FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Reduced-Order Shape Optimization Using Offset Surfaces in Blender

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Lukas Gersthofer
Matrikelnummer 01325669

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: Univ.Ass. Dipl.-Mediensys.wiss. Dr.techn. Przemyslaw Musialski

Wien, 21. Februar 2018

Lukas Gersthofer	Michael Wimmer

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Reduced-Order Shape Optimization Using Offset Surfaces in Blender

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Lukas Gersthofer
Registration Number 01325669

to the Faculty of Informatics

at the TU Wien

Advisor: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: Univ.Ass. Dipl.-Mediensys.wiss. Dr.techn. Przemyslaw Musialski

Vienna, 21st February, 2018

Lukas Gersthofer	Michael Wimmer

Erklärung zur Verfassung der Arbeit

Lukas Gersthofer
Vienna

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 21. Februar 2018

Lukas Gersthofer

Danksagung

Mein Dank geht an Przemyslaw Musialski, der mir die Möglichkeit eröffnet hat, solch ein modernes Thema im Rahmen einer Bachelorarbeit zu behandeln. Im Weiteren möchte ich Christian Hafner für jegliche Fragen zur Implementierung und zum mathematischen Hintergrund danken.
Schließlich gilt mein Dank auch meinen Eltern, ohne deren stetige Unterstützung es mir nicht möglich gewesen wäre einen akademischen Abschluss in Angriff nehmen zu können.

Acknowledgements

I would like to thank Przemyslaw Musialski for the opportunity to work on this thesis. A special thank goes to Christian Hafner for supporting me with questions related to the implementation and mathematical background.
At last I would like to thank my parents for their patience and steady support so that I can realize my own potential.

Kurzfassung

3D Modelle spielten bis vor kurzem nur eine wichtige Rolle in digitalen Welten. Doch seit dem Aufkommen von 3D Druckern und anderen Fabrikationsmethoden werden diese digitalen Modelle zunehmend in die physikalische Welt geholt. Vor allem die rasche Entwicklung in den technischen Fähigkeiten dieser Geräte sowie die sinkenden Preise beschleunigten diesen Trend. Ein großer Nachteil besteht jedoch darin, dass die digitalen 3D Modelle nicht dafür entwickelt wurden, in unserer Welt zu existieren. Die Einbeziehung von physikalischen Eigenschaften wie Masse, Schwerpunkt oder Trägheitsmoment, die wesentlich zum Verhalten der Objekte beitragen, wurden dabei vernachlässigt. Die gedruckten Modelle weisen daher fast immer ein falsches physikalisches Verhalten auf. Die Fähigkeit in einer gewissen Pose stehen, im Wasser schwimmen oder um eine bestimmte Axe stabil rotieren zu können sind Beispiele für das physikalische Verhalten eines Objekts. Nach und nach zog auch die Software nach und es wurden Verfahren entwickelt, um digitale Modelle auf die Fabrikation vorzubereiten indem gewisse physikalische Eigenschaften durch gezielte Optimierung des Volumens angepasst werden. Eine kürzlich präsentierte Methode, auf dem diese Arbeit basiert, ist durch ihre Flexibilität und Schnelligkeit dafür geeignet, um in modernen 3D Modellierungsprogrammen umgesetzt zu werden. Im Zuge dieser Arbeit wurde dieses Verfahren als C/C++ Bibliothek implementiert und anschließend in die freie Modellierungssoftware Blender als Modifier integriert. Diese Integrierung soll einen einfachen Zugang zu einem Optimierungsverfahren von digitalen Modellen für die Farbikation in deren üblichen Modellierungsumgebung gewährleisten.

Abstract

The advance of 3D printers' capabilities and their sinking costs led to a huge trend of personal and commercial fabrication. But those advances were restricted to the hardware side meaning that there was a lack of software to optimize the digital models before printing. This was necessary because physical properties like mass, center of mass or moments of inertia, were neglected in the design of digital 3D models. Those properties play an important role in the behavior of a real-world object. Examples of an objects behavior are the ability to stand in a specific pose, float in the water or stably rotate around a certain axis.
In the last few years methods have been presented to optimize digital models by altering specific regions of their volume in order to change their physical properties and therefore to prepare them for printing. A recently presented method forms the basis of this thesis. Due to its flexibility and performance it is well suited to be integrated into current 3D modeling applications. The algorithm was implemented as a C/C++ library which can be integrated in almost every application. Afterwards this library was integrated into the open source 3D modeling application Blender as a modifier.

Contents

CHAPTER 1

Introduction

Since the beginning of the field of computer graphics, digital models play a central role in representing real-world objects in the digital space. Nowadays, these models are still frequently used for simulation, rendering and visualization. With new 3D scanning technology it is even possible to transfer objects from our real world to the digital one. The field of digital or computational fabrication just started a few years ago to be a hot topic as an area of research. New fabrication methods were developed and allowed for increase in size and complexity of fabricated digital models at moderate costs. 3D printers became very popular among researchers and 3D enthusiasts. The sinking costs and improved precision of fabrication devices led to a new trend and a revolution of digital fabrication [Ger12]. Personal usage of 3D printers rose due to affordable and easily operable devices and publicly available blueprints of real-world objects [Mot11]. A combination of high-resolution fabrication methods and optimization of digital models led to a better design and a more efficient production of new gadgets, toys, and other objects. It replaces long and costly trial-and-error processes.
Unfortunately, a major aspect of real-world objects has been neglected in the design of their digital representations: physical properties. A common representation of digital models in modeling and fabrication are polygonal (mostly triangular) meshes defined by their surface. But those meshes only approximate the real surface up to a certain degree and do not define any physical properties. One exception is the volume enclosed by the surface. It appeared that many fabricated models cannot even stand in an upright position. At that point it was clear that physical properties like mass, center of mass, or moments of inertia, which define how objects behave in the real world, had been neglected in the design of producible models.

To counter this issue, computer graphics researchers developed new methods and tools to alter the physical properties of digital models interactively before fabrication. Although FEM-based tools are also capable of dealing with physical properties of digital models, these methods are considered as rather slow and complex in terms of usage and integration into standard software.
Prévost et al. [PWLSH13] presented an approach to make arbitrary models stand in an upright position or hang in suspension by optimizing their centers of mass. One year later, the Disney researchers Bächer et al. [BWBSH14] developed a method to optimize not only the center of mass but also the moments of inertia, allowing to create spinning tops or yo-yos. The method developed by Musialski et al. [MAB+15] makes use of a different volume representation and dimensionality reduction of the optimization problem, resulting in more flexibility when it comes to defining global goals and better performance due to its independence of object complexity. Therefore, the algorithm is suitable for integration into state-of-the-art 3D modeling software like Maya[1], Blender[2], etc. A follow-up of the work of Bächer et al. [BWBSH14] was presented by Wang et al. [WW16], which is capable of making objects float in a given position and be assembled of slices of plywood.

In this thesis the author presents the implementation of the above mentioned method by Musialski et al. [MAB+15] as an open source library. In addition, the optimization problem was reduced by projecting lower and upper bounds into the objective function (see Section 3.3.2), and a different solver was integrated. The library is written in C/C++ and depends on several external libraries: Eigen[3], libigl[4], Spectra[5], CGAL[6] and NLopt[7]. To make this work publicly available for designers and 3D enthusiasts, the implemented algorithm was integrated into the open source 3D modeling software Blender as a modifier. The library currently supports two target functions to optimize as they are: (1.) static stability, (2.) rotational stability.

The following thesis is structured as follows: In Chapter 2 an overview of state-of-the-art methods in this field is presented. Chapter 3 will give some background information about how the algorithm works i.e. volume representation, performance increase, optimization, etc. The gained knowledge is implemented and some insights into the code are given in Chapter 4. Moreover, a more detailed description of the external libraries used will be given. Chapter 5 will show some results of optimized 3D models as well as screen shots of the actual algorithm integrated in Blender. At last there will be a conclusion and outlook for future work.

[1] http://www.autodesk.com/products/maya/overview
[2] https://www.blender.org/
[3] http://eigen.tuxfamily.org/
[4] http://libigl.github.io/libigl/
[5] https://github.com/yixuan/spectra/
[6] https://www.cgal.org/
[7] http://ab-initio.mit.edu/wiki/index.php/NLopt

CHAPTER 2

Related Work

As described in the introduction, there has been quite an advance in methods to optimize physical properties of digital models before fabrication. Here, just three major publications, besides the one which forms the basic for this work, will be explained in this chapter. They will be ordered chronologically by their publication date. A short description of the methods basic idea should give an overview.

2.1 Make It Stand

One of the first works published in the field of optimizing shapes to fulfill an objective after printing was presented by Prévost et al. [PWLSH13]. The method focuses on the static stability or suspension of 3D models and is based on two main techniques: voxel carving and linear blend skinning deformations. The idea is to modify the interior of the mesh first and then deform its shape slightly while preserving its main appearance. As illustrated in Figure 2.1 the optimization consists of alternating steps of carving and deformation. First the mesh is voxelized and the voxel structure is stored in binary variables α then the model is deformed with respect to its voxelized interior (fixed α variables). After deforming the mesh it is discretized into voxels again but now with fixed deformation handles $\mathcal{H}$. This procedure repeats until the energy difference between iterations vanishes or is lower than a given threshold. The energy of a mesh is defined by how close the current physical properties (here the center of mass) is compared to the optimum.

To achieve static stability a plane is cut through the optimal center of mass c^* (which lies in the middle of the support polygon projected along the gravity direction) which is perpendicular to the projection of the difference of optimal and current center of mass c_0. Voxels which lie in the half-space that contains c_0 are left unchanged whereas voxels on the other side are carved out to translate c_0 in direction of c^*. To visually describe this

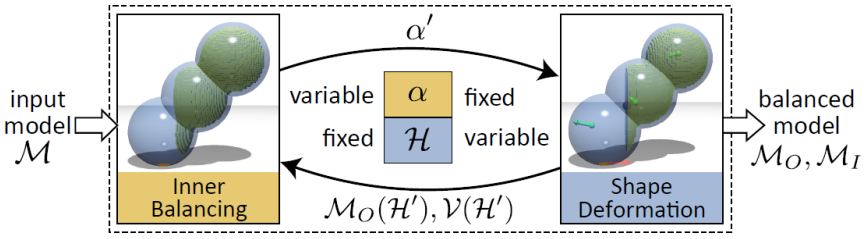


Figure 2.1: One iteration consists of voxelizing the input model, deforming it with the fixed voxels α and voxelize it again with fixed deformation handles $\mathcal{H}$ © [PWLSH13].

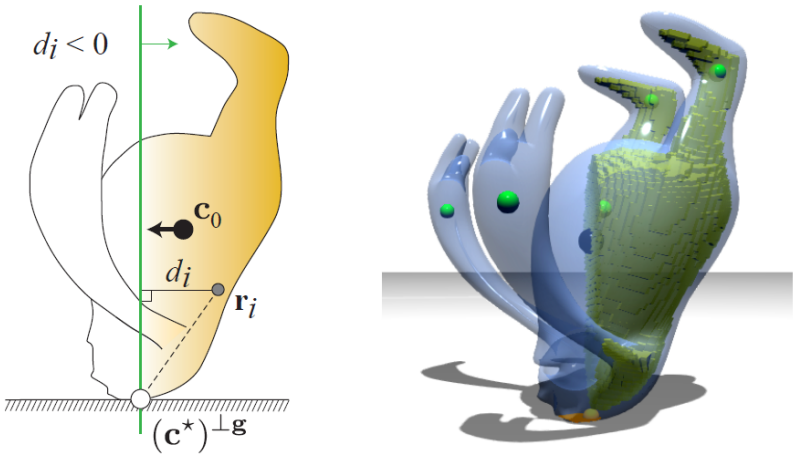


Figure 2.2: Visualization of a voxel carving step (left). A carved model with deform handles (right), empty areas are colored yellow © [PWLSH13].

method and its output see Figure 2.2. Models with multiple bases are also supported. Therefore, a plane for each base is defined.

2.2 Spin-It

Fabricated spinning tops and yo-yos with arbitrary shapes are examples for the method presented by Bächer et al. [BWBSH14]. Their goal was to optimize 3D models to stably rotate around a given axis despite their asymmetric appearance. Therefore, a desired rotation axis in addition to the input mesh must be provided by the user. The model is then optimized such that the given axis is parallel to the main principal axis of inertia and the center of mass is positioned on this axis. This is done by combining three different methods: Hollowing (extends voxel carving by supporting a multi resolution voxelization), cage-based deformations and dual-density optimization. Primarily hollowing is applied to

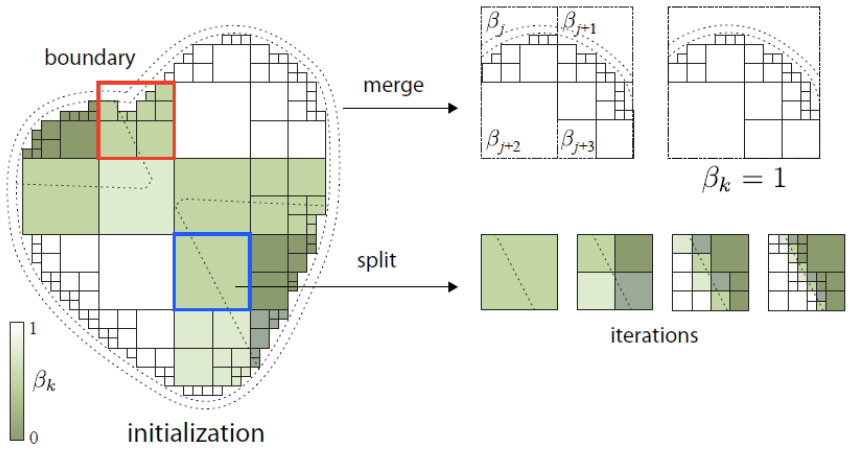


Figure 2.3: Fill variables β_k are displayed in green shades © [BWBSH14].

the model. For some complex models cage-based deformations are used too and for some extraordinary cases it is possible to use two different materials with dissimilar densities.

The filling of a voxel v_k is described by a binary fill variable β_k. The optimization works as follows (visually described in Figure 2.3):

1. *Initialization.* The octree is initialized to a mid-level refinement.
2. *Optimization step.* In this step the binary variables β_k are optimized with a split-and-merge approach. For that case the filling variables are treated as continuous numbers in the interval $[0, 1]$ and represent if a cell needs to be split or merged. Cells with a value of 0 are kept filled and cells with value 1 are kept empty. Everything in between 0 and 1 is either split into multiple subvoxels or merged with its neighbors into one region.
3. *Convergence.* If all fill variables are binary and correspond to the cells at the maximum resolution, the optimization is terminated.

2.3 Buoyancy Optimization.

The science of floating objects dates back to Archimedes and has been studied since then. The principle of buoyancy is used for many applications in the field of computer graphics like simulations and visualizations of realistic water and objects floating on it. In the work of Wang et al. [WW16] an optimization approach is presented to float 3D models on water, either totally or partially submerged. Similar to *Make It Stand* and *Spin-It* it makes use of a voxel carving technique. To improve optimization speed and quality the

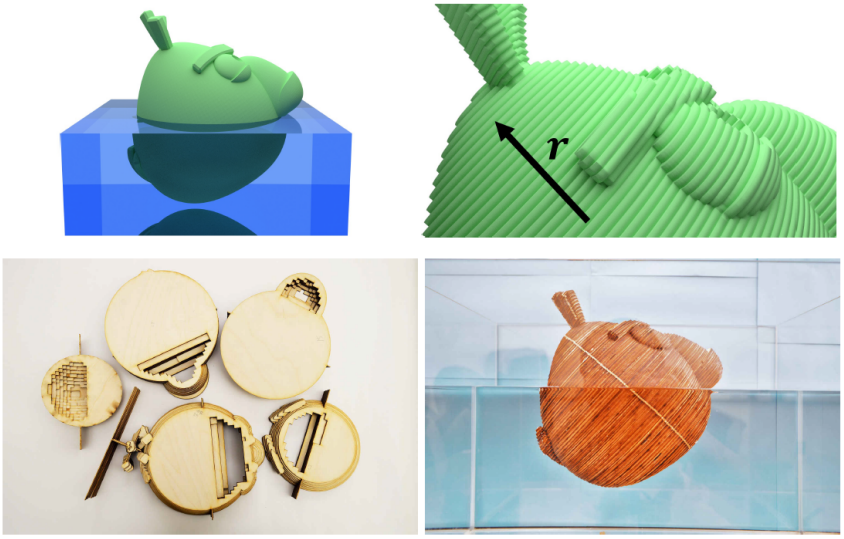


Figure 2.4: Optimized "Angry birds" model in simulation and fabricated with planar-pieces design. © [WW16].

model is discretized with a multi-resolution voxelization method. In order to guarantee that the model can be fabricated a minimal wall thickness is used to create an inner offset surface.
Because of the limitation in size when fabricating models with a 3D printing device, planar-pieces design can be used to first slice the object, print or cut out the slices and then reassemble them. Wang and Whiting used slices of plywood to construct the optimized models. An example is shown in Figure 2.4. Because plywood has a lower density than water additional changes to the model's mass are required. Two different methods are supported: embedding prefabricated metal elements or a prototyped method for high density material casting. The former one relies on the proper placement of metal parts like bolts in specified regions to change the object's mass distribution. Instead of using prefabricated elements a prototyped method for high density material casting is available. Therefore, materials with a high density e.g. concrete are casted into the interior of the model.

CHAPTER 3

Methodology

This chapter will cover the basics to optimize a model according to the work presented in [MAB+15]. A brief discussion about volume representation in spatial and frequency domain is followed by the basic optimization problem. The order of this problem can then be reduced by using a reduced manifold harmonic basis in order to speed up the optimization. Additionally, a trick presented in [MHR+16] eases the process of optimization by altering the objective function to remove lower and upper bounds.

3.1 Volume representation

A volume is intuitively represented as the inside of a closed 2-manifold surface embedded in 3D (non-degenerated surface). To provide more flexibility, a second manifold surface is introduced which does not penetrate the original one. The volume of a solid body is then defined as the space enclosed by those two surfaces.
In the case of open manifolds, the volume is computed by limiting the volume to the extent of the two surfaces. A discrete volume is therefore best represented by two closed manifold meshes which do not penetrate each other. By using offset surfaces (see Section 3.1.1), the number of input surfaces is reduced to one.
One advantage of this representation over a voxel based one used in most of the other methods (see Chapter 2) is, that surfaces are generally smoother and depending on its tessellation the volume computed can be more precise (assuming a uniform voxelization or a multi-resolution voxelization with a limited number of subdivisions). The volume of a closed manifold mesh can be exactly computed by a sum of signed tetrahedron volumes described in Zhang et al. [ZC01].
Optimization of mass properties alters specific regions of the volume. Using a voxel based approach, voxels are carved out of the full volume (see Chapter 2) and voxels left are then 'filled' with material in the fabrication process. In the case of offset surfaces, the

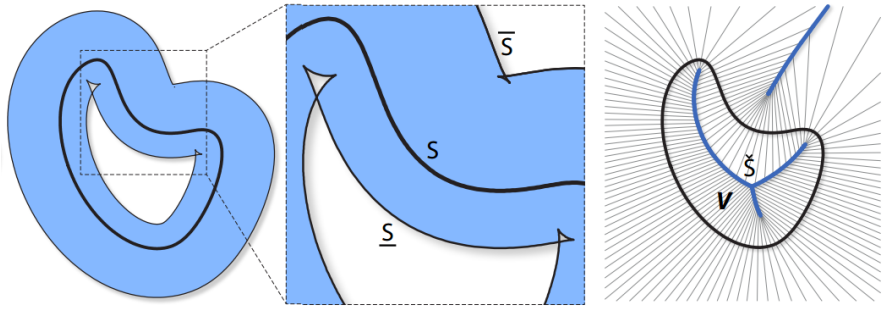


Figure 3.1: Original surface S and inner offset surface $\underline{S}$ penetrate each other and create a self intersection. To avoid this issue offset directions and limits are extracted of the mean curvature skeleton. © [MAB+15]

volume enclosed by two surfaces is altered by changing the surfaces via their offsets. It is possible to either use the original surface together with one offset surface or use both offset surfaces (inner and outer). The latter one allows for a larger design space at the cost of more computational effort and possible deformations of the visual appearance.

3.1.1 Offset surface

An offset or parallel surface is defined by displacement of every point of the original surface by a specific amount (see [Mae99]). On discrete surfaces, like 3D meshes are, this is defined as the displacement of every vertex along a given direction (for example the normal vector) as seen in the equation below:

$$x_i' = x_i + \delta v_i \,, \tag{3.1}$$

where x_i is the position of vertex i, v_i is the offset direction and δ is the amount of displacement. In the following, offset and displacement are used as synonyms.
Of course, the surface may be displaced in both directions: towards the inside and the outside. This requires a basic definition of inside and outside. In the following, the original surface is defined as S, the inner offset surface as $\underline{S}$ and the outer one as $\overline{S}$. The offset direction generally points outwards such that the inner surface is obtained by displacing the surface along the negative offset direction and the outer one along the positive direction.
However, offset surfaces have a major disadvantage: self-intersections. By offsetting vertices of a concave surface (or surface patch) along their normal vector the surfaces might penetrate each other and create self-intersections. This is illustrated on the left side in Figure 3.1. Therefore, offset directions and amount of displacement have to be chosen such that no self-intersections occur. This issue is tackled in the following section.

3.1.2 Mean curvature skeletons

A skeleton extracted from the original shape can be used to estimate the maximum amount of displacement in a specific offset vector for every vertex. The offset directions v_i for a given vertex x_i are defined as follows:

$$v_i = x_i - x_i^{corr}\,, \tag{3.2}$$

where x_i^{corr} describes the corresponding vertex positioned on the skeleton. Afterwards, the maximum displacement along an offset direction is defined as its length.
Basically, any method to extract the skeleton from the shape can be used, but mean curvature skeletons presented by Tagliasacchi et al. [TAOZ12] are less sensitive to surface detail and hence provide a solid approximation of the shapes skeleton even for high frequency surfaces. An example of an extracted skeleton is shown in Figure 3.1 on the right side.
These skeletons are extracted after several applications of constrained Laplacian smoothing of a remeshed input shape. After every iteration a so called meso-skeleton is formed which finally converges to a curve skeleton (see Figure 3.2).

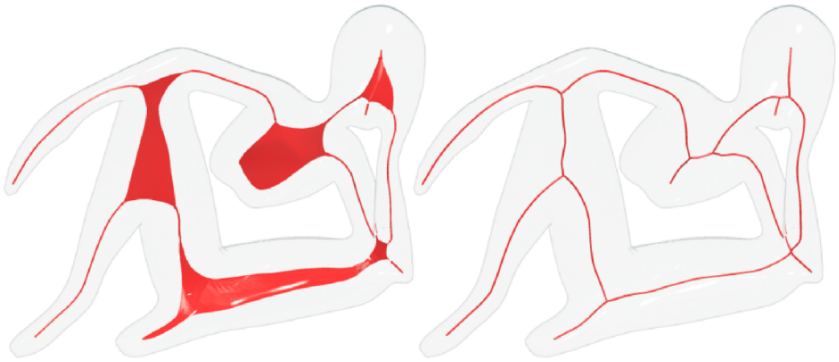


Figure 3.2: Intermediate meso-skeleton on the left side produced by the mean curvature skeleton approach which finally converges to the curve skeleton on the right. © [TAOZ12]

3.2 Manifold harmonics

The Fourier Transform is a popular tool to switch between a space or time domain and a frequency domain. Usually, this holds for signals but can also be extended to 3D shapes [VL08]. Manifold harmonics describe the basic vibration modes across a 3D shape, where low frequent vibration modes correspond to the basic shape and high frequency ones express the surface details. Those vibration modes are extracted as the eigenfunctions of the Laplace-Beltrami operator applied to the given surface. In the discrete case this reduces to computing the eigenvectors of the discrete Laplace-Beltrami operator (matrix) which is defined as follows:

$$L_{i,j} = \begin{cases} \omega_{i,j} & \text{if}(i,j) \in E \\ \sum_{k \in N(i)} -\omega_{i,k} & \text{if } (i=j) \\ 0 & \text{otherwise} \end{cases}, \tag{3.3}$$

where E is the set of edges and $N(i)$ the set of vertices in the one-ring neighborhood. $\omega_{i,j}$ describes a weight function based on geometric properties.
Possibilities for $\omega_{i,j}$ include the uniform weight function ($\omega_{i,j} = 1$) and more sophisticated ones which encapsulate information of the surrounding geometry and achieve better approximation of the Laplacian such as contangent weights proposed by Pinkall and Poltier [PP93] in Equation (3.4).

$$\omega_{i,j} = \frac{1}{2}(\cot \phi^l_{i,j} + \cot \phi^r_{i,j}), \tag{3.4}$$

where $\phi^l_{i,j}$ and $\phi^r_{i,j}$ are the angles opposite to edge (i,j) in the incident triangles on the left or right side respectively.
The resulting matrix L is symmetric positive semi-definite and its eigenvectors γ_i and eigenvalues λ_i are defined as:

$$L\gamma_i = \lambda_i \gamma_i . \tag{3.5}$$

Instead of computing all eigenvectors of this large sparse matrix, only the k first ones corresponding to the lowest eigenvalues are computed hence they approximate the overall shape. This task is solved best using the Implicitly Restarted Arnoldi Method or in the case of a symmetric matrix Implicitly Restarted Lanczos Method[1]. Unfortunately, those methods perform better at the other end of the spectrum thus solving for large eigenvalues. However, by exploiting the shift-and-invert mode any eigenvectors close to a given point can be computed.

Finally, the k first eigenvectors corresponding to the lowest frequencies are put together to a rectangular matrix to form the so called manifold harmonic basis (short MHB):

$$H_k = [\gamma_1 \gamma_2 ... \gamma_k] . \tag{3.6}$$

[1] http://www.caam.rice.edu/software/ARPACK/

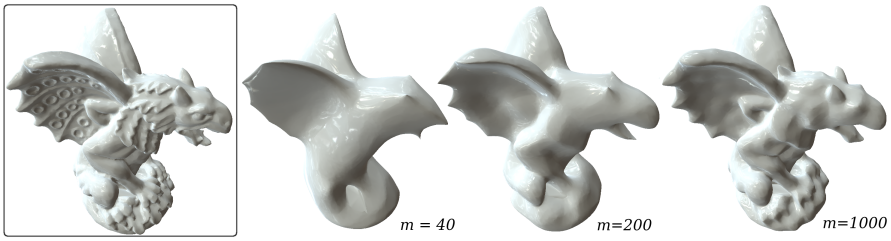


Figure 3.3: Shape of a dragon reconstructed with a different amount of eigenvectors. The more eigenvectors the MHB contains, the more details (high frequencies) are preserved. © [VL08]

This basis is then used to transform a mesh into the frequency domain where it is represented as a linear combination of the first k basis functions and its frequency coefficients α as:

$$\delta = H_k \alpha \,. \tag{3.7}$$

The reconstruction is done by solving the linear system above for the α vector given displacements δ. A shape reconstructed with different numbers of eigenvectors is illustrated in Figure 3.3. Using a MHB including all eigenvectors of the discrete Laplace-Beltrami operator would lead to a perfect reconstruction (no error) of the original shape.

3.3 Non-linear optimization

Minimization of energy terms, so called objective functions, leads to achieving the best possible solution in the given solution space. The objective function measures, how much the physical properties (see Section 3.3.3) of a current configuration differ from the optimum. To minimize this term, the optimization variables—in this case the displacements of the offset surface—are altered, considering dynamic constraints and static bounds. Gradient-based optimization routines offer faster convergence at the cost of finding only local minima. Since the algorithm should perform fast and any valid, near-optimal solution is acceptable, gradient-based methods are prefered.
Based on offset surface representation and assuming that offset directions are already determined (see Section 3.1.2) the optimization problem only depends on the offset values δ_i:

$$\underset{\delta}{\text{Min}}\, f(\delta) \quad \text{s.t.} \quad g_i(\delta) \leq 0 \quad \text{or} \quad g_i(\delta) = 0 \quad \text{and} \quad \delta_l \leq \delta \leq \delta_u\,, \tag{3.8}$$

where f is the objective function which is minimized with respect to changes of the set of displacements δ. The functions g_i define hard equality or inequality constraints and δ_u and δ_l are lower and upper bounds for displacements. These bounds are further described in Section 3.3.2.
Usually, objective function f and constraints g_i are of non-linear form and of high dimensionality due to the displacements' dependence on the number of vertices. This makes the problem expensive to solve, but fortunately the dimensionality of the problem formulation can be drastically reduced.

3.3.1 Dimensionality reduction by using a manifold harmonic basis

With the previously mentioned manifold harmonic basis H_k, a given mesh with n vertices can be reduced to a representation consisting of k frequency coefficients α_k. Hence, the offset δ_i along the displacement direction v_i of the offset surface can also be expressed as a linear combination of the MHB's eigenvectors:

$$\overline{x}_i = x_i + \sum_{j=1}^{k} \alpha_j \gamma_{i,j} v_i\,. \tag{3.9}$$

So instead of finding n displacements δ_i, the problem reduces to finding the k unknown coefficients α_i:

$$\underset{\alpha}{\text{Min}}\, f(\alpha) \quad \text{s.t.} \quad g_i(\alpha) \leq 0 \quad \text{or} \quad g_i(\alpha) = 0 \quad \text{and} \quad \delta_l \leq H_k\alpha \leq \delta_u\,. \tag{3.10}$$

Notable here is, that $k \ll n$ and the new representation in the frequency domain does not depend on the number of vertices anymore. It turned out that around 36 eigenvectors are sufficient to approximate most shapes adequately [MAB+15].

3.3.2 Lower and upper bounds

As mentioned in Section 3.1.1, displacing the surface too much might lead to self-intersections. Therefore, displacements are limited by lower and upper bounds in the optimization process. For the inner offset surface a practical upper bound $\underline{\delta}_u$ per vertex is given by the length of its unnormalized offset vector seen in Equation (3.2). Because fabrication devices such as 3D printers are often limited by a fabrication resolution it is not possible to fabricate objects thinner than this resolution. Also it might happen that the object is too thin in specific regions making it unable to bear its own weight. To compensate these issues, a lower bound for the inner displacements is defined as $\underline{\delta}_l$. If deformations of the visual appearance should be small or negligible, then the outer offsets also needs to be restricted. Again, a minimum wall thickness can be ensured by setting a lower bound $\overline{\delta}_l$. Upper bounds for the outer displacements $\overline{\delta}_u$ are only constrained by the size of printable objects and are therefore not as critical.
The complexity of problems with hard constraints solved by Newton-based solvers often lead to long convergence times or undesirable results which lie outside of the possible solution space due to violation of given hard constraints. To reduce the number of constraints, the lower and upper bounds are encoded into the objective function [MHR+16]. Instead of having the optimization in the form stated in Equation (3.8) the new problem is now given as:

$$\underset{\delta}{\text{Min}}\, f(\varphi(\delta)) \quad \text{s.t.} \quad g_i(\varphi(\delta)) \leq 0 \quad \text{or} \quad g_i(\varphi(\delta)) = 0\,, \tag{3.11}$$

where φ is a function projecting the lower and upper bounds into the objective function and is defined as:

$$\begin{aligned} \varphi(\delta) &= \frac{\delta_u - \delta_l}{\pi} \arctan\left(\delta - o\right) + o \\ o &= \frac{\delta_u - \delta_l}{2}\,. \end{aligned} \tag{3.12}$$

To apply dimensionality reduction in combination with bound encoding the original displacements have to be projected first and then transformed to the frequency coefficients which leads to the final optimization definition:

$$\underset{\delta}{\text{Min}}\, f(\Gamma_k(\varphi(\delta))) \quad \text{s.t.} \quad g_i(\Gamma_k(\varphi(\delta))) \leq 0 \quad \text{or} \quad g_i(\Gamma_k(\varphi(\delta))) = 0\,, \tag{3.13}$$

where Γ_k projects the offsets using the reduced MHB and yields the frequency coefficient vector α.

3.3.3 Physical properties

When it comes to designing the real-world behavior of digital models, a basic understanding of some physical properties is necessary. Therefore, the following sections will deal mainly with the physical properties of real-world objects and how they are computed. However, in this work only mass properties are considered including volume, mass and the first two moments of mass. With this knowledge, objective functions and constraints are derived which are described in Section 3.3.4.

Volume and mass

For objects consisting of only one material (uniform density), volume and mass are linearly related. For analytically defined surface in 3D, its volume is computed as a triple integral over the parameter domain.

Of course, analytically computing a volume integral over a discrete surface is not possible, but by applying the divergence theorem, the volume integral can be reduced to a surface integral. The analytical computation of surface integrals over triangulated surfaces is well defined and can therefore be given in discrete form as:

$$V = \frac{1}{6} \sum_{i \in I} ((v_{i2} - v_{i1}) \times (v_{i3} - v_{i1})) \cdot (v_{i1} + v_{i2} + v_{i3}) \,, \tag{3.14}$$

with v_{ix} describing the x-th vertex position of the i-th triangle and I as the index set of faces.

To now obtain the mass of an object, the density function is integrated over its volume [Dem15]:

$$M = \int_V \rho(\vec{x}) \, \mathrm{d}\vec{x} = \iiint_V \rho(x, y, z) \, \mathrm{d}x \, \mathrm{d}y \, \mathrm{d}z \tag{3.15}$$

For homogeneous objects—density does not vary across the volume—this integral is simplified to a multiplication of density and volume:

$$M = \rho \int_V \mathrm{d}\vec{x} = \rho V \tag{3.16}$$

which leads to a final formula to compute the mass of a triangulated mesh:

$$M = \frac{\rho}{6} \sum_{i \in I} ((v_{i2} - v_{i1}) \times (v_{i3} - v_{i1})) \cdot (v_{i1} + v_{i2} + v_{i3}) \tag{3.17}$$

Center of mass/gravity

The center of mass is defined as the unique point in a solid object, where the weighted relative position of the mass sums to zero. In other words it is the average position of all parts of the system, weighted according to their masses. Analogously, the center of gravity is the average position where gravity cancels out. Sometimes center of gravity is used as a synonym which is true while being in a uniform gravity field. For simplicity, physicists often tend to assume a uniform gravity field on earth, therefore center of mass and center of gravity describe the same point.
The center of mass is computed by utilizing the first moments of mass:

$$M_{\vec{x}} = \int_V \vec{x}\rho(\vec{x})\ \mathrm{d}\vec{x}\,. \tag{3.18}$$

The actual center of mass c is then given by:

$$c = \left(\frac{M_x}{M}\,, \frac{M_y}{M}\,, \frac{M_z}{M}\right). \tag{3.19}$$

Applying the divergence theorem helps to express the volume integral as a surface integral again utilizing the assumption of uniform density. The surface integral is then discretized and stated below [PWLSH13]:

$$\begin{aligned} c &= \frac{\rho}{24M}\sum_{i\in I}((v_{i2}-v_{i1})\times(v_{i3}-v_{i1}))\star g(v_{i1},v_{i2},v_{i3}) \\ g(v_1,v_2,v_3) &= v_1\star v_1 + v_1\star v_2 + v_2\star v_2 + v_2\star v_3 + v_3\star v_3 + v_3\star v_1 \end{aligned} \tag{3.20}$$

where the $\star$ operator defines the component-wise product.

Moment of inertia

Moment of inertia is a measure of an object's resistance to change in rotation direction. Moment of Inertia has the same relationship to angular acceleration as mass has to linear acceleration. As depicted in fig. 3.4 on the right, a body has three principal axes of inertia and their origin is the center of mass c. Every principal axis has a corresponding moment I_x. A rotation around one of the principal axes is considered to be stable if the moments of the other two axes are equal.

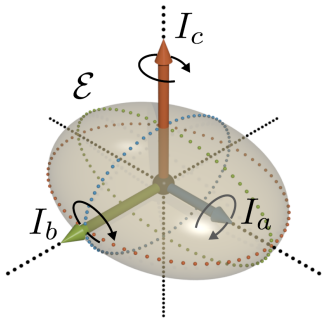


Figure 3.4: Principal axes of rotation with corresponding moments. © [BWBSH14]

For continuous bodies, the moments are derived as the second moment of mass:

$$I_{\vec{x}} = \int_V \vec{x}^2 \rho(\vec{x}) \, \mathrm{d}\vec{x} \,. \tag{3.21}$$

The tensor of inertia is used to compute the total angular momentum L about all three principal axes as:

$$\begin{pmatrix} L_x \\ L_y \\ L_z \end{pmatrix} = \begin{pmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{pmatrix} \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} , \tag{3.22}$$

where the 3x3 matrix forms the tensor of inertia I and ω is the angular velocity vector. It is also worth noting that I is a real symmetric matrix. The elements of I are calculated as follows:

$$\begin{aligned} I_{xx} &= \int_V (y^2 + z^2) \, \mathrm{d}m & I_{xy} &= I_{yx} = \int_V xy \, \mathrm{d}m \\ I_{yy} &= \int_V (x^2 + z^2) \, \mathrm{d}m & I_{xz} &= I_{zx} = \int_V xz \, \mathrm{d}m \\ I_{zz} &= \int_V (x^2 + y^2) \, \mathrm{d}m & I_{yz} &= I_{zy} = \int_V yz \, \mathrm{d}m \end{aligned} \tag{3.23}$$

with $\mathrm{d}m = \rho(x, y, z) \, \mathrm{d}\vec{x}$. I_{xx} is called the moment of inertia about the x-axis, I_{xy} the xy product of inertia and the others follow accordingly.
Again, using the divergence theorem these volume integrals are reduced to surface integrals which lead to a discrete solution. For exact computation of second order mass moments, refer to Bächer et al. [BW+14+].

3.3.4 Global goals

Global goals define the physical behavior of an object during and after the optimization. Global goals encompass objective functions and possibly multiple constraints. The objective functions are specified as at least C^1 continuous non-linear functions (C^1 continuity is needed for gradient-based methods which should be preferably used here). The optimization process minimizes the given objective function with respect to multiple equality and inequality constraints as well as lower and upper bounds. Equality and inequality constraints may be linear or non-linear. Lower and upper bounds are projected into the objective function as described in Section 3.3.2.
In the original paper [MAB+15], such objectives are heavily related to mass properties. This set of global goals encompasses:

- static stability,
- static stability under storage,
- monostatic stability,
- rotational stability,
- specific volume and buoyancy.

However, in this work only static and rotational stability are tackled, hence the others follow the same scheme and can be easily integrated as well.

Static Stability

A statically stable object remains in a given position once placed. In the picture to the right a toy T. rex is shown, which balances his big body on his tiny feet. The static stability is expressed as the deviation between the center of mass c projected along the gravitational axis and the centroid of the area touching the ground called contact area. If the object has multiple contact ares, those regions are joined together by their convex hull. To ensure static stability, the projection of c has to lie in the contact area. This is achieved by placing the centroid of the contact area in the origin and minimizing the distance between the center of mass and the origin in the xy-plane and reducing its height to make it even more stable. The objective function finally is:

Figure 3.5: Toy T. rex optimized to stand stably on his tiny feet. © [PWLSH13]

$$f(\alpha) = c_x^2 + c_y^2 + c_z \,. \tag{3.24}$$

Additionally, the position has to be constrained to the contact area which is defined as the largest inscribing circle of the contact area (g_1). Of course, the center of mass has to be above the ground (g_2):

$$\begin{aligned} g_1(\alpha) &= (c_x + c_y)^2 - (r - \epsilon)^2 \leq 0 \\ g_2(\alpha) &= c_z > 0\,. \end{aligned} \tag{3.25}$$

Rotational Stability

For rotational stability, the axis an object spins about is of important role. The rotation is considered stable, if it spins about the smallest or largest principal axis [GPS02]. For spinning tops for example as seen in the right picture, the desired rotational axis has to coincide with the smallest or largest principal axis. First, the object is placed such the rotation axis coincides with the z-axis (up-axis). Then the goal is to adjust the principal axis to align with the rotation axis.
The according objective function is defined as:

$$f(\alpha) = mc_z + \left(\frac{I_a}{I_{zz}}\right)^2 + \left(\frac{I_b}{I_{zz}}\right)^2 \tag{3.26}$$

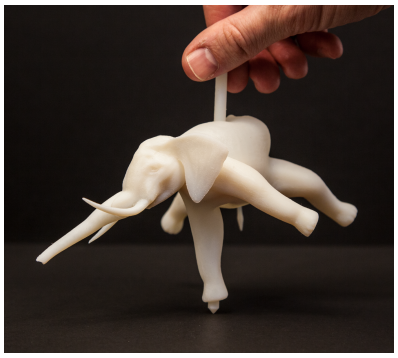


Figure 3.6: Elephant spinning top made by Bächer et al. © [BWBSH14]

where $\{I_a, I_b\}$ is obtained as the eigenvalues of the 2x2 upper-left part of the inertia tensor leading to

$$\{I_a, I_b\} = \frac{1}{2}\left(I_{xx} + I_{yy} \pm \sqrt{I_{xx}^2 + 4I_{xy}^2 - 2I_{xx}I_{yy} + I_{yy}^2}\right). \tag{3.27}$$

The constraints are defined as:

$$\begin{aligned} g_1(\alpha) &= \{c_x, c_y\} = 0 \\ g_2(\alpha) &= \{I_{xz}, I_{yz}\} = 0\,, \\ g_3(\alpha) &= c_z > 0 \end{aligned} \tag{3.28}$$

where the first constraints (g_1) ensures, that the center of mass lies on the rotation axis. The terms I_{xz} and I_{yz} have to vanish (g_2), otherwise they would influence the rotational behavior. And, of course, the center of mass has to be again above the ground (g_3).

CHAPTER 4

Implementation

The practical part of this thesis included an implementation of the described method as a C/C++ library, which was then integrated into the open source modeling software Blender[1].

Note: Modifiers in Blender are plugins written in C and therefore directly compiled into the binaries. These modifiers process a mesh and can be chained, but for this task it was enough to write a single modifier which takes the optimization parameters and then acts on the input mesh. Because Blender is an open source project, everyone can contribute. Originally, it was planned to publish this modifier in order to make it available to people interested in computational fabrication. Unfortunately, the modifier won't make it into the public release builds due to external library dependencies. Although Blender already supports some of the used libraries, it would be too much overhead to support multiple external libraries for only one modifier. Instead, the modified Blender version is now kept as an in-house solution.

In the following sections only the implementation of the library will be explained, since the explanation of the integration of a library is not part of this work.

[1] `https://www.blender.org/`

The library itself maps the processes described in Chapter 3. The basic algorithm is given in the following pseudocode:

1 compute mesh skeleton (mcf-skeleton)
2 compute displacement field $\vec{d} = V_s - V_m$,
  where V_m is the set of mesh vertices and V_s is the set of skeleton vertices
3 compute maximum inner offsets $d_i^{max} = ||\vec{d}||$
4 compute cotangent Laplacian Matrix L for input mesh
5 compute k first eigenvectors γ_i corresponding to low eigenvalues λ_i of L
6 compute manifold harmonic basis $\Gamma_k = [\gamma_1\ \gamma_2\ ...\ \gamma_k]$,
  where γ_i is the eigenvector corresponding to eigenvalue λ_i
7 minimize objective function f given additional constraints c_i

In the following sections, the implementation of these steps are explained in detail.

4.1 Linear algebra and mesh precessing

Eigen[2] was the first choice in terms of linear algebra libraries. This is not only due to its simplicity and high performance, but mainly because some other libraries build on top of it. One of those libraries is libigl[3] which eases geometry processing. For instance, it supports computing the Laplacian matrix L with cotangent weights.

4.2 Mean curvature skeleton

The computation of mean curvature skeletons is implemented in the well-known geometry processing library CGAL. Hence, implementation details have to be extracted from the CGAL documentation[4] or the original paper of Tagliasacchi et al. [TAOZ12].
After extracting a skeleton from the input mesh, the displacement vector field is computed as the difference between the mesh vertices and the corresponding skeleton vertices. The length of all displacement vectors yields the maximum amount of inner displacement per vertex. The remaining parameters, such as maximum outer displacement and minimum displacements (wall thickness), are provided by the user.

[2] http://eigen.tuxfamily.org
[3] http://libigl.github.io/libigl/
[4] https://doc.cgal.org/latest/Surface_mesh_skeletonization/index.html

4.3 Eigendecomposition of Laplacian L

ARPACK[5], a Fortran77 library, is designed to solve large-scale eigenvalue problems. Fortunately, while implementing, an open source project called Spectra[6] was released. This header-only project provides many algorithms implemented in the ARPACK library. A major advantage is, that it is built on top of Eigen.
The most important functionality is to compute the first k eigenvectors corresponding to the lowest eigenvalues of a sparse matrix. Applied to L, this yields the harmonic basis vectors which are then assembled to a matrix to form the manifold harmonic basis H_k.

4.4 Minimizing the objective function

All objective functions and many constraints are formulated in a non-linear way which makes the optimization process more complex and time consuming. Fortunately, all objective functions are at least C^1 continuous, hence gradient-based methods can be applied to find local minima. In particular, sequential least squares quadratic programming (SLSQP) proposed by Dieter Kraft [Kra88] perfectly suits the requirements because besides nonlinear functions it supports nonlinear inequality and equality constraints as well as lower and upper bounds. However, because bounds are not interpreted as hard constraints but rather are appended to the least-squares problem, this routine always led to results violating constraints. After further investigation it turned out, that projecting the bounds into the objective function (see Section 3.3.2) was enough to avoid constraint violations. The library NLopt[7] developed by Steven G. Johnson [Joh08] aims at non-linear optimization. It provides various methods for local and global optimization. A full list of all included algorithms can be obtained from the webpage[8]. Furthermore, it supports the SLSQP routine and was therefore used for the optimization process.

4.5 Limitations

Computation runtime of mean curvature skeletons scales directly with the complexity of the geometry. Objects with a high number of vertices—beginning at around 10.000 vertices—take more time for computing their mean curvature skeleton than for going through the whole process of optimization. Therefore, this is considered as a performance bottleneck of this implementation.
Another issue with more complex objects is the part of minimizing the objective function. Unfortunately, the above mentioned solver runs out of memory when dealing with more complex objects. The amount of necessary allocations of memory during evaluation of the objective function is considered as too high for the optimization routine for larger meshes—again the threshold starts about 10.000 vertices.

[5] http://www.caam.rice.edu/software/ARPACK/
[6] https://spectralib.org/
[7] https://nlopt.readthedocs.io
[8] https://nlopt.readthedocs.io/en/latest/NLopt_Algorithms/

CHAPTER 5

Results

The result of this work is split into two parts: functionality and interaction. While in the first part the focus lies on accuracy and performance of the underlying algorithm, the second part mainly deals with the integration and interaction in Blender.

5.1 Functionality

Two example shapes were optimized using the implemented Blender plugin: a box standing on one of its corners and a spinning top in form of a tilted ellipsoid. The input shapes are described and illustrated in the following sections. To compare the obtained results to the original implementation, the same shapes were optimized with the same parameters and are visualized side by side. Optimization and deformation was applied to the inner offset surface, such that the original surface represents the outer surface and therefore maintains its appearance.

5.1.1 Optimized shapes

Box: The box model shown in Figure 5.1 was optimized to stand stably on its beveled corner. To do so, the inner shape was compressed near this corner in order to lower the center of mass and move the projected center of mass inside the ground polygon. The deformed inner surface after optimization is illustrated in Figure 5.3. In addition, a cut-through visualization is given in Figure 5.5.

Ellipsoid: The second optimized model is visualized in Figure 5.2. It consists of a slightly tilted ellipsoid and a vertical stick representing the rotation axis. After optimizing according to the rotational stability objective, the remaining mass inside the ellipsoid is clumped near the rotation axis. The results are shown in Figure 5.4. In addition, a cut-through visualization is given in Figure 5.6.

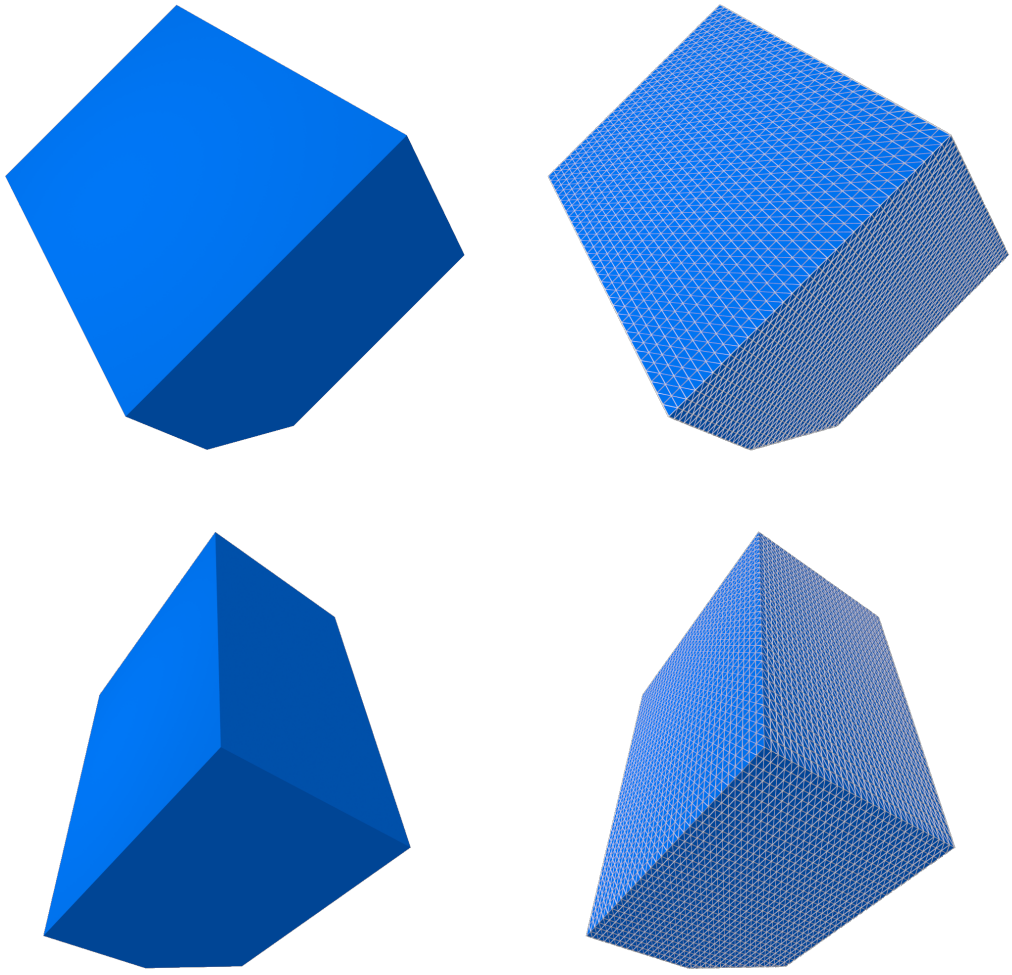


Figure 5.1: Box shape shown from two different viewpoints. Additionally, the tessellation is shown on the right side.

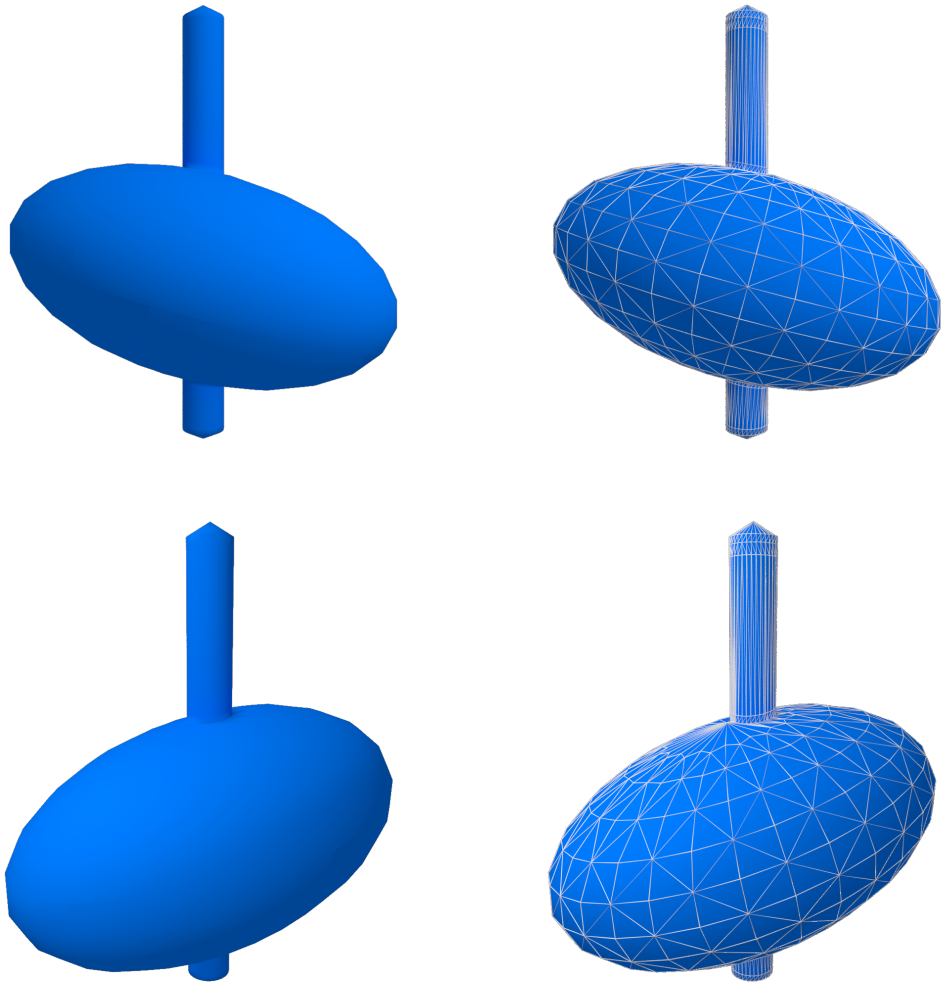
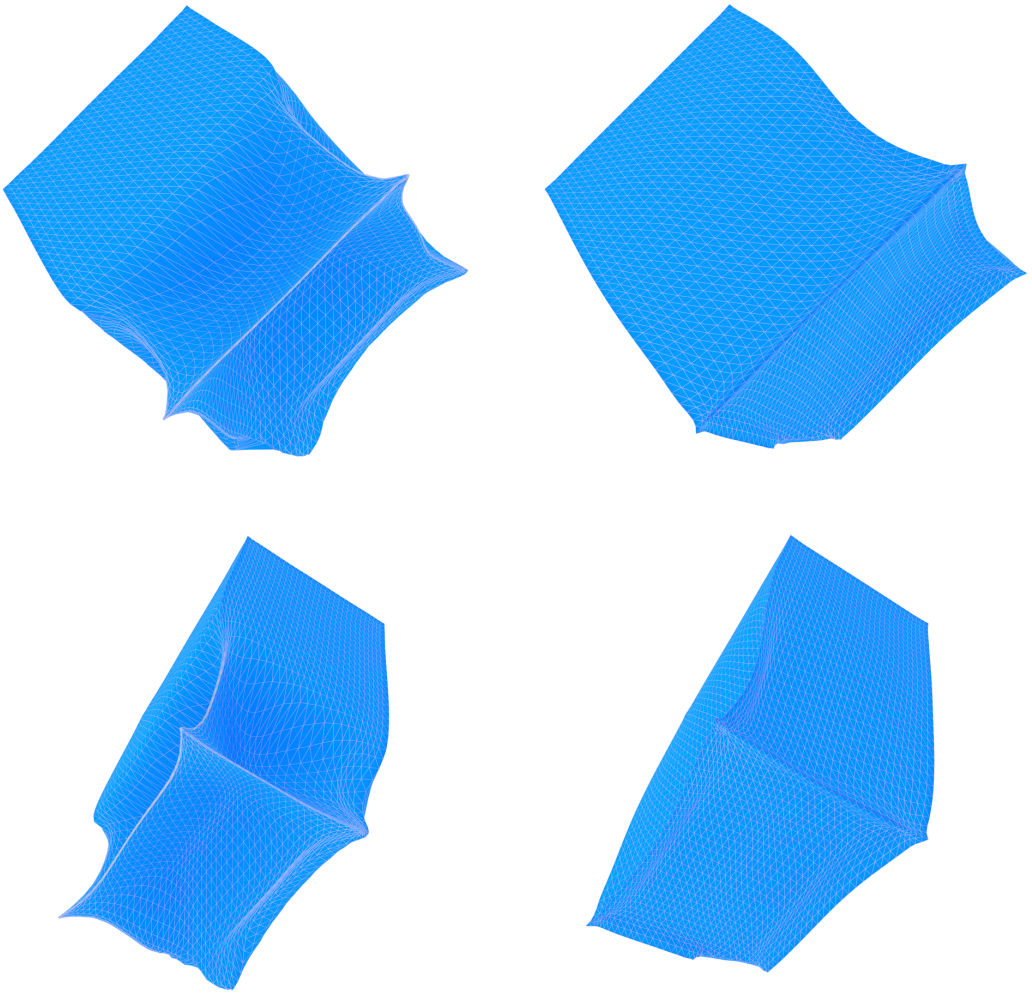
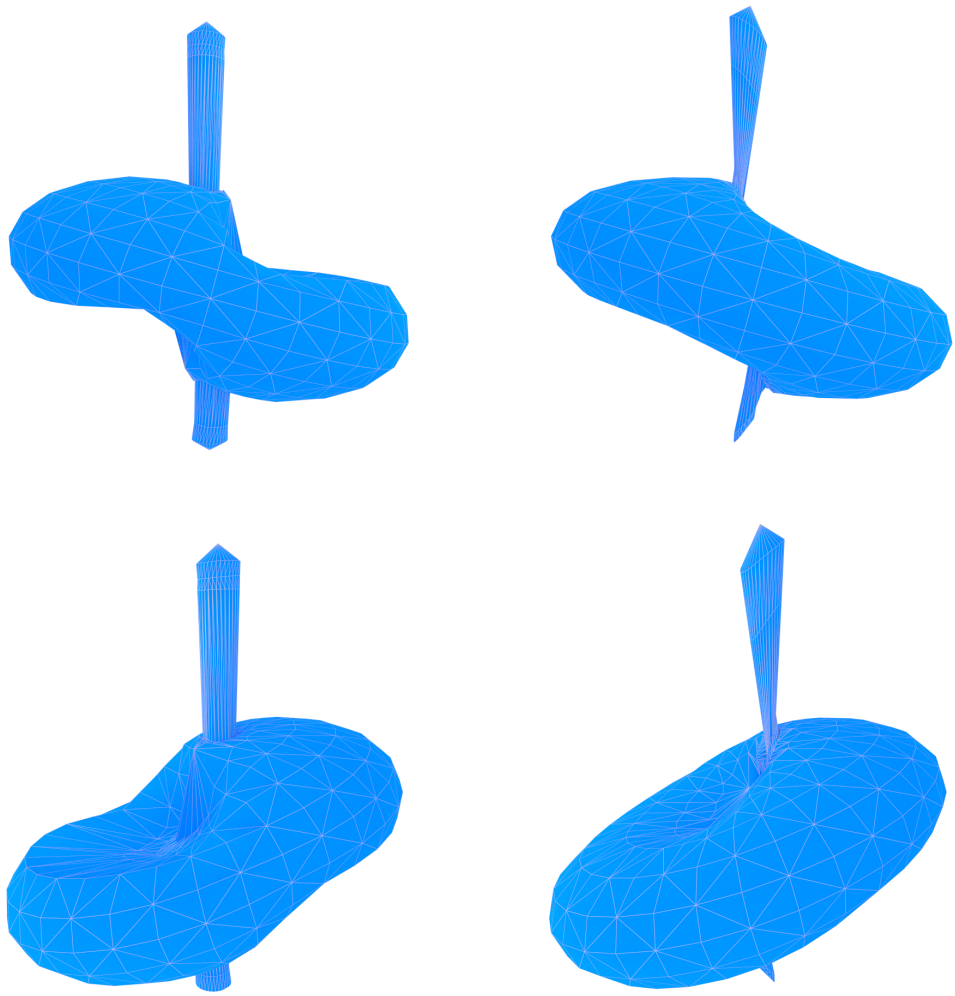


Figure 5.2: Ellipsoid shape shown from two different viewpoints. Additionally, the tessellation is shown on the right side.



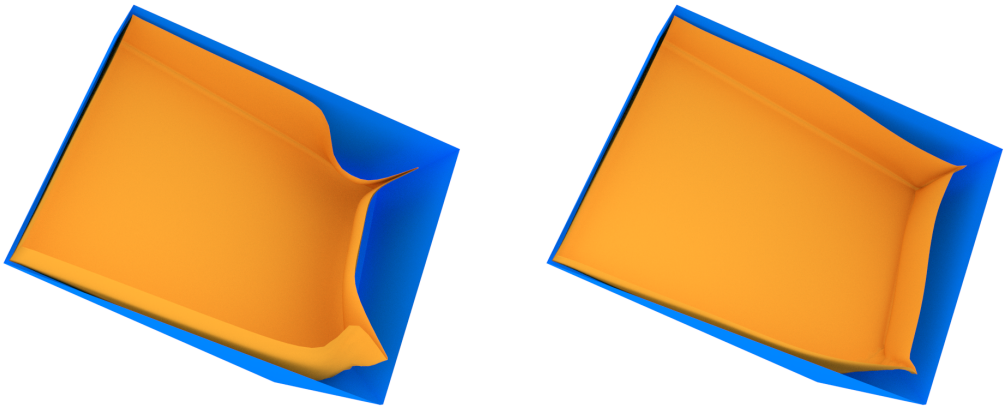
(a) (b)

Figure 5.3: Inner surface of the box model after static stability optimization from two different viewpoints. Left side (a) was generated by the implementation of this work. Right side (b) was optimized by the reference implementation.



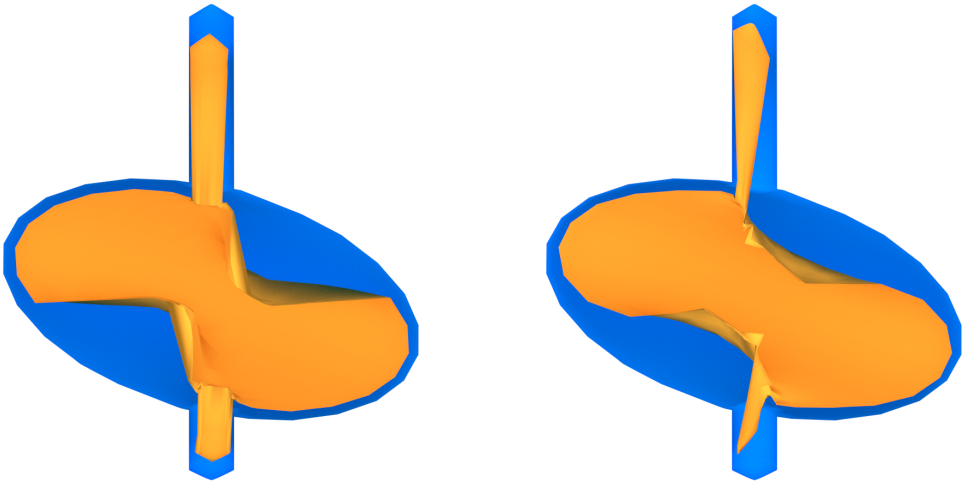
(a) (b)

Figure 5.4: Inner surface of the ellipsoid model after rotational stability optimization from two different viewpoints. Left side (a) was generated by the implementation of this work. Right side (b) was optimized by the reference implementation.



(a) (b)

Figure 5.5: Cut-through visualization of both surfaces of the box model. Volume enclosed by the two surfaces is filled with material in the later fabrication process. Left side (a) was generated by the implementation of this work. Right side (b) was optimized by the reference implementation.



(a) (b)

Figure 5.6: Cut-through visualization of both surfaces of the ellipsoid model. Volume enclosed by the two surfaces is filled with material in the later fabrication process. Left side (a) was generated by the implementation of this work. Right side (b) was optimized by the reference implementation.

Shape / Criteria	Box		Ellipsoid	
	Reference	**This work**	**Reference**	**This work**
Skeleton	3216ms	3216ms	320ms	320ms
Harmonic basis	597ms	315ms	31ms	25ms
Optimization	4402ms	3510ms	1512ms	63ms
Matrix operations	830ms	37ms	136ms	15ms
Total	9045ms	7078ms	1999ms	423ms
Objective value	4.8302	4.7506	15.9203	13.7436

Table 5.1: Performance results of box model (6129 vertices) optimized according to static stability objective (left) and results of ellipsoid model (750 vertices) according to rotational stability objective (right). All numbers are averaged over 5 runs.

5.1.2 Performance and accuracy

The comparison between reference and this implementation shows two major differences:

1. Stronger deformation of inner surface near critical regions
2. Differences in runtime and resulting objective value

According to the resulting inner surfaces illustrated in Figure 5.3 and Figure 5.4, the differences in deformation between both implementations are easily visible. The implementation proposed in this thesis tends to make bigger changes near critical regions instead of an overall shape deformation as seen in the reference implementation. This is caused by the projection of lower and upper bounds into the objective function. Involving the arc tangent gradient into the optimization leads to bigger steps near obviously feasible regions and allows for better exploitation of regions close to the actual bound limits. This results in a slightly better result in terms of objective value and an overall performance boost (see Table 5.1). Furthermore, the total runtime and the runtimes of each task are smaller than in the reference implementation. One exception is the computation of mean curvature skeletons, where both implementations rely on the same library. Although the reference implementation is based on MATLAB and therefore highly parallelized, the optimized C/C++ code was faster in terms of matrix operations. Regarding computation of the manifold harmonic basis and non-linear optimization, one has to mention that MATLAB does not rely on a single solving routine when optimizing, instead it uses a bunch of different solvers, each better suited for a different situation and then chooses whichever fits the best in the current situation. This implies more overhead during computation time and a longer ramp up time and is therefore outperformed by a single predefined solving routine.
All tests were performed on an AMD FX-8320 with 8GB DDR3 1600 MHz.

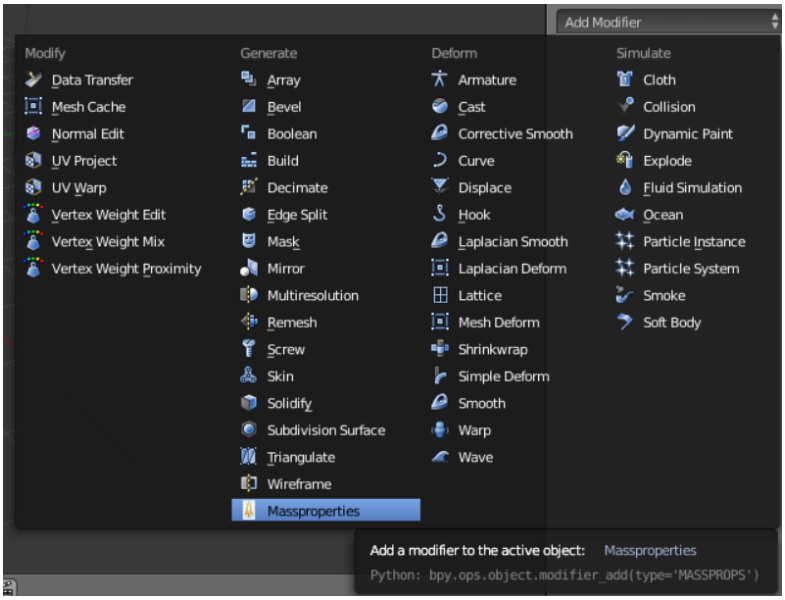


Figure 5.7: The list of modifiers in Blender split into 4 categories.

5.2 Integration and interaction

Modifiers in Blender are programs compiled into the binaries which either alter the appearance of a mesh or generate additional geometry. Each object owns a modifier stack, where multiple modifiers can be executed on top of each other. The list of modifiers in Blender is split into multiple categories: *modify*, *generate*, *deform* and *simulate*. Since the algorithm generates an offset surface, it is available in the *generate* section as shown in Figure 5.7. The name *Massproperties* suits the description of the modifie, because it manipulates the interior of the mesh to change its properties of mass. Data is provided via a custom layout of input controls written in Python. In Figure 5.8, the required input data of the algorithm is depicted. A description of the different input fields is stated in Table 5.2. After hitting the *Apply*-Button, the modifier invokes the optimization routine and stalls the user interface. This might take a few moments (see Section 5.1.2) and after returning the offset surface is added to the scene. Currently the interface only allows for optimizing the shape by displacing the inner surface, although the implemented algorithm is capable of changing the outer or both offset surfaces.

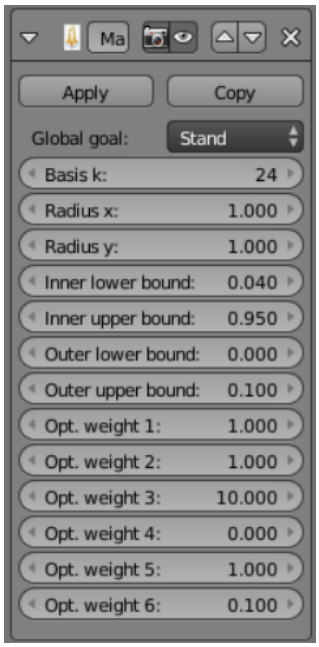


Figure 5.8: Input layout for the Blender modifier *Massproperties*.

Input field	**Description**
Global goal	Type of objective function. Possible values are Stand (static stability) and Spin (rotational stability)
Basis k	Number of basis vectors used to represent the manifold harmonic basis
Radius x	Radius of the contact area on x-axis
Radius y	Radius of the contact area on y-axis
Inner lower bound	Minimum offset for inner surface to ensure minimal wall thickness.
Inner upper bound	Maximum offset for inner surface to avoid self intersections. Factor is multiplied by computed upper bounds (see Section 3.3.2)
Outer lower bound	Minimum offset for outer surface to ensure minimal wall thickness
Outer upper bound	Maximum offset for outer surface to ensure maximal wall thickness
Opt. weight 1	Additional weight to minimize volume in objective function
Opt. weight 2	Additional weight to lower center of mass and penalize positions outside of the projected contact area in objective function
Opt. weight 3	Additional weight to minimize moments of inertia in objective function
Opt. weight 4	Currently unused weight
Opt. weight 5	Additional weight for constraints
Opt. weight 6	Additional weight to penalize outer displacements

Table 5.2: Input fields of the Blender modifier *Massproperties*.

CHAPTER 6

Conclusion

In this work a novel method for optimizing the physical properties of a 3D object before fabrication presented by Musialski et al. [MAB+15] was implemented as a C/C++ library. A further improvement presented in the work [MHR+16] extends the original algorithm and was integrated into the library. Eventually, the library was included in the open source modeling software Blender.

While implementing the algorithm in C/C++, two main problems occurred, which took a long time to solve. Computing just a small number of eigenvectors corresponding to the largest eigenvalues from a sparse matrix was only supported by a subroutine implemented in a Fortran77 library called ARPACK. Luckily, this issue could be solved after a few weeks because the header-only C++ library Spectra was released on Github, which implemented exactly the needed routine. The second problem was concerning the optimization. NLopt was the only open source C/C++ library supporting non-linear objective functions, equality/inequality constraints and lower and upper bounds. Unfortunately, because of too many constraints, the solution always ended up in an invalid configuration and therefore violated the constraints. After integrating the lower and upper bound projection, it finally led to promising results.

Regarding results, the offset surfaces generated by this implementation are quite similar to the original ones presented by the authors. However, the method shown in this work leads to bigger changes to the inner surface near critical regions instead of a more uniform deformation in order to further minimize the objective function.

Integrating support for more objective functions as discussed in the original paper or improving the Blender modifier by creating a more intuitive user interface are considerations for future work.

List of Figures

Bibliography

[BW+14+] Moritz Bächer, Emily Whiting, Bernd Bickel, and Olga Sorkine-Hornung. Mass properties of triangulated solids and their derivatives.

[BWBSH14] Moritz Bächer, Emily Whiting, Bernd Bickel, and Olga Sorkine-Hornung. Spin-it: Optimizing moment of inertia for spinnable objects. *ACM Transactions on Graphics (TOG)*, 33(4):96, 2014.

[Dem15] Wolfgang Demtröder. *Experimentalphysik 1: Mechanik und Wärme*. Springer-Verlag, 2015.

[Ger12] Neil Gershenfeld. How to make almost anything: The digital fabrication revolution. *Foreign Aff.*, 91:43, 2012.

[GPS02] Herbert Goldstein, Charles Poole, and John Safko. Classical mechanics, 2002.

[Joh08] Steven G. Johnson. Nlopt. `http://ab-initio.mit.edu/nlopt`, 2008. Retrieved at: 30.09.2017.

[Kra88] Dieter Kraft. A software package for sequential quadratic programming. *Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt*, 1988.

[MAB+15] Przemyslaw Musialski, Thomas Auzinger, Michael Birsak, Michael Wimmer, and Leif Kobbelt. Reduced-order shape optimization using offset surfaces. *ACM Trans. Graph*, 34(4):102, 2015.

[Mae99] Takashi Maekawa. An overview of offset curves and surfaces. *Computer-Aided Design*, 31(3):165–173, 1999.

[MHR+16] Przemyslaw Musialski, Christian Hafner, Florian Rist, Michael Birsak, Michael Wimmer, and Leif Kobbelt. Non-linear shape optimization using local subspace projections. *ACM Transactions on Graphics (TOG)*, 35(4):87, 2016.

[Mot11] Catarina Mota. The rise of personal fabrication. In *Proceedings of the 8th ACM conference on Creativity and cognition*, pages 279–288. ACM, 2011.

[PP93] Ulrich Pinkall and Konrad Polthier. Computing discrete minimal surfaces and their conjugates. *Experimental mathematics*, 2(1):15–36, 1993.

[PWLSH13] Romain Prévost, Emily Whiting, Sylvain Lefebvre, and Olga Sorkine-Hornung. Make it stand: balancing shapes for 3d fabrication. *ACM Transactions on Graphics (TOG)*, 32(4):81, 2013.

[TAOZ12] Andrea Tagliasacchi, Ibraheem Alhashim, Matt Olson, and Hao Zhang. Mean curvature skeletons. In *Computer Graphics Forum*, volume 31, pages 1735–1744. Wiley Online Library, 2012.

[VL08] Bruno Vallet and Bruno Lévy. Spectral geometry processing with manifold harmonics. In *Computer Graphics Forum*, volume 27, pages 251–260. Wiley Online Library, 2008.

[WW16] Lingfeng Wang and Emily Whiting. Buoyancy optimization for computational fabrication. *Computer Graphics Forum (Proceedings of Eurographics)*, 2016.

[ZC01] Cha Zhang and Tsuhan Chen. Efficient feature extraction for 2d/3d objects in mesh representation. In *Image Processing, 2001. Proceedings. 2001 International Conference on*, volume 3, pages 935–938. IEEE, 2001.