UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO DE CIÊNCIAS FÍSICAS E MATEMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM MATEMÁTICA PURA E APLICADA

Carlos Eduardo Leal de Castro

Optimal rotations for overhang reduction

Florianópolis
2020

Carlos Eduardo Leal de Castro

Optimal rotations for overhang reduction

Dissertação de mestrado apresentada para o Programa de Pós-Graduação em Matemática Pura e Aplicada da Universidade Federal de Santa Catarina, para a obtenção do grau de Mestre em Matemática Pura e Aplicada.
Orientador: Prof. Dr. Leonardo Koller Sacht.

Florianópolis
2020

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Castro, Carlos Eduardo Leal de
   Optimal rotations for overhang reduction / Carlos
Eduardo Leal de Castro ; orientador, Leonardo Koller
Sacht, 2020.
   100 p.

   Dissertação (mestrado) - Universidade Federal de Santa
Catarina, Centro de Ciências Físicas e Matemáticas,
Programa de Pós-Graduação em Matemática Pura e Aplicada,
Florianópolis, 2020.

   Inclui referências.

   1. Matemática Pura e Aplicada. 2. impressão 3D. 3.
rotações. 4. partes em suspensão. 5. otimização. I. Sacht,
Leonardo Koller. II. Universidade Federal de Santa
Catarina. Programa de Pós-Graduação em Matemática Pura e
Aplicada. III. Título.

Carlos Eduardo Leal de Castro

Optimal rotations for overhang reduction

O presente trabalho em nível de mestrado foi avaliado e aprovado por banca examinadora composta pelos seguintes membros:

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi julgado adequado para obtenção do título de mestre em Matemática Pura e Aplicada.

Coordenador do Programa de
Pós-Graduação

Prof. Leonardo Koller Sacht, Dr.
Orientador

Florianópolis, 18 de fevereiro de 2020

This work is dedicated to my mother, my wife, my brothers and Tobias.

ACKNOWLEDGMENTS

I would like to give special thanks to my wife Maíra Sevegnani and my mother Tania Leal for always supporting me in my choices and desires and my dad José Castro for inspiration, peace and illumination. Without you I would not have been able to get here. I would like to thank my brothers João Vitor Leal, Tina Castro, Marcos Castro, my little dog Tobias and my favorite couple Claudinei Sevegnani and Anderson Marcos.

I am very grateful to my advisor, Professor Leonardo Koller Sacht, for guiding me in my studies during the last years and for his friendship and kindness that helped me with problems inside and outside the university. Thank you for introducing me to this research area and your inner world. I am grateful to Professor Marianna Ravara Vago for always being willing to listen to me and help me with my problems. Every math department needs a professor like her. Also, I would like to thank Professor Jorge Massayuki Kondo, for his patience in teaching me how to use a 3D printer and letting us use it for our research and experiments.

I would like to thank all my colleagues for making my master studies more pleasant: Ben-hur Eidt,Victor Espinoza, Daniella Losso, Juan Carlos, Hêrnán Agamez, Everton Boos, Marina Geremia, Rafaela Filippozzi, Welington Grossmann, Mateus Oliveira, Gabriel Schafaschek, Gabriel Michels, Helena Günther, Jean Gengnagel, Sabrina Vigano, Carlos Caldeira, Leandro Correa, Maritza Camilli, Lucas Dodl, Talles Faria and Javier Alfonso. I am always grateful to my music and childhood friends Alexandre Nano, Leandro Pessoa, Danilo Caribé, João Amaro, Cláudio Colares, Frederico Viana, Gutto Leal, Thiago Carvalho, Lucas Azevedo Lobo, Teago Oliveira, Leonardo Brandão, Felipe Dieder, Igor Andrade and my home/team/passion: Bahia.

I would like to thank Professors Fermin Bazán, Eliezer Batista, Ivan Pontual, Vinícius Albani, Douglas Gonçalves, Paulo Carvalho, Antônio Leitão, Maicon Marques, Celso Dória, Melissa Weber, José Pinho, Martin Weilandt, Licio Bezerra, Carmem Gimenez for their classes and teachings. Also, I am grateful to the mathematics department staff Elisa Amaral, Érica Flores, Eduardo Krukoski, Eduardo Bastos and everyone who keeps this department fully functioning.

I am grateful to “Universidade Federal de Santa Catarina” (UFSC) for support and necessary structure for my studies, to “Fundação de Amparo à Pesquisa e Inovação do Estado de Santa Catarina” (FAPESC) and to “Coordenação de Aperfeiçoamento de Pessoal de Nível Superior” (CAPES) for the financial support.

RESUMO

A impressão 3D está ganhando espaço em diversas áreas pelo mundo, sendo utilizada na construção civil, criação de maquetes e modelos, materiais educacionais e para a saúde, como na impressão de partes do corpo para auxílio em cirurgias. Quando a impressora necessita imprimir uma superfície em 3D, algumas de suas partes podem estar suspensas no ar, necessitando de suporte para uma melhor impressão. Esses suportes são destacados do objeto e são descartados, gerando um desperdício de material e dinheiro. Este trabalho propõe uma nova abordagem para o problema de impressão de suportes em superfícies 3D baseado no campo de vetores normais à superfície, que busca de uma orientação global para a superfície, de modo que, após rotacionar a superfície na orientação encontrada, sua impressão gere o mínimo de suporte possível.

Palavras-chave: impressão 3D, normais, suporte, partes em suspensão, otimização.

RESUMO EXPANDIDO

Introdução

A impressão 3D vem ganhando espaço em diversas áreas ao redor do mundo. Por conta de sua versatilidade, ela pode ser usada tanto da indústria, para criação de protótipos e produção em série, quanto na arquitetura, no desenvolvimento de maquetes, entre outras áreas como construção civil, indústrias aeroespaciais, experimentos em física, instrumentos educacionais, bem como peças delicadas, como réplicas de órgãos humanos e próteses,

Quando imprimimos um sólido em uma impressora 3D, algumas de suas partes podem estar suspensas no ar e precisam de suporte para imprimir com mais perfeição. Esses suportes são destacados e descartados, levando a um desperdício de material, tempo de impressão e dinheiro.

Nós propomos um modelo matemático para o problema de suportes em impressão 3D, encontrando uma rotação global ótima para a superfície e minimizando, assim, o volume de suporte necessário para imprimir este objeto.

Objetivos

Esta dissertação de mestrado tem os seguintes objetivos:

- **Estudar e entender o problema de suportes para impressão 3D**
- **Analisar os conceitos matemáticos e referencias principais para o problema**
- **Modelagem do problema de suportes para impressão 3D**
- **Implementação e avaliação do método proposto**

Metodologia

Em nosso trabalho, nós analisamos a geometria da superfície e a conectamos com o problema de suportes em uma impressão 3D.

Após entendermos o problema de suportes para impressão 3D, foi apresentada todas técnicas necessárias para lidar com o problema. A bibliografia necessária para uma completa análise dos conceitos apresentados aqui estão disponíveis na seção de referências.

Com a revisão dos conceitos matemáticos, desenvolvemos um modelo matemático que trabalha o problema de suportes para reduzir a necessidade e a quantidade de suportes em uma impressão 3D.

Os conceitos matemáticos e o modelo desenvolvido nos permite criar estratégias para encontrar uma rotação global que reduz a quantidade de suporte e tempo de impressão. Nós, então, comparamos nossos resultados com outros métodos desenvolvidos anteriormente.

Resultados e discussão

Nós desenvolvemos um método que encontra uma rotação global ótima para as superfícies de modo que esse reposicionamento é impresso com menos volume de suporte e tempo de impressão menor, comparado com uma posição usual da superfície e métodos desenvolvidos anteriormente.

Considerações finais

A impressão 3D é um tema de pesquisa com grande potencial. Acreditamos que o trabalho desenvolvido nessa dissertação tem espaço para resultados ainda melhores, como desenvolvimento de todos os códigos em linguagem de programação robusta (C/C++), funções mais fáceis de manipular computacionalmente, o desenvolvimento de método de optimização global mais eficientes e diretos, além de estratégias que, combinadas com o método desenvolvido neste trabalho, possam gerar ainda mais economia de material e tempo de impressão.

Palavras-chave: impressão 3D, normais, suporte, partes em suspensão, otimização.

ABSTRACT

3D printing is gaining ground in many areas around the world, being used in construction, modeling and mockups, educational purposes and for health research, as well as in the printing of body parts for surgery assistance. When the printer needs to print a 3D surface, some of the parts may be suspended in the air, requiring support for better printing. These supports are detached from the object and are discarded, generating waste of material and money. This thesis proposes a new approach for this problem of 3D surface printing based on the normal field of surfaces, which seeks a global surface orientation, so that, after rotating the surface by the obtained orientation, its printing generates as less support as possible.

Keywords: 3D printing, normal field, support, overhang, optimization.

List of Figures

1.1.1 A plastic filament (a) is horizontally heated by a controlled nozzle (b), allowing the template (c) to be printed over the horizontal platform (e), that moves on the vertical direction. Some sloping parts of the template need supports (d) to be printed. . . . 17

1.1.2 This solid has parts that need support to print correctly. So the printer will build some structures to hold this parts while it is printed. . . . 18

2.1.1 The transformation T applied on vector w returns a vector $T(w)$ that also belongs to $\{u\}^{\perp}$ (a) and applied on a vector x returns a vector $T(x)$ that keeps the angle β between u (b). . . . 29

2.1.2 Rotating u around the x-axis by an angle θ_x (resulting in the vector u') also rotates its yz-projection p_{yz} by the same angle θ_x, taking it to the $z-axis$ (left). The length of projection of u into the yz-plane p_{yz} and its y and z coordinates form a right triangle (right). . . . 30

2.1.3 Rotating u' around the y-axis by an angle $-\theta_y$ results in the vector u'' (left). The length u', the x-coordinate of u and the length of yz-projection, that was rotated into the z-axis, form a right triangle (right). . . . 31

2.2.1 The vector x and its coordinates with $\{u, w, (u \times w)\} \subset \mathbb{R}^3$ basis before (left) and after (right) applied rotation matrix R. Notice that R preserves the coordinate $\alpha_1 u$ and rotate $\alpha_2 w$ and $\alpha_3(u \times w)$ around u by an angle θ. . . . 34

2.3.1 Parametric surface. . . . 41

2.3.2 Example of a parametric cone that opens along the z-axis and its domain. . . . 41

2.3.3 Examples of triangulated surfaces. All these surfaces are available on Libigl website (JACOBSON et. al., 2018). . . . 42

2.3.4 In this example, the *l-th* row of triangle mesh matrix $\mathscr{F}$ stores the row positions of the vertex v_i, v_j and v_k of the vertex matrix $\mathscr{F}$. This face, on the matrix, is represented by $\mathscr{F}_l = (i, j, k)$. . . . 42

2.3.5 Triangular base prism. . . . 43

2.3.6 Associated to this surface, we have the vertex matrix $\mathscr{V}$ and its triangle mesh matrix $\mathscr{F}$. We can associate one normal vector for each triangle on mesh, as defined in definition 2.3.3. . . . 44

3.0.1 Camel surface from Libigl (JACOBSON et al., 2018) in the initial position (left) and after $\theta = \frac{\pi}{3}$ z-axis rotation (right). The red area of both surfaces are those that need support to print. Notice that the support area does not change after this $z-axis$ rotation. 46

3.1.1 This two normals n_1 and n_2 of the bunny surface have different angles. The normal vector n_1 is within the angle range of the printer. The normal vector n_2 is in an overhanging part and will need a support to print correctly. 47

3.1.2 Lion surface from Libigl (JACOBSON et al., 2018) with 0% infill (left) and 30% infill(right). The parts of the surface such that normal angles are above the support angle threshold θ will also need some support to be printed, if the printer infill is set to 0%. 47

3.1.3 The equation (3.1.1) measure the angle between the normal $n(f) \in \mathscr{N}$ of a triangle $f \in \mathscr{F}$ on mesh and the vertical direction $e_3 = (0\ 0\ 1)^T$. 48

3.1.4 The surfaces *Camel, Horse, Bunny, Knight, Max, Cow and Arm* were chosen from Libigl (JACOBSON et al., 2018) library and *heart* (ADOREZOOEY, 2015) and *skull* (AELLIS43, 2014) are available on SketchFab (DENOYEL, PINSON, PASSET, 2012) website. 50

3.1.5 Horse surface at the initial embedding (a) and after rotation (b). Knight at the initial embedding (c) and after rotation (d). 50

3.1.6 This graph represents the value (in a gradient color) of sum of max in equation 3.1.2 for every discretized x-axis and y-axis rotation of *horse-quad.obj* surface. 51

3.1.7 Horse_quad.obj on the initial vertex position (left) and after rotation (center-left). The same surface on the initial position on the printing simulator (center-right) and after rotation (right). In yellow, the software indicates the solid that we want to print and in green the overhang supports. 51

3.1.8 Surface *decimated-max.obj* on the initial vertex position (top-left) and after rotation (top-center-left). The same surface on the initial position on the printing simulator (top-center-right) and after rotation (top-right). In yellow, the software indicates the solid that we want to print and in green the overhang supports. The bottom figure is a photo of the same surface after printed. On the bottom-left, the surface is at the initial vertex positions and on the bottom-right, the surface is positioned after rotation. 53

3.1.9 Surface *Bunny.obj* at the initial vertex position (a) and after rotation (b). In yellow, the software indicates the solid that we want to print and in green the overhang supports. With Slic3r sorftare we can observe that the objective function 3.1.3 returns an orientation of this object with more support than initial position. 53

3.2.1 As the normal vector $n(f)$ approaches the xy-plane, the value of its z-coordinate will approach zero. Note that the sequence in the image $\{n_{zk}\}_{k=\{1,2,3\}} \subset \mathbb{R}$ of z-coordinate values on z-axis is such that $n_{zk} \to 0$ if the sequence of rotation on $n(f)$ leads to the xy-plane. 55

3.2.2 The difference between the surface normal vector $n(f)$ (in wine) and its xy-Projection is a vector with zeros on x and y coordinates and z-coordinate normal vector value on itself. Observe that, the closer to the xy plane is the surface normal, the closer to zero is the difference norm. Consequently, this triangle mesh face is as vertical as possible. 56

3.2.3 Surface *cow.obj* at the initial vertex position (a) and after rotation (b) minimizing the objective function 3.1.3. Even getting less overhanging parts, this objective function does not consider the surface area and height of support, which, in this case, causes the amount of support to be larger than desired. 57

3.3.1 Comparison of layered surface parts with its model surface, leading to a staircase effect. Zooming in and considering that the difference between layers forms a triangle, we can analyse the height h of this triangle, the thickness z_t of the layer and the angle θ between them to minimize the staircase effect. We can see in the figure on the left that the more vertical the surface is, the less it suffers from a staircase effect. This figure was based on the figures in (ALEXANDER; ALLEN; DUTTA, 1998). 58

3.3.2 In this vertical cut of a 3D object, we can see that the surface can be divided in three parts. *Unsupported faces*: parts of object that do not need external support to print; *supported faces*: parts of object that need external support underneath them to print; and *support faces*: parts of object that are touched from above by the support structure. This figure was created based on figures in (ALEXANDER; ALLEN; DUTTA, 1998). 60

3.4.1 2D graph of function value variation in equation 3.2.4 applied to *bunny.obj* Libigl surface. 61
3.4.2 Surfaces at the initial vertex position (top) and after applied a global rotation that minimizes equation 3.2.4 (bottom). *Camel, Cow and Max (human head)* are surfaces available on Libigl (JACOBSON et al., 2018) and *heart* (ADOREZOOEY, 2015) and *skull* (AELLIS43, 2014) are Sketch Fab surfaces. 62
3.4.3 Libigl surface *bunny.obj* on the initial position (a) and after rotation (b). On 3D print sorftware (Sklic3r), the same surface on the initial position (c) and after rotation (d). Also, in figures (c) and (d), the software indicates the surface in yellow and the overhang supports in green. 62
3.4.4 Libigl surface *cow.obj* after rotation using our method (a) and Alexander's method (b). As we can see, these results using our method generate less support for overhanging parts of surface then Alexander's method. 64
3.4.5 *decimated-knight.obj* surface after rotation using our method (a) and Alexander's method (b). As we can be seen, the result using our method generates more overhanging parts area (in brown) than Alexander's method. 64
4.2.1 Objective function 4.0.3 3D graph applied on Knight Libigl (JACOBSON et al., 2018) surface. 81
4.2.2 Knight surface from Libigl (JACOBSON et al., 2018) after global rotation on tests performed on section 3.4 (left) and after finding a local minimum using *fmincon* Matlab function with a chosen initial point (right). The teal surface on the left would be printed in 136 minutes with 3g of material loss (support that would be wasted) and the pink surface on the right would be printed in 149 minutes with 5g of material loss. 81
4.2.3 Dividing the angle ranges on the restrictions of equation 4.0.3 and taking the central point of each set, we got multiple start point of our algorithm. 82
4.2.4 Surfaces on initial position (top) and after optimal rotation (bottom). *Knight, Horse, Bunny and Arm* are surfaces available on Libigl (JACOBSON et al., 2018). 83
4.2.5 Graph of randomly chosen starting points (R.C.S.P.) strategy applied on Knight Libigl surface, with 100 starting points. Each black spot on graphic is an initial point that the algorithm choose randomly as the start point to minimization problem. 83

List of Tables

3.1.1 All these surfaces present a smaller sum in function 3.1.2 with an x-axis and y-axis angle rotation. The surfaces *Camel, Horse, Bunny, Knight, Max, Cow and Arm* were chosen from Libigl (JACOBSON et al., 2018) library and *heart* (ADOREZOOEY, 2015) and *skull* (AELLIS43, 2014) have a Creative Commons Attribution and are freely available surfaces on the SketchFab website. All these surfaces can be seen in figure 3.1.4. 49

3.1.2 All these surfaces present a smaller printint time, weight and length of material after applied a global rotation that minimizes overhang parts. The surfaces *Camel, Horse, Bunny, Knight, Max, Cow and Arm* were chosen from Libigl (JACOBSON et al., 2018) library and *heart* (ADOREZOOEY, 2015) and *skull* (AELLIS43, 2014) are from Sketch Fab website. 52

3.1.3 All these surfaces present a higher printing time after applied a global rotation that minimizes overhang parts, according to the objective function (3.1.3). On first row of this table, *Arm* is the only surface tested that is reducing the amount of support, but does not reduce printing time. These surfaces were chosen from Libigl (JACOBSON et al., 2018). 54

3.4.1 All tested surfaces present a smaller printint time, weight and length of material after applied a global rotation that minimizes equation 3.2.4. The surfaces *Arm, Camel, Horse, Bunny, Knight, Max and Cow* were chosen from the Libigl (JACOBSON et al., 2018) library and *heart* (ADOREZOOEY, 2015) and *skull* (AELLIS43, 2014) are Sketch Fab surfaces. 63

4.1.1 Execution time of the algorithm that finds the smallest value of the function 3.2.4, for each surface tested. $|F|$ is the number of triangular faces of each mesh. 69

4.2.1 This table presents the run time of both strategies to seek the global minimum point and the tests presented in section 3.4. On multiple constraint division (M.C.D.) strategy, we divide the angle intervals into smaller with $h = \frac{\pi}{4}$ spacing on both axes, having 32 different initial points. On randomly chosen starting point (R.C.S.P.) strategy, we collect 32 random points as start point of our algorithm. On tests presented in section 3.4 we run 65341 angles combination and select the one with the lower objective function value. 84

Contents

1 Introduction **17**
1.1 3D Printing support - overview . . . 17
1.2 Related work . . . 18
1.3 Goals and contributions . . . 19
1.4 Structure of the thesis . . . 19

2 Preliminary concepts: rigid transformations and $SO(3)$ space **21**
2.1 Linear and rigid transformations . . . 21
2.2 The rotation space $SO(3)$ and generalized Euler sequences . . . 32
2.3 Spatial surfaces and their representations . . . 40
2.3.1 Triangle mesh representation . . . 41

3 Objective function **45**
3.1 Surface normal angle analysis . . . 47
3.1.1 Initial results . . . 49
3.1.2 Problems in paradise . . . 52
3.2 Surface normal "Least squares" . . . 55
3.2.1 Support volume . . . 57
3.3 Alexander's method . . . 58
3.4 Tests and method comparison . . . 61

4 Optimization method **66**
4.1 The Matlab function *fmincon* and its optimization theory . . . 69
4.1.1 Finite-difference approximation . . . 70
4.1.2 Local and global minima and descent directions . . . 71
4.1.3 Barrier penalty method . . . 73
4.1.4 The BFGS method . . . 75
4.2 Optimization Results . . . 81

5 Conclusion **84**
5.1 Review . . . 84
5.2 Future work . . . 85

References **87**

A Source codes **90**
A.1 Main functions . . . 90
A.1.1 print3Dopt_grid.m . . . 90
A.1.2 print3Dopt_rand.m . . . 92
A.1.3 print3Dopt.m . . . 95
A.2 Auxiliary functions . . . 97
A.2.1 rotatexy.m . . . 97
A.2.2 normalsurf.m . . . 99
A.2.3 areatsurf.m . . . 99

1 Introduction

1.1 3D Printing support - overview

The 3D printing process is gaining space in many areas around the world. Its versatility can be used to print artistic objects, architectural mock ups, civil constructions, aerospace models, parts of physics experiments, educational instruments, as well as delicate objects as prostheses and real representations of human organs.

When we need to print some 3D solid in a 3D printer, some parts of this solid may be suspended in the air and need support for a better print. However, these supports are detached from the final surface and will not be reused, leading to a waste of material, time and money.

Currently, the most affordable 3D printers use a printing technology known as *fused deposition modeling* (FDM). This printing process, as shown in figure 1.1.1, melts a type of polymer, that solidifies on the moving platform, or on the surface itself, printing the desired solid with cross-sectional layers, from bottom to top.

Some parts of the solid need support because they are suspended in the air, which can damage the object or generate wrong prints. These parts are called *overhangs*.

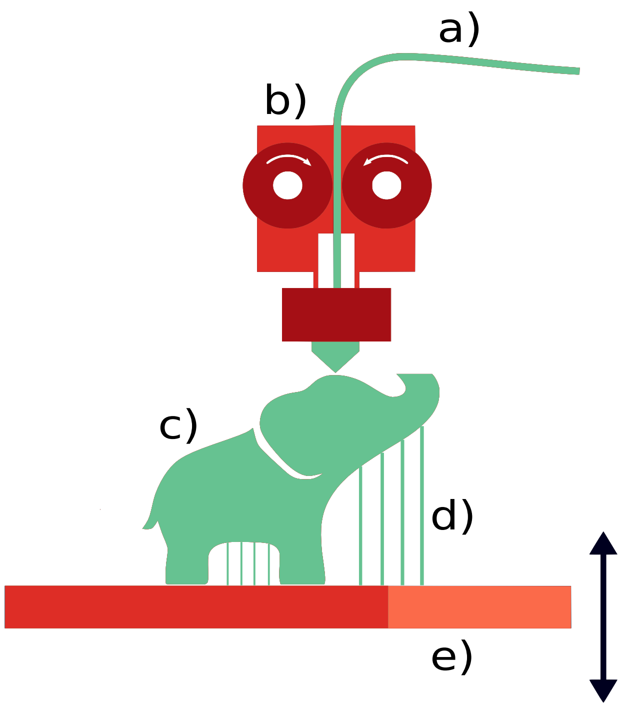

Figure 1.1.1: A plastic filament (a) is horizontally heated by a controlled nozzle (b), allowing the template (c) to be printed over the horizontal platform (e), that moves on the vertical direction. Some sloping parts of the template need supports (d) to be printed.

To avoid overhangs, the 3D printers print columns to support the parts of this solid that have no material underneath them, otherwise the material would be melted and fall on the platform. This extra material must be removed from the solid, leading to a waste of material, time and money. As we can see represented in figure 1.1.2, the solid has overhanging parts that

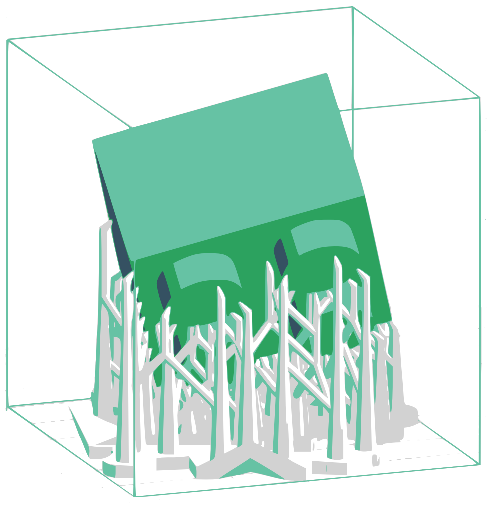

Figure 1.1.2: This solid has parts that need support to print correctly. So the printer will build some structures to hold this parts while it is printed.

need support to print correctly.

However, some overhangs are tolerable. Each printer has a limiting angle to tolerate these overhangs. Given an angle θ, the printer only prints an overhang support if the part of the solid that will be printed forms an angle with the horizontal plane less than θ.

We propose a formulation for the overhang problem based on the normal field of a surface and an optimization method to find a global rotation of a surface that minimizes overhanging parts that cannot be printed without supports. This global rotation does not change the surface since the printed object can be derotated in the real world after printing.

1.2 Related work

During the last years, the problem of reducing overhangs from a surface has been extensively studied and some partial solutions have been proposed.

Some of these solutions (HU et al, 2014), (YAO et al, 2015), (KARASIK; FATTAL; WERMAN, 2019) partition the surface into smaller parts such that each subpart has little or no overhangs. These methods introduce junction lines between the different parts that are clearly visible after printing. Our method does not suffer from this limitation since we search for a global solution for the whole input surface.

In (VANEK; GALICIA; BENES, 2014) the authors propose to change the geometry of the supports to minimize the waste of material. We, instead, propose a general solution that is optimal, in a sense, for most supports generated by common 3D printing software. Other approach to the problem was proposed by (DAI et al, 2014), where the authors design new printing hardware to eliminate overhangs. Our current solution is purely based on software.

Other methods (ZHANG et al, 2015), (MARTÍNEZ et al, 2015), (WANG; ZANNI; KOBBELT, 2016), propose altering the surface to improve the final quality of the printed piece. These improvements can lead to overhang reduction. On the other hand, we directly formulate the overhanging problem and find a specific solution for it.

1.3 Goals and contributions

This thesis has the following goals:

- **Study and understand the 3D printing overhang problem**: In our work, we analyze a given surface and connect this analysis with the overhang problem of 3D printing, which is necessary to understand the difficulties and challenges of the problem.
- **Analyze the mathematical concepts and main references of the problem**: After understanding the overhang problem, we discuss all necessary techniques related to this problem. The bibliography needed for a complete problem analysis is explained in this thesis and additional texts are available in the references section.
- **Model the 3D printing overhang problem**: With the mathematical concepts available, we develop a mathematical model that treats the overhang problem to reduce the need and amount of support for 3D printing.
- **Test and evaluate the proposed method**: The mathematical concepts and the model developed allow us to create a strategy to find the global rotation that reduces the amount of support. We then compare our results with other methods.

1.4 Structure of the thesis

In chapter 2, we present in section 2.1 some initial concepts of orthogonal and rigid transformations.These concepts help us to understand an axial rotation and the rotation space $SO(3)$ that we detail in section 2.2, presenting some results about the representation of elements in this space with well known matrices. In the last part of this chapter, we introduce in section 2.3 the mathematical representation of 3D surfaces and some of their useful elements.

Chapter 3 presents the development of the objective function that we are going to work in this thesis. In section 3.1, using the normal field of a surface we develop an objective function that takes under consideration the angle of each normal vector on the surface. After we identify some problems with the previous representation of the objective function, in section 3.2 we develop a new function that considers the length of z-coordinate on each normal angle of the surface as a way of measuring the need for 3D printing support of a surface. In section 3.3, we study the method developed by (ALEXANDER, ALLEN, DUTTA, 1998), that minimizes the staircase effect in the manufacture of objects, to maximize surface accuracy. Finally, in section 3.4 we performed some tests to compare our results with initial positions of surfaces and the positions found after applying Alexander's method.

Chapter 4 begins with a reformulation of objective function developed in the previous chapter to better understand how to apply an optimization method to solve the problem. Section 4.1 presents the concepts behind the Matlab function *fmincon* and some results that guarantee correctness of the function and section 4.2 shows the results of solving the minimization problem using this function with different initial point strategies.

Finally, chapter 5 has a quick review of this thesis in section 5.1 and the future work in section 5.2 that can be developed from what was discussed here.

2 Preliminary concepts: rigid transformations and $SO(3)$ space

All objects we are working with in this thesis are 3D surfaces. So, we need to define and remember some properties of these elements, for a better analysis of all steps that we will take here. For more details of what we approach in this section, please refer to (BOTSCH et al., 2010), (CARMO, 2006) and (GOMES; VELHO; SOUSA, 2012).

As we will see during this work, some operations that we will apply to surfaces should not change their shape. So, we need to introduce here the important concepts of operators that preserve distances and angles. In addition, all vector norms used in this work will be the norm induced by the usual inner-product of $\mathbb{R}^n$.

2.1 Linear and rigid transformations

Definition 2.1.1. *A* linear operator *T is an operator such that*

1. *the domain $\mathscr{D}(T)$ of T is a vector space and the range $\mathscr{R}(T)$ lies on a vector space over the same field and*
2. *$\forall x, y \in \mathscr{D}(T)$ and scalars α, β,*

$$T(\alpha x + \beta y) = \alpha T(x) + \beta T(y).$$

Example 2.1.1. ***k-Differentiation:*** *Let $\mathscr{C}^k[a,b]$ ($a, b \in \mathbb{R}$) be the vector space of functions that are at least k times differentiable and define the operator T_k by setting*

$$T_k(x(t)) = x^{(k)}(t), \quad \forall t \in [a,b] \subset \mathbb{R}.$$

For a better background of the operations that we will approach in this work, we need to understand the concept of bounded operators. Later, we will talk specifically about a special case of bounded operator: rigid transformations. The main definition and properties of normed spaces and bounded operators can be found in (KREYSZIG, 1978).

Before we approach rigid transformations, we must talk about a special type of bounded operators: *orthogonal transformations*. In order to find a better position of a surface for 3D printing support reduction, we will be looking for an orthogonal transformations that does this.

Definition 2.1.2. *In an inner-product space X, two vectors $x, y \in X$ are said to be orthogonal if $\langle x, y \rangle = 0$.*

It is important to remember that, using the definition of inner-product in a real vector space X, we can measure the smallest angle between two vectors $x, y \in X$, $||x|| \neq 0$ and $||y|| \neq 0$. This angle is defined to be the number $\theta \in [0, \pi]$ such that

$$\cos(\theta) = \frac{\langle x, y \rangle}{||x|| \, ||y||}.$$

Definition 2.1.3. *Given a real inner-product X, the set $B = \{u_1, u_2, \dots, u_n\}, u_i \in X, i = 1, \dots, n$, is called an orthonormal set if*

$$\langle u_i, u_j \rangle = \begin{cases} 1, \text{ when } i = j \\ 0, \text{ when } i \neq j \end{cases}.$$

Definition 2.1.4. *An $n \times n$ matrix is orthogonal if its columns are orthonormal.*

Proposition 2.1.1. *If $A \in \mathbb{R}^{n \times n}$ is an orthogonal matrix, then $A^T A = I_{n \times n}$.*

Proof. Since A is a orthogonal matrix, by definition, its columns are orthonormal. So, if

$$A = [a_1 \; a_2 \; \dots \; a_n]$$

with $a_i \in \mathbb{R}^n$ each column of A, for $i = 1, \dots n$, we have

$$A^T A = \begin{bmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_n^T \end{bmatrix} [a_1 \; a_2 \; \dots \; a_n] = \begin{bmatrix} a_1^T a_1 & a_1^T a_2 & \cdots & a_1^T a_n \\ a_2^T a_1 & a_2^T a_2 & \cdots & a_2^T a_n \\ \vdots & \vdots & \vdots & \vdots \\ a_n^T a_1 & a_n^T a_2 & \cdots & a_n^T a_n \end{bmatrix}.$$

Since $a_i^T a_j = \begin{cases} 1, \text{ if } i = j \\ 0, \text{ if } i \neq j \end{cases}$, we then have

$$A^T A = \begin{bmatrix} a_1^T a_1 & a_1^T a_2 & \cdots & a_1^T a_n \\ a_2^T a_1 & a_2^T a_2 & \cdots & a_2^T a_n \\ \vdots & \vdots & \vdots & \vdots \\ a_n^T a_1 & a_n^T a_2 & \cdots & a_n^T a_n \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} = I_{n\times n}.$$

■

Definition 2.1.5. *A linear transformation $T : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ is called orthogonal if it preserves norms of vectors, i.e.,*

$$||T(x)|| = ||x||, \quad \forall x \in \mathbb{R}^n.$$

Proposition 2.1.2. *Let $T : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ be an orthogonal transformation. Then $\langle T(x), T(y)\rangle = \langle x, y\rangle$, for any $x, y \in \mathbb{R}^n$.*

Proof. Since T is an orthogonal transformation, for any $x, y \in \mathbb{R}^n$ we have $||T(x+y)||^2 = ||x+y||^2$. Expanding this equality, we have:

$$||T(x)||^2 + 2\langle T(x), T(y)\rangle + ||T(y)||^2 = ||x||^2 + 2\langle x, y\rangle + ||y||^2.$$

Using the fact that T is orthogonal, $||T(x)|| = ||x||, \forall x \in \mathbb{R}^n$, we have:

$$||T(x)||^2 + 2\langle T(x), T(y)\rangle + ||T(y)||^2 = ||x||^2 + 2\langle T(x), T(y)\rangle + ||y||^2 = ||x||^2 + 2\langle x, y\rangle + ||y||^2.$$

Then, subtracting $||x||^2$ and $||y||^2$ from both sides of equality and dividing by 2, we obtain

$$\langle T(x), T(y)\rangle = \langle x, y\rangle.$$

■

Theorem 2.1.1. *Let $T : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ be a linear transformation and A a matrix such that $T(x) = Ax$, $\forall x \in \mathbb{R}^n$. Then T is orhogonal if and only if A is orthogonal.*

Proof. ($\Rightarrow$) Suppose that $T : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ is an orthogonal transformation. Then, we have that $\|T(x)\| = \|x\|, \forall x \in \mathbb{R}^n$.

Since $T(x) = Ax$, for a matrix $A \in \mathbb{R}^{n\times n}$ and for all $x \in \mathbb{R}^n$, for an orthonormal basis $\{e_1, e_2, \ldots, e_n\}$ of $\mathbb{R}^n$, $\|T(e_i)\| = \|e_i\|$ and the columns of A are $[T(e_1)\ T(e_2)\ \ldots\ T(e_n)]$.

Notice that:

$$\langle T(e_i), T(e_i)\rangle = \|T(e_i)\|^2 = \|e_i\|^2 = 1.$$

Besides that, using proposition 2.1.2 for $i \neq j$,

$$\langle T(e_i), T(e_j)\rangle = \langle e_i, e_j\rangle = 0.$$

So, the matrix A is orthogonal.

($\Leftarrow$) Suppose that the matrix $A \in \mathbb{R}^{n\times n}$ asscoaited to the linear transformation T is orthogonal. So, we have that the columns of A are orthonormal and:

$$|T(x)\|^2 = \|Ax\|^2 = (Ax)^T Ax = x^T \underbrace{A^T A}_{I_{n\times n}} x = x^T x = \|x\|^2.$$

Therefore, $\|T(x)\| = \|x\|$ and T is an orthogonal transformation. ■

Next example has a hint to what we are seeking for at this beginning of work. To be able to find a better position to reduce support in 3D printing, all we need is to find an embedding of the surface in space that has less overhanging parts that require support. To do this, we need to consider some *rotation matrices in* $\mathbb{R}^3$.

Example 2.1.2.
Rotation matrices in $\mathbb{R}^3$*: A vector $u \in \mathbb{R}^3$ can be rotated couterclockwise through an angle θ around a coordinate axis by means of multiplication $R \cdot u \in \mathbb{R}^3$ in which $R \in \mathbb{R}^{n\times n}$ is an appropriate orthogonal matrix as described below.*

- ***Rotation around the x-Axis***

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{pmatrix}$$

- ***Rotation around the y-Axis***

$$R_y = \begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{pmatrix}$$

- ***Rotation around the z-Axis***

$$R_z = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Observe that, if $u = (u_x \ \ u_y \ \ u_z)^T$:

$$\begin{aligned}
\|R_x u\| &= \left\| \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix} \right\| = \left\| \begin{pmatrix} u_x \\ \cos(\theta)u_y - \sin(\theta)u_z \\ \sin(\theta)u_y + \cos(\theta)u_z \end{pmatrix} \right\| \\
&= (u_x^2 + (\cos(\theta)u_y - \sin(\theta)u_z)^2 + (\sin(\theta)u_y + \cos(\theta)u_z)^2)^{\frac{1}{2}} \\
&= (u_x^2 + \cos^2(\theta)u_y^2 - 2\cos(\theta)\sin(\theta)u_y u_z + \sin^2(\theta)u_z^2 + \sin^2(\theta)u_y^2 + \\
&\quad +2\sin(\theta)\cos(\theta)u_y u_z + \cos^2(\theta)u_z^2)^{\frac{1}{2}} \\
&= (u_x^2 + \underbrace{(\cos^2(\theta) + \sin^2(\theta))}_{=1} u_y^2 + \underbrace{(\cos^2(\theta) + \sin^2(\theta))}_{=1} u_z^2)^{\frac{1}{2}} \\
&= (u_x^2 + u_y^2 + u_z^2)^{\frac{1}{2}} = \|u\|
\end{aligned}$$

Therefore $\|R_x u\| = \|u\|$, R_x is an orthogonal matrix and, by theorem 2.1.1, R_x can be the matrix associated with an orthogonal transformation. Similarly, we can prove that R_y and R_z are orthogonal matrices and also can be the matrices associated with orthogonal transformations.

Lemma 2.1.1. *Let $n \in \mathbb{N}$ and $R_1, R_2, \ldots, R_n \in \mathbb{R}^{n\times n}$. If $R_1, R_2, \ldots R_n$ are orthogonal then $R = R_n R_{n-1} \cdots R_2 R_1$ is orthogonal.*

Proof. To prove that $R = R_n R_{n-1} \cdots R_2 R_1$ is orthogonal we only need to prove that $R^T R = I_{n\times n}$. Since $R_i, i = 1, \ldots n$ are orthogonal, we have that $R_i^T R_i = I_{n\times n}$, with $i = 1, \ldots, n$.

Then,

$$\begin{aligned}
R^T R &= (R_n R_{n-1} \cdots R_2 R_1)^T (R_n R_{n-1} \cdots R_2 R_1) = R_1^T R_2^T \cdots R_{n-1}^T \underbrace{R_n^T R_n}_{I_{n\times n}} R_{n-1} \cdots R_2 R_1 \\
&= R_1^T R_2^T \cdots \underbrace{R_{n-1}^T R_{n-1}}_{I_{n\times n}} \cdots R_2 R_1 \\
&\;\;\vdots \\
&= R_1^T R_1 = I_{n\times n}
\end{aligned}$$

■

As mentioned before, we need to find a transformation that changes the position of a surface but does not change its shape. So, we will introduce now the concept of rigid transformations. These functions only change the positions of objects, leaving their shape and size unchanged.

Definition 2.1.6. *Consider a normed space A , $T : A \longrightarrow A$ a rigid transformation and $x, y \in A$. A transformation is called* rigid *if and only if it preserves*

1. *Distances between points, i.e,*

$$\|T(x-y)\| = \|x-y\|;$$

2. *Angles between vectors, i.e.,*

$$\langle T(x), T(y)\rangle = \langle x, y\rangle.$$

Also, a ridid transformation $T : \mathbb{R}^n \longrightarrow \mathbb{R}^n$, where $T(x) = Ax$ with $A \in \mathbb{R}^{n\times n}$, is called orientation-preserving *if $det(A) > 0$.*

A simple lemma is required to prove a theorem that has an important role in this work.

Lemma 2.1.2 (Eigenvalue of a matrix)**.** *Let $A = (a_{ij}) \in \mathbb{R}^{n\times n}$. Then A has at least one eigenvalue.*

Proof. Given $\lambda \in \mathbb{C}, x \in \mathbb{R}^n, x \neq 0$ and consider the equation

$$Ax = \lambda x. \tag{2.1.1}$$

Equation 2.1.1 can be written as

$$Ax - \lambda x = 0 \Rightarrow (A - \lambda I_{n\times n})x = 0. \tag{2.1.2}$$

If $\det(A - \lambda I_{n\times n}) \neq 0$, the system in equation 2.1.2 would have no solution because $A - \lambda I_{n\times n}$ would be singular and $(A - \lambda I_{n\times n})x = 0 \Leftrightarrow x = 0$.

So, $\det(A - \lambda I_{n\times n})$ must be zero in order to 2.1.2 have a solution $x \neq 0$. This leads to the characteristic equation

$$\det(A - \lambda I_{n\times n}) = \begin{vmatrix} a_{11} - \lambda & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} - \lambda & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} - \lambda \end{vmatrix} = 0,$$

and gives us a polynomial in λ of degree n.

Therefore, by the fundamental theorem of algebra, there exists at least one $\lambda \in \mathbb{C}$ eigenvalue. ■

Observation 2.1.1. *Note that, when the matrix A belongs to $\mathbb{R}^{3\times 3}$, it has at least one real eigenvalue.*

Indeed, when we set its characteristic equation

$$\det(A - \lambda I_{3\times 3}) = \begin{vmatrix} a_{11} - \lambda & a_{12} & a_{13} \\ a_{21} & a_{22} - \lambda & a_{23} \\ a_{31} & a_{32} & a_{33} - \lambda \end{vmatrix} = 0,$$

we have the characteristic polynomial of matrix A, $p(\lambda) = \det(A - \lambda I_{3\times 3})$ and it is a degree 3 *polynomial. When we expand the determinant, we have that the leading term of the characteristic polynomial of A is $-\lambda^3$, i.e.,*

$$p(\lambda) = -\lambda^3 + lowerterms.$$

But, note that, if we increase λ to a sufficiently big value, and decrease λ to a sufficiently small value, or yet,

$$\lim_{\lambda \to \infty} p(\lambda) = -\infty,$$

$$\lim_{\lambda \to -\infty} p(\lambda) = \infty,$$

the intermediate value theorem guarantee that $\exists \lambda^* \in \mathbb{R}$ such that $p(\lambda^*) = 0$. Therefore, $A \in \mathbb{R}^{3\times 3}$ has at least one real eigenvalue.

We now generalize the idea of rotation by an angle around an arbitrary axis:

Definition 2.1.7. *Given any vector $u \in \mathbb{R}^3$, $\|u\| = 1$ and an angle $\theta \in (-\pi, \pi]$. An* axial rotation *is a rigid orientation-preserving transformation $T : \mathbb{R}^3 \longrightarrow \mathbb{R}^3$ that rotates any vector $x \in \mathbb{R}^3, x \neq (0\ 0\ 0)^T$ around u by an angle θ.*

Theorem 2.1.2. *Every orientation-preserving linear rigid transformation in 3D is an axial rotation.*

Proof. Let $T : \mathbb{R}^3 \longrightarrow \mathbb{R}^3$ be an orientation-preserving linear rigid transformation, and then

$$\|T(x)\| = \|x\|, \forall x \in \mathbb{R}^3,$$

and consider $\lambda \in \mathbb{R}$ a real eigenvalue of T associated to the eigenvector $u \in \mathbb{R}^3, \|u\| = 1$. Since λ is an eigenvalue of T, we have $T(u) = \lambda u$ and applying the norm to both sides of the equality,

$$\|T(u)\| = \|\lambda u\| = |\lambda|\|u\|.$$

Since $\|T(x)\| = \|x\|, \forall x \in \mathbb{R}^3$, it also applies to u and matching both equations, we have

$$\|u\| = |\lambda|\|u\| \Rightarrow |\lambda| = 1 \Rightarrow \lambda = \pm 1 \text{ (since } \lambda \in \mathbb{R}).$$

If $\lambda = 1$, then $T(u) = u$ and, using proposition 2.1.2, we have that T preserves angles (and inner product). So, for any $w \in \mathbb{R}^3 \cap \{u\}^\perp$ (e.g. any vector in plane of all vectors in $\mathbb{R}^3$

that $w \perp u$),

$$\langle T(w), T(u)\rangle = \langle w, u\rangle = 0.$$

Also, given any vector $x \in \mathbb{R}^3$, we have that

$$\langle u, x\rangle = \|x\|\|u\|\cos(\beta) = \|x\|\cos(\beta),$$

with β the smallest angle between u and x. Note that, as $T(u) = u$ and T preserves inner product, we have that

$$\langle u, x\rangle = \langle T(u), T(x)\rangle = \langle u, T(x)\rangle.$$

Then $\langle u, x\rangle = \langle u, T(x)\rangle$.

It means that T keeps vectors in the orthogonal plane of u and preserves the angle between x and u, as we can see on figure 2.1.1. Therefore, as any vector $x \in \mathbb{R}^3, x \neq 0$, can be

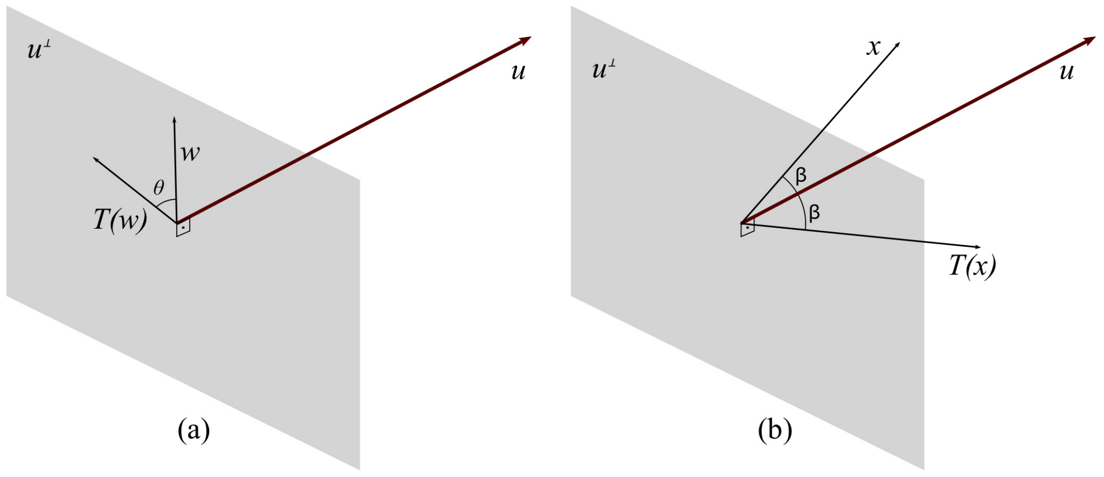

Figure 2.1.1: The transformation T applied on vector w returns a vector $T(w)$ that also belongs to $\{u\}^\perp$ (a) and applied on a vector x returns a vector $T(x)$ that keeps the angle β between u (b).

written as a linear combination of u and its projection in $\{u\}^\perp$ and the transformation T rotates any vector in $\{u\}^\perp$ by an angle θ, we have that T must be a rotation around the u-axis by an angle θ.

if $\lambda = -1$, $T(u) = -u$ and then using inner product preservation, we have that T takes the $\mathbb{R}^3 \cap \{u\}^\perp$ plane into itself but reverses orientation on this plane. Then, as T is an orientation-preserving transformation, we have that T can not have $\lambda = -1$ as its eigenvalue.

Analyzing the cases with $\lambda = \pm 1$, we have that T must be a rotation around u-axis and given the arbitrariness of the choice of T, we conclude that all orientation-preserving linear rigid transformation in 3D is an axial rotation. ■

We will show how to construct the generic axial rotation matrix by an angle θ around an axis u. This construction is important and will help us to understand the space $SO(3)$ of all rotations in $\mathbb{R}^3$.

Consider any vector $u \in \mathbb{R}^3, u = (u_x\ u_y\ u_z)^T$, with $\|u\| = 1$. The first step to find the general rotation $R_u(\theta)$ is to rotate u around $x-axis$ by an angle $\theta_x \in \mathbb{R}$ so that u is mapped to the $xz-plane$, resulting on the vector u' (figure 2.1.2). Note that the projection p_{yz} of u on the $yz-plane$ will also be rotated by the angle θ_x and is mapped to the $z-axis$, as we can see in figure 2.1.2. We can also see that the length of this projection is

$$\|p_{yz}\| = \sqrt{u_y^2 + u_z^2}.$$

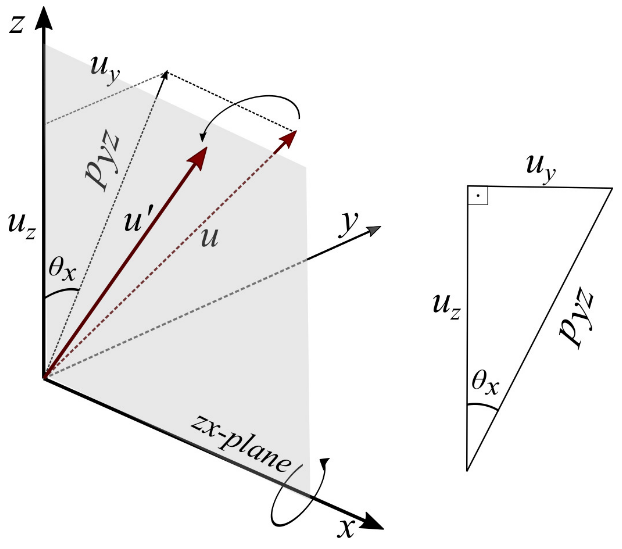

Figure 2.1.2: Rotating u around the x-axis by an angle θ_x (resulting in the vector u') also rotates its yz-projection p_{yz} by the same angle θ_x, taking it to the $z-axis$ (left). The length of projection of u into the yz-plane p_{yz} and its y and z coordinates form a right triangle (right).

So, since

$$\sin(\theta_x) = \frac{u_y}{p_{yz}} \quad \text{and} \quad \cos(\theta_x) = \frac{u_z}{p_{yz}},$$

we have the rotation matrix around the $x-axis$:

$$R_x(\theta_x) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{u_z}{p_{yz}} & -\frac{u_y}{p_{yz}} \\ 0 & \frac{u_y}{p_{yz}} & \frac{u_z}{p_{yz}} \end{pmatrix}.$$

Now, we will rotate u' around the $y-axis$ by an angle $-\theta_y \in \mathbb{R}$ to map its to the $z-axis$, as shown in figure 2.1.3. Remember that $||u'|| = 1$ and as we initially rotate u around $x-axis$ and map it to the $xz-plane$, we have that the new rotation angle satisfies

$$\sin(\theta_y) = \frac{u_x}{||u||} = u_x \quad \text{and} \quad \cos(\theta_y) = \frac{p_{yz}}{||u||} = p_{yz}$$

and the new rotation matrix around $y-axis$ is

$$R_y(-\theta_y) = \begin{pmatrix} p_{yz} & 0 & -u_x \\ 0 & 1 & 0 \\ u_x & 0 & p_{yz} \end{pmatrix}.$$

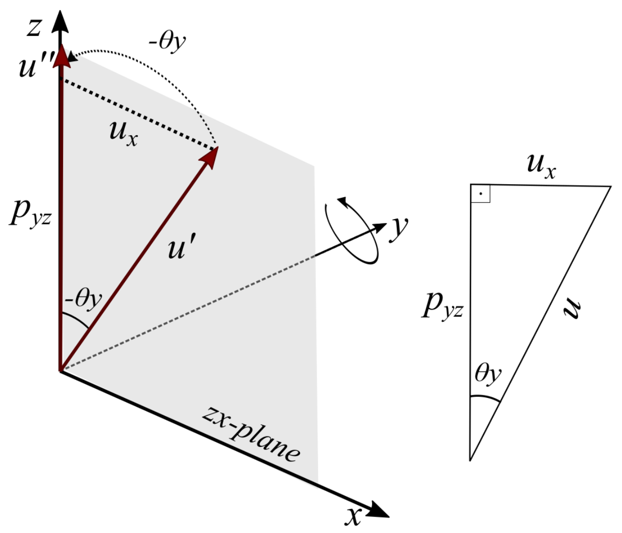

Figure 2.1.3: Rotating u' around the y-axis by an angle $-\theta_y$ results in the vector u'' (left). The length u', the x-coordinate of u and the length of yz-projection, that was rotated into the z-axis, form a right triangle (right).

Now, with u on the $z-axis$ (represented by u'' in figure 2.1.3), we can rotate by the initial angle θ around $z-axis$ and reverse all steps that we did before, constructing the rotation matrix around an arbitrary axis as

$$R_u(\theta) = R_x^{-1}(\theta_x)R_y^{-1}(-\theta_y)R_z(\theta)R_y(-\theta_y)R_x(\theta_x),$$

and getting the result:

$$R_u(\theta) = \begin{pmatrix} u_x^2(1-\cos(\theta))+\cos(\theta) & u_xu_y(1-\cos(\theta))+u_z\sin(\theta) & u_xu_z(1-\cos(\theta))-u_y\sin(\theta) \\ u_xu_y(1-\cos(\theta))-u_z\sin(\theta) & u_y^2(1-\cos(\theta))+\cos(\theta) & u_yu_z(1-\cos(\theta))+u_x\sin(\theta) \\ u_xu_z(1-\cos(\theta))+u_y\sin(\theta) & u_yu_z(1-\cos(\theta))-u_x\sin(\theta) & u_z^2(1-\cos(\theta))+\cos(\theta) \end{pmatrix}. \tag{2.1.3}$$

Observation 2.1.2. *Notice that we can write the equation 2.1.3 as*

$$\begin{aligned} R_u(\theta) &= \begin{pmatrix} u_x^2(1-\cos(\theta))+\cos(\theta) & u_xu_y(1-\cos(\theta))+u_z\sin(\theta) & u_xu_z(1-\cos(\theta))-u_y\sin(\theta) \\ u_xu_y(1-\cos(\theta))-u_z\sin(\theta) & u_y^2(1-\cos(\theta))+\cos(\theta) & u_yu_z(1-\cos(\theta))+u_x\sin(\theta) \\ u_xu_z(1-\cos(\theta))+u_y\sin(\theta) & u_yu_z(1-\cos(\theta))-u_x\sin(\theta) & u_z^2(1-\cos(\theta))+\cos(\theta) \end{pmatrix} \\ &= \underbrace{\begin{pmatrix} \cos(\theta) & 0 & 0 \\ 0 & \cos(\theta) & 0 \\ 0 & 0 & \cos(\theta) \end{pmatrix}}_{=\cos(\theta)I_{3\times 3}} + (1-\cos(\theta)) \underbrace{\begin{pmatrix} u_x^2 & u_xu_y & u_xu_z \\ u_xu_y & u_y^2 & u_yu_z \\ u_xu_z & u_yu_z & u_z^2 \end{pmatrix}}_{=uu^T} \\ &\quad -\sin(\theta) \underbrace{\begin{pmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{pmatrix}}_{=:U} \\ &= \cos(\theta)I_{3\times 3} + (1-\cos(\theta))uu^T - \sin(\theta)U. \end{aligned} \tag{2.1.4}$$

This way of writing the equation 2.1.3 will help us with the main theorem on the next section.

2.2 The rotation space $SO(3)$ and generalized Euler sequences

Before we move on to the objective functions we are going to work with, let us understand the space where the solution of the first objective of this work lives, the space of all rotations in $\mathbb{R}^3$.

Definition 2.2.1. *The set that describes all rotations in 3D is given by:*

$$SO(3) = \{R \in \mathbb{R}^{3\times 3} \mid R^T R = I_{3\times 3}, det(R) = 1\}$$

Theorem 2.2.1 (Euler's Rotational Theorem)**.** *Given any $R \in SO(3)$, $\exists u \in \mathbb{R}^3$, $\|u\| = 1$ and an angle $\theta \in (-\pi, \pi]$ such that $R = R_u(\theta)$, as in 2.1.3.*

Proof. To prove this theorem, given any $R \in SO(3)$, if we find a vector $u \in \mathbb{R}^3$ such that $Ru = u$, we will have found a vector that is invariant by R and thus we will be able to finding the axis of rotation (hence the angle).

By definition of $SO(3)$ and orthogonal transformations, $R^T = R^{-1}$ and using determinant properties $\det(AB) = \det(A)\det(B)$ and $\det(-A) = (-1)^n \det(A), \forall A, B \in \mathbb{R}^{n\times n}$, we have that:

$$\begin{aligned}
\det(R - I_{3\times 3}) &= \det\left((R - I_{3\times 3})^T\right) = \det\left(R^T - I_{3\times 3}\right) = \det\left(R^T - R^T R\right) \\
&= \det\left(R^T (I_{3\times 3} - R)\right) = \det\left(-R^T (R - I_{3\times 3})\right) \\
&= \det\left(-R^T\right)\det(R - I_{3\times 3}) = -\det\left(R^T\right)\det(R - I_{3\times 3}) \\
&= -\det(R - I_{3\times 3}).
\end{aligned}$$

Thus, $\det(R - I_{3\times 3}) = -\det(R - I_{3\times 3})$ and this is only possible if $\det(R - I_{3\times 3}) = 0$. Then $(R - I_{3\times 3})$ is singular, $\mathcal{N}(R - I_{3\times 3}) \neq \emptyset$ and there exists at least one $u \in \mathbb{R}^3$ such that $u \in \mathcal{N}(R - I_{3\times 3})$ and $(R - I_{3\times 3})u = 0$. So,

$$(R - I_{3\times 3})u = 0 \Leftrightarrow Ru = u.$$

Notice that, for any vector $w \in \{u\}^\perp$, we have that

$$0 = \langle u, w \rangle = \langle Ru, w \rangle.$$

Then, R takes the $\{u\}^\perp$ into itself and $\exists \theta \in \mathbb{R}$ that R rotates any vector in $\{u\}^\perp$ around u-axis (see figure 2.1.1 (a)). Also, given any $x \in \mathbb{R}^3, x \neq 0$ and $w \in \{u\}^\perp$, we have that $\{u, w, (u \times w)\} \subset \mathbb{R}^3$ forms a basis to $\mathbb{R}^3$. So, we can write x as $x = \alpha_1 u + \alpha_2 w + \alpha_3 (u \times w), \alpha_i \in \mathbb{R}, i = \{1, 2, 3\}$

and we have that

$$\begin{aligned} Rx &= R(\alpha_1 u + \alpha_2 w + \alpha_3(u \times w)) = \alpha_1 Ru + \alpha_2 Rw + \alpha_3 R(u \times w) \\ &= \alpha_1 u + R(\alpha_2 w + \alpha_3(u \times w)). \end{aligned}$$

So, R preserves $\alpha_1 u$ and keeps $\alpha_2 w$ and $\alpha_3(u \times w)$ on $\{u\}^\perp$. As R also preserves the angle between $\alpha_2 w$ and $\alpha_3(u \times w)$, we have that R rotates those around u by an angle θ, as we can see on figure 2.2.1.

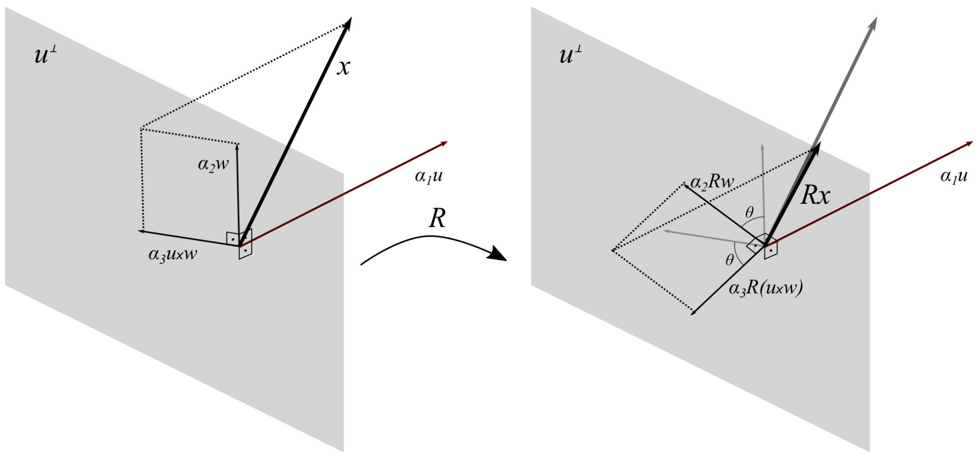

Figure 2.2.1: The vector x and its coordinates with $\{u, w, (u \times w)\} \subset \mathbb{R}^3$ basis before (left) and after (right) applied rotation matrix R. Notice that R preserves the coordinate $\alpha_1 u$ and rotate $\alpha_2 w$ and $\alpha_3(u \times w)$ around u by an angle θ.

Since $\langle u, x\rangle = \langle Ru, Rx\rangle = \langle u, Rx\rangle$ and $||Rx|| = ||x||$, we have that R preserves the angle between u and x. So, given the arbitrariness of the choice of x and R and the fact that R rotates any vector in $\{u\}^\perp$ around u by an angle $\theta \in \mathbb{R}$, we have that R is a rotation around u by the angle θ and $R = R_u(\theta)$.

■

Theorem 2.2.1 will have an important role on the proof of the next theorem.

Definition 2.2.2. *Consider three axis* $u_1, u_2, u_3 \in \mathbb{R}^3, ||u_i|| = 1, i = 1,2,3,$ *such that* $u_1^T u_2 = u_2^T u_3 = 0$. *Then, a generalized Euler sequence is given by*

$$R_{u_3}(\theta_3) R_{u_2}(\theta_2) R_{u_1}(\theta_1) \in SO(3),$$

for $\theta_1, \theta_2, \theta_3 \in \mathbb{R}$ *and* $R_{u_i}(\theta_i) \in SO(3), i = 1,2,3.$

Theorem 2.2.2. *Consider three axis* $u_1, u_2, u_3 \in \mathbb{R}^3, \|u_i\| = 1, u_i = (u_i^x\ u_i^y\ u_i^z)^T, i = 1,2,3,$ *and let* $\alpha \in (-\pi, \pi]$ *be the unique angle such that*

$$u_1^T u_2 = u_2^T u_3 = 0, \quad \sin(\alpha) = u_3^T U_1 u_2 \quad \textit{and} \quad \cos(\alpha) = u_3^T u_1,$$

where

$$U_1 = \begin{pmatrix} 0 & -u_1^z & u_1^y \\ u_1^z & 0 & -u_1^x \\ -u_1^y & u_1^x & 0 \end{pmatrix}.$$

Then, for any $R \in SO(3)$, *there exists* $(\theta_1, \theta_2, \theta_3) \in (-\pi, \pi] \times [-\alpha, \pi - \alpha] \times (-\pi, \pi]$ *and* $(\theta_1, \theta_2, \theta_3) \in (-\pi, \pi] \times [-\pi - \alpha, -\alpha] \times (-\pi, \pi]$ *such that*

$$R = R_{u_3}(\theta_3) R_{u_2}(\theta_2) R_{u_1}(\theta_1).$$

Proof. To prove this theorem, we have to show that the transformations

1. $R_{u_3}(\theta_3) R_{u_2}(\theta_2) R_{u_1}(\theta_1) : (-\pi, \pi] \times [-\alpha, \pi - \alpha] \times (-\pi, \pi] \longrightarrow SO(3)$ and
2. $R_{u_3}(\theta_3) R_{u_2}(\theta_2) R_{u_1}(\theta_1) : (-\pi, \pi] \times [-\pi - \alpha, -\alpha] \times (-\pi, \pi] \longrightarrow SO(3)$

are surjective.

So, let any $R \in SO(3)$. Let us build a generalized Euler sequence such that $R = R_{u_3}(\theta_3) R_{u_2}(\theta_2) R_{u_1}(\theta_1)$.

- Finding $\theta_1 \in (-\pi, \pi]$:

Suppose for the moment that given any $R \in SO(3)$, we have three axes $u_i \in \mathbb{R}^3, \|u_1\| = 1$ and three angles $\theta_i \in \mathbb{R}$ with $i = 1,2,3$ such that $R = R_{u_3}(\theta_3) R_{u_2}(\theta_2) R_{u_1}(\theta_1)$. If we transpose this equation, we have $R^T = R_{u_1}^T(\theta_1) R_{u_2}^T(\theta_2) R_{u_3}^T(\theta_3)$. So, multiplying both sides by $u_2^T R_{u_1}(\theta_1)$ on the left and by u_3 on the right, we have:

$$\begin{aligned} u_2^T R_{u_1}(\theta_1) R^T u_3 &= u_2^T \underbrace{R_{u_1}(\theta_1) R_{u_1}^T(\theta_1)}_{I_{3\times 3}} R_{u_2}^T(\theta_2) R_{u_3}^T(\theta_3) u_3 \\ &= u_2^T R_{u_2}^T(\theta_2) R_{u_3}^T(\theta_3) u_3 \\ &= \left(R_{u_2}(\theta_2) u_2\right)^T R_{u_3}^T(\theta_3) u_3. \end{aligned}$$

Using equation 2.1.3 for $R_u(\theta)$, any $u \in \mathbb{R}^3, \|u\| = 1$ and any $\theta \in \mathbb{R}$, notice that:

$$R_u(\theta)u =$$

$$= \begin{pmatrix} u_x^3(1-\cos(\theta)) + u_x\cos(\theta) + u_xu_y^2(1-\cos(\theta)) + u_yu_z\sin(\theta) + u_xu_z^2(1-\cos(\theta)) - u_yu_z\sin(\theta) \\ u_x^2u_y(1-\cos(\theta)) - u_xu_z\sin(\theta) + u_y^3(1-\cos(\theta)) + u_y\cos(\theta) + u_yu_z^2(1-\cos(\theta)) + u_xu_z\sin(\theta) \\ u_x^2u_z(1-\cos(\theta)) + u_xu_y\sin(\theta) + u_y^2u_z(1-\cos(\theta)) - u_xu_y\sin(\theta) + u_z^3(1-\cos(\theta)) + u_z\cos(\theta) \end{pmatrix}$$

$$= \begin{pmatrix} u_x^3 - u_x^3\cos(\theta) + u_x\cos(\theta) + u_xu_y^2 - u_xu_y^2\cos(\theta) + u_yu_z\sin(\theta) + u_xu_z^2 - u_xu_z^2\cos(\theta) - u_yu_z\sin(\theta) \\ u_x^2u_y - u_x^2u_y\cos(\theta) - u_xu_z\sin(\theta) + u_y^3 - u_y^3\cos(\theta) + u_y\cos(\theta) + u_yu_z^2 - u_yu_z^2\cos(\theta) + u_xu_z\sin(\theta) \\ u_x^2u_z - u_x^2u_z\cos(\theta) + u_xu_y\sin(\theta) + u_y^2u_z - u_y^2u_z\cos(\theta) - u_xu_y\sin(\theta) + u_z^3 - u_z^3\cos(\theta) + u_z\cos(\theta) \end{pmatrix}$$

$$= \begin{pmatrix} u_x(u_x^2 - u_x^2\cos(\theta) + \cos(\theta) + u_y^2 - u_y^2\cos(\theta)) + u_z^2 - u_z^2\cos(\theta)) \\ u_y(u_x^2 - u_x^2\cos(\theta)) + u_y^2 - u_y^2\cos(\theta)) + \cos(\theta) + u_z^2 - u_z^2\cos(\theta)) \\ u_z(u_x^2 - u_x^2\cos(\theta) + u_y^2 - u_y^2\cos(\theta) + u_z^2 - u_z^2\cos(\theta) + \cos(\theta)) \end{pmatrix}$$

$$= \begin{pmatrix} u_x\left(1 + \cos(\theta)(-u_x^2 + 1 - u_y^2 - u_z^2)\right) \\ u_y\left(1 + \cos(\theta)(-u_x^2 + 1 - u_y^2 - u_z^2)\right) \\ u_z\left(1 + \cos(\theta)\underbrace{(-u_x^2 + 1 - u_y^2 - u_z^2)}_{=0}\right) \end{pmatrix} = \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix} = u$$

So, $R_u(\theta)u = u$ for any $u \in \mathbb{R}^3, \|u\| = 1$ and $\theta \in \mathbb{R}$.

Also, notice that, since $R_u(\theta)u = u$ and $R_u^T(\theta)R_u(\theta) = I_{3\times3}$ then, multiplying $R_u(\theta)u = u$ on the left by $R_u^T(\theta)$ we obtain

$$\underbrace{R_u^T(\theta)R_u(\theta)}_{I_{3\times3}}u = R_u^T(\theta)u \Rightarrow R_u^T(\theta)u = u,$$

and then u is invariant by $R_u^T(\theta)R_u(\theta)$ and $R_u(\theta)$.

Therefore, we have

$$\begin{aligned} u_2^T R_{u_1}(\theta_1) R^T u_3 &= u_2^T \underbrace{R_{u_1}(\theta_1) R_{u_1}^T(\theta_1)}_{I_{3\times 3}} R_{u_2}^T(\theta_2) R_{u_3}^T(\theta_3) u_3 \\ &= u_2^T R_{u_2}^T(\theta_2) R_{u_3}^T(\theta_3) u_3 \\ &= \underbrace{\left(R_{u_2}(\theta_2) u_2\right)^T}_{=u_2^T} \underbrace{R_{u_3}^T(\theta_3) u_3}_{=u_3}. \\ &= u_2^T u_3 = 0 \end{aligned} \quad . \qquad (2.2.1)$$

Writing the matrix $R_{u_1}(\theta_1) = \cos(\theta_1) I_{3\times 3} + (1-\cos(\theta_1)) u_1 u_1^T - \sin(\theta_1) U_1$, we have:

$$\begin{aligned} & u_2^T \left(\cos(\theta_1) I_{3\times 3} + (1-\cos(\theta_1)) u_1 u_1^T - \sin(\theta_1) U_1\right) R^T u_3 = 0 & \Leftrightarrow \\ \Leftrightarrow \quad & \cos(\theta_1) u_2^T R^T u_3 + (1-\cos(\theta_1)) \underbrace{u_2^T u_1}_{=0} u_1^T R^T u_3 - \sin(\theta_1) u_2^T U_1 R^T u_3 = 0 & \Leftrightarrow \\ \Leftrightarrow \quad & \cos(\theta_1) u_2^T R^T u_3 = \sin(\theta_1) u_2^T U_1 R^T u_3. \end{aligned}$$

So, we have four cases to analyse:

1. If $u_2^T R^T u_3 \neq 0$ and $u_2^T U_1 R^T u_3 \neq 0$, it is impossible to have $\cos(\theta_1) = 0$ (if $\cos(\theta_1) = 0$, we would have $\sin(\theta_1) = 0 = \cos(\theta_1)$, and this could not happen). So, we have:

$$\tan(\theta_1) = \frac{u_2^T R^T u_3}{u_2^T U_1 R^T u_3} \neq 0$$

and it is possible to find an angle $\theta_1 \in \left(-\pi, -\frac{\pi}{2}\right) \cup \left(-\frac{\pi}{2}, 0\right) \cup \left(0, \frac{\pi}{2}\right) \cup \left(\frac{\pi}{2}, \pi\right)$, that satisfies this equation.

2. If $u_2^T R^T u_3 = 0$ and $u_2^T U_1 R^T u_3 \neq 0$, we have that $\sin(\theta_1) = 0$ and this is only possible if $\theta_1 = -\pi$, $\theta_1 = 0$ or $\theta_1 = \pi$.

3. If $u_2^T R^T u_3 \neq 0$ and $u_2^T U_1 R^T u_3 = 0$, we have that $\cos(\theta_1) = 0$ and this is only possible if $\theta_1 = -\frac{\pi}{2}$ and $\theta_1 = \frac{\pi}{2}$.

4. If $u_2^T R^T u_3 = u_2^T U_1 R^T u_3 = 0$, we can choose any angle $\theta_1 \in (-\pi, \pi]$.

Therefore, in all cases we can find a solution for the equation above.

- Finding $\theta_2 \in [\alpha - \pi, \alpha]$ and $\theta_2 \in [\alpha, \pi + \alpha]$:

  Let $\theta_1 \in (-\pi, \pi]$ and consider a vector $w \in \mathbb{R}^3$ such that

$$w = R_{u_1}(\theta_1) R^T u_3.$$

Since $u_1 \perp u_2$ and $U_2 u_1$ represents the croos product between u_2 and u_1, $\{u_1, U_2 u_1, u_2\}$ is an orthonormal basis for $\mathbb{R}^3$. So, since we have $u_2^T u_3 = 0$, it follows that $u_3 \in span\{u_1, U_2 u_1\}$. Notice that, by the trigonometric identity $\sin(\alpha)^2 + \cos(\alpha)^2 = 1$, we have that

$$1 = \sin(\alpha)^2 + \cos(\alpha)^2 = \sin(\alpha)\sin(\alpha) + \cos(\alpha)\cos(\alpha)$$

Then, taking an $\alpha \in (-\pi, \pi]$ such that

$$\sin(\alpha) = u_3^T U_1 u_2 \text{ and } \cos(\alpha) = u_3^T u_1$$

we have

$$\begin{aligned}
1 &= \sin(\alpha)^2 + \cos(\alpha)^2 \\
&= \sin(\alpha)\sin(\alpha) + \cos(\alpha)\cos(\alpha) \\
&= \sin(\alpha) u_3^T U_1 u_2 + \cos(\alpha) u_3^T u_1 \\
&= u_3^T \left(\sin(\alpha) U_1 u_2 + \cos(\alpha) u_1 \right).
\end{aligned}$$

Then, as $\|u_3\|^2 = u_3^T u_3 = 1$, we have that $u_3 = \sin(\alpha) U_1 u_2 + \cos(\alpha) u_1$. From equation 2.1.4, we have

$$R_{u_2}^T(\alpha) = \cos(\alpha) I_{3\times 3} + (1 - \cos(\alpha)) u_2 u_2^T + \sin(\alpha) U_2.$$

Multiplying on the right by u_1, we have

$$\begin{aligned}
R_{u_2}^T(\alpha) u_1 &= \cos(\alpha) u_1 + (1 - \cos(\alpha)) u_2 \underbrace{u_2^T u_1}_{=0} + \sin(\alpha) U_2 u_1 \\
&= \cos(\alpha) u_1 + \sin(\alpha) U_2 u_1 \\
&= u_3.
\end{aligned}$$

Notice that $\|w\| = \|R_{u_1}(\theta_1) R^T u_3\| = 1$ and $u_2 \perp w$ by equation 2.2.1. Since $u_2 \perp w$, we have that $w \in span\{u_1, U_2 u_1\}$. So, just as we did for u_3 before, taking an angle $\gamma \in (-\pi, \pi]$

that

$$\sin(\gamma) = w^T U_1 u_2 \text{ and } \cos(\gamma) = u_3^T u_1,$$

we can write w as

$$w = \cos(\gamma) u_1 + \sin(\gamma) U_2 u_1 = R_{u_2}^T(\gamma) u_1.$$

Setting $\theta_2 = \gamma - \alpha$ we have

$$\begin{aligned} R_{u_2}(\theta_2) w &= R_{u_2}(\gamma - \alpha) R_{u_2}^T(\gamma) u_1 \\ &= R_{u_2}(-\alpha) u_1 = R_{u_2}^T(\alpha) u_1 = u_3 \end{aligned}$$

Therefore, we have that

$$u_3 = R_{u_2}(\theta_2) w = R_{u_2}(\theta_2) R_{u_1}(\theta_1) R^T u_3.$$

We have shown that it always possible to find $\theta_1 \in (-\pi, \pi]$ and $\theta_2 \in [-\alpha, \pi - \alpha]$ if $(\theta_2 + \alpha) \in [0, \pi]$ or $\theta_2 \in [-\pi - \alpha, -\alpha]$ if $(\theta_2 + \alpha) \in [-\pi, 0]$ such that

$$u_3 = R_{u_2}(\theta_2) R_{u_1}(\theta_1) R^T u_3.$$

Now, let us find θ_3.

- Finding $\theta_3 \in (-\pi, \pi]$:

Since $R_{u_2}(\theta_2) R_{u_1}(\theta_1) R^T \in SO(3)$ and $u_3 = R_{u_2}(\theta_2) R_{u_1}(\theta_1) R^T u_3$, using the same construction on theorem 2.2.1, $\exists \theta_3 \in (-\pi, \pi]$ such that

$$R_{u_3}(-\theta_3) = R_{u_2}(\theta_2) R_{u_1}(\theta_1) R^T.$$

So, multiplying both sides of the equality by R on the right and $R_{u_3}^T(-\theta_3)$ on the left, we have

$$\begin{aligned} \underbrace{R_{u_3}^T(-\theta_3) R_{u_3}(-\theta_3)}_{I_{3\times 3}} R &= R_{u_3}^T(-\theta_3) R_{u_2}(\theta_2) R_{u_1}(\theta_1) \underbrace{R^T R}_{I_{3\times 3}} \quad \Leftrightarrow \\ \Leftrightarrow \qquad R &= R_{u_3}^T(-\theta_3) R_{u_2}(\theta_2) R_{u_1}(\theta_1). \end{aligned}$$

Since $R_{u_3}^T(-\theta_3) = R_{u_3}(\theta_3)$ we were able to find $\theta_1 \in (-\pi, \pi], \theta_2 \in [-\alpha, \pi - \alpha]$ or $\theta_2 \in$

$[-\pi-\alpha,-\alpha]$ and $\theta_3 \in (-\pi,\pi]$ such that

$$R = R_{u_3}(\theta_3)R_{u_2}(\theta_2)R_{u_1}(\theta_1).$$

■

2.3 Spatial surfaces and their representations

Since the most relevant 3D objects in our work are 3D surfaces, we need to understand the concept of surfaces. We can define a *topological surface* as a subset $S \subset \mathbb{R}^3$ that is locally homeomorphic to an open set of the Euclidian plane $\mathbb{R}^2$, e.g., $\forall x \in S$, existis $U \subset S$ and $V \subset \mathbb{R}^2$, U,V open sets and $x \in U$ such that $\exists \Phi : V \longrightarrow U$ that has the following properties:

- Φ is a bijection
- Φ is continuous and
- Φ^{-1} is continuous.

There are two ways to describe a surface: *parametric* and *implicit*.

Definition 2.3.1. *A* Parametric surface S *is described by a transformation* $f : U \subset \mathbb{R}^2 \longrightarrow \mathbb{R}^3$

$$f(u,v) = \begin{pmatrix} f_1(u,v) \\ f_2(u,v) \\ f_3(u,v) \end{pmatrix}$$

with $f_i : U \subset \mathbb{R}^2 \longrightarrow \mathbb{R}, i = \{1,2,3\}$*, as illustrated in figure 2.3.1.*

.

Example 2.3.1. *Consider the function* $f : [0,1] \times [0,2\pi] \longrightarrow \mathbb{R}^3$ *given by*

$$f(u,v) = \begin{pmatrix} u\cos(v) \\ u\sin(v) \\ u \end{pmatrix}.$$

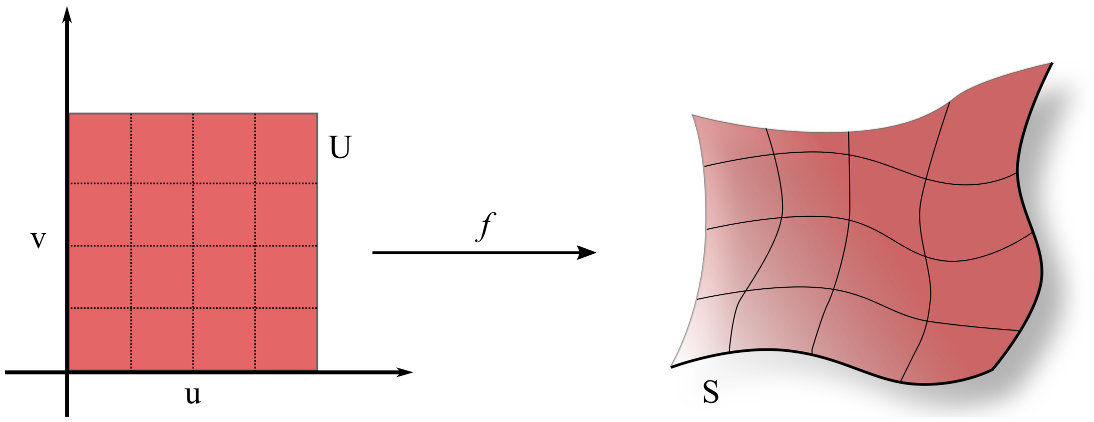

Figure 2.3.1: Parametric surface.

The function f is the parametric representation of a cone that opens along the z-axis, as can be seen in figure 2.3.2.

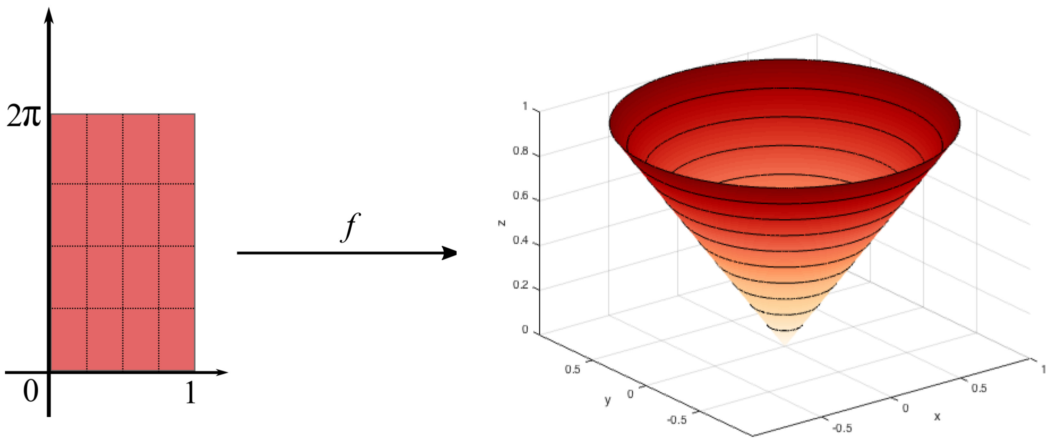

Figure 2.3.2: Example of a parametric cone that opens along the z-axis and its domain.

2.3.1 Triangle mesh representation

A *2D triangulation* of a region in space is a collection $\mathscr{F} = \{f_i\}$ of triangles, in which, given two different triangles f_i, f_j, $i \neq j$ in the triangulation $\mathscr{F}$ such that $f_i \cap f_j \neq \emptyset$, we have $f_i \cap f_j$ is a commom vertex or $f_i \cap f_j$ is a commom edge.

Since we are interested to work with 3D objects, the concept of 2D triangulation is fundamental to help us to develop our work in the computer. Using the idea of triangulation, we can define a *triangulated surface*.

Definition 2.3.2. *A* Triangulated Surface *is a triangulation of the space that covers a given surface partly or totally, as ilustrated on figure 2.3.3.*

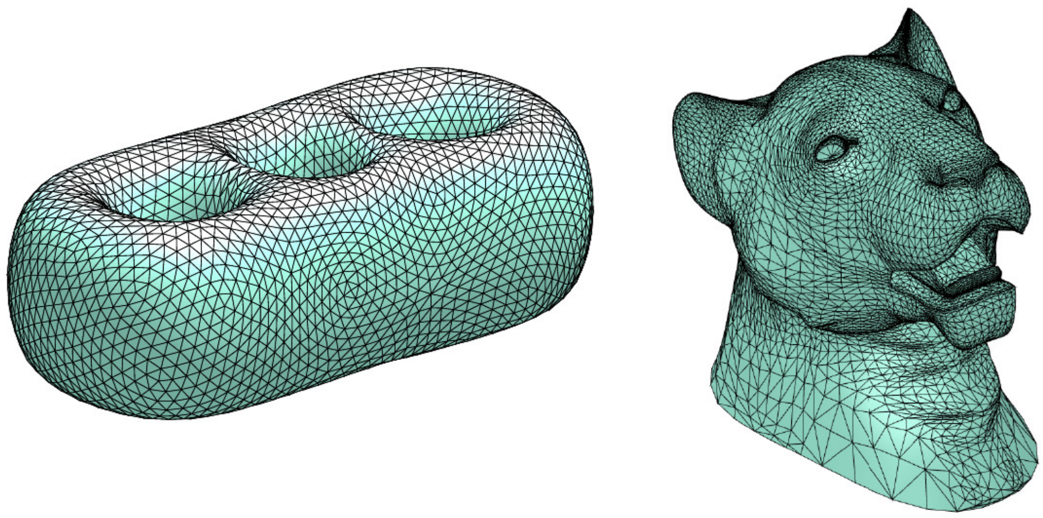

Figure 2.3.3: Examples of triangulated surfaces. All these surfaces are available on Libigl website (JACOBSON et. al., 2018).

To better analyze our solid in the computer, we need to remember some properties of surfaces and their discretizations. This analysis will help us to understand a different way to compare surface angles and the threshold angles for overhangs.

A solid can be represented in the computer by its vertex matrix $\mathscr{V} \in \mathbb{R}^{n\times 3}$, $\mathscr{V} = \{v_1, v_2, \dots, v_n\}$, where $n = |\mathscr{V}|$ is the number of vertices of the surface, in which each vertex $v_i \in \mathbb{R}^{1\times 3}$ has the x-, y-, z- coordinates of the vertex $i = 1, \dots, n$. A way (BOTSCH et al, 2010) to represent a surface mesh is storing the set of vertex indices on a matrix $\mathscr{F} \in \mathbb{R}^{m\times 3}$, $m = |\mathscr{F}|$, that represents the connection between the vertices and originates the triangular faces, i.e., the matrix $\mathscr{F}$ has in each row the indices of rows of $\mathscr{V}$ that, together, form a triangle mesh. In figure 2.3.4, we can see an example of an individual triangle mesh and its vertices.

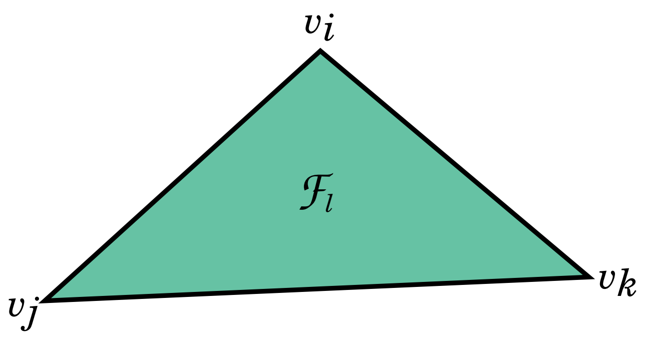

Figure 2.3.4: In this example, the *l-th* row of triangle mesh matrix $\mathscr{F}$ stores the row positions of the vertex v_i, v_j and v_k of the vertex matrix $\mathscr{F}$. This face, on the matrix, is represented by $\mathscr{F}_l = (i, j, k)$.

Example 2.3.2. *Consider the triangular base prism surface in figure 2.3.5.*

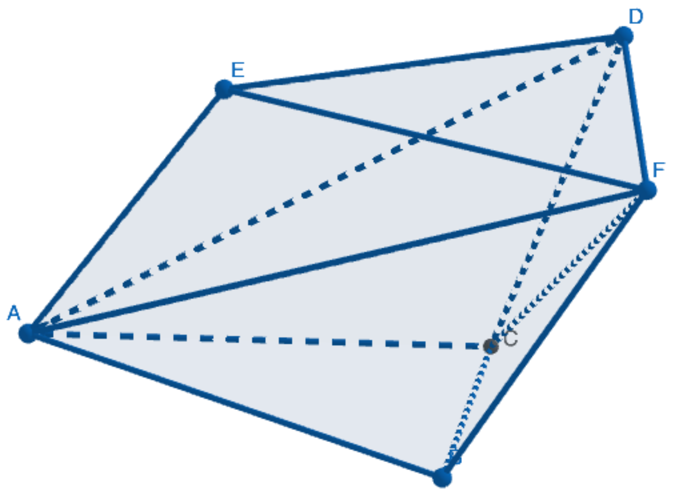

Figure 2.3.5: Triangular base prism.

Its vertex and face matrices $\mathscr{V} \in \mathbb{R}^{6\times 3}$ *and* $\mathscr{F} \in \mathbb{R}^{8\times 3}$ *are*

$$\mathscr{V} = \begin{pmatrix} -1.27 & 0 & 0 \\ 0 & -1.3 & 0 \\ 0 & 0 & 0 \\ 0.48 & -1.15 & 1.22 \\ -0.69 & -0.4 & 0.82 \\ 0.6 & -1.63 & 0.93 \end{pmatrix} \text{ and } \mathscr{F} = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 6 \\ 1 & 5 & 6 \\ 1 & 3 & 4 \\ 1 & 4 & 5 \\ 2 & 3 & 6 \\ 3 & 4 & 6 \\ 4 & 5 & 6 \end{pmatrix}.$$

Using the triangle mesh representation of a triangulated surface, we can define a crucial element of our analysis.

Definition 2.3.3. *Given a vertex matrix* $\mathscr{V} \in \mathbb{R}^{n\times 3}, n = |\mathscr{V}|$ *and a triangle* $f = (i, j, k)$ *of its triangle mesh* $\mathscr{F} \in \mathbb{R}^{m\times 3}$*, the* normal *vector of the triangle f is*

$$n(f) = \frac{(v_j - v_i) \wedge (v_k - v_i)}{\|(v_j - v_i) \wedge (v_k - v_i)\|}, \tag{2.3.1}$$

where $\wedge : \mathbb{R}^3 \longrightarrow \mathbb{R}^3$ *is the Euclidean cross product.*

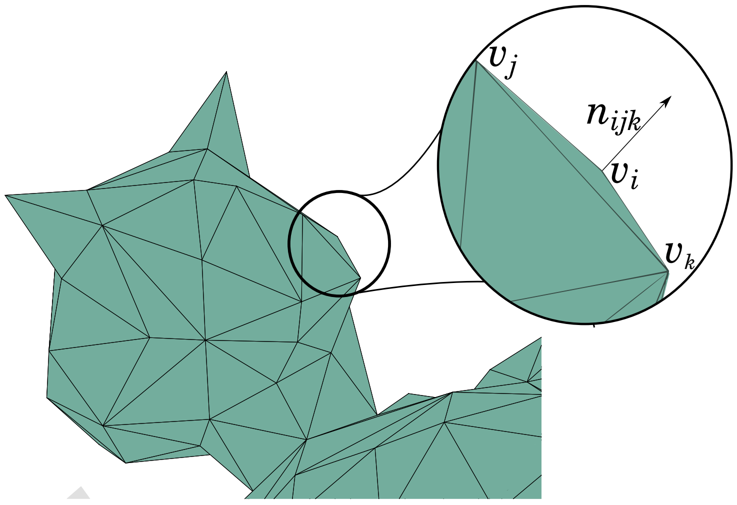

Figure 2.3.6: Associated to this surface, we have the vertex matrix $\mathscr{V}$ and its triangle mesh matrix $\mathscr{F}$. We can associate one normal vector for each triangle on mesh, as defined in definition 2.3.3.

Each triangle in the mesh has a normal vector associated, as shown in figure 2.3.6 and the set of all normal vectors in a triangle mesh is called by *Normal Field*. We can organize them in the matrix of normals $\mathscr{N} \in \mathbb{R}^{m\times 3}$.

3 Objective function

Our purpose here is to find an embedding of the surface that minimizes the amount of support in a 3D printing process. These supports are worthless and, after printing the object, the printing user will detach them and throw them away, leading to a waste of material and money.

With that in mind, we seek to find a transformation of a surface that does not change distances and angles. So what we are looking for is a rigid transformation, as in definition 2.1.6. Also, we saw in theorem 2.1.2 that every orientation-preserving linear rigid transformation is an axial rotation and, finally, with theorem 2.2.2, we can represent any rotation on $SO(3)$ with a generalized Euler sequence, given by a matrix product of three axial rotation matrices, respecting the conditions of the same theorem.

Notice that, if we choose the canonical basis in $\mathbb{R}^3$, $e_1 = (1\,0\,0)^T, e_2 = (0\,1\,0)^T$ and $e_3 = (0\ 0\ 1)^T$ as the three axes of the generalized Euler sequence, according to theorem 2.2.2, we have to cosider an angle $\alpha \in (-\pi, \pi]$ such that

$$u_1^T u_2 = u_2^T u_3 = 0, \quad \sin(\alpha) = u_3^T U_1 u_2 \quad \text{and} \quad \cos(\alpha) = u_3^T u_1,$$

to be able to represent any rotation on $SO(3)$ with these three axes. Since $e_1^T e_2 = e_2^T e_3 = 0$ and $\cos(\alpha) = e_3^T e_1 = 0$ let us analize the sin condition to find α:

$$\begin{aligned} \sin(\alpha) = e_3^T E_1 e_2 &= \begin{pmatrix} 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = 1. \end{aligned}$$

We have that $\alpha \in (-\pi, \pi]$ is an angle that $\cos(\alpha) = 0$, $\sin(\alpha) = 1 \Rightarrow \alpha = \dfrac{\pi}{2}$.

So, by the theorem 2.2.2, for any $R \in SO(3)$, we can find

$$(\theta_1, \theta_2, \theta_3) \in (-\pi, \pi] \times \left[-\frac{\pi}{2}, \frac{\pi}{2}\right] \times (-\pi, \pi]$$

that $R = R_{e_3}(\theta_3)R_{e_2}(\theta_2)R_{e_1}(\theta_1)$. But, for our purpose of minimizing overhang support in the 3D printing process, we have to note the following: given any surface $\mathscr{V} \in \mathbb{R}^{|V|\times 3}$ and any vertex $v \in \mathscr{V}, v = (x \;\; y \;\; z)^T$, if we rotate $\mathscr{V}$ by any angle $\theta \in (-\pi, \pi]$ around e_3, we have that

$$R_{e_3}(\theta)v = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \cos(\theta)x - \sin(\theta)y \\ \sin(\theta)x + \cos(\theta)y \\ z \end{pmatrix}.$$

So, the $z-axis$ rotation does not change the value of $z-coordinate$ and, if the surface needs to print a support, this particular rotation will not change the need for this support. On figure 3.0.1, we can see the Camel surface on initial position and after $z-axis$ rotation.

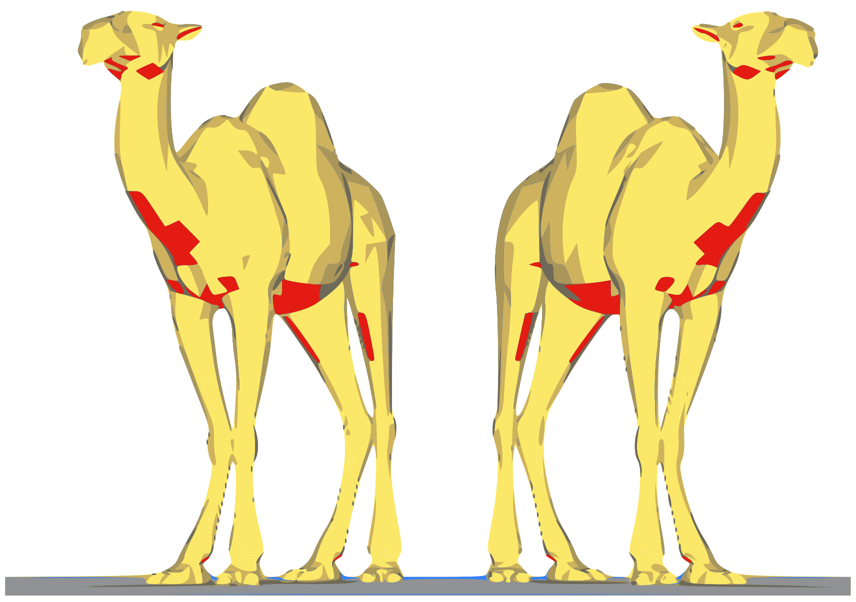

Figure 3.0.1: Camel surface from Libigl (JACOBSON et al., 2018) in the initial position (left) and after $\theta = \dfrac{\pi}{3}$ z-axis rotation (right). The red area of both surfaces are those that need support to print. Notice that the support area does not change after this $z-axis$ rotation.

Therefore, we only need to find a rotation $R = R_{e_2}(\theta_2)R_{e_1}(\theta_1)$, with $(\theta_1, \theta_2) \in (-\pi, \pi] \times \left[-\dfrac{\pi}{2}, \dfrac{\pi}{2}\right]$ that, applied to a surface, finds an orientation that reduces the ammount of support.

In (VANEK; GALICIA; BENES, 2014), before building the overhang support proposed by them, they use the results of (ALEXANDER; ALLEN; DUTTA, 1998) to find a better orientation for the tested surfaces. As we will see in this section, our method provides a surface orientation that results in a lower amount of support volume and printing time.

3.1 Surface normal angle analysis

We can analyze overhanging parts of the solid that is going to be printed using the normal field of its surface, as illustrated in figure 3.1.1: the faces that have normals belonging to the white region of the sphere on the right can be printed without supports and the ones that have normals in the green region need supports to be printed.

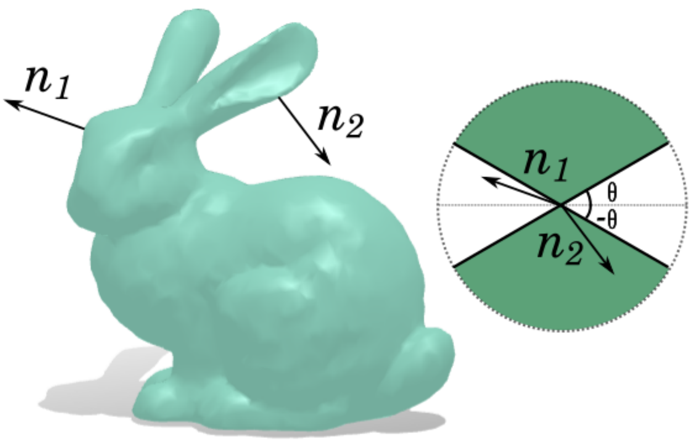

Figure 3.1.1: This two normals n_1 and n_2 of the bunny surface have different angles. The normal vector n_1 is within the angle range of the printer. The normal vector n_2 is in an overhanging part and will need a support to print correctly.

At the moment of printing, the user can choose the amount of surface infill, ranging from 0% to 100%. In our work we consider 0% infill, which corresponds to the worst possible case in terms of support needed for printing, as we can see on figure 3.1.2. So the parts of the surface that have normal vector pointing upwards will also be considered in the problem.

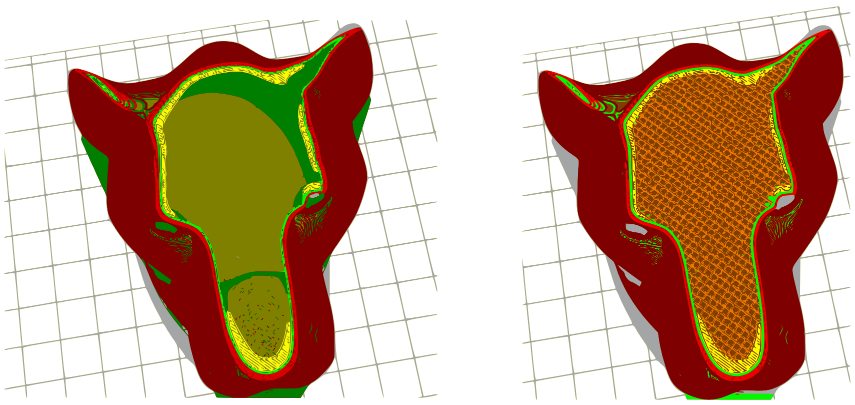

Figure 3.1.2: Lion surface from Libigl (JACOBSON et al., 2018) with 0% infill (left) and 30% infill(right). The parts of the surface such that normal angles are above the support angle threshold θ will also need some support to be printed, if the printer infill is set to 0%.

Given a triangle mesh matrix $\mathscr{F}$ we can find the angle of the normal of each triangle $f \in \mathscr{F}$ with the vertical direction, defining the function $\alpha : \mathscr{F} \longrightarrow \mathbb{R}$,

$$\alpha(f) = \arccos\left(n(f)^T e_3\right), \tag{3.1.1}$$

where e_3 is the unit vector $e_3 = (0,0,1)^T$ and $n(f) \in \mathbb{R}^3$ is the (unit) normal vector of triangle f, as you can be seen in figure 3.1.3. So, given the normal field $\mathscr{N}$, the angle of each normal

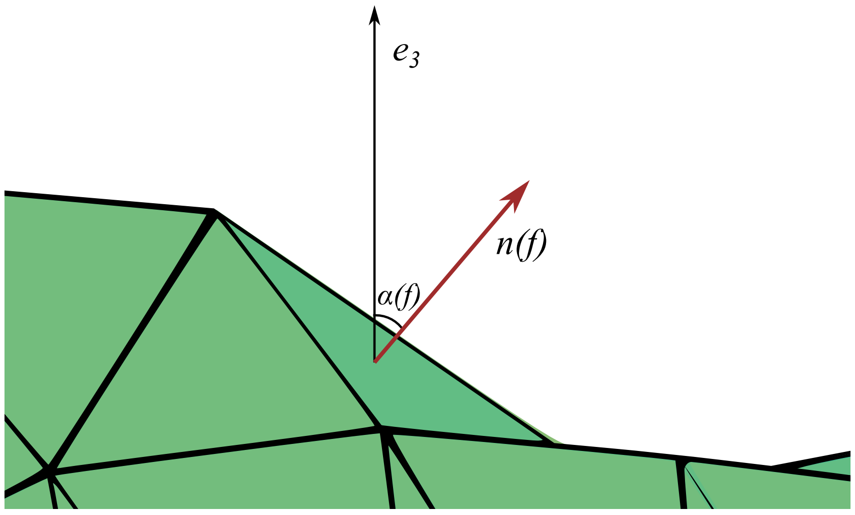

Figure 3.1.3: The equation (3.1.1) measure the angle between the normal $n(f) \in \mathscr{N}$ of a triangle $f \in \mathscr{F}$ on mesh and the vertical direction $e_3 = (0\ 0\ 1)^T$.

and the printer limit angle θ for overhang, we can measure the residual error of each normal angle and the threshold angle and sum over all triangle of the mesh:

$$\sum_{f \in \mathscr{F}} \max\left\{\left|\alpha(f) - \frac{\pi}{2}\right| - \theta, 0\right\}. \tag{3.1.2}$$

Since we want to find a rotation $R \in SO(3)$ that when applied to the surface normal field minimizes the sum in equation 3.1.2, we want to obtain the solution for

$$\begin{aligned} \min_{R} \quad & \sum_{f \in \mathscr{F}} \max\left\{\left|\alpha(R \cdot f) - \frac{\pi}{2}\right| - \theta, 0\right\} \\ s.t. \quad & R \in SO(3) \end{aligned}. \tag{3.1.3}$$

3.1.1 Initial results

The libraries we are working with are Libigl (JACOBSON et al., 2018) (C++) and *gptoolbox* (JACOBSON et. al., 2018) (Matlab). Our main job will be to analyze the surface to be printed and return new vertex positions $\mathscr{V}$ to be printed with minimal overhang support. Initially, we are seeking for a global rotation of the surface that, given the normal field of the surface, finds a new normal field that minimizes the overhanging parts of the surface. To make the tests more reliable, we preprocessed all surfaces in our tests to be centered at the origin.

Using some surfaces available in *Libigl* and some freely available surfaces on Sketch Fab website (DENOYEL, PINSON, PASSET, 2012), we were able to perform some tests to evaluate if it is possible to find rotated surfaces with normal fields that solve the problem (3.1.3). Initial results show that for every libgl surface tested we find an x-axis and y-axis angle rotation that minimize the value of function (3.1.2), as can be seen in table 3.1.1.

Surface(.obj)	**x-axis angle**	**y-axis angle**	**Initial**	**Minim**
camel_b	136^o	-2^o	235.29	150.19
horse_quad	-60^o	-30^o	565.53	215.28
bunny	124^o	46^o	812.80	346.33
decimated-knight	-92^o	30^o	120.44	43.18
decimated-max	64^o	-10^o	$1.0517 \cdot 10^3$	431.93
cow	112^o	42^o	883.05	266.99
arm	-100^o	54^o	$2.1418 \cdot 10^3$	409.98
heart	-128^o	50^o	$1.5740 \cdot 10^3$	831.86
skull	-32^o	14^o	295.14	268.10

Table 3.1.1: All these surfaces present a smaller sum in function 3.1.2 with an x-axis and y-axis angle rotation. The surfaces *Camel, Horse, Bunny, Knight, Max, Cow and Arm* were chosen from Libigl (JACOBSON et al., 2018) library and *heart* (ADOREZOOEY, 2015) and *skull* (AELLIS43, 2014) have a Creative Commons Attribution and are freely available surfaces on the SketchFab website. All these surfaces can be seen in figure 3.1.4.

These tests were done in MATLAB, using some of gptoolbox functions to read the vertex matrix $\mathscr{V}$ and triangle mesh matrix $\mathscr{F}$ of each surface file (.obj extension). Using these matrices, we can calculate the sum in (3.1.2) and test for all possible x-axis and y-axis angle combination with a $\Delta\theta = \dfrac{\pi}{180}$ step for both variables. We also show in figure 3.1.6 2D graphs that show the variation of function value (3.1.2) on these combinations.

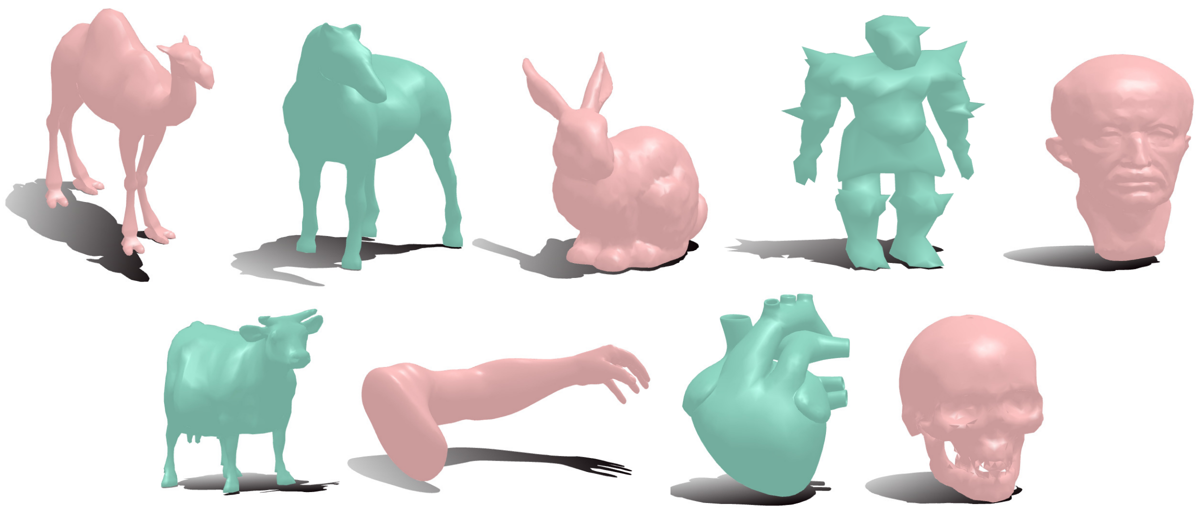

Figure 3.1.4: The surfaces *Camel, Horse, Bunny, Knight, Max, Cow and Arm* were chosen from Libigl (JACOBSON et al., 2018) library and *heart* (ADOREZOOEY, 2015) and *skull* (AELLIS43, 2014) are available on SketchFab (DENOYEL, PINSON, PASSET, 2012) website.

As we can see in figure 3.1.6, the rotation of x-axis and y-axis, and its combination, generates regions of minimal value of equation (3.1.2) and has a periodicity, what indicates more than one minimizer. The minimal value in equation (3.1.3), rotation result of cow.obj and decimated-knight.obj found with our initial tests can be seen in figure 3.1.5. Both results

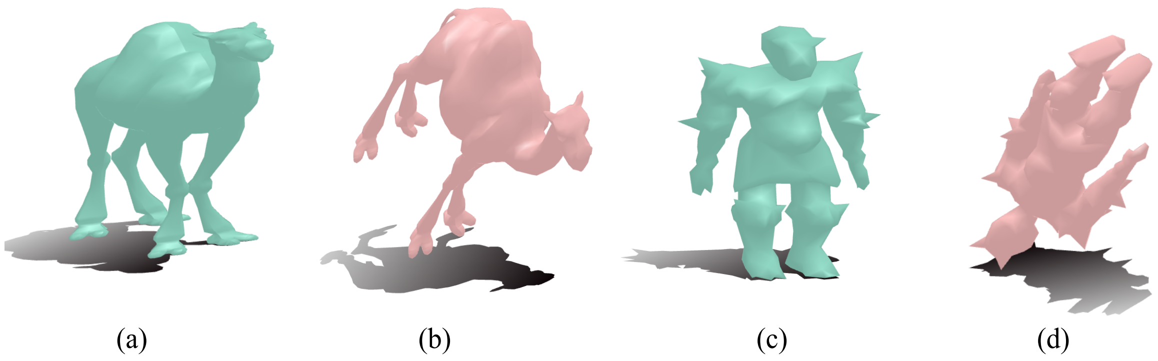

Figure 3.1.5: Horse surface at the initial embedding (a) and after rotation (b). Knight at the initial embedding (c) and after rotation (d).

show that small rotations applied to the surfaces can lead to normal fields that result in less overhanging parts.

3D printing software, such as Ultimaker Cura (ELSERMAN, BRUIJN, WIJNIA, 2011), SLic3r (RANELLUCCI, LENOX, 2011), and others, allow us to simulate solids being printed and their overhang supports. So, we can validate our tests before printing them. With these 3D printer softwares, we can simulate, and then measure the time of each print on

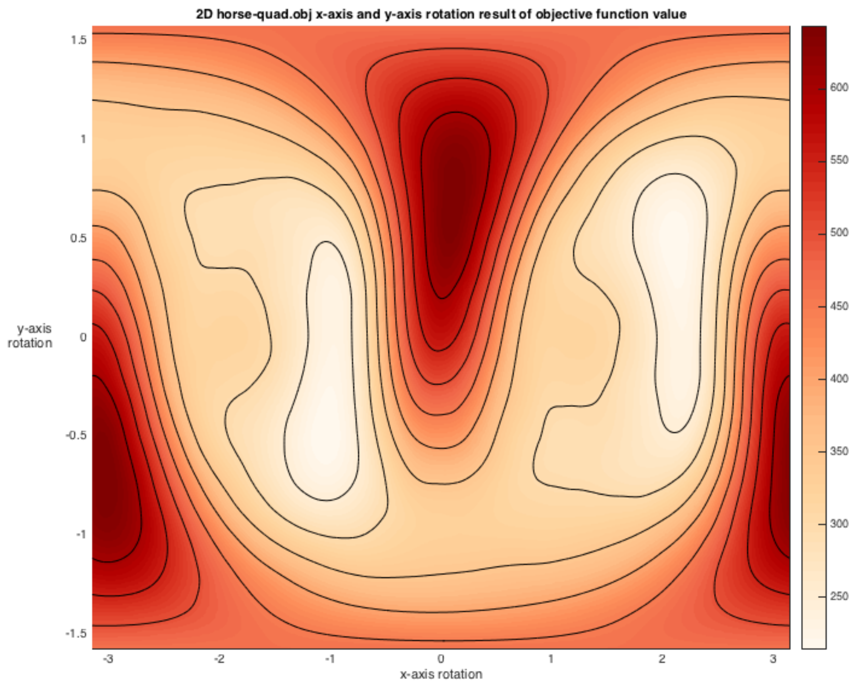

Figure 3.1.6: This graph represents the value (in a gradient color) of sum of max in equation 3.1.2 for every discretized x-axis and y-axis rotation of *horse-quad.obj* surface.

a 3D printer, as well as the total weight and length of material used to print the surfaces. Some results are shown on table 3.1.2.

We can also see in figures 3.1.7 and 3.1.8 that the vertex positions that result in the minimal value of equation (3.1.2) generate less supports for overhanging parts. Figure 3.1.8 also shows physical prints of the input and output surfaces.

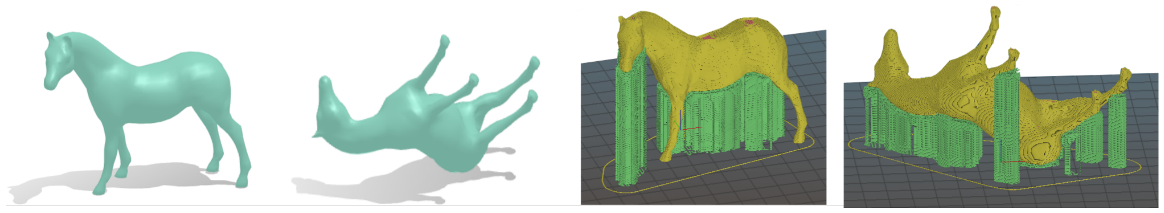

Figure 3.1.7: Horse_quad.obj on the initial vertex position (left) and after rotation (center-left). The same surface on the initial position on the printing simulator (center-right) and after rotation (right). In yellow, the software indicates the solid that we want to print and in green the overhang supports.

Surface	Material	Usual orientation	New orientation
Human skull	Length	5.38m	4.53m
	Weight	16g	14g
	Time	1h51min	1h44min
Horse	Length	6.09m	5.41m
	Weight	18g	16g
	Time	2h06min	2h04min
Max	Length	10.05m	7.47m
	Weight	30g	22g
	Time	3h08min	2h36min
Human heart	Length	13.14m	11.39m
	Weight	39g	34g
	Time	4h37min	4h20min
Knight	Length	5.79m	5.62m
	Weight	17g	16g
	Time	2h17min	2h11min

Table 3.1.2: All these surfaces present a smaller printint time, weight and length of material after applied a global rotation that minimizes overhang parts. The surfaces *Camel, Horse, Bunny, Knight, Max, Cow and Arm* were chosen from Libigl (JACOBSON et al., 2018) library and *heart* (ADOREZOOEY, 2015) and *skull* (AELLIS43, 2014) are from Sketch Fab website.

3.1.2 Problems in paradise

As we saw before, table 3.1.1 illustrates that with the objective function 3.1.3 we are be able to find rotation angles and a minimal value of the function for all surfaces tested. But, the bad news is that when we put the results on a printing software, we could find some surfaces that, after rotation by the minimizer of the function, needed more support than the initial positions. Also, we could find some surfaces that this objective function minimizes the amount of support printed, but does not reduce the printing time. We can see in table 3.1.3 that the proposed objective function 3.1.3 does not minimize the amount of support printed for some surfaces. We can see in figure 3.1.9 an example of surface for which the amount of support is not minimized.

Besides that, in order to obtain the actual optimal rotation (instead of looping over a finite set of angles), we want to use an optimization method that uses a descent direction to find

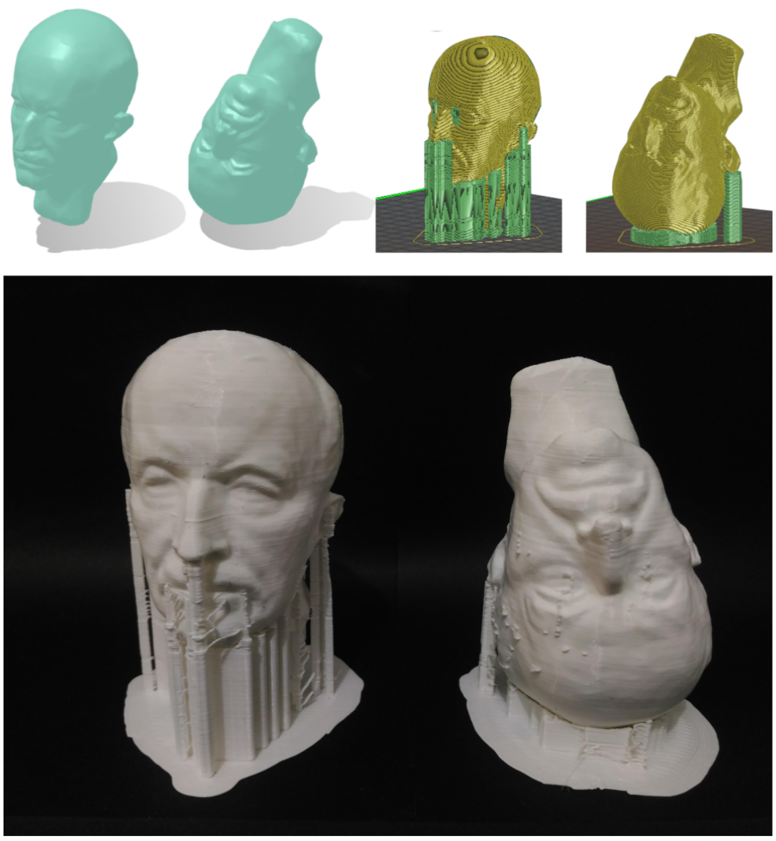

Figure 3.1.8: Surface *decimated-max.obj* on the initial vertex position (top-left) and after rotation (top-center-left). The same surface on the initial position on the printing simulator (top-center-right) and after rotation (top-right). In yellow, the software indicates the solid that we want to print and in green the overhang supports. The bottom figure is a photo of the same surface after printed. On the bottom-left, the surface is at the initial vertex positions and on the bottom-right, the surface is positioned after rotation.

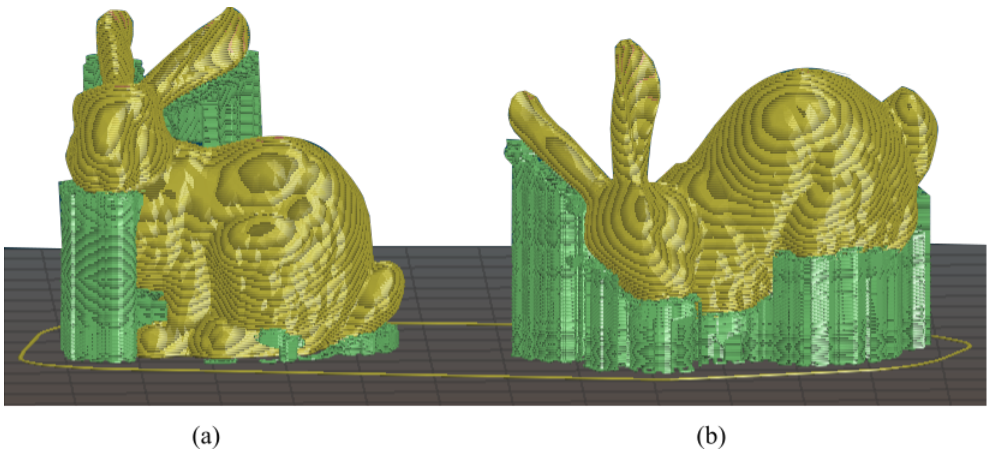

Figure 3.1.9: Surface *Bunny.obj* at the initial vertex position (a) and after rotation (b). In yellow, the software indicates the solid that we want to print and in green the overhang supports. With Slic3r sorftare we can observe that the objective function 3.1.3 returns an orientation of this object with more support than initial position.

Surface	Material	Usual orientation	New orientation
Arm	Length	1.86m	1.23m
	Weight	6g	4g
	Time	39min	54min
Bunny	Length	7.85m	8.78m
	Weight	23g	26g
	Time	2h41min	3h01min
Camel	Length	6.04m	6.65m
	Weight	18g	20g
	Time	2h10min	2h24min
Cow	Length	7.16m	8.35m
	Weight	21g	25g
	Time	2h31min	2h50min

Table 3.1.3: All these surfaces present a higher printing time after applied a global rotation that minimizes overhang parts, according to the objective function (3.1.3). On first row of this table, *Arm* is the only surface tested that is reducing the amount of support, but does not reduce printing time. These surfaces were chosen from Libigl (JACOBSON et al., 2018).

this minimizer. So, even if the level curves of figure 3.1.6 show that the objective function 3.1.3 seems to be smooth enough to find a gradient vector, inner functions of 3.1.3 such as $\max(\cdot)$ and $|\cdot|$ will make the gradient impossible to calculate. We can see this difficulty when we take derivatives of equation 3.1.3 and differentiate $\alpha(u) = \arccos(u)$ function, that is

$$\frac{d}{du}\alpha(u) = \frac{d}{du}\left[\arccos(u)\right] = \frac{-1}{\sqrt{1-u}}u'.$$

In equation 3.1.1, we see that if $u = n(f)^T e_3 \to 1$ (i.e. the normal vector of some face $f \in \mathscr{F}$ of surface approaches to z-axis), we have that $\frac{-1}{\sqrt{1-u}} \to -\infty$, leading us to further problems in its differentiability and perhaps an approximation of gradient calculation.

3.2 Surface normal "Least squares"

We propose here to continue analyzing overhanging parts of our printing solid using its normal field. Consider the "*xy*-Projection function"

$$\begin{array}{rccc} Proj_{xy}: & \mathbb{R}^3 & \longrightarrow & \mathbb{R}^3 \\ & (x\ y\ z)^T & \longmapsto & (x\ y\ 0)^T \end{array}$$

that maps a vector in $\mathbb{R}^3$ into its projection on *xy*-plane. Given a triangle face $f \in \mathscr{F}$ and its normal $n(f) \in \mathscr{N} \subset \mathbb{R}^3$, composing the normal function and "*xy*-Projection function" we have

$$\begin{array}{rccccc} Proj_{xy} \circ n: & \mathscr{F} & \longrightarrow & \mathbb{R}^3 & \longrightarrow & \mathbb{R}^3 \\ & f & \longmapsto & n(f) := \left(n_x\ n_y\ n_z\right)^T & \longmapsto & \left(n_x\ n_y\ 0\right)^T \end{array} . \qquad (3.2.1)$$

Analysing figure 3.1.1, we can see that the more surface normals are within the white region of the angle cone, the less we have surface overhang, and consequently, the object will need less support to print correctly. Taking this into consideration, if we rotate any normal surface vector in any direction so that it approaches the *xy*-plane, this vector will enter the support-free white zone (see figure 3.1.1) and its *z*-coordinate (n_z) will approach zero, as we can see in figure 3.2.1.

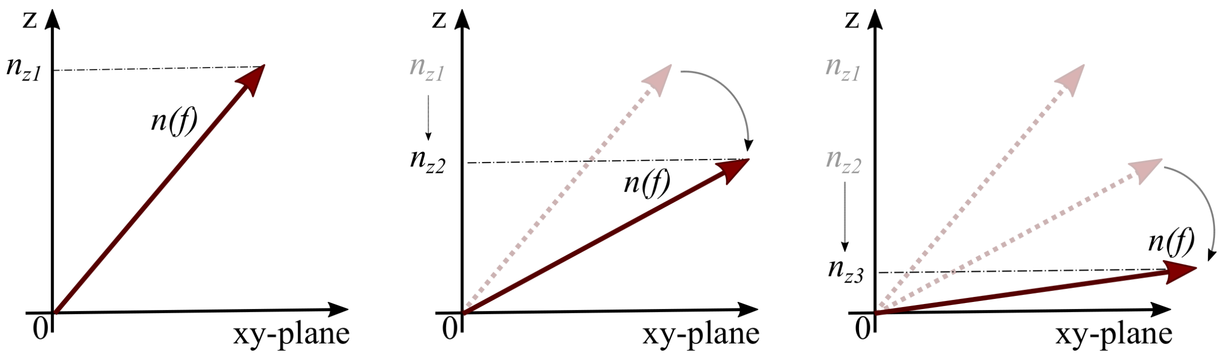

Figure 3.2.1: As the normal vector $n(f)$ approaches the *xy*-plane, the value of its z-coordinate will approach zero. Note that the sequence in the image $\{n_{zk}\}_{k=\{1,2,3\}} \subset \mathbb{R}$ of *z*-coordinate values on *z*-axis is such that $n_{zk} \to 0$ if the sequence of rotation on $n(f)$ leads to the *xy*-plane.

So, for any surface and its normal field $\mathscr{N}$, if we define an objective function that searches for a surface orientation such that its normal vectors are as horizontal as possible (i.e. find an embedding of the surface with the z-coordinates of the normal vectors as close to zero as possible), we will find an embedding for the surface that minimizes its overhanging parts.

Therefore, given a triangle mesh face $f \in \mathscr{F}$ of the surface, its normal vector $(n_x^f\ n_y^f\ n_y^f) = n(f) \in \mathscr{N}$ and the normal xy-Projection function $Proj_{xy}(n(f)) \in \mathbb{R}^3$, taking the difference between them, we have

$$n(f) - Proj_{xy}(n(f)) = \begin{pmatrix} n_x^f \\ n_y^f \\ n_z^f \end{pmatrix} - \begin{pmatrix} n_x^f \\ n_y^f \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ n_z^f \end{pmatrix},$$

(see figure 3.2.2 for an illustration). Taking the norm of this result, squaring it and summing

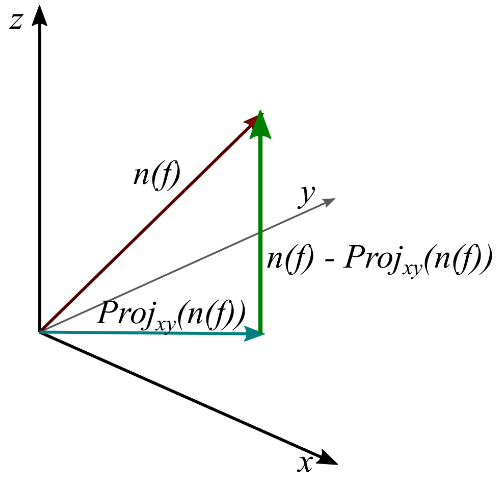

Figure 3.2.2: The difference between the surface normal vector $n(f)$ (in wine) and its xy-Projection is a vector with zeros on x and y coordinates and z-coordinate normal vector value on itself. Observe that, the closer to the xy plane is the surface normal, the closer to zero is the difference norm. Consequently, this triangle mesh face is as vertical as possible.

over all triangles of the mesh, we get

$$\sum_{f \in \mathscr{F}} \|n(f) - Proj_{xy}(n(f))\|^2. \tag{3.2.2}$$

We want to find a rotation $R \in SO(3)$ applied to the surface that minimizes the sum in equation 3.2.2. We then obtain the following optimization problem:

$$\begin{aligned} \min_{R} \quad & \sum_{f \in \mathscr{F}} \|R \cdot n(f) - Proj_{xy}(R \cdot n(f))\|^2 \\ s.t. \quad & R \in SO(3) \end{aligned} \quad . \tag{3.2.3}$$

3.2.1 Support volume

Some tests using the objective function 3.1.3 reveal that, even getting less overhanging parts on all surfaces tested, there are some other factors to be considered. Looking at figure 3.2.3, we observe that the rotation procedure (b) positioned overhanging parts higher than in the initial position (a). This will create taller overhang support in the printing procedure and, eventually, more support volume and material waste.

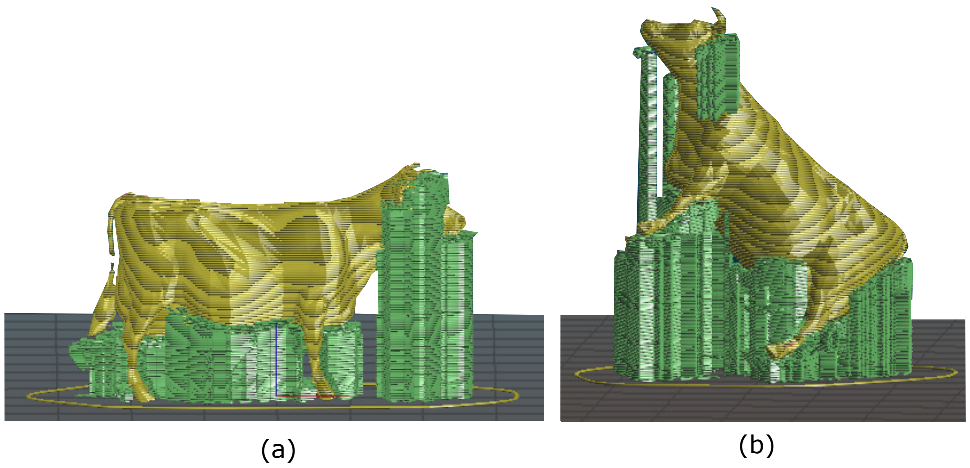

Figure 3.2.3: Surface *cow.obj* at the initial vertex position (a) and after rotation (b) minimizing the objective function 3.1.3. Even getting less overhanging parts, this objective function does not consider the surface area and height of support, which, in this case, causes the amount of support to be larger than desired.

Another thing that we have to consider is that the objective function 3.1.3 and the new function 3.2.3 are analyzing only the normal vector of each triangle on the surface mesh and does not consider whether the triangle has huge or small area.

This means that, for each triangle in equation 3.2.3, we need to multiply the norm by a *"weight"* that takes into account the average height of triangle vertices and its area. We have to see that the triangle height will change when the rotation matrix $R \in \mathbb{R}^{3\times 3}$ is applied, but the area of each triangle will not change with this transformation (see the definition of rigid transformation on section 2.1). So, the equation we want to minimize will be

$$\min_{R} \sum_{f \in \mathscr{F}} \left[e_3^T \left(R \cdot \frac{1}{3} \left(v_{f_1} + v_{f_2} + v_{f_3} \right) \right) - min_z^{\mathscr{V}} \right] \cdot A_f \cdot \| R \cdot n(f) - Proj_{xy}(R \cdot n(f)) \|^2, \quad s.t. \quad R \in SO(3) \tag{3.2.4}$$

where $e_3 = (0\ 0\ 1)^T$, $v_{f1}, v_{f2}, v_{f3} \in \mathbb{R}^3$, A_f are vertices and area of triangle $f \in \mathscr{F}$, respectively, and $min_z^{\mathscr{V}} \in \mathbb{R}$ is the minimal value of all z-coordinates of the vertices of $\mathscr{V}$. Since the barycenter of the surface is positioned at $(0\ 0\ 0)^T$, some parts of the surface are below the xy-plane of $\mathbb{R}^3$. So, to be able to calculate the real height of each triangle, we need to compensate this error of positioning adding the value of $min_z^{\mathscr{V}}$.

Note that this weight parameter that we have proposed is close to the volume of the support to be printed. To be exactly the volume of the support, we have to consider some other factors, such as the shape of support, if the support will intersect the surface (and then the support is not printed from the platform base, but from some part of the surface itself) and if this part of surface really needs a support. Here, what we are doing is to consider that all triangle faces of surface mesh needs some kind of support to print and searching for an orientation that minimizes this "maximal need for support".

3.3 Alexander's method

Before creating their support structures, (VANEK; GALICIA; BENNES, 2014) states that the simplest way to reduce the need for supports is to find an orientation for the surfaces using (ALEXANDER; ALLEN; DUTTA, 1998) method.

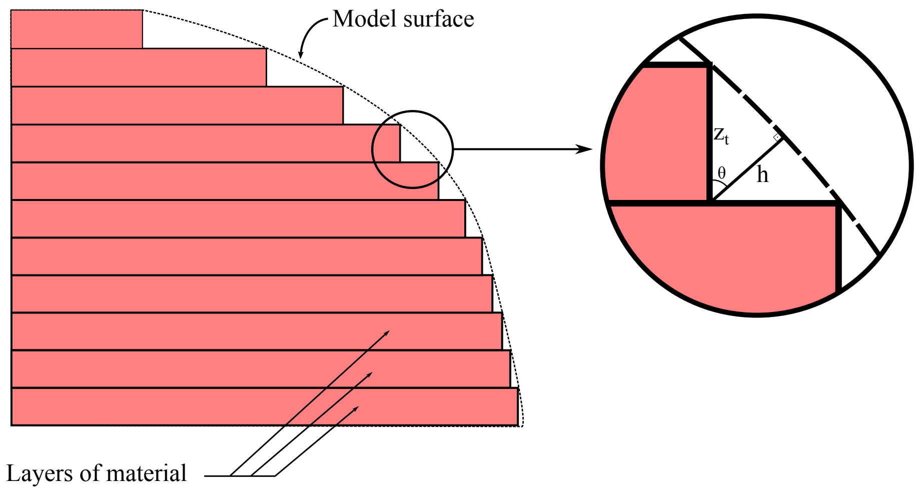

Figure 3.3.1: Comparison of layered surface parts with its model surface, leading to a staircase effect. Zooming in and considering that the difference between layers forms a triangle, we can analyse the height h of this triangle, the thickness z_t of the layer and the angle θ between them to minimize the staircase effect. We can see in the figure on the left that the more vertical the surface is, the less it suffers from a staircase effect. This figure was based on the figures in (ALEXANDER; ALLEN; DUTTA, 1998).

The main objective of this method is to minimize the staircase effect, as shown in figure 3.3.1, to maximize surface accuracy, and thus the surface must be oriented as vertically as possible.

So, if we look at the triangle formed by the thickness of layer z_t, the height h and the angle θ between them, we have that

$$h_{fac} = \begin{cases} z_t|\cos(\theta)|, & \text{if } |\cos(\theta)| \neq 1 \\ 0, & \text{if } |\cos(\theta)| = 1 \end{cases}. \tag{3.3.1}$$

Considering the normal vector to the surface $\hat{n} \in \mathbb{R}^3, \|\hat{n}\| = 1$ and the build direction $\hat{b} \in \mathbb{R}^3, \|\hat{b}\| = 1$, we have that

$$|\cos(\theta)| = |\langle \hat{n}, \hat{b} \rangle|. \tag{3.3.2}$$

Plugging 3.3.2 into 3.3.1, we have

$$h_{fac} = z_t|\langle \hat{n}, \hat{b} \rangle| \tag{3.3.3}$$

for $|\cos(\theta)| \neq 1$.

Since 3D objects have multiple polygonal faces, the value of equation 3.3.3 has to be weighted. So, for each face, this method multiplies its face area A_{fac} into the objective function, getting

$$h_{fac} = A_{fac}\left(z_t|\langle \hat{n}, \hat{b} \rangle|\right). \tag{3.3.4}$$

Also, this method considers three types of faces on the surface: *unsupported faces, faces that need support* and *faces touched from above by the support structure*, as illustrated in figure 3.3.2. Taking this into consideration, the method uses two ways to apply different weights to different types of faces.

For faces that need support to print, it is assumed that when supports touch the surface, they add some additional inaccuracy and equation 3.3.3 is increased by an amount $R \in \mathbb{R}^+$, getting

$$h_{fac_s} = A_{fac}\left(z_t|\langle \hat{n}, \hat{b} \rangle| + R\right). \tag{3.3.5}$$

For faces that are touched by support from above, a piece of support structure is projected to those faces. So, to determine the contact area of this structure and supposing that it is a prism with a rectangular basis with sides of length p and q, given $e_1, e_2 \in \mathbb{R}^3$ unit vectors in x and y directions, respectively, $\hat{b}$ the vector parallel to the build direction and $\hat{n}$ the normal

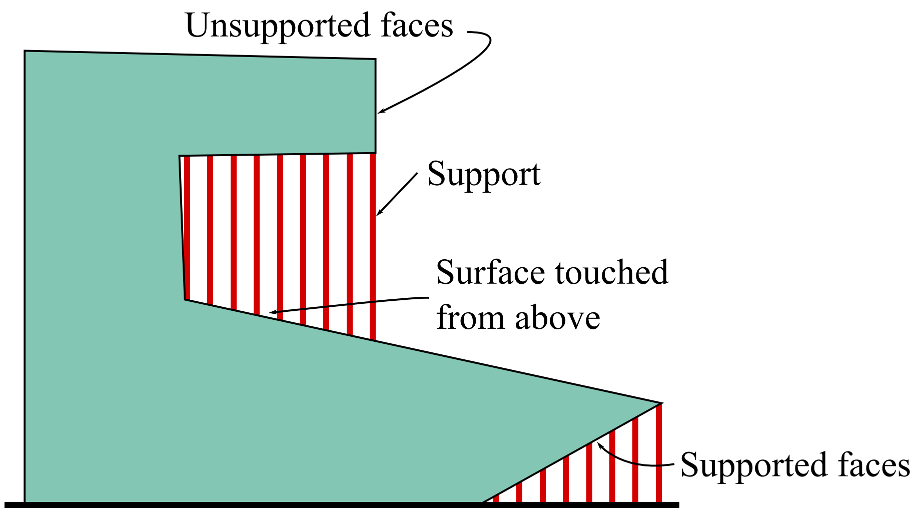

Figure 3.3.2: In this vertical cut of a 3D object, we can see that the surface can be divided in three parts. *Unsupported faces*: parts of object that do not need external support to print; *supported faces*: parts of object that need external support underneath them to print; and *support faces*: parts of object that are touched from above by the support structure. This figure was created based on figures in (ALEXANDER; ALLEN; DUTTA, 1998).

vector of face, we calculate the ray that defines the support doing

$$\tilde{x} = pe_1,$$

$$\tilde{b}_x = -\left(\frac{\langle \hat{n}, \tilde{x} \rangle}{\langle \hat{n}, \hat{b} \rangle}\right),$$

$$\bar{x} = \tilde{x} + \tilde{b}_x$$

and simillary for $\tilde{y}, \tilde{b}_y$ and $\bar{y}$. So, the area of contact of support structure is $A_{proj} = |\bar{x} \times \bar{y}|$ and they calculate the area following the condition

$$A_{proj} = \begin{cases} |\bar{x} \times \bar{y}|, & \text{if } A_{proj} \leq A_{fac} \\ A_{fac}, & \text{otherwise.} \end{cases}$$

So, to increase the value of function to faces that is touched by the support from above, the weight condition in those cases are given by

$$h_{fac_{ab}} = RA_{proj}. \tag{3.3.6}$$

Finally, taking the results of equations 3.3.4, 3.3.5 and 3.3.6 and summing over all faces of surface, we obtain the objective function of Alexander's method:

$$\min_{\mathscr{N}} \sum_{i=1}^{N-N_{sup}} A_i\left(z_t|\langle\hat{n}_i,\hat{b}\rangle|\right)+\sum_{i=1}^{N_{sup}} A_i\left(z_t|\langle\hat{n}_i,\hat{b}\rangle|+R\right)+\sum_{i=1}^{N_{rays}} RA_{proj}, \tag{3.3.7}$$

where $\mathscr{N}$ is the normal field of surface, N is the total number of polygonal faces, N_{sup} is the number of faces that need support underneath them and N_{rays} is the number of faces that are touched by support from above.

3.4 Tests and method comparison

These tests were also done on MATLAB, using some of *gptoolbox* functions (JACOBSON et. al., 2018). Also, using the vertices matrix $\mathscr{V}$, triangle mesh matrix $\mathscr{F}$ of some Libigl (JACOBSON et. al., 2018) surfaces and freely available *SketchFab website* surfaces, we could calculate the sum in 3.2.4 and test for rotation matrix around x and y axis, and their combinations, with a step of $\Delta\theta = \dfrac{\pi}{180}$ for both axis.

Figure 3.4.1 shows a 2D graph that illustrates the variation of the function value in equation 3.2.4. This graph shows that our new objective function 3.2.4 has regions with function minimal value. The results of finding a minimal value of function in equation 3.2.4 in some

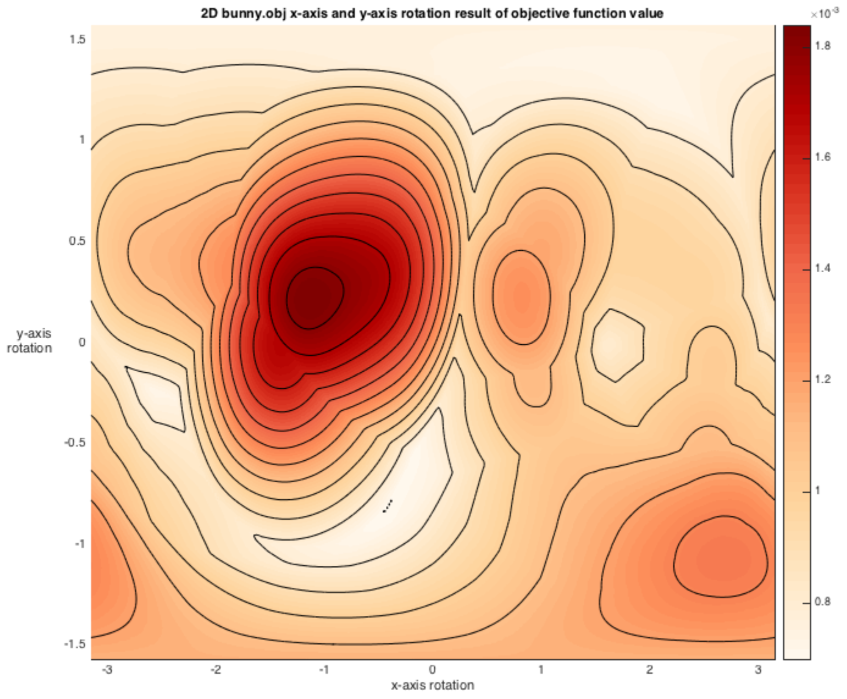

Figure 3.4.1: 2D graph of function value variation in equation 3.2.4 applied to *bunny.obj* Libigl surface.

surfaces can be seen in figure 3.4.2.

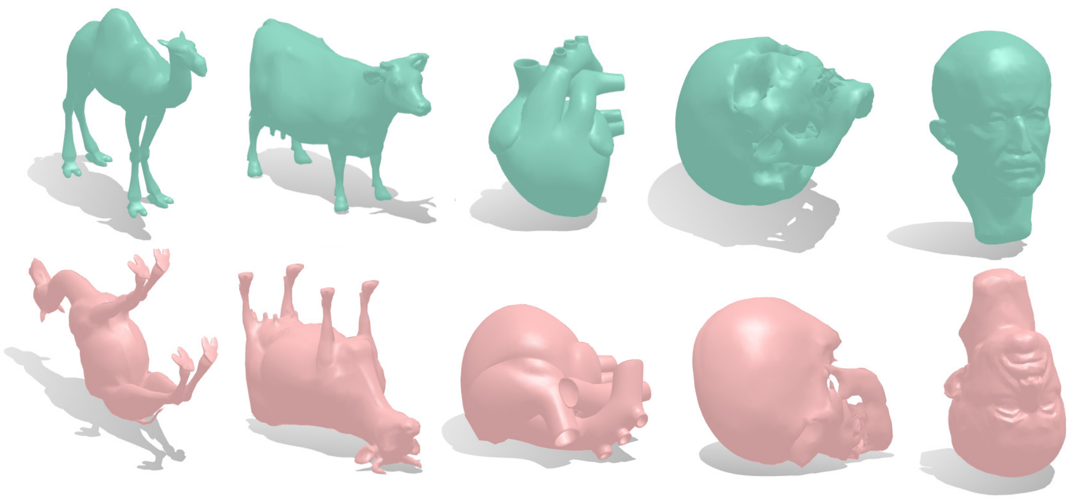

Figure 3.4.2: Surfaces at the initial vertex position (top) and after applied a global rotation that minimizes equation 3.2.4 (bottom). *Camel, Cow and Max (human head)* are surfaces available on Libigl (JACOBSON et al., 2018) and *heart* (ADOREZOOEY, 2015) and *skull* (AELLIS43, 2014) are Sketch Fab surfaces.

Using Ultimaker Cura 3D printing software we can simulate and visualize our tests and obtain the length of polymer that we need to print the solid, as well as the weight and time of printing.

We can also see that the surface *bunny.obj* in figure 3.4.3 with 3D printing software (Slic3r) generates less support for overhangs after rotation then at the initial position.

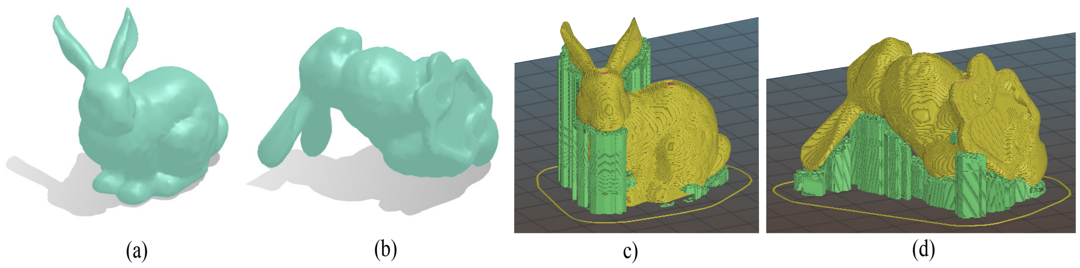

Figure 3.4.3: Libigl surface *bunny.obj* on the initial position (a) and after rotation (b). On 3D print sorftware (Sklic3r), the same surface on the initial position (c) and after rotation (d). Also, in figures (c) and (d), the software indicates the surface in yellow and the overhang supports in green.

As the solid, in any orientation, has the same weight and need of polymer to print, the difference between this data will show us the residual material to print, i.e. the less weight and length of material the print has, the less support the 3D object need to print. As we can see in table 3.4.1, all surfaces tested present less length, weight and printing time after applied global rotation that minimizes equation 3.2.4.

In table 3.4.1, we can also compare the results obtained using our equation 3.2.4 with the results obtained using Alexander's method (ALEXANDER; ALLEN; DUTTA, 1998) and their equation 3.3.7.

Surface	Material	Usual orientation	Our orientation	Alexander's orientation
Arm	Length	4.82m	3.32m	4.14m
	Weight	14g	10g	12g
	Time	1h40min	1h16min	1h45min
Human skull	Length	5.11m	4.30m	5.60m
	Weight	15g	13g	17g
	Time	1h48min	1h40min	1h52min
Horse	Length	5.80m	3.51m	6.39m
	Weight	17g	10g	19g
	Time	2h03min	1h33min	2h12min
Max	Length	9.47m	6.43m	6.46m
	Weight	28g	19g	19g
	Time	3h03min	2h15min	2h15min
Human heart	Length	12.85m	10.74m	11.31m
	Weight	38g	32g	34g
	Time	4h34min	4h10min	4h21min
Knight	Length	5.57m	4.09m	5.65m
	Weight	17g	12g	17g
	Time	2h14min	1h42min	2h18min
Bunny	Length	7.52m	7.10m	8.63m
	Weight	22g	21g	26g
	Time	2h38min	2h30min	2h54min
Camel	Length	5.81m	3.88m	5.79m
	Weight	17g	12g	17g
	Time	2h08min	1h34min	2h10min
Cow	Length	6.86m	4.65m	6.73m
	Weight	20g	14g	20g
	Time	2h28min	2h04min	2h24min

Table 3.4.1: All tested surfaces present a smaller printint time, weight and length of material after applied a global rotation that minimizes equation 3.2.4. The surfaces *Arm, Camel, Horse, Bunny, Knight, Max and Cow* were chosen from the Libigl (JACOBSON et al., 2018) library and *heart* (ADOREZOOEY, 2015) and *skull* (AELLIS43, 2014) are Sketch Fab surfaces.

Alexander's method depends on the layer thickness z_t, the contact area of support structure, that has rectangular basis with sides of length p and q and a penalty value R for parts of surfaces that need support. So, putting $z_t, p, q = 0.1cm$ and $R = 10$, we were able to test this method and compare the results with ours, using the same surfaces that we used before.

As we can see in table 3.4.1, our method presents better length, weight of material and printing time if we compare with Alexander method in all surfaces tested except *Max*, that

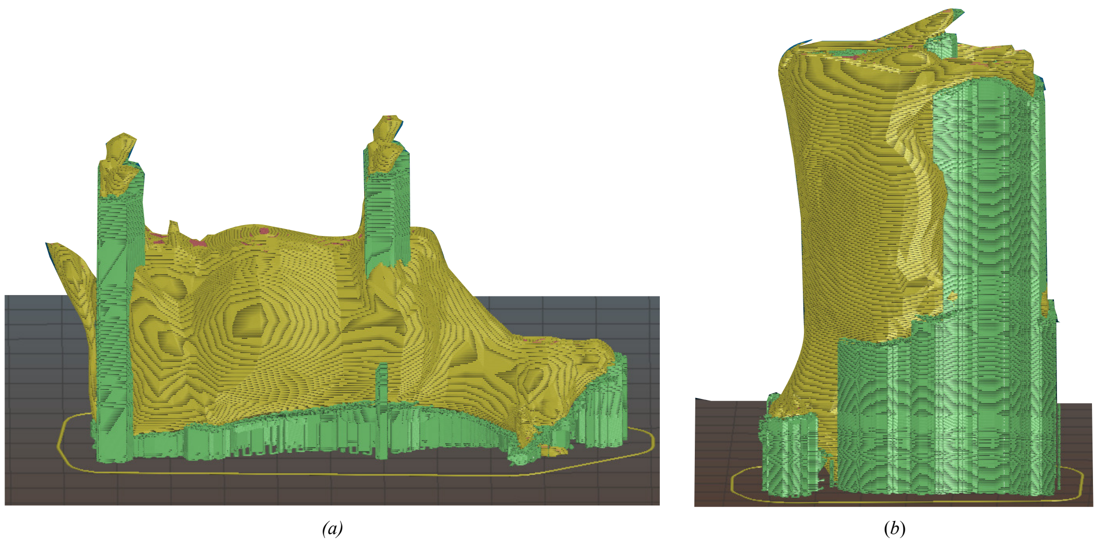

Figure 3.4.4: Libigl surface *cow.obj* after rotation using our method (a) and Alexander's method (b). As we can see, these results using our method generate less support for overhanging parts of surface then Alexander's method.

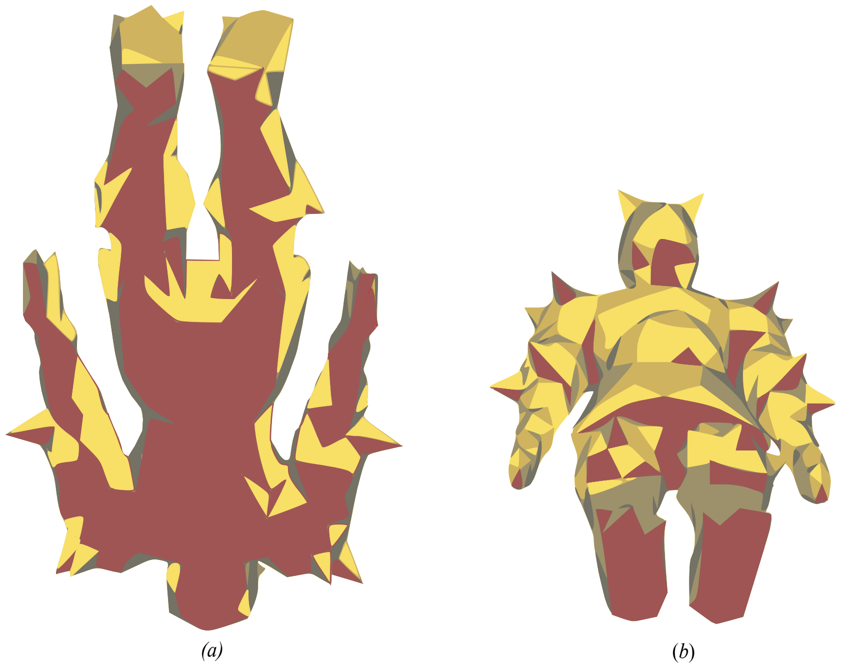

Figure 3.4.5: *decimated-knight.obj* surface after rotation using our method (a) and Alexander's method (b). As we can be seen, the result using our method generates more overhanging parts area (in brown) than Alexander's method.

presents the same result. In figure 3.4.4 we can make a visual comparison between ours and Alexander's results.

Although our results reduce the amount of support and printing time, in comparison with Alexander's method, their results reduce the contact area of support. In figure 3.4.5 the area that needs support of the *Knight* surface is painted in brown. However, while this may seem like a bad aspect of our method, there are several post-processing methods that completely eliminate artefacts caused by the support, such as the one developed in (HOUCK et al., 2019).

4 Optimization method

Before we study and apply an optimization method, let us expand our objective function to see what method we can use. We want to minimize

$$\min_{R} \sum_{f\in\mathscr{F}} \left[e_3^T \left(R \cdot \frac{1}{3} \left(v_{f_1} + v_{f_2} + v_{f_3} \right) \right) - min_z^{\mathscr{V}} \right] \cdot A_f \cdot \|R\cdot n(f) - Proj_{xy}(R\cdot n(f))\|^2,$$
$$s.t. \quad R \in SO(3)$$

Let us consider $n(f) = \begin{pmatrix} n_{f_x} \\ n_{f_y} \\ n_{f_z} \end{pmatrix}, v_{f_1} = \begin{pmatrix} x_{f_1} \\ y_{f_1} \\ z_{f_1} \end{pmatrix}, v_{f_2} = \begin{pmatrix} x_{f_2} \\ y_{f_2} \\ z_{f_2} \end{pmatrix}, v_{f_3} = \begin{pmatrix} x_{f_3} \\ y_{f_3} \\ z_{f_3} \end{pmatrix}, e_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$.
Consider also that the matrix $R \in SO(3)$ that we are looking for is a product of Euclidian axis rotation matrices $R_{e_1}(\theta_1)$ and $R_{e_2}(\theta_2)$, as shown in section 3. So, we are looking for a matrix $R = R_{e_2}(\theta_2)R_{e_1}(\theta_1)$, with $(\theta_1, \theta_2) \in (-\pi, \pi] \times \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$.

Next, since Euclidian axis rotation matrices $R_{e_1}(\theta_1)$ and $R_{e_2}(\theta_2)$ have the expressions

$$R_{e_1}(\theta_1) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_1) & -\sin(\theta_1) \\ 0 & \sin(\theta_1) & \cos(\theta_1) \end{pmatrix}$$

and

$$R_{e_2}(\theta_2) = \begin{pmatrix} \cos(\theta_2) & 0 & \sin(\theta_2) \\ 0 & 1 & 0 \\ -\sin(\theta_2) & 0 & \cos(\theta_2) \end{pmatrix}.$$

The matrix that we are looking for is

$$R = R_{e_2}(\theta_2)R_{e_1}(\theta_1) = \begin{pmatrix} \cos(\theta_2) & \sin(\theta_1)\sin(\theta_2) & \cos(\theta_1)\sin(\theta_2) \\ 0 & \cos(\theta_1) & -\sin(\theta_1) \\ -\sin(\theta_2) & \sin(\theta_1)\cos(\theta_2) & \cos(\theta_1)\cos(\theta_2) \end{pmatrix}. \tag{4.0.1}$$

Applying matrix (4.0.1) to the normal vector $n(f)$ we have

$$\begin{aligned}
R\cdot n(f) &= \begin{pmatrix} \cos(\theta_2) & \sin(\theta_1)\sin(\theta_2) & \cos(\theta_1)\sin(\theta_2) \\ 0 & \cos(\theta_1) & -\sin(\theta_1) \\ -\sin(\theta_2) & \sin(\theta_1)\cos(\theta_2) & \cos(\theta_1)\cos(\theta_2) \end{pmatrix} \begin{pmatrix} n_{f_x} \\ n_{f_y} \\ n_{f_z} \end{pmatrix} \\
&= \begin{pmatrix} \cos(\theta_2)n_{f_x} + \sin(\theta_2)\sin(\theta_1)n_{f_y} + \cos(\theta_1)\sin(\theta_2)n_{f_z} \\ \cos(\theta_1)n_{f_y} - \sin(\theta_1)n_{f_z} \\ -\sin(\theta_2)n_{f_x} + \cos(\theta_2)\sin(\theta_1)n_{f_y} + \cos(\theta_1)\cos(\theta_2)n_{f_z} \end{pmatrix}
\end{aligned}$$

So, since $R\cdot n(f) - Proj_{xy}(R\cdot n(f))$ results in 0 in the x and y coordinates and

$$-\sin(\theta_2)n_{f_x} + \cos(\theta_2)\sin(\theta_1)n_{f_y} + \cos(\theta_1)\cos(\theta_2)n_{f_z}$$

in the z-coordinate of the resultant vector, we have that its norm is

$$\begin{aligned}
\|R\cdot n(f) - Proj_{xy}(R\cdot n(f))\|^2 &= \left[-\sin(\theta_2)n_{f_x} + \cos(\theta_2)\sin(\theta_1)n_{f_y} + \cos(\theta_1)\cos(\theta_2)n_{f_z}\right]^2 \\
&= \sin^2(\theta_2)n_{f_x}^2 - 2\sin(\theta_2)\cos(\theta_2)\sin(\theta_1)n_{f_x}n_{f_y} + \\
&\quad + \cos^2(\theta_2)\sin^2(\theta_1)n_{f_y}^2 + 2\cos^2(\theta_2)\sin(\theta_1)\cos(\theta_1)n_{f_y}n_{f_z} - \\
&\quad - 2\sin(\theta_2)\cos(\theta_2)\cos(\theta_1)n_{f_x}n_{f_z} + \cos^2(\theta_1)\cos^2(\theta_2)n_{f_z}^2.
\end{aligned} \tag{4.0.2}$$

Also, since $\underbrace{\frac{1}{3}\left(v_{f_1} + v_{f_2} + v_{f_3}\right)}_{:=b_f} = \begin{pmatrix} \frac{x_{f_1}+x_{f_2}+x_{f_3}}{3} \\ \frac{y_{f_1}+y_{f_2}+y_{f_3}}{3} \\ \frac{z_{f_1}+z_{f_2}+z_{f_3}}{3} \end{pmatrix}$, we have that

$$e_3^T R\cdot b_f = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}^T \begin{pmatrix} \cos(\theta_2) & \sin(\theta_1)\sin(\theta_2) & \cos(\theta_1)\sin(\theta_2) \\ 0 & \cos(\theta_1) & -\sin(\theta_1) \\ -\sin(\theta_2) & \sin(\theta_1)\cos(\theta_2) & \cos(\theta_1)\cos(\theta_2) \end{pmatrix} \begin{pmatrix} \frac{x_{f_1}+x_{f_2}+x_{f_3}}{3} \\ \frac{y_{f_1}+y_{f_2}+y_{f_3}}{3} \\ \frac{z_{f_1}+z_{f_2}+z_{f_3}}{3} \end{pmatrix}$$

$$= -\sin(\theta_2)\left(\frac{x_{f_1}+x_{f_2}+x_{f_3}}{3}\right)+\sin(\theta_1)\cos(\theta_2)\left(\frac{y_{f_1}+y_{f_2}+y_{f_3}}{3}\right)+ \\ +\cos(\theta_1)\cos(\theta_2)\left(\frac{z_{f_1}+z_{f_2}+z_{f_3}}{3}\right).$$

Besides that, we need to subtract the minimal value of all z-coordinates of $\mathscr{V}$. So we can do that by subtracting from all terms of the sum the minimal z-coordinates value of $\mathscr{V}$ after applied rotation matrix R to it. We then obtain

$$e_3^T R\cdot b_f - min_z^{\mathscr{V}} = -\sin(\theta_2)\left(\frac{x_{f_1}+x_{f_2}+x_{f_3}}{3}\right)+\sin(\theta_1)\cos(\theta_2)\left(\frac{y_{f_1}+y_{f_2}+y_{f_3}}{3}\right)+ \\ +\cos(\theta_1)\cos(\theta_2)\left(\frac{z_{f_1}+z_{f_2}+z_{f_3}}{3}\right)-\min\left\{\left(\mathscr{V}\cdot R^T\right)\cdot e_3\right\}.$$

Notice that, if $\mathscr{V}\in\mathbb{R}^{n\times 3}$ and $R\in\mathbb{R}^{3\times 3}$, we have that the transposed vertices matrix $\mathscr{V}^T = (v_1\ \ v_2\ \ \cdots\ \ v_n)$, $v_i\in\mathbb{R}^3$, is a list of column vectors. When we apply the rotation matrix to $\mathscr{V}^T$ we have

$$R\cdot\mathscr{V}^T = R\cdot(v_1\ \ v_2\ \ \cdots\ \ v_n) = (Rv_1\ \ Rv_2\ \ \cdots\ \ Rv_n),$$

with $R\cdot v_j\in\mathbb{R}^3$ as the rotation of each vertice of the surface by R. Taking the transpose of $R\cdot\mathscr{V}^T$ we obtain $\left(R\cdot\mathscr{V}^T\right)^T = \mathscr{V}\cdot R^T\in\mathbb{R}^{n\times 3}$. Multiplying by $e_3=(0,0,1)^T$, we have that $\left(\mathscr{V}\cdot R^T\right)\cdot e_3\in\mathbb{R}^n$ and has the z-coordinates of all vertices of $\mathscr{V}$. Since we are working with surfaces that have their barycenters centered at the origin of $\mathbb{R}^3$, the function $\min\left\{\left(\mathscr{V}\cdot R^T\right)\cdot e_3\right\}$ has a negative value and, subtracting it in the function, will give us the height of each triangle as if the surface was on top of the printer plate.

Therefore, we can rewrite our objective function 3.2.4 as

$$\begin{aligned} \min_{\theta_1,\theta_2} \quad & \sum_{f\in\mathscr{F}} A_f T_f(\theta_1,\theta_2), \\ s.t. \quad & (\theta_1,\theta_2)\in(-\pi,\pi]\times\left[-\frac{\pi}{2},\frac{\pi}{2}\right] \end{aligned} \tag{4.0.3}$$

where $T_f(\theta_1,\theta_2) = \left(e_3^T R \cdot b_f - min_z^{\mathcal{V}}\right) \cdot \|R \cdot n(f) - Proj_{xy}(R \cdot n(f))\|^2$ and

$$\begin{aligned} T_f(\theta_1,\theta_2) \;=\; & \Bigg[-\sin(\theta_2)\left(\frac{x_{f_1}+x_{f_2}+x_{f_3}}{3}\right)+\sin(\theta_1)\cos(\theta_2)\left(\frac{y_{f_1}+y_{f_2}+y_{f_3}}{3}\right)+ \\ & +\cos(\theta_1)\cos(\theta_2)\left(\frac{z_{f_1}+z_{f_2}+z_{f_3}}{3}\right)-\min\left\{\left(\mathcal{V}\cdot R^T\right)\cdot e_3\right\}\Bigg] \\ & \Big[\sin^2(\theta_2)n_{f_x}^2+\cos^2(\theta_1)\cos^2(\theta_2)n_{f_z}^2-2\sin(\theta_2)\cos(\theta_2)\sin(\theta_1)n_{f_x}n_{f_y}+ \\ & +\cos^2(\theta_2)\sin^2(\theta_1)n_{f_y}^2+2\cos^2(\theta_2)\sin(\theta_1)\cos(\theta_1)n_{f_y}n_{f_z}- \\ & -2\sin(\theta_2)\cos(\theta_2)\cos(\theta_1)n_{f_x}n_{f_z}\Big]. \end{aligned} \tag{4.0.4}$$

4.1 The Matlab function *fmincon* and its optimization theory

Until now, our tests were searching for the minimal value of function 3.2.4 by searching a combination of angles $(\theta_1,\theta_2) \in (-\pi,\pi] \times \left[-\frac{\pi}{2},\frac{\pi}{2}\right]$ such that the rotation matrix $R = R_{e_2}(\theta_2)R_{e_1}(\theta_1) \in SO(3)$ is applied to the surface and generates a new global position in space that generates less support in the 3D printing process. By partitioning the angle ranges $(-\pi,\pi]$ and $\left[-\frac{\pi}{2},\frac{\pi}{2}\right]$ with a spacing of $\Delta\theta_1 = \Delta\theta_2 = \frac{\pi}{180}$ on each interval, we were able to evaluate the objective function with each combination of these angles and store the minimum value of all those function evaluations. With that, we find the minimum value of objective function 3.2.4 by evaluating 65341 angle combinations, giving us an slow method. The execution time of all tests can be seen in table 4.1.1.

Surface	$\|F\|$	Time	**Surface**	$\|F\|$	Time	**Surface**	$\|F\|$	Time
Arm	16618	839.92s	Max	10540	467.67s	Bunny	6966	299.01s
Human skull	4986	240.30s	Human heart	14720	687.55s	Camel	3576	210.65s
Horse	4796	282.88s	Knight	1000	56.89s	Cow	5520	251.59s

Table 4.1.1: Execution time of the algorithm that finds the smallest value of the function 3.2.4, for each surface tested. $|F|$ is the number of triangular faces of each mesh.

Replacing function 3.2.4 by function 4.0.3 was important to see how can we find the solution to this problem more efficiently. The problem presented in 4.0.3 seems to be a finite sum of a $\sin(\cdot)$ and $\cos(\cdot)$ combination with two variables (θ_1, θ_2) subject to a box condition $(\theta_1, \theta_2) \in (-\pi, \pi] \times \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$.

To solve the minimization problem presented by equation (4.0.3) we use a Matlab function called *fmincon*. This function is able to find a local minimum value of constrained nonlinear multivariable functions. We used the default parameter of *fmincon* that finds the solution of a minimization problem using an *Interior-point* algorithm with logarithmic barrier penalty method to deal with the constraints of the problem and descent direction given by Broyden–Fletcher–Goldfarb–Shanno (BFGS) Quasi-Newton method. Notice that our objective function 4.0.3 has a term

$$min_z^{\mathscr{V}} = \min\left\{\left(\mathscr{V} \cdot R^T\right) \cdot e_3\right\}$$

that is difficult to differentiate. This leads us to approximate the gradient of the function by the finite difference method.

Before we show our results with Matlab's *fmincon* function, let us see a little of optimization theory to understand how *fmincon* works. The main definitions, results and applications that we will use here can be seen in (POWELL, 1965), (NOCEDAL, WRIGHT, 2006), (IZMAILOV, SOLODOV, 2007), (BYRD, GILBERT, NOCEDAL, 2000), (BERAHAS, BYRD, NOCEDAL, 2019), among others.

4.1.1 Finite-difference approximation

This technique is based on Taylor's theorem (see (NOCEDAL, WRIGHT, 2006)). When $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ is twice differentiable and Lipschitz continuous, we have that

$$f(x+\varepsilon e_i) = f(x) + \varepsilon \frac{\partial f}{\partial x_i}(x) + \frac{1}{2}\varepsilon^2 \frac{\partial^2 f}{\partial x_i^2}(x) + O(\varepsilon^3)$$

and

$$f(x-\varepsilon e_i) = f(x) - \varepsilon \frac{\partial f}{\partial x_i}(x) + \frac{1}{2}\varepsilon^2 \frac{\partial^2 f}{\partial x_i^2}(x) + O(\varepsilon^3),$$

for an $\varepsilon > 0$ and $e_i = (0 \;\; 0 \;\; \cdots \;\; 1 \;\; \cdots \;\; 0)^T$ with one at its ith-coordinate, for each $i = 1, \ldots n$. Subtracting both equations, we have

$$f(x+\varepsilon e_i) - f(x-\varepsilon e_i) = 2\varepsilon \frac{\partial f}{\partial x_i}(x) + O(\varepsilon^3).$$

Dividing the remaining equation by 2ε, we got

$$\frac{f(x+\varepsilon e_i)-f(x-\varepsilon e_i)}{2\varepsilon}=\frac{\partial f}{\partial x_i}(x)+O(\varepsilon^2)$$

that gives us the *central-difference* formula, given by

$$\frac{\partial f_\varepsilon}{\partial x_i}(x)\approx\frac{f(x+\varepsilon e_i)-f(x-\varepsilon e_i)}{2\varepsilon}, \tag{4.1.1}$$

with a truncation error in order of $O(\varepsilon^2)$ for each $i=1,\dots,n$. So, putting each one into a vector, we define the *central-difference gradient* by

$$\nabla_\varepsilon f(x)=\begin{pmatrix}\frac{\partial f_\varepsilon}{\partial x_1}(x)\\ \frac{\partial f_\varepsilon}{\partial x_2}(x)\\ \vdots\\ \frac{\partial f_\varepsilon}{\partial x_n}(x).\end{pmatrix} \tag{4.1.2}$$

With that in mind, let us talk about some preliminary optmization theory. Here, we are looking for a point that is a global minimum value of equation 4.0.3.

4.1.2 Local and global minima and descent directions

A general formulation of a minimization problem subject to constraints on the variables is

$$\begin{aligned}\min_x &\quad f(x)\\ s.t. &\quad x\in\Omega,\end{aligned} \tag{4.1.3}$$

where $f:\mathbb{R}^n\longrightarrow\mathbb{R}$ and $\Omega\in\mathbb{R}^n$. In our case

$$\Omega=\left\{x=(x_1,x_2)\in\mathbb{R}^2\middle|-\pi<x_1\leq\pi\text{ and }-\frac{\pi}{2}\leq x_2\leq\frac{\pi}{2}\right\}.$$

That leads us to the following definition.

Definition 4.1.1 (Global and Local minimum)**.** *Consider the minimization problem 4.1.3.*

- *We say that $x_* \in \Omega$ is the global minimum of f in Ω if*

$$f(x_*) \leq f(x), \forall x \in \Omega.$$

- *We say that $x_* \in \Omega$ is a local minimum of f in Ω if there exists $\varepsilon \in \mathbb{R}, \varepsilon > 0$ such that*

$$f(x_*) \leq f(x), \forall x \in \Omega \cap B(x_*, \varepsilon),$$

where $B(x_*, \varepsilon) = \left\{x \in \Omega \mid \|x - x_*\| < \varepsilon\right\}$.

Definition 4.1.2. *Consider the minimization problem 4.1.3. We say that $d \in \mathbb{R}^n$ is a descent direction for f from a point $x_0 \in \mathbb{R}^n$ if there exists $\overline{\varepsilon} \in \mathbb{R}, \overline{\varepsilon} > 0$ such that*

$$f(x_0 + td) < f(x_0), \forall t \in (0, \overline{\varepsilon}].$$

Definition 4.1.3. *Consider the minimization problem 4.1.3. We say that a direction $d \in \mathbb{R}^n$ is feasible from a point $x_0 \in \Omega$ if there exists $\overline{t} \in \mathbb{R}, \overline{t} > 0$ such that $x + td \in \Omega, \forall t \in (0, \overline{t}]$.*

Proposition 4.1.1. *If $f \in \mathscr{C}^1$ and $\nabla f(x)^T d < 0$, then d is a descent direction.*

Proof. By Taylor's Theorem

$$\frac{f(x+td) - f(x)}{t} = \nabla f(x)^T d + \frac{o(|t| \cdot \|d\|)}{t},$$

for any $t \in (0,1)$.

Notice that, if there is an $\overline{\varepsilon} > 0$ is small enough so that $t < \overline{\varepsilon}$ and

$$\left|\frac{o(|t| \cdot \|d\|)}{t}\right| < \frac{1}{2}\left|\nabla f(x)^T d\right|,$$

then $\nabla f(x)^T d + \frac{o(|t| \cdot \|d\|)}{t} < \nabla f(x)^T d + \frac{1}{2}\nabla f(x)^T d = \frac{3}{2}\nabla f(x)^T d < 0$.

Therefore $\frac{f(x+td) - f(x)}{t} < 0 \Rightarrow f(x+td) < f(x)$, for $t \in (0, \overline{\varepsilon})$. ■

4.1.3 Barrier penalty method

In the general formulation of the minimization problem (4.1.3), suppose the constraints on the variables are described by

$$\Omega = \left\{ x \in \mathbb{R}^n \mid g(x) \leq 0 \right\},$$

where $g : \mathbb{R}^n \longrightarrow \mathbb{R}^m$. In our case, the functions in the constraint set are linear box conditions $x_1 - \pi \leq 0, -x_1 - \pi < 0, x_2 - \dfrac{\pi}{2} \leq 0$ and $-x_2 - \dfrac{\pi}{2} \leq 0$.

The main idea of internal penalty methods is that we can approximate the original problem by a sequence of unconstrained problems. We add a penalty function to the objective function that goes to infinity when any sequence approaches the boundary of the feasible set Ω.

Definition 4.1.4. *A function $\phi : \Omega \longrightarrow \mathbb{R}$ is called a* barrier function *of the set Ω if it is continuous on the interior of Ω and $\phi(x_k) \to +\infty \ (k \to \infty)$ for every sequence $\{x_k\} \subset \Omega$ such that $x_k \to \tilde{x}$ with $g_i(\tilde{x}) = 0$ for $i \in \{1, 2, \ldots, m\}$.*

In the case of *fmincon*, the logarithmic barrier function is used, i.e., if $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ is the objective function for minimization and there are constraints conditions $\Omega = \left\{ x \in \mathbb{R}^n \mid g(x) \leq 0 \right\}$ for the problem variables, the method adds the function $\phi : \Omega \longrightarrow \mathbb{R}$ defined by

$$\phi(x) = -\sum_{i=1}^{m} \ln\left(-g_i(x)\right)$$

to the objective function and converts the initial constrained minimization problem 4.1.3 into an unconstrained minimization problem

$$\begin{aligned} \min_{x} \quad & f(x) + \alpha\phi(x) \\ s.t. \quad & x \in \mathbb{R}^n, \end{aligned} \tag{4.1.4}$$

where $\alpha > 0$ is a penalty parameter.

The next two results will discuss the convergence of the use of algorithms using internal penalty barrier functions.

Proposition 4.1.2. *Let $\{x_k\}$ be a sequence generated by an algorithm using internal penalty barrier function, where x_k is a global solution of problem 4.1.4 with $\alpha = \alpha_k$ and $\alpha_{k+1} < \alpha_k, \forall k$. Then, for all k we have*

$$\phi(x_{k+1}) \geq \phi(x_k) \tag{4.1.5a}$$

$$f(x_{k+1}) \leq f(x_k) \tag{4.1.6a}$$

Proof. Given the optimality of x_{k+1} in 4.1.4 with $\alpha = \alpha_{k+1}$ we have

$$f(x_{k+1}) + \alpha_{k+1}\phi(x_{k+1}) \leq f(x_k) + \alpha_{k+1}\phi(x_k).$$

Also, by the optimality of x_k in 4.1.4 we have

$$f(x_k) + \alpha_k\phi(x_k) \leq f(x_{k+1}) + \alpha_k\phi(x_{k+1}).$$

Summing these two equations, we have

$$\begin{aligned}
f(x_{k+1}) + \alpha_{k+1}\phi(x_{k+1}) + f(x_k) + \alpha_k\phi(x_k) &\leq f(x_k) + \alpha_{k+1}\phi(x_k) + f(x_{k+1}) + \alpha_k\phi(x_{k+1}) \\
\alpha_k\phi(x_k) - \alpha_{k+1}\phi(x_k) &\leq \alpha_k\phi(x_{k+1}) - \alpha_{k+1}\phi(x_{k+1}) \\
(\alpha_k - \alpha_{k+1})\phi(x_k) &\leq (\alpha_k - \alpha_{k+1})\phi(x_{k+1})
\end{aligned}$$

Since $\alpha_{k+1} < \alpha_k$ we have that $\phi(x_k) \leq \phi(x_{k+1})$ and then

$$\begin{aligned}
f(x_{k+1}) + \alpha_{k+1}\phi(x_{k+1}) &\leq f(x_k) + \alpha_{k+1}\phi(x_k) \\
f(x_{k+1}) - f(x_k) &\leq \alpha_{k+1}\phi(x_k) - \alpha_{k+1}\phi(x_{k+1}) \\
&= \alpha_{k+1}(\phi(x_k) - \phi(x_{k+1})) \\
&\leq 0.
\end{aligned}$$

■

This leads us to the convergence theorem.

Theorem 4.1.1. *Let $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ and $g : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ be continuous functions. Suppose that the constraint set $\Omega = \{x \in \mathbb{R}^n | g(x) \leq 0\}$ is such that $int(\Omega) = \{x \in \mathbb{R}^n | g(x) < 0\} \neq \emptyset$ and*

$$\inf_{x \in int(\Omega)} f(x) = \inf_{x \in \Omega} f(x) > -\infty.$$

Then all accumulation points of a sequence $\{x_k\}$ generated by an algorithm using internal penalty barrier method, where x_k is a global solution of the problem 4.1.4 with $\alpha = \alpha_k$ and $\alpha_k \to 0 \ \ (k \to \infty)$ is a global solution of problem 4.1.3.

Proof. Let $\bar{x} \in D$ be an accumulation point of the sequence $\{x_k\}$. If $f(\bar{x}) = \inf_{x \in int(\Omega)} f(x)$, then the proof is over. If $f(\bar{x}) > \underbrace{\inf_{x \in int(\Omega)} f(x)}_{:=v}$ let $p \in \mathbb{N}^*$ such that we can define

$$\delta = \frac{(f(\bar{x}) - v)}{p} > 0.$$

and take a $\tilde{x} \in int(\Omega)$ such that $f(\tilde{x}) \leq v - \delta$.

By equation 4.1.6 in proposition 4.1.2, the sequence $\{f(x_k)\}$ is non-increasing. Then the sequence $\{f(x_{k_i})\}$, with $x_{k_i} \to \bar{x} \ \ (i \to \infty)$ is also non-increasing and

$$f(x_{k_i}) \geq \lim_{i \to \infty} f(x_{k_i}) = f(\bar{x}) = v + p\delta.$$

Now, since $f(\tilde{x}) \leq v - \delta$ and $f(x_{k_i}) \geq v + p\delta$ we have that

$$f(x_{k_i}) - f(\tilde{x}) \geq \delta - v + v + p\delta = (1+p)\delta > \delta.$$

So, as x_{k_i} is a global solution for 4.1.4 with i sufficiently large and by equation 4.1.5, we have that

$$\begin{aligned} 0 &\geq f(x_{k_i}) + \alpha_{k_i}\phi(x_{k_i}) - f(\tilde{x}) - t_{k_i}\phi(\tilde{x}) \\ &> f(x_{k_i}) - f(\tilde{x}) + t_{k_i}\phi(x_{k_i}) - \alpha_{k_i}\phi(\tilde{x}) \\ &> \delta + \alpha_{k_i}\underbrace{(\phi(x_{k_i}) - \phi(\tilde{x}))}_{>0 \text{ (by eq. 4.1.5)}} \\ &> \delta > 0. \end{aligned}$$

So, since $0 < \delta \leq 0$ and $f(\bar{x}) = v + p\delta$, we have that $f(\bar{x}) = v = \inf_{x \in \Omega} f(x)$. ■

4.1.4 The BFGS method

The Matlab function *fmincon* uses the BFGS method to find a descent direction to solve optimization problems. This method was developed by Broyden, Fletcher, Goldfarb, and Shanno as a Quasi-Newton method that only requires the gradient (or its approximation) of the objective function to find a descent direction.

Let x_k be the result of a minimization step of an optimization problem. By Taylor's Theorem, we can approximate $f(x)$ by

$$f(x) \approx f(x_k) + \nabla f(x_k)(x - x_k) + \frac{1}{2}(x - x_k)^T \nabla^2 f(x_k)(x - x_k).$$

Defining the quadratic model $q_k(x) = f(x_k) + \nabla f(x_k)(x - x_k) + \frac{1}{2}(x - x_k)^T \nabla^2 f(x_k)(x - x_k)$, we have that if $q_k(\overline{x_k})$ is the local minimum of the quadratic model, we have that

$$0 = \nabla(q_k(\overline{x_k})) = \nabla^2 f(x_k) \underbrace{(\overline{x_k} - x_k)}_{d_k} + \nabla f(x_k).$$

So, we can write explicitly the solution of the minimizer of this quadratic model as

$$d_k = -\left(\nabla^2 f(x_k)\right)^{-1} \nabla f(x_k) \tag{4.1.7}$$

and d_k is used as a descent direction in the optimization problem

$$x_{k+1} = x_k + t_k d_k,$$

where the step size $0 < t_k \leq 1$ is chosen to satisfy Wolfe conditions

$$f(x_k + t_k d_k) \leq f(x_k) + \lambda_1 t_k \nabla f(x_k)^T d_k \tag{4.1.8a}$$

$$\nabla f(x_k + t_k d_k)^T d_k \geq \lambda_2 \nabla f(x_k)^T d_k, \tag{4.1.8b}$$

where $0 \leq \lambda_1 < \lambda_2 < 1$.

The idea of BFGS Quasi-Newton method is that we can use an approximation $B_k \in \mathbb{R}^{n \times n}$ of the Hessian $\nabla^2 f(x_k)$ that is computed at every iteration by calculating

$$B_{k+1} = B_k + \frac{\left(\nabla f(x_{k+1}) - \nabla f(x_k)\right)\left(\nabla f(x_{k+1}) - \nabla f(x_k)\right)^T}{\left(\nabla f(x_{k+1}) - \nabla f(x_k)\right)^T (x_{k+1} - x_k)} - \frac{B_k (x_{k+1} - x_k)(x_{k+1} - x_k)^T B_k}{(x_{k+1} - x_k)^T B_k (x_{k+1} - x_k)}. \tag{4.1.9}$$

To calculate $(B_{k+1})^{-1}$, we can use Sherman– Morrison–Woodbury formula (see (NO-

CEDAL, WRIGHT, 2006) in Appendix A.28 section for more information), that gives us

$$
\begin{aligned}
(B_{k+1})^{-1} \quad = \quad & \left(I - \frac{(x_{k+1}-x_k)\left(\nabla f(x_{k+1}) - \nabla f(x_k)\right)^T}{\left(\nabla f(x_{k+1}) - \nabla f(x_k)\right)^T (x_{k+1}-x_k)} \right) (B_k)^{-1} \left(I - \frac{\left(\nabla f(x_{k+1}) - \nabla f(x_k)\right)(x_{k+1}-x_k)^T}{\left(\nabla f(x_{k+1}) - \nabla f(x_k)\right)^T (x_{k+1}-x_k)} \right) \\
& + \frac{(x_{k+1}-x_k)(x_{k+1}-x_k)^T}{\left(\nabla f(x_{k+1}) - \nabla f(x_k)\right)^T (x_{k+1}-x_k)}
\end{aligned}
\tag{4.1.10}
$$

Observation 4.1.1. *Notice that, if the matrix $(\nabla^2 f(x))^{-1} \in \mathbb{R}^{n\times n}$ or its approximation $B_k \in \mathbb{R}^{n\times n}$ are* symmetric positive definite (S.P.D.) *matrices, the direction given by equation 4.1.7 is a descent direction. Indeed, by definition, if $A \in \mathbb{R}^{n\times n}$ is S.D.P., $x^T A x > 0, \forall x \in \mathbb{R}^n, x \neq 0$. So, by proposition 4.1.1, if $\nabla f(x)^T d < 0$ then d is a descent direction. Taking $d = d_k = -\left(\nabla^2 f(x_k)\right)^{-1} \nabla f(x_k)$ or $d = d_k = -(B_k)^{-1} \nabla f(x_k)$ we have that*

$$
\nabla f(x_k)^T d_k = -\underbrace{\nabla f(x_k)^T \left(\nabla^2 f(x_k)\right)^{-1} \nabla f(x_k)}_{>0} < 0
$$

or

$$
\nabla f(x_k)^T d_k = -\underbrace{\nabla f(x_k)^T (B_k)^{-1} \nabla f(x_k)}_{>0} < 0.
$$

The next proposition will guarantee that all $B_i \in \mathbb{R}^{n\times n}$ of each iterate of BFGS method are S.P.D. matrices.

Proposition 4.1.3. *In BFGS updating iterate, if $B_k \in \mathbb{R}^{n\times n}$ is S.P.D. and*

$$
\left(\nabla f(x_{k+1}) - \nabla f(x_k)\right)^T (x_{k+1} - x_k) > 0,
$$

then $B_{k+1} \in \mathbb{R}^{n\times n}$ is S.P.D..

Proof. Defining $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$ and $s_k = (x_{k+1} - x_k)$, to show that B_{k+1} is S.P.D., we need to show that for any vector $v \in \mathbb{R}^n$ $(v \neq 0)$, $v^T B_{k+1} v > 0$.

Considering any $v \in \mathbb{R}^3, v \neq 0$ and using the BFGS formula for B_{k+1} 4.1.9, we have:

$$\begin{aligned} v^T B_{k+1} v &= v^T B_k v + \frac{\left(v^T y_k\right)^2}{\underbrace{y_k^T s_k}_{>0}} - \frac{\left(v^T B_k s_k\right)^2}{\underbrace{s_k^T B_k s_k}_{>0}} \\ &= \frac{\left(v^T y_k\right)^2}{y_k^T s_k} + \frac{(v^T B_k v)(s_k^T B_k s_k) - \left(v^T B_k s_k\right)^2}{s_k^T B_k s_k} \end{aligned}$$

By Cauchy-Schwarz, we have that

$$\left|v^T B_k s_k\right|^2 \leq \left|v^T B_k v\right| \cdot \left|s_k^T B_k s_k\right|.$$

So, we have two possibilities:

- if $\left(v^T B_k v\right)\left(s_k^T B_k s_k\right) = \left(v^T B_k s_k\right)^2$ then

$$v^T B_{k+1} v = \frac{\left(v^T y_k\right)^2}{y_k^T s_k} > 0;$$

- if $\left(v^T B_k s_k\right)^2 < \left(v^T B_k v\right)\left(s_k^T B_k s_k\right)$ then

$$\left(v^T B_k v\right)\left(s_k^T B_k s_k\right) - \left(v^T B_k s_k\right)^2 > 0$$

and then

$$v^T B_{k+1} v = \frac{\left(v^T y_k\right)^2}{y_k^T s_k} + \frac{(v^T B_k v)(s_k^T B_k s_k) - \left(v^T B_k s_k\right)^2}{s_k^T B_k s_k} > 0.$$

In all cases, we have that $v^T B_{k+1} v > 0$ and, therefore, B_{k+1} is S.P.D.. ■

Observation 4.1.2. *It is well known from Linear Algebra that if B_{k+1} is S.P.D., then $(B_{k+1})^{-1}$ is also symmetric positive definite. For more informations, see (NOCEDAL, WRIGHT, 2006) or any Linear Algebra book.*

The global convergence of methods that use BFGS to find a descent direction is guaranteed by the next theorem.

Theorem 4.1.2. *Let $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ differentiable and $\|\nabla f(x)\| < L\|x\|$ for a $L > 0$. Suppose that the step of any algorithm with BFGS uses Wolfe conditions to calculate its step size. Suppose,* *also, that the matrices $(B_k)^{-1}$ produced by this algorithm satisfy*

$$\| (B_k)^{-1} \| \leq \beta \text{ and } \langle (B_k)^{-1} d, d \rangle \geq \gamma \|d\|^2,$$

for some $\beta > 0$ and $\gamma > 0$ and $\forall d \in \mathbb{R}^n$.

Then, the sequence $\{x_k\}$ generated by this algorithm has an accumulation point or if the function f is bounded below in $\mathbb{R}^n$, we have

$$\{\nabla f(x_k)\} \to 0 \ \ (k \to \infty).$$

Proof. Suppose that $\nabla f(x_k) \neq 0, \forall k$. By Wolfe condition 4.1.8, we have that

$$\begin{aligned} f(\overbrace{x_k + t_k d_k}^{x_{k+1}}) &\leq f(x_k) + \lambda_1 t_k \nabla f(x_k)^T d_k \\ &= f(x_k) - \lambda_1 t_k \nabla f(x_k)^T (B_k)^{-1} \nabla f(x_k). \end{aligned}$$

Notice that $\langle (B_k)^{-1} d, d \rangle \geq \gamma \|d\|^2, \forall d \in \mathbb{R}^n$. So, this inequality holds for $\nabla f(x_k)$. Then,

$$\begin{aligned} f(\overbrace{x_k + t_k d_k}^{x_{k+1}}) &= f(x_k) - \lambda_1 t_k \nabla f(x_k)^T (B_k)^{-1} \nabla f(x_k) \\ &\leq f(x_k) - \lambda_1 t_k \gamma \|\nabla f(x_k)\|^2 \\ &= f(x_k) - \lambda_1 t_k \gamma \|B_k \left((B_k)^{-1} \nabla f(x_k) \right) \|^2 \\ &= f(x_k) - \lambda_1 t_k \gamma \|B_k d_k\|^2 \\ &\leq f(x_k) - \lambda_1 t_k \gamma \|B_k\|^2 \|d_k\|^2 \end{aligned}$$

Notice that, if $\|B_k^{-1}\| \leq \beta$ then $\|B_k\| \geq \dfrac{1}{\beta}$. Indeed, as $B_k \cdot B_k^{-1} = I_{n \times n}$, we have that $\|B_k \cdot B_k^{-1}\| = \|I_{n\times n}\| = 1$. So, using the statement $\|B_k^{-1}\| \leq \beta$, we have that

$$1 = \|B_k \cdot B_k^{-1}\| \leq \|B_k\| \|B_k^{-1}\| \leq \|B_k\| \beta \Rightarrow$$

$$\Rightarrow \frac{1}{\beta} \leq \|B_k\|.$$

So, we have that $\|B_k\| \geq \dfrac{1}{\beta} \Rightarrow \|B_k\|^2 \geq \dfrac{1}{\beta^2} \Rightarrow -\|B_k\|^2 \leq -\dfrac{1}{\beta^2}$. Then

$$\begin{aligned} f(x_{k+1}) &\leq f(x_k) - \lambda_1 t_k \gamma \|B_k\|^2 \|d_k\|^2 \\ &\leq f(x_k) - \lambda_1 t_k \frac{\gamma}{\beta^2} \|d_k\|^2. \end{aligned}$$

Since the step size satisfies $0 < t_k$ and the sequence $\{t_k\}$ does not converge to 0, we have that we can find an $\overline{\varepsilon} > 0$ such that $|t_k - 0| = |t_k| > \overline{\varepsilon}, \forall k$. So, it is always possible to choose $t = \dfrac{\max\{\overline{\varepsilon}, t_1, t_2, \ldots, t_k\}}{2}$ such that $t_k \geq t > 0$ that does not depend on k. So, since the sequence $\{f(x_k)\}$ is non-increasing, if f has a lower bound or if $\{x_k\}$ has an accumulation point, we have that $f(x_{k+1}) - f(x_k) \to 0 \ (k \to \infty)$ and then $\{d_k\} \to 0 \ (k \to \infty)$.

Since $d_k = -(B_k)^{-1} \nabla f(x_k)$ we have that $\nabla f(x_k) = -B_k d_k \to 0 \ (k \to \infty)$. ■

We can now present the algorithm used by Matlab function *fmincon*, set as default, that uses barrier penalty method for the constraints, *central-difference* method to approximate the gradient of the objective function and Quasi-Newton BFGS method to find a descent direction:

Algorithm 1: *fmincon* algorithm for problem 4.0.3

```
initialization;
$\mathscr{V}; \mathscr{F}; k \longleftarrow 0; TOL > 0; \alpha_0 \in \mathbb{R}; x_0 = (x_1^0, x_2^0)^T \in \Omega;$
$F_f(x_0) = A_f T_f(x_0)$ as in 4.0.4;
$\phi(x_0) = \alpha_0 \left( \ln\left(\pi - x_1^0\right) + \ln\left(\pi + x_1^0\right) + \ln\left(\frac{\pi}{2} - x_2^0\right) + \ln\left(\frac{\pi}{2} + x_2^0\right) \right);$
$g(x_0) = \sum_{f \in \mathscr{F}} F_f(x_0) - \phi(x_0);$
$\nabla_\varepsilon g(x_0)$ as in 4.1.2;
Inverse Hessian approximation $B_0^{-1} \in \mathbb{R}^{n \times n}$;
$d_0 = -(B_0)^{-1} \nabla_\varepsilon g(x_0);$
while $\|d_k\| > TOL$ or $-\nabla_\varepsilon g(x_k)^T d_k > TOL$ or $\|\nabla_\varepsilon g(x_0)\| > TOL$ do
    1. Compute search direction
           $d_k = -(B_k)^{-1} \nabla_\varepsilon g(x_k);$
    2. Set $x_{k+1} = x_k + t_k d_k$ where $t_k$ satisfies the Wolfe conditions 4.1.8;
    3. Compute $(B_{k+1})^{-1}$ by 4.1.10;
    4. Choose $\alpha_{k+1} < \alpha_k$;
    5. Compute $\nabla_\varepsilon g(x_{k+1})$ as in 4.1.2;
    6. $k \longleftarrow k + 1$;
    7. Return to step 1;
end
```

4.2 Optimization Results

We saw in the previous section that with *fmincon*, we are able to find a local minimum for the problem 4.0.3. Looking at the graph of the function 4.0.3 in figure 4.2.1, applied to the

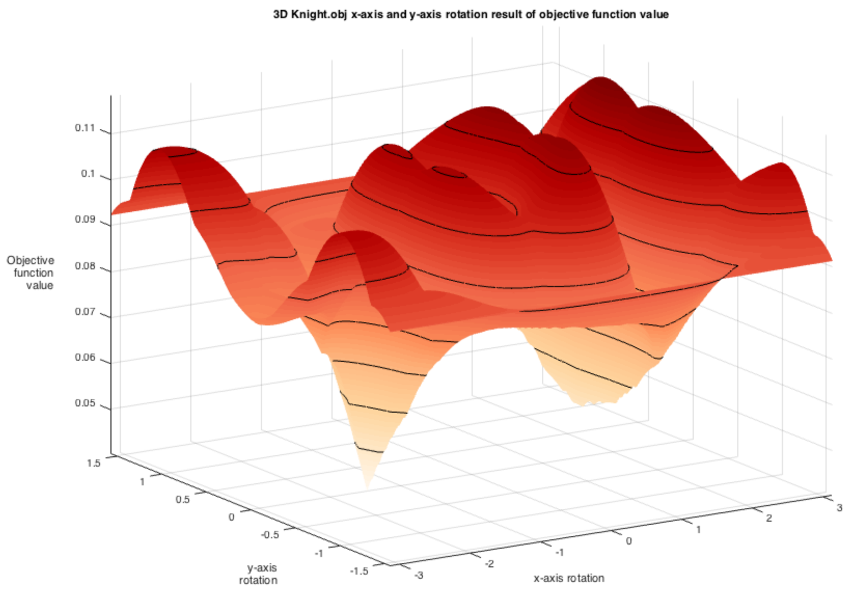

Figure 4.2.1: Objective function 4.0.3 3D graph applied on Knight Libigl (JACOBSON et al., 2018) surface.

Knight surface, we notice that the objective function is not convex and has some different local minima. So, if we use a function that calculates a local minimum, with an initial point that is sufficiently far from the global minimum, the solver will find a local minimum, instead of the global one. We can see in figure 4.2.2 that, if we choose an initial point that is not close enough

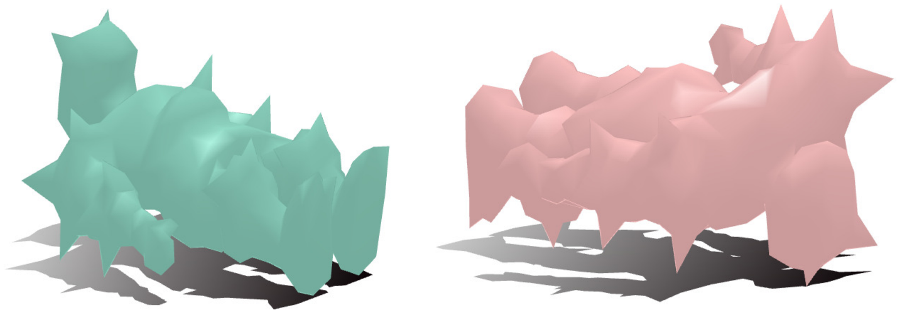

Figure 4.2.2: Knight surface from Libigl (JACOBSON et al., 2018) after global rotation on tests performed on section 3.4 (left) and after finding a local minimum using *fmincon* Matlab function with a chosen initial point (right). The teal surface on the left would be printed in 136 minutes with 3g of material loss (support that would be wasted) and the pink surface on the right would be printed in 149 minutes with 5g of material loss.

of global minimum, *fmincon* find a local minimum but this is not the global minimum, since we can find another surface position that improves the obtained result..

In the tests in section 3.4 we only find global minimum candidates for each surface. So, to be able to find the same (or better) results that we found in the previous tests, we peform different strategies using *fmincon* minimization solver. The tests that we did were based on multistart points for global search. Notice that, the angle range of the objective function can be easily divided in smaller intervals, with size $h > 0$. So, to the first multistart strategy, we take the central point of each subset of angle ranges as starting point, as we can see in figure 4.2.3.

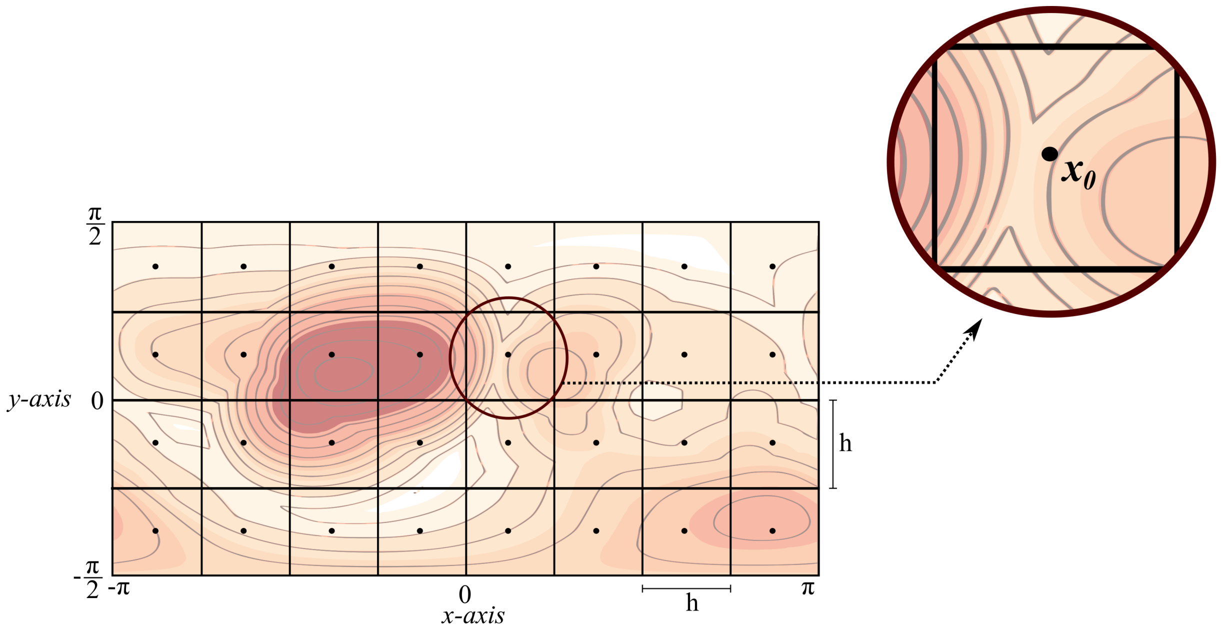

Figure 4.2.3: Dividing the angle ranges on the restrictions of equation 4.0.3 and taking the central point of each set, we got multiple start point of our algorithm.

Setting $h = \dfrac{\pi}{4}$, we have 32 different initial points and we got results that are close to what was found in the previous tests, but much faster. Also, these results has the same amount of *length of material, weight and printing time* as compared to the tests done in the section 3.4 and was presented on table 3.4.1, what indicates that these points could be the global minimum of 4.0.3.

The second test that we did was also a multistart method, inspired by the multi-start method presented in (MARTÍ, RESENDE, RIBEIRO, 2013), that chooses random starting points to the minimization problem. In every test we have different multiple start points. The final result has reached the same minimum points as presented in table 3.4.1 in all surfaces and every time that we run the algorithm. We can see in figure 4.2.5 one of the tests with randomly chosen starting points (R.C.S.P.), with 32 initial points. We can see some of optimization results in figure 4.2.4

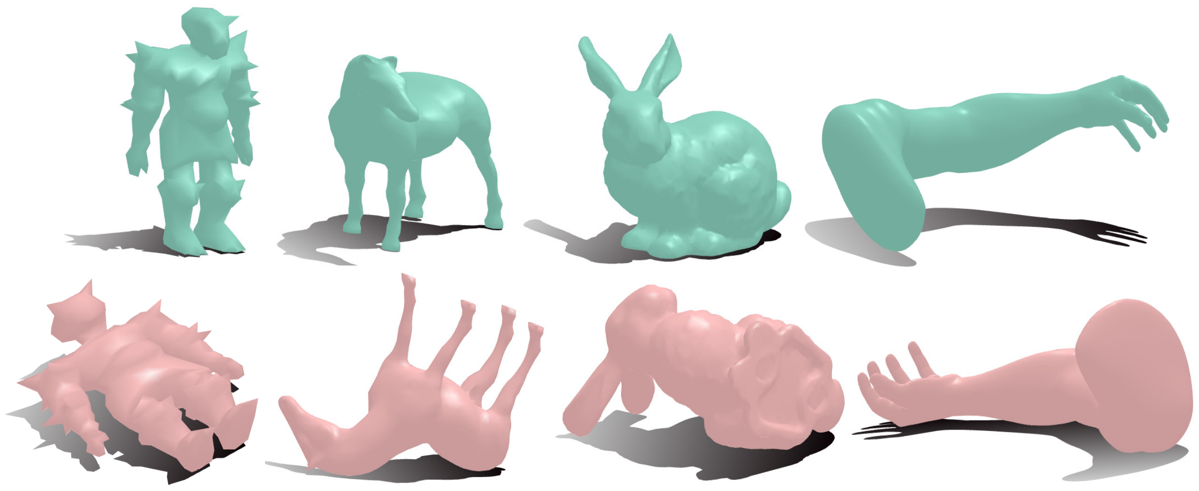

Figure 4.2.4: Surfaces on initial position (top) and after optimal rotation (bottom). *Knight, Horse, Bunny and Arm* are surfaces available on Libigl (JACOBSON et al., 2018).

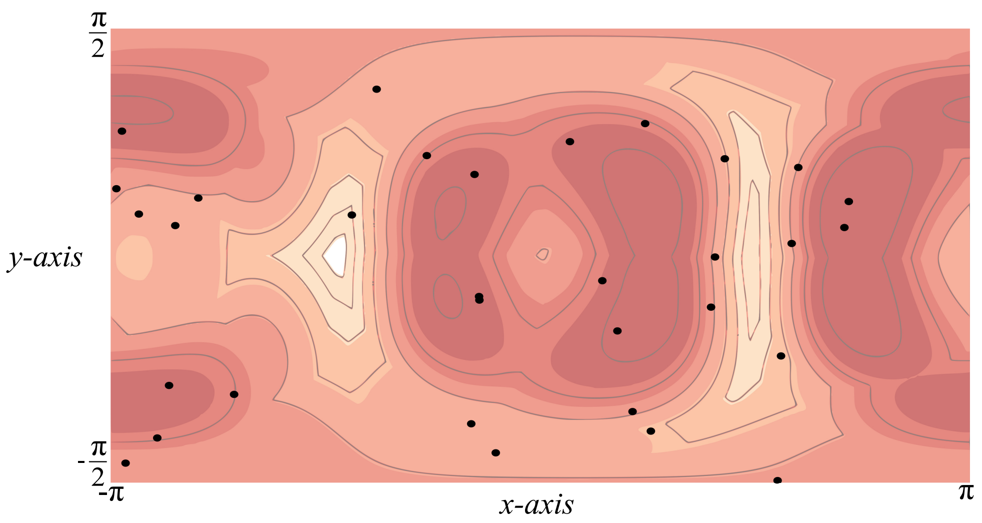

Figure 4.2.5: Graph of randomly chosen starting points (R.C.S.P.) strategy applied on Knight Libigl surface, with 100 starting points. Each black spot on graphic is an initial point that the algorithm choose randomly as the start point to minimization problem.

The table with the optimal global rotation angles and the time spent of multiple constraint division (M.C.D.), randomly chosen starting points (R.C.S.P.) strategies and the tests presented in section 3.4 to find the global minimum of problem 4.0.3 are available in table 4.2.1.

Surface	x_{min}	Time Tests (sec 3.4)	Time (M.C.D.)	Time (R.C.S.P.)
Arm	$\begin{pmatrix} -2.7444 & 0.0799 \end{pmatrix}$	839.97s	17.56s	17.79s
Human skull	$\begin{pmatrix} 3.1004 & -0.4287 \end{pmatrix}$	240.30s	9.46s	9.17s
Horse	$\begin{pmatrix} -1.4316 & 0.2639 \end{pmatrix}$	282.88s	10.91s	9.68s
Max	$\begin{pmatrix} -2.9084 & -0.0235 \end{pmatrix}$	467.67s	7.16s	9.95s
Human heart	$\begin{pmatrix} -2.6011 & 0.0370 \end{pmatrix}$	687.55s	13.03s	12.94s
Knight	$\begin{pmatrix} -1.4726 & 0.0074 \end{pmatrix}$	56.89s	5.21s	4.36s
Bunny	$\begin{pmatrix} -0.1788 & 0.5579 \end{pmatrix}$	299.01s	16.41s	12.91s
Camel	$\begin{pmatrix} 2.9598 & -1.4669 \end{pmatrix}$	210.65s	5.16s	5.35s
Cow	$\begin{pmatrix} -1.5708 & -0.0924 \end{pmatrix}$	251.59s	6.81s	7.37s

Table 4.2.1: This table presents the run time of both strategies to seek the global minimum point and the tests presented in section 3.4. On multiple constraint division (M.C.D.) strategy, we divide the angle intervals into smaller with $h = \frac{\pi}{4}$ spacing on both axes, having 32 different initial points. On randomly chosen starting point (R.C.S.P.) strategy, we collect 32 random points as start point of our algorithm. On tests presented in section 3.4 we run 65341 angles combination and select the one with the lower objective function value.

5 Conclusion

5.1 Review

In this section, we review the main aspects of our work. We first reviewed some of the main concepts of orthogonal and rigid transformations, presenting results that served as basis for modelling our problem. With these concepts, in particular the theorem of rotation matrices representation with Euler sequences 2.2.2, we could identify the variables of our problem in order to create a minimization problem.

We modeled our minimization problem based on preliminary concepts that were reviewed before. We presented two ways to model the overhanging problem in 3D printing. The angle-based approach shows that we can reduce the suspended area of the surface, but this does not mean a reduction of support in 3D printing. This model presents better results with some surfaces but not in general. So, we formulate a model that considers the the z-coordinate norm of the normal surface vector with a weight factor that takes into consideration the height of each triangle on mesh. This modelling of the problem presents better results in all surfaces tested, compared with initial positions of surfaces, in terms of printing time and support quantity. With a quick review of (ALEXANDER, ALLEN, DUTTA, 1998) method to surface position, we were able to compare both methods and conclude that our method has better results compared with theirs.

With the modelled objective function 3.2.4 and its explicit representation 4.0.3, we reviewed the concepts used by the Matlab function *fmincon*, that finds a local minimum of the objective function with logarithmic barrier function to deal with the constraints of the problem, finite differences to approximate the gradient of the function and BFGS method to find a descent direction for the algorithm. We then apply *fmincon* to our objective function with two multistart strategies for initial points: multiple constraint division (M.C.D.), that divides the constraint domain of problem variables into smaller intervals and take the center of this set as start point of algorithm; randomly chosen starting points (R.C.S.P), that randomly chooses starting points on constraint domain of problem and run the algorithm with them. With them, we find strong candidates of global solution to the minimization problem, whose tests showed to be very close to the tests previously performed.

In conclusion, we develop a method that finds a global rotation of the surfaces that reduces the amount of support in 3D printing and presents better results than previous work.

5.2 Future work

We think that the work developed in this thesis has space for better results and the main future works that can be developed are:

- **Develop all Matlab code in C/C++**: All tests and optmization codes were done in Matlab. It would be interesting for future applications to develop the method proposed here in a more robust programming language such as C ++.

- **Fully differentiable weight function**: On this thesis, we consider a weight function for each triangle on the surface mesh, that takes into consideration the triangle height with respect to the printer plate. This weight function has a factor that is difficult to differentiate and forces us to approximate the gradient on minimization method with finite differences. So, it will be interesting to develop a weight function that is easy to differentiate, leading to a much precise and faster optmization method to solve the problem.

- **Global optimization method**: We used Matlab's *fmincon* function to solve the minimization problem, but this method finds a local minimum of the objective function. Even using two multistart strategies to initialize our minimization search and always finding the same values for all tests and surfaces, this does not prove that the solution found for the problem is, indeed, the global minimum of the objective function. We intend to investigate global optimization methods to apply into this minimization problem or try to prove that the methods that we used so far are sufficient to find a global minimum.

- **Horizontal partitioning of the surface:** Even after finding a position in space that minimizes the amount of support in 3D printing, some of the overhanging parts can still remain at a distance from the printer plate that can generate a lot of support. We believe that one way to solve this problem is to find an optimal height for a horizontal partition of the surface. We consider a horizontal partition because the seam that will separate the two parts of the partitioned surface will be in the same direction as the printing direction and could be hidden.

References

ADOREZOOEY. Human heart. *Sketchfab*, Paris, 2015. Available from: https://sketchfab.com/3d-models/human-heart-a294c26cb0e34c8f97d72e4dc4e9fdd5. Access in: nov. 29, 2019.

AELLIS43. Human skull (untextured). *Sketchfab*, Paris, 2014. Available from: https://sketchfab.com/3d-models/human-skull-untextured-6521abbdccb24e11915e7d9c7905e719. Access in: nov. 29, 2019.

ALEXANDER, P.; ALLEN, S.; DUTTA, D. Part orientation and build cost determination in layered manufacturing. *Computer-Aided Design*, Elsevier, v. 30, n. 5, p. 343–356, 1998.

BERAHAS, A. S.; BYRD, R. H.; NOCEDAL, J. Derivative-free optimization of noisy functions via quasi-newton methods. *SIAM Journal on Optimization*, SIAM, v. 29, n. 2, p. 965–993, 2019.

BOLLAPRAGADA, R.; WILD, S. M. Adaptive sampling quasi-newton methods for derivative-free stochastic optimization. *arXiv preprint arXiv:1910.13516*, 2019.

BORWEIN, J.; LEWIS, A. S. *Convex analysis and nonlinear optimization: theory and examples*. New York: Springer Science & Business Media, 2010.

BUSS, S. R. *3D computer graphics: a mathematical introduction with OpenGL*. United Kingdom: Cambridge University Press, 2003.

BYRD, R. H.; GILBERT, J. C.; NOCEDAL, J. A trust region method based on interior point techniques for nonlinear programming. *Mathematical programming*, Springer, v. 89, n. 1, p. 149–185, 2000.

CARMO, M. P. d. *Geometria Diferencial de Curvas e Superficies, 2a ediçao*. Rio de Janeiro: Sociedade Brasileira de Matemática, 2006.

CHEN, X. Superlinear convergence of smoothing quasi-newton methods for nonsmooth equations. *Journal of Computational and Applied Mathematics*, Elsevier, v. 80, n. 1, p. 105–126, 1997.

DAI, C. et al. Support-free volume printing by multi-axis motion. *ACM Transactions on Graphics (TOG)*, ACM, v. 37, n. 4, p. 134, 2018.

DENOYEL, A.; PINSON, C.; PASSET, P.-A. Sketchfab. *Sketchfab*, Paris, 2012. Available from: https://sketchfab.com/. Access in: nov. 29, 2019.

ELSERMAN, M.; BRUIJN, E. de; WIJNIA, S. *Ultimaker*. 2011. Https://ultimaker.com.

GOMES, J.; VELHO, L.; SOUSA, M. C. *Computer graphics: theory and practice*. United States: AK Peters/CRC Press, 2012.

GOSAVI, A.; PHATAKWALA, S. A finite-differences derivative-descent approach for estimating form error in precision-manufactured parts. *Journal of Manufacturing Science and Engineering*, American Society of Mechanical Engineers, v. 128, n. 1, p. 355–359, 2006.

HOUCK, H. A. et al. Light-stabilized dynamic materials. *Journal of the American Chemical Society*, ACS Publications, v. 141, n. 31, p. 12329–12337, 2019.

HU, R. et al. Approximate pyramidal shape decomposition. *ACM Trans. Graph.*, Citeseer, v. 33, n. 6, p. 213–1, 2014.

IZMAILOV, A.; SOLODOV, M. *Otimização, volume 2: métodos computacionais*. Rio de Janeiro: IMPA, 2007.

JACOBSON, A. et al. *gptoolbox: Geometry Processing Toolbox*. 2018. Http://github.com/alecjacobson/gptoolbox.

JACOBSON, A.; PANOZZO, D. et al. *libigl: A simple C++ geometry processing library*. 2018. Https://libigl.github.io/.

KARASIK, E.; FATTAL, R.; WERMAN, M. Object partitioning for support-free 3d-printing. *Computer Graphics Forum*, v. 38, n. 2, p. 305–316, 2019.

KOVÁCS, E. Rotation about an arbitrary axis and reflection through an arbitrary plane. *Annales Mathematicae et Informaticae*, v. 40, p. 175–186, 2012.

KREYSZIG, E. *Introductory functional analysis with applications, v. 1*. New York: Wiley, 1978.

LARSON, J.; MENICKELLY, M.; WILD, S. M. Derivative-free optimization methods. *Acta Numerica*, Cambridge University Press, v. 28, p. 287–404, 2019.

MARTÍ, R.; RESENDE, M. G.; RIBEIRO, C. C. Multi-start methods for combinatorial optimization. *European Journal of Operational Research*, Elsevier, v. 226, n. 1, p. 1–8, 2013.

MARTÍNEZ, J. et al. Polyhedral voronoi diagrams for additive manufacturing. *ACM Transactions on Graphics (TOG)*, ACM, v. 37, n. 4, p. 129, 2018.

NOCEDAL, J.; WRIGHT, S. *Numerical optimization*. New York: Springer Science & Business Media, 2006.

POWELL, M. A method for minimizing a sum of squares of non-linear functions without calculating derivatives. *The Computer Journal*, The British Computer Society, v. 7, n. 4, p. 303–307, 1965.

RANELLUCCI, A.; LENOX, J. *Slic3r: Open source 3D printing toolbox*. 2011. Https://slic3r.org.

RUITER, A. H. de; FORBES, J. R. Generalized euler sequences revisited. *The Journal of the Astronautical Sciences*, Springer, v. 62, n. 1, p. 1–20, 2015.

SHANNO, D. F. Conditioning of quasi-newton methods for function minimization. *Mathematics of computation*, v. 24, n. 111, p. 647–656, 1970.

SORKINE, O.; ALEXA, M. As-rigid-as-possible surface modeling. v. 4, p. 109–116, 2007.

VANEK, J.; GALICIA, J. A. G.; BENES, B. Clever support: Efficient support structure generation for digital fabrication. v. 33, n. 5, p. 117–125, 2014.

WANG, W. M.; ZANNI, C.; KOBBELT, L. Improved surface quality in 3d printing by optimizing the printing direction. v. 35, n. 2, p. 59–70, 2016.

WITTENBURG, J.; LILOV, L. Decomposition of a finite rotation into three rotations about given axes. *Multibody System Dynamics*, Springer, v. 9, n. 4, p. 353–375, 2003.

YAGOU, H.; OHTAKE, Y.; BELYAEV, A. Mesh smoothing via mean and median filtering applied to face normals. p. 124–131, 2002.

YAO, M. et al. Level-set-based partitioning and packing optimization of a printable model. *ACM Transactions on Graphics (TOG)*, ACM, v. 34, n. 6, p. 214, 2015.

ZHANG, X. et al. Perceptual models of preference in 3d printing direction. *ACM Transactions on Graphics (TOG)*, ACM, v. 34, n. 6, p. 215, 2015.

ZHENG, Y. et al. Bilateral normal filtering for mesh denoising. *IEEE Transactions on Visualization and Computer Graphics*, IEEE, v. 17, n. 10, p. 1521–1530, 2010.

A Source codes

In this section, we present some of the source codes written in Matlab to apply the optimization method to find the global minimum of problem 4.0.3. The functions are divided in two parts: *main functions* and *auxiliary functions*. Other auxiliary functions were used, such as *readOBJ.m*, *tsurf.m* and are available on *gptoolbox* (JACOBSON et al., 2018) website. Also, the surfaces that were used for tests here are available on *libigl* (JACOBSON, PANOZZO et al., 2018) website.

A.1 Main functions

The main functions use *fmincon* to find the minimum of problem 4.0.3. The Matlab function *print3Dopt_grid.m* uses the multiple constraint division (M.C.D.) strategy for multistart initial point. The Matlab function *print3Dopt_rand.m* uses the randomly chosen starting points (R.C.S.P) strategy also for multistart initial point. The Matlab function *print3Dopt.m* can be used if the user knows what starting point to use for the algorithm find the global minimum.

A.1.1 print3Dopt grid.m

```
function [Xmin,Vnew,F,minim] = print3Dopt_grid(surface)
%    This algorithm uses the GPTOOLBOX (JACOBSON, 2018) readOBJ function
% to read solids in .obj format and searches for the best orientation in
% space that solves the minimization problem:
%
%                    min sum W*||N-PrN||,
%
% where N is the normal field of surface, PrN is the projection of the
% normal field on the xy-plan and W is a weight function for each
% triangle of the mesh. This function uses fmincon to solve the minimiza-
% tion problem with a uniform grid center multistart point strategy.
%
%Syntax:
%
% [Xmin,V,F,minim] = print3Dopt_grid(surface)
%
%Input:
```

```
%
%   surface     path to .obj file
%
%Outputs:
%
%   Xmin        2 by 1 vector solution of the minimization problem
%   V           #V by 3 list of vertices at the optimal position
%   F           #F by 3 list of triangle indices
%   minim       objective function value at the optimal point
%
%JACOBSON, A. gptoolbox: Geometry processing toolbox.
%http://github.com/alecjacobson/gptoolbox, 2018.
%

[V,F]=readOBJ(surface); %% Reading .obj file from path
minim=1e10; %% Setting a high minimum initial value
Xmin=zeros(2,1); %% Alocating the optimal point
h=pi/4; %% Setting the lenght to partition the grid domain

% % % %Necessary parameters for fmincon
LB=[-pi;-pi/2]; %%Left box condition  %
UB=-LB; %% Right box condition        %
A=[];                                 %
B=[];                                 %
Aeq=[];                               %
Beq=[];                               %
% % % % % % % % % % % % % % % % % % % %



% % % % % % % % % % % % % % % % % % % % % Begining of multistart strategy
for ii=-pi:h:pi-h
    for jj = -pi/2:h:pi/2-h
        x0=[(ii+h)/2;(jj+h)/2]; %% Setting the initial point for fmincon

        Vh = rotatexy(V,F,[0;0],'center');%% Translating the surface
                                          % barycenter to the origin.
        N = normalsurf(Vh,F); %% Generating the normal field of the surface
        Area = areatsurf(Vh,F); %% Generating a vector with triangle mesh
                                % areas
% % % % % % % % % % % % % % % % %%fmincon applied to the objective function
        options = optimoptions('fmincon','Display','none');              %
        [X,fval] = fmincon(@(x)(Area.*(-sin(x(2,1))*Vh(F(:,1),1)+ ...    %
            cos(x(2,1))*sin(x(1,1))*Vh(F(:,1),2) + cos(x(1,1))*...       %
            cos(x(2,1))*Vh(F(:,1),3) - min(-sin(x(2,1))*Vh(F(:,1),1)+ ... %
            cos(x(2,1))*sin(x(1,1))*Vh(F(:,1),2) + cos(x(1,1))*...       %
            cos(x(2,1))*Vh(F(:,1),3))))'*((sin(x(2,1)).^2).*(N(:,1).^2)+...
            (cos(x(1,1)).^2)*(cos(x(2,1)).^2).*(N(:,3).^2) - ...         %
            2.*sin(x(2,1)).*cos(x(2,1)).*sin(x(1,1)).*N(:,1).*N(:,2) + ...%
            (cos(x(2,1)).^2).*(sin(x(1,1)).^2).*(N(:,2).^2) + ...        %
            2.*(cos(x(2,1)).^2).*sin(x(1,1)).*cos(x(1,1)).*...           %
            N(:,2).*N(:,3) - 2.*sin(x(2,1)).*cos(x(2,1)).*...            %
            cos(x(1,1)).*N(:,1).*N(:,3)),x0,A,B,Aeq,Beq,LB,UB,[],options);%
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %

% % %Getting the minimal value of function evaluation and the optimal point
        if fval < minim                                                  %
            minim = fval;                                                %
            Xmin=X;                                                      %
        end                                                              %
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %

    end
end
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
Vnew = rotatexy(V,F,Xmin,'plate');%% Rotating the surface by optimal angles
                                   % given by fmincon results, centering the
                                   % surface barycenter at the origin and
                                   % adjusting the surface to printer plate
end
```

A.1.2 print3Dopt_rand.m

```
function [Xmin,Vnew,F,minim] = print3Dopt_rand(surface)
%    This algorithm uses the GPTOOLBOX (JACOBSON, 2018) readOBJ function
% to read solids in .obj format and searches for the best orientation in
% space that solves the minimization problem:
```

```
%
%                    min sum W*||N-PrN||,
%
% where N is the normal field of surface, PrN is the projection of the
% normal field on the xy-plan and W is a weight function for each
% triangle of the mesh. This function uses fmincon to solve the minimiza-
% tion problem with a randomic multistart point strategy.
%
%Syntax:
%
% [Xmin,V,F,minim] = print3Dopt_rand(surface)
%
%Input:
%
%   surface     path to .obj file
%
%Outputs:
%
%   Xmin        2 by 1 vector solution of the minimization problem
%   V           #V by 3 list of vertices at the optimal position
%   F           #F by 3 list of triangle indices
%   minim       objective function value at the optimal point
%
%JACOBSON, A. gptoolbox: Geometry processing toolbox.
%http://github.com/alecjacobson/gptoolbox, 2018.
%

[V,F]=readOBJ(surface); %% Reading .obj file from path
minim=1e10; %% Setting a high minimum initial value
Xmin=zeros(2,1); %% Alocating the optimal point

m=32; %% Setting the number of random choices for initial point
% multistart = zeros(m,2); %%Creating a vector with random initial points

% % % %Necessary parameters for fmincon
LB=[-pi;-pi/2]; %%Left box condition  %
UB=-LB; %% Right box condition        %
A=[];                                 %
B=[];                                 %
```

```
Aeq=[];                                      %
Beq=[];                                      %
% % % % % % % % % % % % % % % % % % % % %

% % % % % % % % % % % % % % % % % % % % % % Begining of multistart strategy
for r=1:m

% % % % % % % % % % % % % % % % %Setting the random initial point
    ii=2*pi.*rand(1,1) -pi;                              %% x-coordinate
    jj=pi.*rand(1,1) - pi/2;                             %% y-coordinate
    x0=[ii;jj];%% Setting the initial point with random choices
%     multistart(r,:) = x0';%% Adding the initial point on vector
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %

    Vh = rotatexy(V,F,[0;0],'center');%% Translating the surface
                                      % barycenter to the origin.
    N = normalsurf(Vh,F); %% Generating the normal field of the surface
    Area = areatsurf(Vh,F); %% Generating a vector with triangle mesh
                              % areas

% % % % % % % % % % % % % % % % % % % %%fmincon applied on objective function
    options = optimoptions('fmincon','Display','none');                %
    [X,fval] = fmincon(@(x)(Area.*(-sin(x(2,1))*Vh(F(:,1),1)+ ...      %
        cos(x(2,1))*sin(x(1,1))*Vh(F(:,1),2) + ...                     %
        cos(x(1,1))*cos(x(2,1))*Vh(F(:,1),3) - ...                     %
        min(-sin(x(2,1))*Vh(F(:,1),1)+ ...                             %
        cos(x(2,1))*sin(x(1,1))*Vh(F(:,1),2) + ...                     %
        cos(x(1,1))*cos(x(2,1))*...                                    %
        Vh(F(:,1),3))))'*((sin(x(2,1)).^2).*(N(:,1).^2) + ...          %
        (cos(x(1,1)).^2)*(cos(x(2,1)).^2).*(N(:,3).^2) - ...           %
        2.*sin(x(2,1)).*cos(x(2,1)).*sin(x(1,1)).*N(:,1).*N(:,2) + ... %
        (cos(x(2,1)).^2).*(sin(x(1,1)).^2).*(N(:,2).^2) + ...          %
        2.*(cos(x(2,1)).^2).*sin(x(1,1)).*cos(x(1,1)).*N(:,2).*N(:,3) - ...
        2.*sin(x(2,1)).*cos(x(2,1))...                                 %
        .*cos(x(1,1)).*N(:,1).*N(:,3)),x0,A,B,Aeq,Beq,LB,UB,[],options); %
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %

% % %Getting the minimal value of function evaluation and the optimal point
        if fval < minim                                                %
```

```
            minim = fval;                                                %
            Xmin=X;                                                      %
        end                                                              %
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %

end
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %

Vnew = rotatexy(V,F,Xmin,'plate');%% Rotating the surface by optimal angles
                                   % given by fmincon results, centering the
                                   % surface barycenter at the origin and
                                   % adjusting the surface to printer plate
end
```

A.1.3 print3Dopt.m

```
function [X,Vnew,F,fval,exitflag,output] = print3Dopt(surface,x0)
%    This algorithm uses the GPTOOLBOX (JACOBSON, 2018) readOBJ function
% to read solids in .obj format and searches for the best orientation in
% space that solves the minimization problem:
%
%                   min sum W*||N-PrN||,
%
% where N is the normal field of surface, PrN is the projection of the
% normal field on the xy-plan and W is a weight function for each
% triangle of the mesh. This function uses fmincon to solve the minimiza-
% tion problem.
%
%Syntax:
%
% [X,V,F,fval,exitflag,output] = print3Dopt(surface,x0)
%
%Input:
%
%   surface     path to .obj file
%   x0          initial point for fmincon
%
%Outputs:
```

```
%
%   X            2 by 1 vector solution of minimization problem
%   V            #V by 3 list of vertices at the optimal position
%   F            #F by 3 list of triangle indices
%   fval         objective function value at the optimal point
%   exitflag     a value that describes the exit condition of fmincon
%   output       a structure output with information about the optimization
%                process.
%
%JACOBSON, A. gptoolbox: Geometry processing toolbox.
%http://github.com/alecjacobson/gptoolbox, 2018.
%

[V,F]=readOBJ(surface); %% Reading .obj file from path

% % % %Necessary parameters for fmincon
LB=[-pi;-pi/2]; %%Left box condition  %
UB=-LB; %% Right box condition        %
A=[];                                 %
B=[];                                 %
Aeq=[];                               %
Beq=[];                               %
% % % % % % % % % % % % % % % % % % % %



Vh = rotatexy(V,F,[0;0],'center'); %% Translating the surface barycenter to
                                   % the origin.
N = normalsurf(Vh,F); %% Generating the normal field of the surface
Area = areatsurf(Vh,F); %% Generating a vector with triangle mesh areas

% % % % % % % % % % % % % % % % % % %%fmincon applied to objective function
options = optimoptions('fmincon','Display','none');                        %
[X,fval,exitflag,output] = fmincon(@(x)(Area.*(-sin(x(2,1))*...            %
    Vh(F(:,1),1)+ cos(x(2,1))*sin(x(1,1))*Vh(F(:,1),2) + ...               %
      cos(x(1,1))*cos(x(2,1))*Vh(F(:,1),3) - min(-sin(x(2,1))...           %
      *Vh(F(:,1),1)+ cos(x(2,1))*sin(x(1,1))*Vh(F(:,1),2) + ...            %
      cos(x(1,1))*cos(x(2,1))*Vh(F(:,1),3))))'*((sin(x(2,1)).^2)...        %
      .*(N(:,1).^2) + (cos(x(1,1)).^2)*(cos(x(2,1)).^2)...                 %
      .*(N(:,3).^2) - 2.*sin(x(2,1)).*cos(x(2,1)).*sin(x(1,1))...          %
      .*N(:,1).*N(:,2) + (cos(x(2,1)).^2).*(sin(x(1,1)).^2)...          %
      .*(N(:,2).^2) + 2.*(cos(x(2,1)).^2).*sin(x(1,1))...               %
      .*cos(x(1,1)).*N(:,2).*N(:,3) - 2.*sin(x(2,1))...                 %
      .*cos(x(2,1)).*cos(x(1,1)).*N(:,1)...                             %
      .*N(:,3)),x0,A,B,Aeq,Beq,LB,UB,[],options);                       %
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %

Vnew = rotatexy(V,F,X,'plate'); %% Rotating the surface by optimal angles
                                 % given by fmincon results, centering the
                                 % surface barycenter at the origin and
                                 % adjusting the surface to printer plate
end
```

A.2 Auxiliary functions

A.2.1 rotatexy.m

```
function Vnew = rotatexy(V,F,theta,option)
%Rotates a surface V around x and y axis by angle thetax,thetay
%
%Sintax:
%
%   Vnew = rotatexy(V,F,theta)
%
% Inputs:
%
%   V            #V by 3 matrix of surface's vertex coordinates.
%   F            #F by 3  matrix of indices of surface's triangle
%                corners.
%   theta        2 by 1 vector with x and y rotation angle.
%   option       final position option of surface.
%                Set 'plate' if the surface need to be positioned on printer
%                plate.
%
% Output:
%
%   Vnew         #V x 3 matrix of surface's vertex coordinates after rotation.
%
```

```
% Making the input vector a column vector
if size(theta,2) > 1                    %
    theta = theta';                     %
end                                     %
% % % % % % % % % % % % % % % % % % % % %

thetax = theta(1,1); % Separating coordinates
thetay = theta(2,1); %

% % % % Translating the barycenter of the surface to the origin
Xbari = (V(F(:,1),1) + V(F(:,2),1) + V(F(:,3),1))/3;          %
Ybari = (V(F(:,1),2) + V(F(:,2),2) + V(F(:,3),2))/3;          %
Zbari = (V(F(:,1),3) + V(F(:,2),3) + V(F(:,3),3))/3;          %
X = sum(Xbari)/length(Xbari);                                  %
Y = sum(Ybari)/length(Ybari);                                  %
Z = sum(Zbari)/length(Zbari);                                  %
                                                               %
T=[1,0,0,-X;0,1,0,-Y;0,0,1,-Z;0,0,0,1];                        %
TV = T*[V ones(size(V,1),1)]';                                 %
TV = TV';                                                      %
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %

 % % % % % % % % % % % % % % % Setting and applying the rotation matrix
ROT1 = [cos(thetay) 0 sin(thetay)  0;...                               %
        0 1 0 0;...                                                    %
        -sin(thetay) 0 cos(thetay)  0;...                              %
        0 0 0 1];                                                      %
                                                                       %
ROT2 = [1 0 0 0;...                                                    %
        0 cos(thetax) -sin(thetax)  0;...                              %
        0 sin(thetax)  cos(thetax)  0;...                              %
        0 0 0 1];                                                      %
                                                                       %
ROT = ROT1*ROT2;                                                       %
Vnew = ROT*TV';  % Applying the rotation matrix                        %
if strcmp(option,'plate') == 1 % Positioning the surface after rotation
   T2=[1,0,0,0;0,1,0,0;0,0,1,-min(Vnew(3,:));0,0,0,1];                 %
   Vnew = T2*Vnew;                                                     %
```

```
end                                                                      %
Vnew = Vnew';       % New surface vertices matrix after rotation         %
Vnew = Vnew(:,1:3);%                                                     %
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %

end
```

A.2.2 normalsurf.m

```
function N = normalsurf(V,F)
%Find the Normal field of surface V and triangle index F.
%
%Sintax:
%
%   N = normalsurf(V,F)
%
% Inputs:
%
%   V       #V by 3 matrix of surface's vertex coordinates.
%   F       #F by 3  matrix of indices of surface's triangle corners.

% Output:
%   N       #F by 3 matrix of Normal field of surface.
%

v = V(F(:,3),:) - V(F(:,1),:); % Setting triangle's edges
w = V(F(:,2),:) - V(F(:,1),:); % Setting triangle's edges
vxw=cross(v,w); % Cross product of triangle's edges and finding normals
N=normr(vxw); % Normalizing the normals
end
```

A.2.3 areatsurf.m

```
function A = areatsurf(V,F)
%Creates a vector with all triangle mesh areas of surface.
%
%Sintax:
```

```
%
%   A = areatsurf(V,F)
%
% Inputs:
%
%   V           #V by 3 matrix of surface's vertex coordinates.
%   F           #F by 3  matrix of indices of surface's triangle corners.
%
% Output:
%
%   A           #F by 1 vector of all triangle mesh areas of V.
%

% Calculating parameters for triangle area using Heron's formula
a = sqrt((V(F(:,2),1) - V(F(:,1),1)).^2 + ...  %triangle edge a
         (V(F(:,2),2) - V(F(:,1),2)).^2 + ...                 %
         (V(F(:,2),3) - V(F(:,1),3)).^2);                     %
b = sqrt((V(F(:,3),1) - V(F(:,1),1)).^2 + ...  %triangle edge b
         (V(F(:,3),2) - V(F(:,1),2)).^2+ ...                  %
         (V(F(:,3),3) - V(F(:,1),3)).^2);                     %
c = sqrt((V(F(:,3),1) - V(F(:,2),1)).^2 + ...  %triangle edge c
         (V(F(:,3),2) - V(F(:,2),2)).^2+ ...                  %
         (V(F(:,3),3) - V(F(:,2),3)).^2);                     %
p = (a+b+c)./2; %Semiperimeter                                %
                                                              %
A=sqrt(p.*(p-a).*(p-b).*(p-c)); % Calculating area of triangles
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %

end
```