run `.\run.cmd`. This is a pre-configured demo that generates 100 fibers of 15 line segments and includes the visualizations for the process of point distribution inside slices and the process of creating fibers.

A.2 How to build the application

If the user wishes to build his or her own executables or is not able to execute the quick demo, due to a different operating system, he or she should follow these next few steps, that explain how to build the project. First, it is important to have *The Visualization Toolkit* (version 9.1.0) from `https://vtk.org/download/` installed as well as *CMake* (version 3.22.0) or *cmake-gui* of the same version from `https://cmake.org/download/`. It is recommended to have the exact versions of the software installed, as these were used during development and will ensure full compatibility.

Next, the user must carefully read the `README.md` file inside the `src` sub-folder. This file describes in detail the installation of the projects *vtkVisualDebugger* and *Muscle Wrapping 2.0* as well as the complete build configuration for *The Visualization Toolkit* and *CMake*.

Lastly, to successfully execute the project, it is also necessary to have at least a C++20 compiler installed because the project now uses some new libraries.

There is another way to quickly build the project which is only possible for users with permissions to the private `Muscle Wrapping 2.0` project found at `https://gitlab.com/besoft/muscle-wrapping-2.0`. They can do so easily by checking out the `HoangBP` branch which contains the latest additions. It is still necessary to have a C++20 compiler configured.

A.3 How to run the application

A compiled Windows 10 executable can be found at the following path: `./Application_and_libraries/bin/MuscleDecompositionTest.exe`. The usage and options of the executable program is described in Listing A.1.

Listing A.1: *Parameters of the executable*

```
Usage:
MuscleDecompositionTest <muscle> [-n <num-of-fibers>]
    [-r <resolution>] [-f <output>] [-x]
    [-v <vis-mode>] -o <origin>... -i <insertion>...
MuscleDecompositionTest (-h | --help)
MuscleDecompositionTest --version

Options:
-o <origin>          File w/ origin attachment area points
-i <insertion>       File w/ insertion attachment area points
-n <num-of-fibers>   Number of the fibers [default:10]
-r <resolution>      Resolution of the fibers [default:15]
-f <output>          Output file with the produced fibers
-v <vis-mode>        0(none), 1(output),2 (debug) [default:1]
-h --help            Show this help
--version            Show version
```

Once the file is running, a window with visualization appears. Using the `Esc` key, it is possible to see the execution of the program step by step and the `X` key skips all the visualization windows. For more controls, the user can click the `F1` key.

Structure of the submitted file

B

- `A19B0054_prilohy.zip` ..... Compressed submitted file
- `Text_thesis`
  - `Latex` ..... LaTeX project
    - `img` ..... Images used in the thesis
    - `texmf` ..... GTAmerica font
    - `thesis.tex` ..... LaTeX thesis source code
  - `BP_Hoang.pdf` ..... E-version of the bachelor's thesis
- `Application_and_libraries`
  - `bin` ..... Executables and binaries
    - `run.cmd` ..... Demo run
    - `MuscleDecompositionText.exe` .... Compiled project executable
  - `src` ..... Project source code
    - `README.md` ..... Complete build instructions
- `Input_data` ..... Includes input files necessary for execution
- `Results`
  - `poll_results.xlsx` ..... Processed survey results
  - `times.xlsx` ..... Processed time measurments
  - `violin.xlsx` ..... Processed fiber lengths
- `Readme.txt` ..... Structure of the submitted file

Bibliography

[Ang+19] ANGLES, Baptiste et al. VIPER. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*. 2019, vol. 2, no. 2, pp. 1–26. Available from DOI: 10.1145/3340260.

[Bri07] BRIDSON, Robert. Fast Poisson disk sampling in arbitrary dimensions. In: *ACM SIGGRAPH 2007 sketches on - SIGGRAPH '07*. ACM Press, 2007. Available from DOI: 10.1145/1278780.1278807.

[CB13] CHOI, Hon Fai; BLEMKER, Silvia S. Skeletal Muscle Fascicle Arrangements Can Be Reconstructed Using a Laplacian Vector Field Simulation. *PLoS ONE*. 2013, vol. 8, no. 10, e77576. Available from DOI: 10.1371/journal.pone.0077576.

[Coo86] COOK, Robert L. Stochastic sampling in computer graphics. *ACM Transactions on Graphics*. 1986, vol. 5, no. 1, pp. 51–72. Available from DOI: 10.1145/7529.8927.

[KK14] KOHOUT, J.; KUKAČKA, M. Real-Time Modelling of Fibrous Muscle. *Computer Graphics Forum*. 2014, vol. 33, no. 8, pp. 1–15. Available from DOI: 10.1111/cgf.12354.

[Kor+20] KORPITSCH, Thorsten; TAKAHASHI, Shigeo; GRÖLLER, Eduard; WU, Hsiang-Yun. Simulated Annealing to Unfold 3D Meshes and Assign Glue Tabs. *Journal of WSCG*. 2020, vol. 28, no. 1-2, pp. 47–56. Available from DOI: 10.24132/jwscg.2020.28.6.

[Kuh55] KUHN, H. W. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*. 1955, vol. 2, no. 1-2, pp. 83–97. Available from DOI: 10.1002/nav.3800020109.

[MK20] MODENESE, Luca; KOHOUT, Josef. Automated Generation of Three-Dimensional Complex Muscle Geometries for Use in Personalised Musculoskeletal Models. *Annals of Biomedical Engineering*. 2020, vol. 48, no. 6, pp. 1793–1804. Available from DOI: 10.1007/s10439-020-02490-4.

[SE03] SCHNEIDER, PHILIP J.; EBERLY, DAVID H. MATRICES, VECTOR ALGEBRA, AND TRANSFORMATIONS. In: *Geometric Tools for Computer Graphics*. Elsevier, 2003, pp. 109–169. Available from DOI: 10.1016/b978-155860594-7/50007-2.

List of Figures

List of Tables

List of Listings