

**DESIGN OF IMPLANTS FOR SKULL RECONSTRUCTIVE
SURGERY**

by

DUMERY CORENTIN

**A THESIS SUBMITTED FOR THE DEGREE OF
MASTER OF COMPUTING**

in

COMPUTER SCIENCE

in the

GRADUATE DIVISION

of the

NATIONAL UNIVERSITY OF SINGAPORE

2020

Supervisor:


Professor Leow Wee Kheng

Declaration

I hereby declare that this thesis is my original work and it has
been written by me in its entirety. I have duly
acknowledged all the sources of information which have
been used in the thesis.

This thesis has also not been submitted for any
degree in any university previously.

Corentin DUMERY



DUMERY Corentin

25 March 2020

Acknowledgments

First, I would like to thank my supervisor, Associate Prof. Leow Wee Kheng, for guiding me throughout this thesis. It was due to his advice and insightful ideas that I was able to overcome complex problems encountered during this thesis.

I am also very grateful towards Jiong Le Lam, Lim Jing, and Osteopore as a whole for helping me understand the challenges of cranioplasty. Their assistance allowed me to better comprehend the practical constraints on implants and give me the motivation to pursue this thesis.

Contents

Acknowledgments	i
Abstract	v
List of Figures	vi
List of Tables	viii
1 Introduction	1
1.1 Motivation	1
1.2 Thesis Objectives	3
1.3 Thesis Synopsis	4
2 Literature Review	5
2.1 Mesh Generation	5
2.2 Surface Flattening	6
2.2.1 Flattening algorithms	6
2.2.2 Flattening error metrics	7
2.2.3 Cutting paths	7
3 Implant design	9

3.1	Implant Design Method	9
3.1.1	Input definition	9
3.1.2	Segmentation	11
3.1.3	Top layer reconstruction	12
3.1.4	Inner layer reconstruction	13
3.1.5	Sides of the implant	23
3.2	Flattening Method	23
3.2.1	Flattening Algorithm	24
3.2.2	Cutting Path	25
3.2.3	Edge smoothing	26
4	Experiments and Discussion	28
4.1	Overview	28
4.2	Data preparation	28
4.2.1	Input used	29
4.2.2	Selection of Area of Interest	30
4.3	Implant Design Experiment	33
4.3.1	Experiment	33
4.3.2	Implant Evaluation	36
4.4	Implant Flattening Experiment	38
4.4.1	Flattening Experiment	38
4.4.2	Flattening Evaluation	39
5	Conclusion and Future Work	41

5.1	Conclusion	41
5.2	Future Research Directions	42
	Bibliography	43

Abstract

Design of Implants for Skull Reconstructive Surgery

by

DUMERY Corentin

Master of Computing in Computer Science

National University of Singapore

The design of 3D cranioplasty implants is a challenging task that bears huge consequences. The geometry of the skull needs to be reconstructed in order to create a perfect-fit implant. Implant manufacturing imposes new constraints on 3D printing methods. Consequently, generated implants need to be flattened and smoothed.

A novel method is proposed to reconstruct the inner layer of a defect skull. The curvature and thickness of the skull are preserved. This information is used in combination with previous studies reconstructing the outer layer to generate an initial implant design. Through the use of flattening methods, the implant is then modified to match implant printing constraints.

This work aims to greatly simplify cranioplasty implants production. This will make implants more affordable and could allow for a more widespread and systematic use in skull injuries.

Key words : Skull reconstruction, Mesh Generation, 3D Geometry, Cranioplasty, Implant, Surface Flattening

List of Figures

1.1	3D printing of a flattened cranioplasty implant. (Source: osteopore.com)	2
3.1	An edge loop.	10
3.2	Reconstruction ambiguity.	10
3.3	Problem definition from user.	10
3.4	Loop filling method	15
3.5	Importance of identical subdivision on both sides of an edge	16
3.6	Line stitching method	17
3.7	Subdivision on small triangles, where at least one side has no new vertex.	18
3.8	Recursive subdivision scheme	19
3.9	Blue vertex can be projected on the right triangle.	20
3.10	Thickness estimation method	21
3.11	Flattening distortion	24
3.12	Cutting method	26
3.13	Cutting effect on flattening semi-sphere (resp. 1/4/7/10 cuts).	26
4.1	Initial skull	29
4.2	Patient 1	30
4.3	Artificial hole mesh	30

4.4	Patient 2	30
4.5	A possible edge loop definition for Patient 1	33
4.6	Simplified mesh from Patient 1 seen from below, the side, and up. . . .	34
4.7	Simplified mesh from Patient 2.	34
4.8	Identified regions separated	35
4.9	Implant for patient 1	35
4.10	Implant for patient 2	36
4.11	Comparison of generated inner layer and original layer, Patient 1 . . .	37
4.12	Comparison of generated inner layer and original layer, Patient 2 . . .	37
4.13	Inner reconstruction filling inner skull layer	38
4.14	Flattened inner layer of the implant without and with cut, Patient 1. .	40
4.15	Flattened inner layer of the implant without and with cut, Patient 2. .	40

List of Tables

4.1	Simplification comparison	32
4.2	Simplification results	33
4.3	Flattening evaluation	39

Chapter 1

Introduction

1.1 Motivation

Head injury is one of the leading causes of disabilities. Most often, it is the consequence of a motor vehicle accident, violence, or falling. It can lead to skull fracture and concussion. In the most extreme cases, the skull's structure may be damaged up to the point that it will lead to more complications if left unchecked. There may also be visible defects that will impact the patient's life negatively. A complex surgical operation is needed to repair the damaged skull.

An efficient way to restore the skull's mechanical strength and shape is the use of a cranial implant. If the skull has been greatly damaged, the implant can be used as a replacement of a whole section of the patient's skull. Through the use of implants, modern cranioplasty achieves great robustness and adaptability even in the most extreme cases. However, it is important to note that traumatic brain injuries are prevalent in low and middle income countries which constitute 90% of total cases [1]. There are two obstacles to a general and systematic use of cranial implants in these countries.

Traditional implant materials include titanium and a variety of thermoplastic polymers. These extremely expensive materials are not within affordable range for most of the patients' families. The implant's material needs to be resistant to infection, not conductive of heat or cold, and resist biological constraints. Thankfully,



Figure 1.1: 3D printing of a flattened cranioplasty implant. (Source: osteopore.com)

different materials have been successfully used, and they have helped reduce the cost of cranial implants. [1] shows a successful example of the use of plexiglass and explains how this material could be used at a larger scale. The other factor hindering the spread of cranioplasty is the lack of knowledge in implant design. 3D printing technologies are now increasingly available in the developing world, but as long as implant design methods are lacking the progression of cranioplasty will be limited. Currently, every implant has to be manually customized to fit the patient's skull and this process takes a lot of time and technical knowledge.

The design of cranial implants is a challenging task that bears huge consequences. Before anything, it must not push against the patient's brain more than the initial skull would. This can occur if the implant is too thick. It is therefore crucial to determine a precise estimation of the skull's thickness. Furthermore, if the implant does not fit the skull defect perfectly, it may lead to leaks. In the worst cases, the surgeon may not be able to insert the implant if its shape does not fit. Lastly, the skull's curvature needs to be approximated correctly in order to reconstruct the normal shape of the patient's head. Failing to do so may result in a noticeable defect which will impact the patient's social life after surgery.

While it may be easy for humans to intuitively tell whether a 3D implant fits into a defect skull and has appropriate curvature, it is a much more challenging task to describe a general procedure to generate 3D implants. The defect skulls vary widely and building an automatic implant generation program is a daunting task.

Furthermore, another constraint on the generation of a 3D model is that it must be fit to be 3D printed into an implant. In general-use 3D printing, overhanging features require the use of a support material during construction to prevent the structure from collapsing. However, in order to maximize the mechanical strength of the implant and with regard to specific printing constraints, implant printing cannot use any support material. As a result, we cannot simply print the generated 3D model directly.

In order to remove overhanging features during 3D printing, one solution is the use of flattening methods. The mesh's shape is modified to lie on a flat surface so that there is no need for support material, as shown in Figure 1.1. However, the flattened implant needs to be manually bent back to its original shape after being printed. Furthermore, most 3D surfaces cannot be flattened without distortion. [2] describes developable meshes, which can be flattened with perfect precision, but it also shows there are strong geometrical constraints on such meshes. In the case of a 3D implant, it is impossible to alter the shape of the implant to make it more easily flattenable. This imperfection is worsened by the fact that the process of manually bending the mesh back to its pre-flattening shape is rather imprecise and its outcome is hard to predict. One of the objectives of implant flattening is to design a flattening process such that the flattened mesh can be consistently bent back. The result should not heavily depend on manual handcrafting skills as the process should be made easy by this novel approach.

1.2 Thesis Objectives

Implant design consists in various operations that will greatly impact the final implant. With regard to the challenges described in the previous section, the precise objectives that will remain the main concern throughout the whole process will be:

- The generation of a volumetric cranioplasty implant with exact fit.
- The research and development of a flattening algorithm for volumetric implants.

In order to determine the current industry's needs and constraints, this project

was conducted in collaboration with Osteopore, a company that specialises in the production of 3D printed implants for craniofacial surgery. Osteopore provides patients with specific implants which are as fitting as possible to the defect skull. The implant design process is mostly performed with different 3D modelling software. The 3D engineer places the necessary vertices and creates most of the implant through 3D editing software. However, with the rising interest that Osteopore has received, it has been harder to keep up with the demand. Hence the need for a program that automatically performs the usual tasks with similar precision and faster execution. This would make the production easier and allow the company to provide its services to a wider audience.

1.3 Thesis Synopsis

The rest of this thesis is organized as follows. In Chapter 2, we conduct a literature review of existing mesh generation and surface flattening techniques. Chapter 3 describes a complete implant design method and shows how flattening can be satisfyingly applied to an implant layer. In Chapter 4, the approach described in Chapter 3 is tested and its results are evaluated. We conclude the thesis and discuss further directions for future research in Chapter 5.

Chapter 2

Literature Review

3D cranioplasty implant generation is a complex and specific problem that was rarely addressed during previous studies. However, some of the challenges of 3D implant design will be similar to more conventional problems. Namely, mesh generation approaches can be used to create the 3D meshes that make up the implant. Surface Flattening methods are used to address the implant's printing constraints.

2.1 Mesh Generation

There exists a wide range of methods to generate mesh structures. In the case of implant generation, the objective of this process is to fill the gap in the defect skull and approximate the initial curvature.

Loop filling consists in generating a 3D mesh structure that fills the inside of a loop. There is no constraint on the vertices inside the loop other than the fact that the mesh is connected to the whole loop and forms a correct structure. The most notorious algorithms are inspired by the Advancing-front method. [3] described this algorithm as an iterative progression of a front. The input loop gives the initial front, and the front shrieks as vertices and triangles are added at every iteration. [4] went further by re-estimating the normals and positions after filling the hole in order to generate a mesh that approximates local curvature.

Delaunay triangulation can also be used to perform loop filling. It produces a triangulation that maximizes the minimum angle of all the angles of the generated

triangles. [5] uses this property to bring an improvement to the advancing-front method. A Delaunay triangulation is computed and defines initial areas to help in adding vertices as the front advances.

Some studies have attempted to reconstruct a skull’s surface from a defect model. [6] has described an approach to reconstruct the outer layer of the skull. There is no straightforward way to make an implant from this work, but the outer layer can be used as part of an implant. It remains to generate the sides and the inner layer of the implant.

2.2 Surface Flattening

Many studies in the past decades have developed the field of surface flattening. It is used in many different ways and for various purposes. Metrics and cutting paths methods have jointly been described to further improve the quality of flattening algorithms.

2.2.1 Flattening algorithms

Flattening is consistently used in various fields and with different objectives. It consists in generating a flat layer from a 3D surface, usually in order to make an item’s production easier. However, flattening comes at a cost and some distortion is introduced by any algorithm. The object then needs to be bent back to its original shape, which will be possible only if the amount of added distortion is minimal. There exists a great variety of flattening algorithms which lead to very different results. [7] has described a method optimized to preserve the areas and shapes of triangles in order to flatten sheet-metal components. Their method is centered around an energy-based optimization process where vertices and edges are modelled as weights and strings. As the strings approach their equilibrium, the flattening accuracy increases. [8] presents an alternative method based on an estimation of local curvature and an estimation of the mesh’s major direction. However, this study is specifically tailored for a certain type of garment design and is not easily adapted to objects with a different topology. Surface flattening is also used for a

broad spectrum of applications in other parts of computer science. [9] has presented a flattening algorithm applied to perform mesh repair. Here, flattening is used to identify holes and flipped triangles are corrected. [10] has similarly defined a method centered around a novel flattening approach to generate a grid to support an irregular surface, like the stylized roof of an airport. In both of these contexts, no cut can be added to the mesh and the flattening’s purpose is not to preserve mesh curvature, but rather to build a 2D representation of the 3D surface which is then numerically reconstructed.

2.2.2 Flattening error metrics

The great variety in flattening methods is associated with a variety of uses and objectives. Existing contributions aim to optimize one factor which is relevant to their respective product. Flattening quality can therefore be assessed through different metrics. Previous studies have defined error measurements using various parameters based on what is relevant to a given use. Among the most relevant to our study, the metric defined by [11] shows how well a flattening process has preserved the input triangles’ angles. This angle preservation metric can be described as the angle difference between the input and the output. An algorithm is subsequently defined to optimize flattening based on this metric. There is no guarantee on distance or area preservation, however. This algorithm has been improved with ABF++ [12], but its design is still centered around angle preservation. Similarly, other studies have defined and optimized precise metrics. [13] uses an area preservation metric to ensure the mesh’s area suffers little distortion, and [14] has taken yet another approach by focusing on distance preservation. These approaches are all relevant to their respective application, but may not be optimal when applied for other purposes.

2.2.3 Cutting paths

One of the most efficient ways to reduce the amount of distortion introduced by flattening is the addition of cutting paths. Cutting between two triangles allows them to be flattened in different positions on the 2D plane, hence relieving constraints

CHAPTER 2. LITERATURE REVIEW

on their vertices, which leads to a smaller distortion. For most uses however, it is a challenge to tie the different pieces back together. Therefore, studies have been conducted to find the best cutting paths and to minimize their length. [15] has successfully designed an algorithm to find cutting paths to a boundary. This work can be applied to meshes with a disk-topology, but other works such as [16] [8] [17] [9] are applied to 3D surfaces which are not closed. [18] has showed this can be applied to closed surfaces, even though these are the hardest to flatten. However, this method introduces many cuts and their length is non-negligible, which could be problematic for some cases as shown by [7] [16].

The existing approaches aim to flatten 3D surfaces without constraint. Flattening an implant raises new challenges, as it is important that the cut introduced on top of the implant is the same as that on the bottom. Neither the disk flattening nor the closed surface flattening methods can be directly applied.

Chapter 3

Implant design

3.1 Implant Design Method

3.1.1 Input definition

The real-world objective of generating an implant needs to be converted into a computable problem. Before anything, the area that will be covered by an implant needs to be precisely defined through user-input. The problem's input must allow for a precise definition of the implant area, and be easily obtained through common 3D editing software.

To define the borders of the implant, an edge loop structure can be used. An edge loop is defined as an ordered set of connected points, with the additional constraint that the first and last points are also connected in the mesh structure, as shown in Figure 3.1. One edge loop is used to describe the location on the outer skull where the top border of the implant should be, and another edge loop is used on the inner skull for the bottom border. We will now refer to these two edge loops as the top loop and the bottom loop.

When storing these inputs, the edge loops need to be represented as a set of edges and not a set of vertices. While it may seem easy to reconstruct a loop from a convex set of 2D points, it is not possible on any general shape and it is made even more complicated in 3D. Experience has shown that a set of vertices could not appropriately represent an edge loop. Figure 3.2 shows a possible ambiguity when

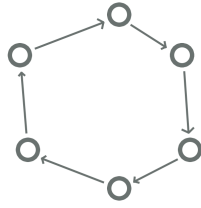


Figure 3.1: An edge loop.

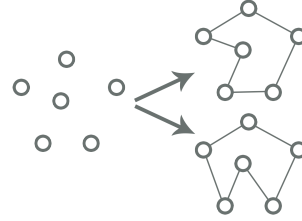


Figure 3.2: Reconstruction ambiguity.

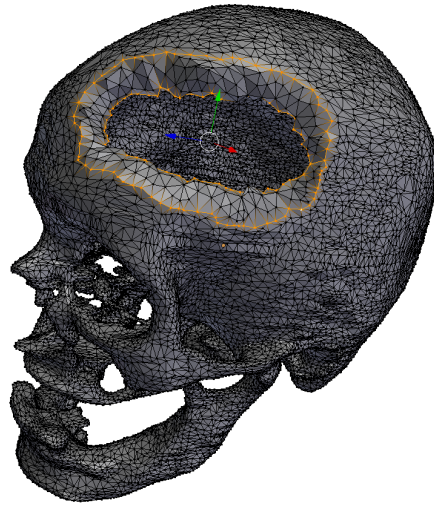


Figure 3.3: Problem definition from user.

reconstructing the edge loop from a set of vertices.

This definition of the input allows us to define the implant generation problem. Given the top loop, the bottom loop, and the defect skull as input as illustrated in Figure 3.3, the implant reconstruction problem is defined by the following objectives:

- Reconstruct the outer layer of the skull to fill the top loop
- Generate an appropriate mesh to bridge the top and bottom loops
- Reconstruct the inner layer of the skull to fill the bottom loop

3.1.2 Segmentation

Before reconstructing the top and bottom layers of the skull, it is important to identify the different regions of the input skull. In particular, this section will describe a method to identify the inner and outer layers of the input skull, as well as the middle mesh connecting the two layers in the defect zone.

To achieve this, an algorithm inspired by Union-Find will be used to segment the mesh into three areas. The mesh is described as a graph where the vertices are connected only if they are also connected in the mesh. However, the particularity here is that if a triangle has a vertex in the loops, then it is not taken into account in the propagation. This prevents the algorithm to propagate through the loops and therefore allows segmentation. The complete process is described in detail in Algorithm 1.

After vertices are assigned to their respective regions in Algorithm 1, no region is assigned to the top and bottom loops' vertices. Thanks to this, we may add triangles to the component as long as at least one of them was identified as part of that component. As a result, a vertex on a loop may be part of two different components, but a triangle may not by construction of the segmentation algorithm.

It may occur that this algorithm finds more than three regions. This may be due to a faulty mesh. For example, a test on real world data revealed that some triangles were found inside the skull between the two layers. The previous algorithm subsequently identified this 3D scanning artifact as a region. However, these components are not relevant to our study, so even if more than three regions are found this segmentation can be considered successful. The remaining components can be ignored.

It remains to identify which region is relevant. A possible approach to identify the outer and inner part of the skull could be to shoot rays from the center of the skull. The last triangle encountered by the ray is part of the outer surface, and the first is part of the inner surface. However, the problem defined previously is centered around the input of a bottom and a top edge loop, which can be used here. We can simply pick arbitrarily one point from each loop and identify components as shown

Algorithm 1: Segmentation Algorithm

Input: Mesh \mathcal{M} , edge loops \mathcal{E}_0 and \mathcal{E}_1
Output: Top, middle and bottom meshes

```

1: Function Segment( $\mathcal{M}, \mathcal{E}_0, \mathcal{E}_1$ ):
2:    $regions \leftarrow []$ 
3:    $border \leftarrow \mathcal{E}_0 \cup \mathcal{E}_1$ 
4:   for every triangle  $ABC$  do
5:     if  $A, B$  or  $C$  is part of the border then
6:       | do nothing and continue to next triangle
7:     if no region was assigned to  $A, B$  or  $C$  then
8:       | assign next available Id to  $A, B$  and  $C$ 
9:     else
10:      |  $k \leftarrow \min(\text{regions of } A, B \text{ or } C)$ 
11:      | assign region  $k$  to  $A, B$  and  $C$ 
12:      | if any of  $A, B$  or  $C$  changed from region  $r$  to  $k$  then
13:      | | change every vertex in region  $r$  to region  $k$ 
14:   for every triangle  $ABC$  do
15:     | let  $k_a, k_b, k_c$  be the regions of its vertices
16:     | add triangle to  $mesh_{k_a}, mesh_{k_b}, mesh_{k_c}$ 
17:   Identify  $k_{mid}$  such that  $mesh_{k_{mid}}$  contains a vertex from  $\mathcal{E}_0$  and one from  $\mathcal{E}_1$ 
18:   Identify  $k_{bot} \neq k_{mid}$  such that  $mesh_{k_{bot}}$  contains a vertex from  $\mathcal{E}_0$ 
19:   Identify  $k_{top} \neq k_{mid}$  such that  $mesh_{k_{top}}$  contains a vertex from  $\mathcal{E}_1$ 
20:   return  $mesh_{k_{top}}, mesh_{k_{mid}}, mesh_{k_{bot}}$ 

```

in lines 17-19 of Algorithm 1. This allows the unique identification of different mesh regions.

3.1.3 Top layer reconstruction

Existing work in the area of skull reconstruction can be used to describe a method to compute the top layer of an implant. In particular, [6] describes a method based on surface interpolation. Symmetry constraints further improve the accuracy of the method and produce more symmetric and normal-looking results. This approach yields satisfactory results on any part of the skull, under the assumption that enough symmetry constraints are provided. This can be a problem if a big portion of the skull is missing and no symmetric point can be found. Under the assumption that a sufficient part of the input skull is not damaged, the results from this approach are still satisfying.

3.1.4 Inner layer reconstruction

It is highly important for the inner part of the implant to be a good approximation of the patient's skull curvature. Failing to reproduce a realistic curvature may lead to complications during the surgical operation or later in the patient's life.

The skull's thickness is also an important factor. There can be important variations of the skull's thickness from one region to another, with values typically ranging from 3 mm to 14 mm. In order to approximate the skull's curvature, its thickness will need to be taken into account.

In this section, we will research ways to approximate the inner surface of the original non-defective skull in the area defined by the bottom loop. In order to reconstruct the bottom layer of the implant, we will use information given by the user input and the reconstructed top layer that was computed as explained in the previous section.

The objective of this section will be the generation of a mesh that satisfies the following properties:

1. The mesh is an appropriate filling of the bottom loop. In particular, no triangle is added outside of the loop and there is no hole in the mesh.
2. The mesh's curvature is an accurate approximation of what the skull's curvature was before it was damaged.
3. The distance between any point of the mesh and the top layer of the implant is a correct approximation of the skull's thickness in that point.

To reconstruct the inner layer of the skull, a series of operations will be applied. These operations will be described in the next paragraphs. The overall idea of the approach used is to:

- Fill the loop to get an initial layer
- Subdivide the mesh to improve the distribution of vertices

- Estimate thickness in every generated point
- Move the vertices to approximate the skull's thickness

3.1.4.1 Loop filling

In order to satisfy the first objective, a topologically coherent layer has to be generated. To do so, an intuitive idea is to fill the input loop given by the user. Evenly filling a 3D loop is a complex problem that has been the center-point of a number of research papers. Our research has identified two main approaches.

- The Advancing Front method considers the hole to fill as a front. It identifies an adequate vertex to advance the front and adds triangles there. Since the front is advanced through short steps, the mesh is evenly distributed and has enough vertices. The challenge here is that different sides of the front should converge to the same location in 3D.
- Delaunay Triangulation can also be applied to this 3D case. It adds triangle between three points such that no fourth point can be in the smallest sphere containing the three points. This approach leads to satisfying evenly distributed meshes, but it considers the edge loop only as a set of points. As a result, it may add triangles outside of the loop. It is possible to remove these triangles by checking their positions with regard to the loop, but it is not a simple task in 3D with a loop that may take any shape.

Our algorithm is inspired by the Advancing Front algorithm to fill the hole, in the sense that triangles are added iteratively and a front is gradually updated. However, to ensure the front converges to a single point, we will not add new vertices when the front advances. While the loop is not completely filled, a triangle is created around the vertex with the smallest angle in the loop and the said vertex is removed from the front, as illustrated by Figure 3.4 (b) and (c). The resulting algorithm is described in Algorithm 2.

After filling this loop, the bottom layer satisfies the first objective of a correct mesh. It could as a result be used to 3D print an implant, but this implant would

Algorithm 2: Loop filling Algorithm

Input: edge loop \mathcal{E}
Output: triangulation filling \mathcal{E}

```

1: Function FillLoop( $\mathcal{E}$ ):
2:    $front \leftarrow \mathcal{E}$ 
3:    $triangulation \leftarrow \emptyset$ 
4:   while  $|front| > 2$  do
5:     let  $(x, y), (y, z)$  be the two consecutive edges with smallest angle in  $front$ 
6:     add triangle  $(x, y, z)$  to  $triangulation$ 
7:     replace  $(x, y), (y, z)$  with  $(x, z)$  in  $front$ 
8:   return  $triangulation$ 

```

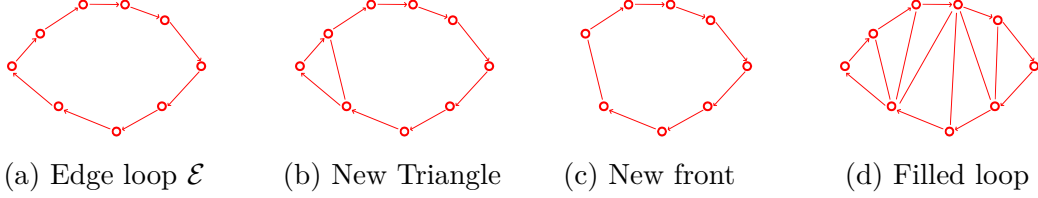


Figure 3.4: Loop filling method

not follow the right curvature and would press on the patient's brain. More precision is needed to represent the bottom layer. To do so, vertices will be added and evenly distributed across the layer.

3.1.4.2 Subdivision

The previous section has shown how to generate an initial bottom layer, but it does not follow the skull's curvature yet. To achieve this, the layer's vertices can be moved along their normal vectors to correctly match the thickness of the skull in this given point. However, our loop filling process does not add any vertex to the loop. The distribution of vertices and triangles on the surface is uneven, and triangles introduced in the end of the loop-filling process tend to be significantly larger. A subdivision is needed to evenly distribute vertices and triangles along the surface. Our objective is to generate triangles of approximately equal areas and angles along the whole layer.

General subdivision algorithms are usually used to make a surface smoother, and they will therefore not only add vertices but also change their positions. For example,

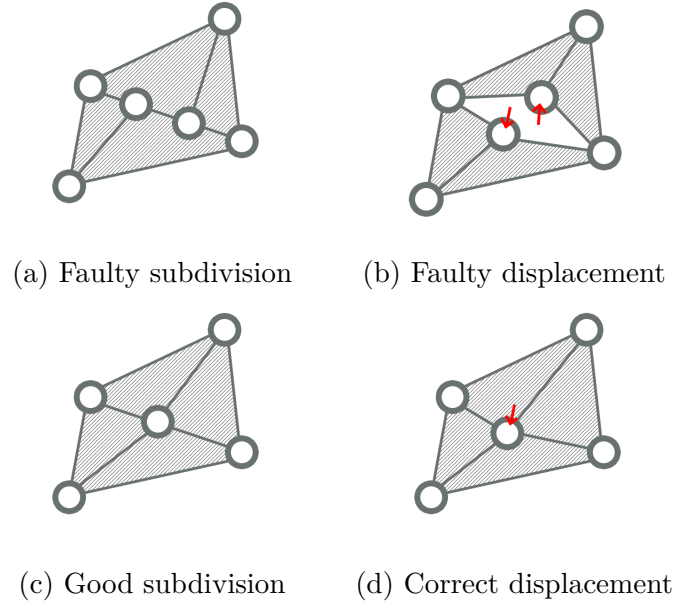


Figure 3.5: Importance of identical subdivision on both sides of an edge

Catmull-Clark subdivision adds vertices on the edges and moves the original vertices to be the average of its newly introduced neighbors. It does not, however, take into account the size of the triangles it receives as input, and applies the same treatment regardless.

If a triangle is divided in n smaller triangles regardless of its size, the triangles generated from the a relatively big triangle will still be bigger than those of a smaller triangle. Instead, the number of generated triangles needs to depend on the area of the triangle we would like to subdivide. Hence, Catmull-Clark subdivision will not be used and instead we will develop a novel approach.

Since an objective of this operation is to ensure the edges of our mesh are of similar lengths across the layer, some of the existing edges need to be divided into smaller segments. However most edges are part of two triangles, and if one triangle adds new vertices on this edge, the other triangle has to use the same ones. If this constraint is not met and the two triangles create vertices in different locations, the newly introduced vertices will not be properly connected to the mesh. An immediate consequence of this is the apparition of holes when transformations are applied to the mesh, as shown in Figure 3.5 (b). The objective of this subdivision is to prepare this mesh so that in the next step vertices can be moved to approximate

Algorithm 3: Subdivision functions

Input: A,B two 3D points, *margin* float number
Output: Set of points dividing $[AB]$

1: **Function** DivLine(*A*, *B*, *margin*):

2: $k \leftarrow \frac{\|A-B\|}{margin} - 1$

3: **for** $i \leftarrow 1$ **to** k **do** add new vertex in $A + \frac{i}{k+1} \times (B - A)$

4: **return** new vertices

Input: Coplanar lines AB and $A'B'$, *margin* float number
Output: Triangulation between AB and $A'B'$

5: **Function** StitchLines(*A*, *B*, *A'*, *B'*, *margin*):

6: $line_0 \leftarrow \text{DivLine}(A, B, margin)$

7: $line_1 \leftarrow \text{DivLine}(A', B', margin)$

8: let p and p' pointers to elements of $line_0$, $line_1$, starting at the beginning

9: **while** p or p' has not reached the end of its line **do**

10: **if** $\frac{p}{|line_0|} < \frac{p'}{|line_1|}$ **then**

11: add triangle $(line_0[p], line_1[p'], line_0[p+1])$

12: increment p

13: **else**

14: add triangle $(line_1[p'], line_1[p'+1], line_0[p])$

15: increment p'

16: **return** new triangles

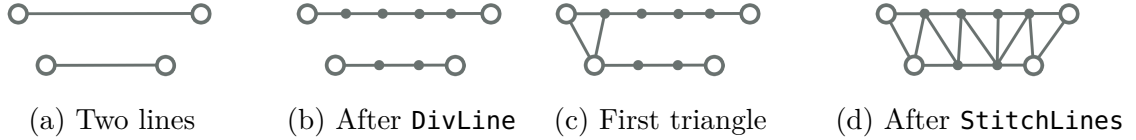


Figure 3.6: Line stitching method

the curvature of the skull, so this problem needs to be avoided.

To ensure this constraint is always satisfied, a function **DivLine** is defined to divide an edge into a number of segments. This number depends only on the length of the edge. If the number of segments to introduce is 0, the distance between A and B is already small and no vertex is added. Note that if any triangle applies this function to one of its edges, the positions of the newly introduced vertices will be identical to that of its neighbor sharing the same edge because this function depends only on the edge and the global margin.

From the previous function, a simple algorithm to stitch two lines together can

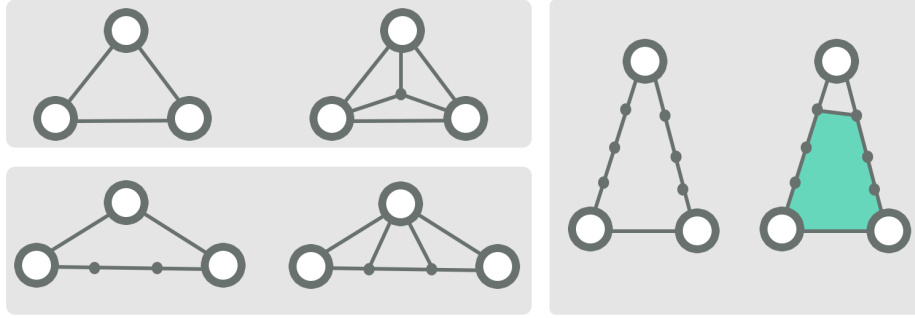


Figure 3.7: Subdivision on small triangles, where at least one side has no new vertex.

be defined. Applying the **DivLine** function allows the use of the newly introduced vertices without having to take the neighborhood into account. Then, triangles are added from one side to the other using the newly introduced vertices. The result of **StitchLines** is shown in Figure 3.6 (d).

The overall subdivision process is described in Algorithm 4. **DivLine** is applied to every edge of a triangle, and a subdivision scheme is applied. To define this subdivision, a recursive approach is used to adapt the process to triangles of varying size. Subdividing a big triangle will need a greater number of recursive calls than a small triangle. This will help reach a constant complexity on the whole layer. An adaptive subdivision scheme is applied based on the number of segments generated by **DivLine**.

When one or two edges of a triangle are too small to be subdivided, it is still important to take into account the new vertices on the remaining edges. The triangle cannot be left as is because if a neighboring triangle introduces a vertex, it will lead to the problem described in Figure 3.5 (b). With this in mind, the scheme shown in Figure 3.7 can be applied. In this representation, the green area is triangulated with the stitching function.

The recursive case is more complex. First, a smaller triangle is generated inside the triangle that is being subdivided as shown in Figure 3.8 (b). Then, in (c), the **DivLine** function generates new vertices on the small triangle's edges and the corners are triangulated. This leaves only four areas shown in (d), 3 of which (in green) are composed of approximately parallel lines and can be divided in triangles

Algorithm 4: Subdivision algorithm

Input: Mesh \mathcal{M}
Output: Subdivided mesh

```

1: Function Subdivide( $\mathcal{M}$ ):
2:   for every triangle  $ABC$  do
3:     Apply DivLine to its edges
4:     Apply subdivision scheme based on number of vertices introduced on
       each side
5:   return subdivided triangles

```

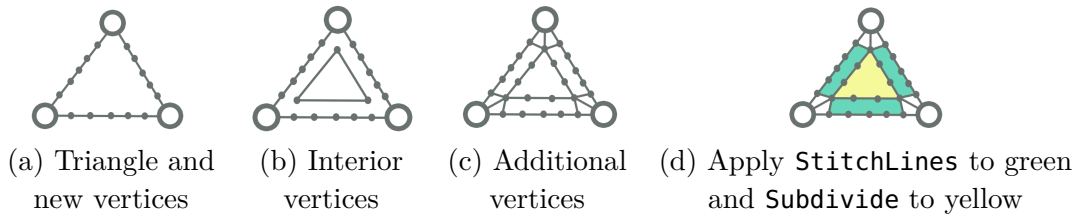


Figure 3.8: Recursive subdivision scheme

using the **StitchLines** function. Finally, this process is iteratively applied to the triangle in yellow. Since this triangle is smaller than the original one, its edges are subdivided in fewer points by the **DivLine** function. This ensures the process eventually terminates.

3.1.4.3 Thickness Estimation

The bottom loop is successfully filled through the operations described previously. However, the points added during this step were introduced without regard to the curvature of the skull. It remains to displace these vertices to better approximate the skull's inner surface.

To do so, the thickness of the skull has to be estimated in the area that is to be reconstructed. Indeed, the thickness of a human skull varies widely: it ranges from 3 mm in the temporal area to 14 mm in the occipital skull. As a result, a wide implant going from the side of the patient's skull to the top will not have a constant thickness.

Firstly, thickness needs to be defined. It is crucial to make thickness measurement in a given vertex easy to compute and fairly accurate. The thickness of a vertex in

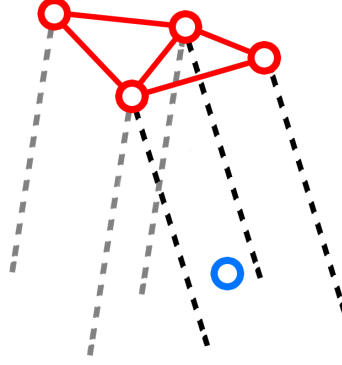


Figure 3.9: Blue vertex can be projected on the right triangle.

Algorithm 5: Thickness propagation algorithm

Input: Mesh \mathcal{M} , partial thickness mapping
Output: Complete thickness mapping on \mathcal{M}

- 1: **Function** `propagThick(\mathcal{M}, t):`
- 2: **while** *thickness mapping changes* **do**
- 3: **for** *every vertex* **do**
- 4: **if** *one or several neighbors have defined thickness* **then**
- 5: vertex thickness \leftarrow average of defined neighbors' thickness
- 6: **return** thickness mapping

one layer depends on the position of the other layer's triangles. One way to measure thickness of a vertex is to consider the triangle on the other layer such that the vertex can be projected on that triangle along the triangle's normal, as illustrated in Figure 3.9. Thickness can then be defined as the distance between that triangle and the vertex. The intuition behind this approach is that such a triangle should be approximately parallel to the area around the vertex, making it a good thickness estimator.

Some other approaches can be used to measure thickness. The simplest method would be to simply consider the closest vertex in the other layer and define thickness as the distance to that vertex. Similarly, the closest triangle can be considered. However, experiments tend to show that the approach described in Figure 3.9 gives more satisfying results and introduces less noise in the generated layer.

However, with the satisfying thickness measurement approach defined previously

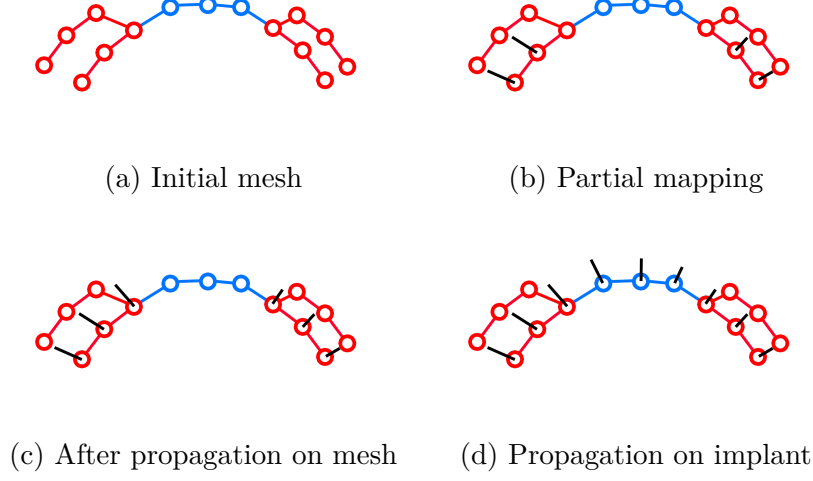


Figure 3.10: Thickness estimation method

it is important to note that some vertices may have undefined thickness. This occurs when no triangle is found to satisfy the situation shown in Figure 3.9. To tackle this issue, let us assume the thickness in one given point is close to that of its neighborhood. It is therefore possible to extrapolate and approximate the thickness in one point as that of its neighbors who have a defined thickness. If a vertex has an undefined thickness and none of its neighbors has a defined thickness, the vertex waits one round. This can be achieved with the `propagThick` function described in Algorithm 5.

The previous algorithm allows an estimation a thickness on all vertices as long as it is defined on some vertices. This can be used on the bottom mesh to obtain a complete thickness mapping, as illustrated by going from Figure 3.10 (b) to (c). Note here that the vertices on the bottom loop are part of both the inner layer of the skull and the initial bottom layer of the loop. As a result, the result shown in Figure 3.10 (c) can be used as a partial thickness mapping of the bottom layer of the implant. By using the thickness propagation algorithm on the bottom layer this time, this thickness mapping is extended to the whole layer from the loops as shown in Figure 3.10 (d). Through these operations, the thickness of the bottom layer is deduced from that of the bottom mesh.

This thickness propagation process allows for a global estimation of thickness

and a definition on the bottom layer of the implant. It remains to displace vertices to match the thickness mapping with regards to the implant's top layer.

3.1.4.4 Displacement

The inner skull's curvature can be reconstructed by approximating thickness and displacing vertices to match this thickness. With the thickness estimation described in the previous section, it remains to move the vertices to generate the right curvature.

To do so for any vertex, let us consider the triangle where the vertex can be projected as shown in Figure 3.9. Since this is how thickness was initially defined on the skull, it can be used on the implant to reproduce similar thickness. Since the top layer of the implant was previously computed, it can be used to project the bottom layer's vertices on and adjust thickness accordingly. A satisfying method to perform this is to move the vertex along the candidate triangle's normal axis such that the distance between the triangle and the vertex equals the estimated thickness of that vertex. This can be seen as the reverse operation of the thickness estimation performed on the skull. Distance between the two layers of the implant is modified to fit the thickness estimation.

As explained earlier, some vertices may not have a triangle on which they can be projected. In such isolated cases, the vertex can simply be moved to the average position of its neighbors. More generally, since this method moves vertices along the triangles' normal axes, there is no guarantee that it will not lead to flipped triangles. Furthermore, the geometric displacement can be influenced by the structure of the other layer and may as a result be noisy. Both of these drawbacks can be solved with a simple smoothing algorithm that replaces every vertex's position with the average of its neighbors' positions and its own. This will lead to a smooth surface without flipped triangles or any defect.

Finally, the generated layer is a perfect filling of the input bottom loop with an approximation of the initial skull's curvature. Therefore, it meets the objectives defined for the bottom layer of the implant and can be used as such. It remains to generate the sides of the implant to bridge the gap between the top and bottom

loops.

3.1.5 Sides of the implant

Reconstructing the area between the two edge loops can be performed in different ways. This step aims to accurately model the shape of the hole in the patient’s defect skull. A simple approach is to directly use the area between the two edge loops that was found during the segmentation process. The immediate advantage of this method is the perfect fit of the implant in the defect skull in theory. However, in practise it may be important to slightly adapt this area between the edge loops to facilitate the insertion of the implant by the surgeon. In simple cases, simply reducing the area covered by the implant while conserving the shape can be satisfactory. However, more complex cases with complex curvature on the sides of the defect area may require human intervention to manually build this part of the implant.

3.2 Flattening Method

A cranioplasty implant is made with a specific material, and its manufacturing process is adapted to the material’s constraints. In particular, implant manufacturers need to apply transformations to the implant mesh in order to make it lie flat on the 3D printing plane. This allows for greater mechanical strength and removes the need for scaffolding material during printing. The implant is then bent back to its original shape by hand.

There exists various kinds of flattening methods in literature. Their objective is to flatten a layer with a disk-like topology while introducing as little distortion as possible. The implant that needs to be flattened does not have a disk-like topology, however. Flattening algorithms cannot directly be applied. Instead, and since the previous sections have described how to independently reconstruct top and bottom layers, it is possible to apply a flattening algorithm to each layer and stitch them back together once flattened.

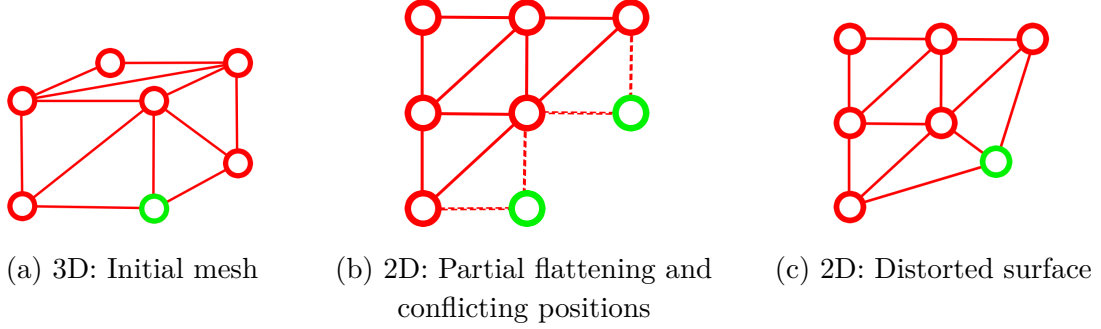


Figure 3.11: Flattening distortion

3.2.1 Flattening Algorithm

Flattening one single layer of the implant consists in converting a 3D surface with a disk-like topology into a 2D surface. The objective of this operation is to generate a flat layer that will still correspond to the original mesh in the sense that it can be bent back to its original shape after being 3D printed.

However when a mesh is flattened onto the UV plane, it may not be possible to perfectly represent the 3D object in 2D. More precisely, flattened triangles will be distorted and their geometry may be altered. Figure 3.11 shows an example of distortion induced by flattening. In (b), only the green vertex remains to flatten onto the 2D plane and the two triangles that include him in the 3D mesh lead to conflicting positions in 2D. If the average of these positions is chosen, both triangles suffer distortion as shown in Figure 3.11 (c). In general, distortion will be introduced to any mesh with non-zero Gaussian curvature. This distortion can be measured in terms of angle, area, or lengths.

The algorithm used for this study is inspired by the one described in [7]. An initial seed triangle is chosen. The seed triangle is translated and rotated to lie in the 2D plane in $(0, 0)$. Then, the algorithm propagates to the neighboring triangles. When 2D coordinates of a vertex are computed, two cases can occur:

- The vertex was never encountered before in the process. In that case, assign it the right coordinates to perfectly flatten the current triangle.
- The vertex was flattened before and has conflicting 2D coordinates. In that

case, set its new coordinates as an average of all candidates.

The second case is the source of all distortion introduced in the mesh during flattening. Our experiments indicate that choosing the seed triangle as the one furthest from the boundary yields satisfactory results. It may be interesting to consider other triangles as seed, such as the one with the highest Gaussian curvature. Flattening algorithms can also be improved through the use of cutting paths methods.

3.2.2 Cutting Path

The flattening algorithm presented in the previous section adds distortion to the input layer. As a result, it may not be possible to bend the implant back to its original shape. In order to improve the flattening quality and remove part of the distortion, a cutting path can be added to the layer prior to flattening.

A layer is described as a 3D mesh and a boundary surrounding it. In the case of this study, the boundary is initially defined as the bottom or top edge loop based on which layer is being flattened. Let us define a cutting path as a path on the mesh from a given vertex to the boundary. To help ease flattening constraints, all the vertices on the cutting path are split in two. That way, the second case described in the flattening algorithm will be less frequent. This will release strain over the whole mesh and reduce the overall distortion.

The first challenge of cutting paths is the selection of an optimal path. [15] has described an algorithm that identifies points with the highest curvature and defines a cutting path that goes through all of them before reaching the boundary. In our experiments, we have obtained satisfactory results with a simpler approach by identifying the single point which is furthest from the boundary. Then, the shortest path from that point to the boundary is chosen as cutting path.

Once the cutting path is chosen, the vertices are split and the two sides separated. Our approach is to identify each side of the cut with a propagation algorithm. This process is detailed in Figure 3.12. First, as seen in (b), the two vertices opposing the first edge of the cutting path are identified in green and blue. Then, a propagation

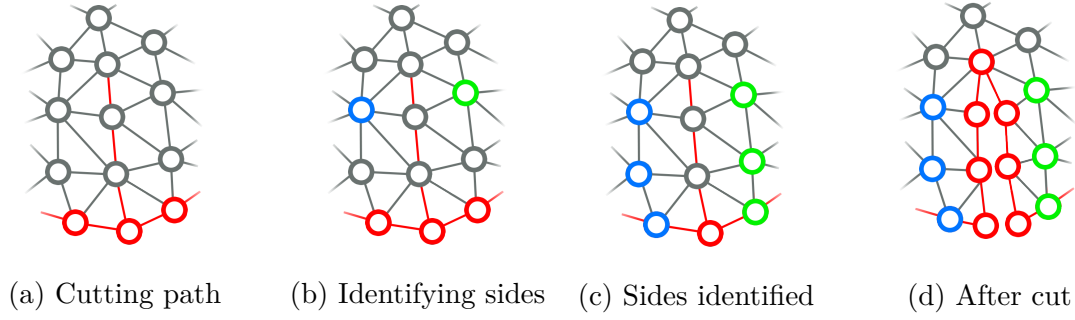


Figure 3.12: Cutting method

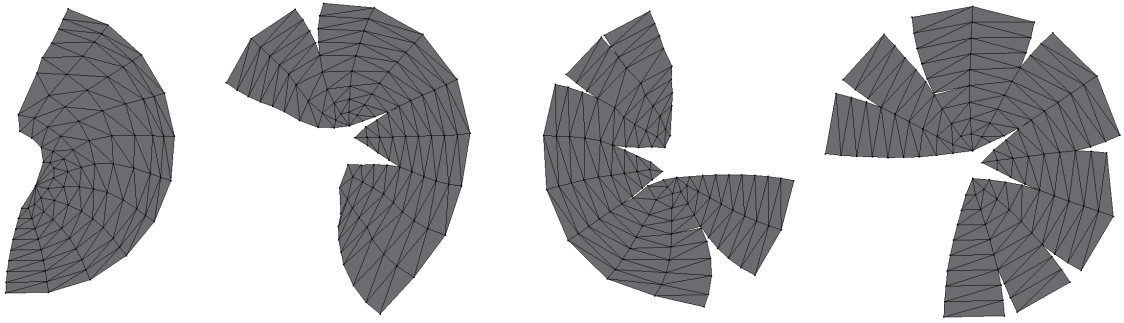


Figure 3.13: Cutting effect on flattening semi-sphere (resp. 1/4/7/10 cuts).

on the mesh through the triangles neighboring the cutting path is performed to identify the two sides of the path, shown in (c). Finally, vertices on the cutting path are split into two and properly stitched with one of the two sides (d). A new boundary is computed to include the newly split vertices.

To demonstrate the effect of cutting paths, flattening was performed on a semi-sphere model. The results in Figure 3.13 show the efficiency of cutting paths. With fewer cuts, triangles are unevenly distorted. The flattened result with 10 cuts is a much better result as triangles' areas and angles are preserved.

3.2.3 Edge smoothing

Because of the practical constraints on cranioplasty implants, the border of the implant needs to be smooth. 3D meshes are represented with triangles and will consequently have sharp edges. To minimize this drawback, it is possible to apply

CHAPTER 3. IMPLANT DESIGN

an edge smoothing algorithm. A simple way to achieve this is to replace every vertex by a weighted average of itself and its neighbors. This allows for the definition of a smoothing coefficient $s \in [0, 1]$. More precisely, given a vertex x and its neighbors in the boundary x_{left} and x_{right} , the following transformation is applied:

$$x' = x \times (1 - s) + x_{left} \times \frac{s}{2} + x_{right} \times \frac{s}{2}$$

This method tends to make the boundary slightly smaller. In the case of an implant, this is desirable as it means the implant will be slightly easier to apply while remaining of an approximately correct shape. There is no guarantee that the new position x' does not overlap another triangle, but in practise this formula has proven to be efficient.

Chapter 4

Experiments and Discussion

4.1 Overview

A method to automatically generate an implant and flatten its layers was described in Chapter 3. Estimating the efficiency of this approach remains a challenge that needs to be addressed.

The implant design method revolves around curvature estimation. The generated curvature can be compared with that of a non-defect skull. Assessing implant quality is hard to compute but it is quite an intuitive task that can be done manually. In the context of a real implant, verification by a human of the quality of the implant would be essential.

Flattening methods, on the other hand, can be tested and evaluated through a wide range of metrics. By comparing associated triangles before and after flattening, geometric distortion indicators can be used to evaluate our approach.

4.2 Data preparation

Prior to any experiment, an input needs to be prepared. 3D meshes are often made up of hundreds of thousands of triangles. Since our approach performs complex operations, the input mesh needs to be adapted to our use. In particular, any triangle

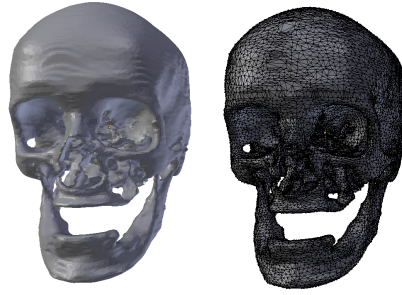


Figure 4.1: Initial skull

which is far from the area of interest should be removed.

4.2.1 Input used

Inputs used in this experiment were generated from a publicly available skull model (Figure 4.1). A hole is created in the skull's surface by selecting and removing an area on both the inner and outer layer. This allows for the simulation of a defect skull. The advantage of this method over the use of actual defect skull models is that the initial non-defect skull can be used for evaluation purposes. The implant that is generated should match the initial skull's curvature which is considered ground truth. More precisely, the generation of an input goes through the following steps:

- Select area on outer skull layer and delete its vertices
- Select area on inner skull layer and delete its vertices
- Bridge the gap with an initial filling
- Smooth the newly introduced area to replicate a real defect skull
- Select bottom and top edge loops

This method makes it easy to test the quality of the proposed algorithm. For example, an input was generated to resemble a small skull defect and is shown in Figure 4.2. It will be referred to as Patient 1. The defect skull is convincing and Figure 4.3 shows the mesh surrounding the hole is satisfying.

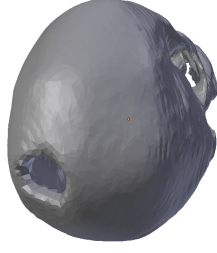


Figure 4.2: Patient 1

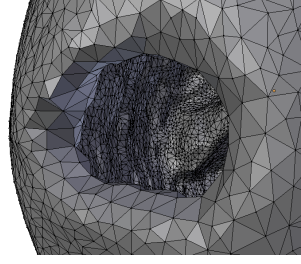


Figure 4.3: Artificial hole mesh

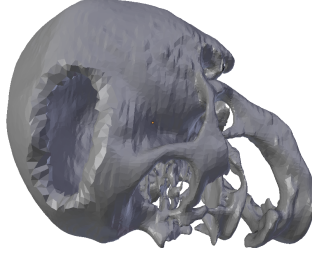


Figure 4.4: Patient 2

For comparison purposes, an input for Patient 2 is also generated. It is shown in Figure 4.4. This input is much larger and the defect area has a stronger curvature. It will be used to test the robustness of our approach and its ability to handle more challenging cases.

4.2.2 Selection of Area of Interest

Our real-world input data is quite large. A 3D scan of a skull easily surpasses 20 Mb, which corresponds to a mesh of potentially over hundreds of thousands of vertices and triangles. This leads to extensive computing time for any operation. Initially, a scan contains mesh information about the teeth or the eye sockets of the patient. These areas require tremendous amounts of triangles to be described in 3D and they are irrelevant to our study. As such, and with regards to what was requested by Osteopore, the first objective is the generation of a smaller mesh to work on.

To delete non-essential triangles, an initial area of interest needs to be defined. It can be arbitrarily selected through a 3D editing software such as Blender. In the case of this study, the edge loops are used to estimate the initial area of interest.

Algorithm 6: Extremums Simplification

Input: Mesh \mathcal{M} , Vertices of interest IV , float $margin$

Output: Purified Mesh

```

1: Function extrSimpl( $\mathcal{M}, IV, margin$ ):
2:   Compute extremums values of  $IV$ 
3:   for all vertices do
4:     if vertex not contained in extremums  $\pm margin$  then
5:       delete this vertex
6:   return purified mesh
    
```

However, as shown in the previous sections, the area around the edge loops is needed to estimate the implant's thickness. This information must not be lost and the triangles that are at a distance to the interesting area below a threshold must be kept.

In order to achieve the best trade-off in efficiency and computation time, different approaches are described. During this step, complexity will be key as the amount of vertices is significant. With this in mind, let us denote $|IV|$ the number of interesting vertices given as input, and $|V|$ the total number of vertices. The quantity $|IV|$ is assumed to be far smaller than $|V|$ but non-neglectable.

4.2.2.1 Extremums simplification

Initially, the most important factor is time complexity. Optimally, a simplification should run in $O(|V|)$. With this in mind, a fast method is described in Algorithm 6. It relies on one pass on the vertices of interest to find the interesting extremums, and one pass on all vertices to filter out those out of these extreme ranges.

In terms of complexity, Algorithm 6 goes through IV to find its extremums in $O(|IV|)$ time. Then, going through the vertices to delete them takes $O(|V|)$. The overall time complexity is therefore in $O(|V| + |IV|)$.

Regarding the efficiency of this method, it can be noted that vertices preserved may be as far as $margin$ on every axis. The distance of any vertex to an interesting vertex is therefore at most $\sqrt{3} \times margin$.

Algorithm 7: Distance Simplification**Input:** Mesh \mathcal{M} , Vertices of interest IV , float $margin$ **Output:** Purified Mesh

```

1: Function distSimpl( $\mathcal{M}, IV, margin$ ):
2:   for all vertices  $v$  in  $\mathcal{M}$  do
3:     for all vertices  $v'$  in  $IV$  do
4:       if  $\|v - v'\| < margin$  then
5:         keep  $v$ , continue to next  $v$ 
6:       delete  $v$ 
7:   return purified mesh

```

4.2.2.2 Distance simplification

A more straightforward but costly algorithm is to check, for every vertex, if there exists an interesting vertex within $margin$ distance. The approach is given in Algorithm 7. It compares all vertices to IV in $O(|V| \times |IV|)$ time. It is slower than the previous approach by a significant factor, but any vertex kept is at distance at most $margin$, which is our objective.

4.2.2.3 Comparison

Distance	$< margin$	$m. < d < \sqrt{3}m.$	$> \sqrt{3}margin$	Complexity
extrSimpl	Kept	No guarantee	Deleted	$O(V + IV)$
distSimpl	Kept	Deleted	Deleted	$O(V \times IV)$

Table 4.1: Simplification comparison

In practise, **distSimpl** is too slow to be used directly on the input mesh. However, **distSimpl** can efficiently be used on the output of **extrSimpl**. If we denote $|V'|$ the amount of vertices from the output of **extrSimpl**, the time complexity of this approach becomes $O(|V| + |IV| + |V'| \times |IV|)$. Our experiments shows this greatly reduces computation time without reducing the effectiveness of **distSimpl**, as shown in Table 4.2. Even after **extrSimpl** removed 76% of vertices, **distSimpl** removes an additional 65.73% of vertices. This shows this method gives significantly better results. This approach will therefore be used.

The results in Table 4.2 will however vary on other parameters such as the margin or the relative size of the area of interest. In general the efficiency achieved in our

Method	Input size	Output size	Time	T. removed
extrSimpl	151351	35904	4.46s	76.28 %
distSimpl	151351	12306	1008s	91 %
distSimpl (after extrSimpl)	35904	12306	200s	65.73 %

Table 4.2: Simplification results

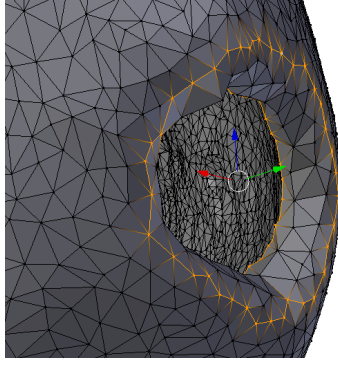


Figure 4.5: A possible edge loop definition for Patient 1

experiments has ranged from 70% to 99%.

4.3 Implant Design Experiment

4.3.1 Experiment

This experiment will focus on the generation of an implant for Patient 1 as shown in Figure 4.2. We define the edge loops as shown in Figure 4.5. For this experiment, the outer edge loop will be defined a bit larger than the defect area. This will allow for an implant with a bigger support. Such a choice can be made by the user when they select the edge loops.

With these edge loops defined, the mesh of interest can be extracted as explained in the previous section. This process achieves a final eradication rate of 98,58% on Patient 1 with a margin set to 10 mm. This excellent rate is explained by the fact that the area that requires an implant is very small when compared to the size of the skull, so very few triangles need to be kept. It takes 21 seconds to run when using

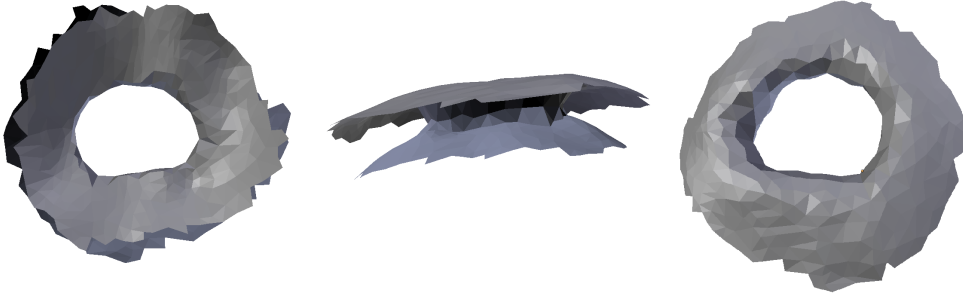


Figure 4.6: Simplified mesh from Patient 1 seen from below, the side, and up.



Figure 4.7: Simplified mesh from Patient 2.

the longer and more efficient distance based approach. Note that the margin used may be satisfactory or not based on the scale used to describe 3D vertices. To ensure simplification is successful, the purified mesh needs to be visualized. The result of this experiment is shown in Figure 4.6. The figure shows that the neighborhood of the area that requires an implant was preserved. It is wide enough to extract a robust approximation of the skull's local thickness. Therefore, we can conclude that the margin chosen is sufficient.

Similarly, applying this method with the same 10 mm margin on Patient 2 removes 97% of triangles in under 38 seconds. The simplified mesh, shown in Figure 4.7, is also large enough to allow for an accurate thickness estimation.

Before any operation can be applied, the different parts of the skull must be identified. The application of the segmentation process described previously on Patient 1 yields the result shown in Figure 4.8. The inner surface of the skull, the

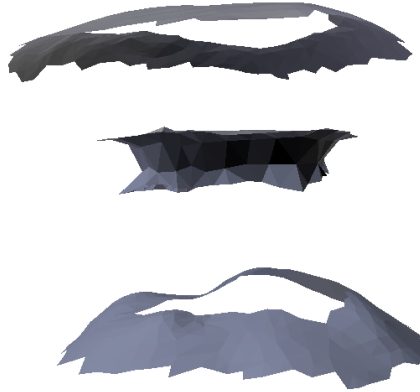


Figure 4.8: Identified regions separated

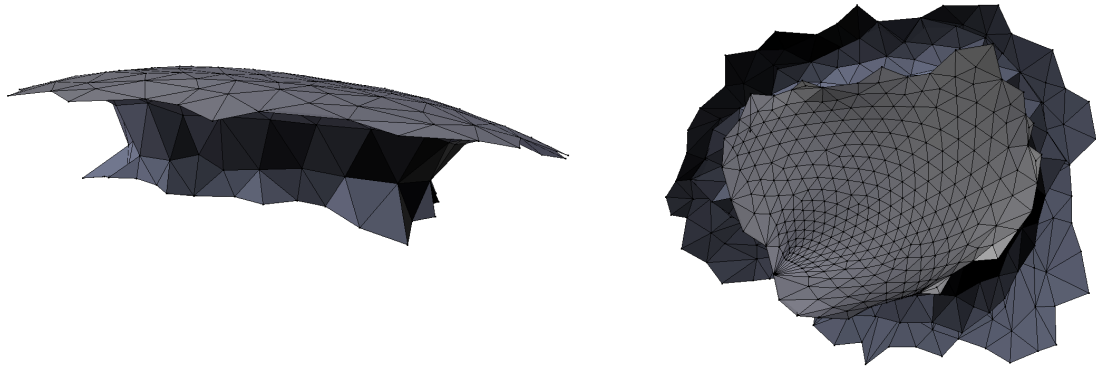


Figure 4.9: Implant for patient 1

outer surface, and the middle section can easily be identified.

This experiment assumes the outer part of the skull was reconstructed with the process described in [6]. The side of the implant is directly derived from the middle section identified with the segmentation. The bottom layer is reconstructed using the approach described in Chapter 3.

The final implant generated for Patient 1 is shown in Figure 4.9. It correctly approximates the skull's inner surface, and the inner layer is smooth. In total, this implant generation took 8 seconds, excluding the initial simplification step. This is a satisfactory result.

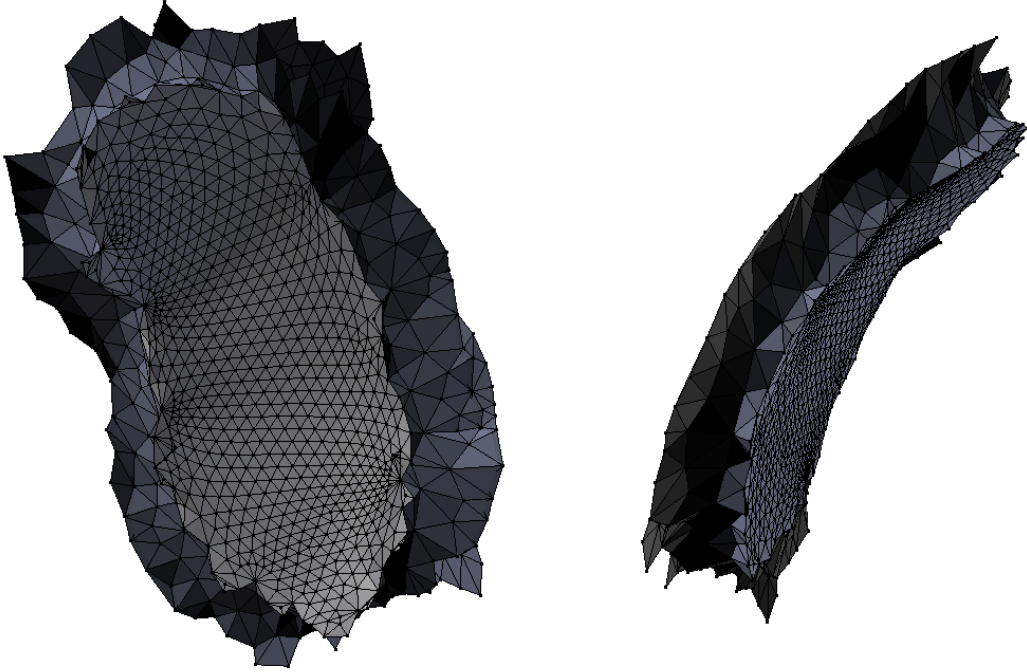


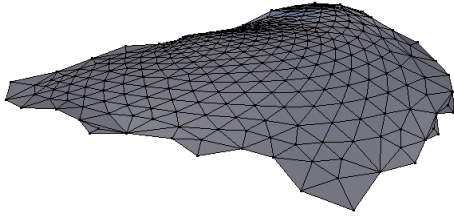
Figure 4.10: Implant for patient 2

In comparison, the implant generation for Patient 2 took 38 seconds. The result shown in Figure 4.10 also has a satisfying curvature. The inner layer and outer layer maintain an appropriate thickness throughout the implant and the implant is smooth.

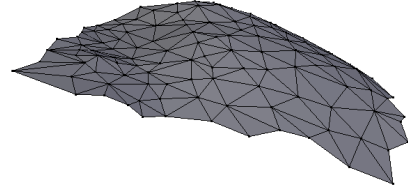
4.3.2 Implant Evaluation

To assess the quality of the generated implant, it can be compared with the initial inner skull layer from the input skull. The results in Figure 4.11 indicate that the generated layer is very similar to its original equivalent. The curvature of the implant seems a bit smoother but is overall equivalent. The same comparison is performed on Patient 2 in Figure 4.12. The implant has successfully adapted to the stronger curvature of this skull area.

To further analyse the quality of the inner surface, the inner layer of the implant can be visualized jointly with the inner layer of the defect skull as identified during

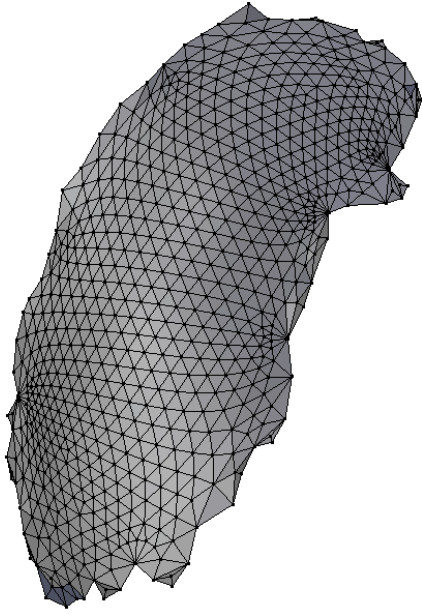


(a) Implant inner layer

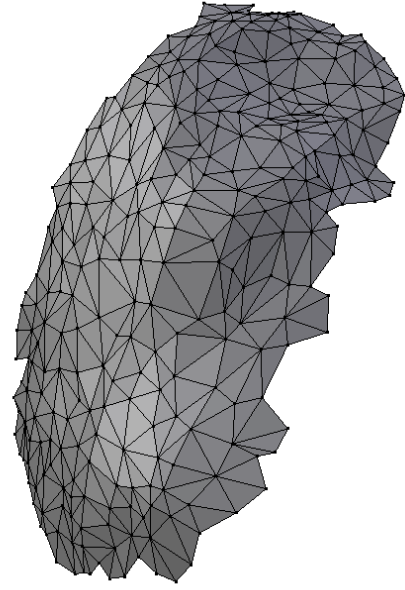


(b) Equivalent mesh from the original skull

Figure 4.11: Comparison of generated inner layer and original layer, Patient 1



(a) Implant inner layer



(b) Equivalent mesh from the original skull

Figure 4.12: Comparison of generated inner layer and original layer, Patient 2

segmentation. The results can be seen in Figure 4.13 (a) and (b). The reconstructed layers match the skull appropriately and the local curvature is approximately preserved. This experiment shows the reconstructed inner layer is an appropriate reconstruction of the skull's inner layer.

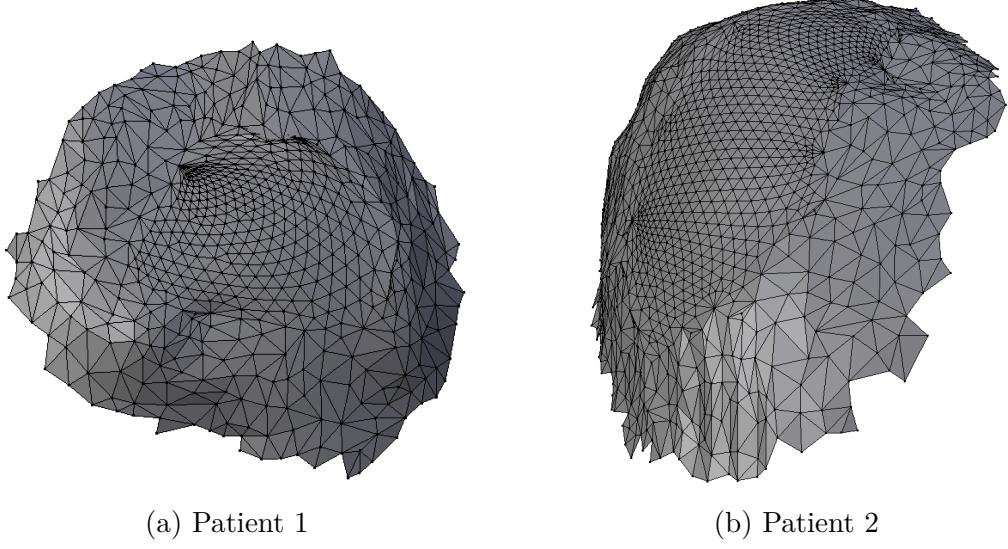


Figure 4.13: Inner reconstruction filling inner skull layer

4.4 Implant Flattening Experiment

4.4.1 Flattening Experiment

In order to better assess flattening quality, metrics need to be defined. Each newly generated flat triangle will be compared to its former equivalent. First, area distortion on a given triangle is defined as:

$$areaDistortion = \frac{|newArea - oldArea|}{oldArea}$$

Similarly, angle distortion can be written as:

$$angleDistortion = \frac{|newAngle - oldAngle|}{oldAngle}$$

Both of these error metrics are defined on a single triangle. It is therefore possible to compute the average distortion on the whole flattened mesh to get an estimation of the overall flattening quality. It is also interesting to compare the maximum distortion on any triangle in the mesh.

CHAPTER 4. EXPERIMENTS AND DISCUSSION

A cut was applied to test the usefulness of cutting path on an implant. Flattening quality would improve if more cuts were added, but it is unpractical to print an implant with several cuts and as such we will add no more than a single cut.

4.4.2 Flattening Evaluation

Patient	Avg. area d.	Max area d.	Avg. angle d.	Max angle d.
1	8.09 %	98.62 %	6.4 %	168.02 %
1 - with cut	4.68 %	98.65 %	3.69 %	129.52 %
2	4.24 %	260.98 %	3.11 %	356.4 %
2 - with cut	4.07 %	339.14 %	3.08 %	404.9 %

Table 4.3: Flattening evaluation

The results presented in Table 4.3 show that the average of any metric is within a 10% distortion threshold. It is also interesting to note that the cutting path added to the implant generated for Patient 1 led to a significant decrease of distortion. We conclude that the cutting path method is efficient. The improvement is much lesser for Patient 2. A possible explanation is a poor choice of cutting path which does not relieve much flattening constraint.

The maximum of both area and angle distortion remains high. This means one of the triangles was greatly modified. However, it can be noted that if a very small sliver triangle suffers from a little distortion in distance, the angle and area may vary widely. The fact that this distortion is not visible on the flattened layer leads us to believe that this maximum distortion can be attributed to such sliver triangles. We can conclude from this that our mesh generation could be improved, as it generates some sliver triangles.

The flattened layers, shown in Figure 4.14 and Figure 4.15, should also be evaluated manually to check the flattened layer has a similar shape to that of the initial implant. For both Patient 1 and Patient 2, the surface flattening process has preserved the overall structure of the mesh. Adding a cut to the mesh proves to be useful as the cut is visible on the flattened result. The triangles in the region around the cut have kept a more uniform aspect. This shows the amount of distortion has

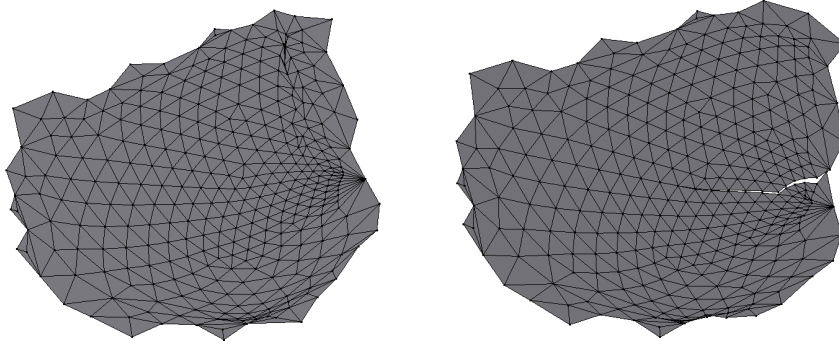


Figure 4.14: Flattened inner layer of the implant without and with cut, Patient 1.

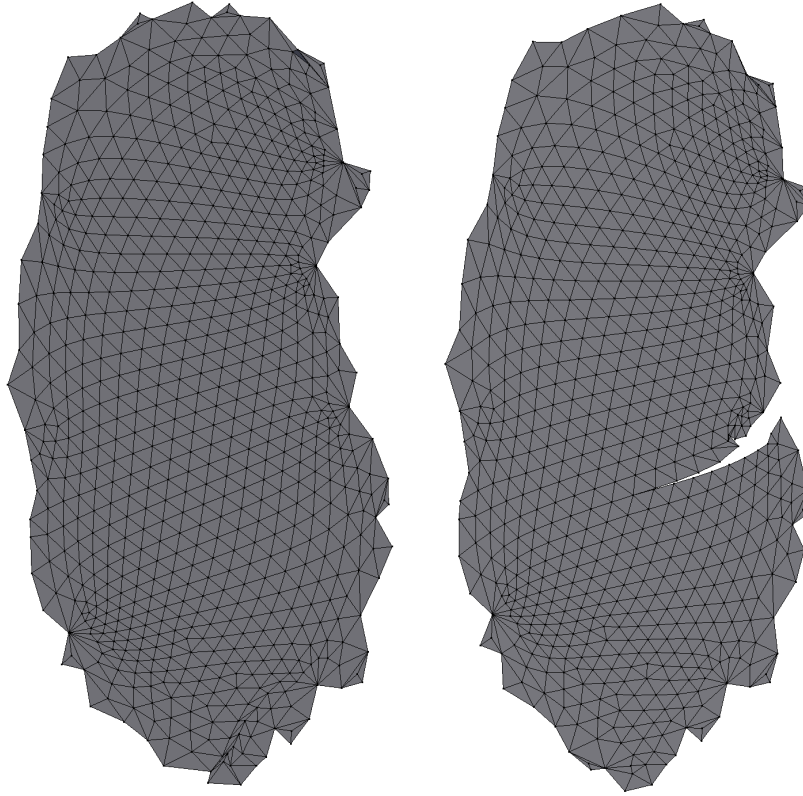


Figure 4.15: Flattened inner layer of the implant without and with cut, Patient 2.

decreased overall. This is especially obvious with Patient 1, which is coherent with the results in Table 4.3.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Cranioplasty implants are an efficient but costly way to treat head injuries. This study aims to provide an efficient and simple-to-use implant generation procedure. We have shown, in the previous sections, how to compute a 3D implant from an input skull and two edge loops. From a user perspective, the input method consists in simply selecting two loops. This is both easy and precise.

Through thickness estimation and mesh generation, the inner layer of the skull is reconstructed and matches the curvature of the outer layer. This allows for a perfect fit implant that can accurately replace the original skull.

Implant generation is performed in a reasonable amount of time thanks to a simplification of the input skull. Human input is only needed at the very beginning for inputs and in the end to check the accuracy of the generated implant.

Then, flattening methods are applied to the implant to help generate a printable implant. Results are improved through the use of cutting paths. This method reduces the amount of distortion introduced by the flattening algorithm. The resulting layer is then smoothed to make the implant easier to use during a surgical operation.

We hope this work can lay a foundation for more in-depth analysis of cranioplasty implants generation. Overall, this would help make implants more affordable and of a better quality for skull injury patients across the world. We believe this has the

potential to make implants more commonly used and safer.

5.2 Future Research Directions

The procedure described in this study creates a perfect fit implant. However, it is not always desirable in cranioplasty to perfectly match the input skull. In particular, the surgeon may find it difficult to insert such an implant into a defect skull. Future research could improve on the work exposed in this study by adapting the generation of the sides of the implant to make it possible to insert the implant no matter what input is given by the user. Further work could also be conducted to implement additional practical constraints such as a smoothness requirements for implants.

The flattening procedure presented can also be improved. In particular, the cutting path on the implant will have consequences on the final implant. Experiments on implant 3D printing have shown that some cutting paths were likely to lead to a bump after bending back the implant to its original shape. In particular, it seems curved path may yield better results. In some cases, the addition of a cutting path may not be justified, and there is currently no simple way to estimate the optimal number of cutting paths. Further research in this domain could help make implants minimizing flattening distortion and easy to bend back.

Our study has shown how to flatten a layer efficiently. It remains to develop a procedure to flatten the whole implant without altering the implant. In particular, when a cut is introduced on one layer, there is currently no guarantee that the same cut will be added on the other layer. Research is needed to develop an algorithm to jointly flatten the top and bottom layers. These improvements would greatly reduce the overall imprecision and distortion introduced by the flattening process.

Bibliography

- [1] K. Mohammad, “Customised cranioplasty implant for decompressive craniectomy patients ? a technical note”, *Turkish Neurosurgery*, vol. 29, p. 148, 2019.
- [2] C. C. L. Wang, “Towards flattenable mesh surfaces”, *Computer-Aided Design*, vol. 40, no. 1, pp. 109–122, 2008.
- [3] P. Parikh and R. Loehner, “Generation of three-dimensional unstructured grids by the advancing-front method”, *International Journal for Numerical Methods in Fluids*, vol. 8, no. 10, pp. 1135–1149, 1988.
- [4] S. G. W. Zhao and H. Lin, “A robust hole-filling algorithm for triangular mesh”, *The Visual Computer*, vol. 23, no. 12, pp. 987–997, 2007.
- [5] P. L. George and E. Seveno, “The advancing-front mesh generation method revisited”, *International Journal for Numerical Methods in Engineering*, vol. 37, no. 21, pp. 3605–3619, 1994.
- [6] L. C. et al, “Flip-avoiding interpolating surface registration for skull reconstruction”, *The International Journal of Medical Robotics and Computer Assisted Surgery*, vol. 14, no. 4, p. 1906, 2018.
- [7] J. X. Q. Liu and Z. Wu, “An energy-based surface flattening method for flat pattern development of sheet metal components”, *The International Journal of Advanced Manufacturing Technology*, vol. 68, no. 5, pp. 1155–1166, 2013.
- [8] J. Fang and Y. Ding, “Energy-based optimal darted pattern for garment design”, *International Journal of Clothing Science and Technology*, vol. 26, no. 2, pp. 164–183, 2014.
- [9] S. Oh, “A new triangular mesh repairing method using a mesh distortion energy minimization-based mesh flattening method”, *Advances in Engineering Software*, vol. 131, pp. 48–59, 2019.

BIBLIOGRAPHY

- [10] B. G. et al, “Grid generation on free-form surface using guide line advancing and surface flattening method”, *Advances in Engineering Software*, vol. 110, pp. 98–109, 2017.
- [11] A. Sheffer and E. de Sturler, “Parameterization of faceted surfaces for meshing using angle-based flattening”, *Engineering with Computers*, vol. 17, no. 3, pp. 326–337, 2001.
- [12] A. S. et al, “Abf++: Fast and robust angle based flattening”, *ACM Transactions on Graphics (TOG)*, vol. 24, no. 2, pp. 311–330, 2005.
- [13] G. Z. et al, “Area-preserving surface flattening using lie advection”, *Anonymous Berlin, Heidelberg: Springer Berlin Heidelberg*, pp. 335–342, 2011.
- [14] O. F. L. Saroul and R. D. Hersch, “Distance preserving flattening of surface sections”, *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 1, pp. 26–35, 2006.
- [15] C. C. L. W. et al, “Reduce the stretch in surface flattening by finding cutting paths to the surface boundary”, *Computer-Aided Design*, vol. 36, no. 8, pp. 665–677, 2004.
- [16] C. L. S. Zhang and X. Wang, “An optimal flattening algorithm for ship hull plate fabricated by thermal forming”, *Journal of Marine Science and Technology*, vol. 24, no. 1, pp. 134–151, 2019.
- [17] J. L. D. Zhang and J. Wang, “Design patterns of soft products using surface flattening”, *Journal of Computing and Information Science in Engineering*, vol. 18, no. 2, 2018.
- [18] C. G. M. Ben-Chen and G. Bunin, “Conformal flattening by curvature prescription and metric scaling”, *Computer Graphics Forum*, vol. 27, no. 2, pp. 449–458, 2008.