Shape design, repair and optimization

by

Siqi Wang

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

August, 2025

Professor Denis Zorin

Professor Daniele Panozzo

Dedication

To my parents with love, and my advisors for their invaluable guidance.

Acknowledgements

This dissertation would not have been possible without the support, guidance, and encouragement of many individuals. I would like to take this opportunity to express my deepest gratitude to them.

First and foremost, I would like to express my sincere appreciation to my advisors, Prof. Daniele Panozzo and Prof. Denis Zorin, for their invaluable guidance, mentorship, and continuous support throughout my PhD. Your expertise, insightful discussions, and constructive feedback have greatly shaped both my research and personal growth. Thank you for being not only outstanding advisors but also wonderful friends.

I am deeply grateful to the members of the Geometric Computing Lab (GCL). Thank you, Dr. Daniel Zint, for your guidance, which made our research project run smoothly. Thank you, Dr. Michael Tao, for being a supportive mentor and friend—I have learned so much from you, not only about innovative research ideas but also about essential coding skills. Thank you, Prof. Teseo Schneider, for your meticulous guidance during my early projects, which laid a solid foundation for my research. Thank you, Ryan Capouellez and Dr. Davi Colli Tozoni, for sharing the office with me and for the daily discussions. Thank you, Yunfan Zhou, for always being optimistic and cheering me up whenever I felt down during my research.

I am also thankful to Prof. Riccardo Lattanzi, Dr. José E. Cruz Serrallés and Dr. Ilias Giannakopoulos from the Department of Radiology at NYU Langone Health for being excellent collaborators and for greatly expanding my knowledge of biomedical imaging.

I was fortunate to have two incredible internships as a PhD student, at Adobe Research and Roblox. To Prof. Alec Jacobson, thank you for your kind guidance and exceptional support during my internship at Adobe and further collaborations. To the entire geometry team at Roblox, and especially Dr. Hsueh-Ti Derek Liu and Janos Meny, thank you for teaching me how to grow as a researcher, and thank you for being outstanding advisors and collaborators during my internship and the subsequent paper submission.

I would also like to thank all my collaborators and co-authors: Prof. Alec Jacobson, Prof. Riccardo Lattanzi, Prof. Teseo Schneider, Dr. Chenxi Liu, Dr. José E. Cruz Serrallés, Dr. Ilias Giannakopoulos, Dr. Daniel Zint, Dr. Hsueh-Ti Derek Liu, Janos Meny, Dr. Mehdi Rahimzadeh, Dr. Izak Grguric, Dr. Chase Tymms, Dr. Zizhou Huang and Damien Chen.

I extend a special thank you to Prof. Bo Zhu, who believed in my potentials, and first introduced me to the field of Computer Graphics and Fluid Simulation as an undergraduate.

Thank you to the NYU IT High-Performance Computing team, especially Shenglong Wang, for their support and for providing the computational resources necessary for my research.

I would also like to acknowledge Prof. Ken Perlin, Prof. Riccardo Lattanzi, and Prof. Alec Jacobson for generously serving on my dissertation committee and providing invaluable feedback and suggestions.

I am deeply grateful to Dingyi Zhuang for being a patient and supportive partner, offering constant encouragement during the most challenging moments of my PhD.

Finally, and most importantly, I am profoundly grateful to my parents for their unwavering love, patience, and constant support. Their encouragement and belief in me have been the foundation of my academic journey, and I dedicate this achievement to them.

Abstract

Digital geometric models are fundamental to modern engineering, media, and manufacturing. However, models created by artists in-the-wild often contain ambiguities that precludes their use in simulation and manufacturing, while complex designs may need to be simplified for efficiency or functionally optimized to meet competing aesthetic and performance goals. This necessity for robust, useful, and high-performing geometry creates a critical need for advanced computational techniques that can automatically repair, simplify, and optimize digital shapes. Our research addresses these challenges by developing a suite of shape processing and optimization methods designed to enhance the quality and functionality of geometric models for a range of applications.

This thesis delivers solutions across three key areas. First, we present a Bézier curve simplification framework that simplifies complex vector graphics while preserving visual fidelity by defining a curve-to-curve distance metric and repeatedly conducting local segment removal operations. Second, we propose a solid or shell labeling technique for artist-created surface meshes that lack a well-defined interior, guided by a sparse set of user inputs. These labels reduce ambiguity and enable the construction of valid volumetric meshes for downstream applications. Finally, we introduce two powerful shape optimization frameworks: one that leverages neural network-based models to independently control the tactile properties and visual appearance of a texture, and another that optimizes the geometry and position of radiofrequency (RF) receive coil arrays to increase signal-to-noise ratio (SNR) in magnetic resonance imaging (MRI).

Contents

List of Figures

List of Tables

1 Introduction

Contemporary advancements across various domains, from artistic endeavors to advanced engineering, such as 3D printing, CNC milling, and biomedical manufacturing, have made it easier and more cost-effective to produce highly customized and high-performing objects, ranging from intricate animated assets to precise medical devices. This drives continuous innovation in computational methodologies. However, the lifecycle of digital shapes, from initial conceptualization to final fabrication, is often challenged by complexities such as eliminating redundancy, rectifying inherent imperfections in acquired data, and optimizing the geometric representation to achieve specific functional or physical properties. These challenges demand sophisticated computational tools that can automate and enhance the design, refinement, and analysis processes beyond what traditional manual or heuristic approaches can offer.

In this context, computational design and geometric processing plays a pivotal role, offering a broad spectrum of techniques that assist designers and engineers in creating new and improved objects and tools tailored to diverse needs. The primary objective is to develop and apply robust algorithms capable of precisely defining, analyzing, and repairing geometric data. This involves simulating various behaviors, extracting key quantities of interest, and intelligently adjusting the shape or its representation to improve an objective function, which measures the quality and performance of the digital structure.

The recent rise of advanced computational methods, including sophisticated optimization algorithms, machine learning techniques, and robust simulation frameworks, has led to significant breakthroughs in tackling these problems. These approaches have been highly successful in material design [175, 199], structural mechanics [79, 181], and fluid dynamics [117], but have yet to be effectively applied to magnetic resonance imaging (MRI) coil design. This opens up substantial potential for enhancing coil performance by parameterizing and optimizing the geometric features of a given coil configuration.

Beyond engineering performance, the realm of perceptual design introduces distinct challenges. Tactile textures, for example, are ubiquitous across fabricated objects—from toys and woven fabrics to human skin and manufactured surfaces—often serving specific practical or aesthetic purposes. Despite the common task of creating a particular tactile feeling, existing work has largely overlooked the ability to independently control tactile and visual properties. Advanced fabrication technologies like high-resolution 3D printing make it possible by the highly flexible control of both visual and tactile texture.

Similarly, in the domain of vector graphics processing, maintaining geometric efficiency without sacrificing artistic intent poses a significant hurdle. While artists craft designs using vector graphics, these assets can frequently become overly dense or detailed through various processes, such as aggressive upsampling before pointwise filters, inadvertent anchor point accumulation during curve manipulation, or inefficient raster art vectorization. Consequently, simplification becomes a core sub-routine in popular vector graphics editing tools (e.g., Inkscape, Adobe Illustrator, CorelDRAW). However, current simplification methods often leave substantial room for improvement, particularly when aiming for highly accurate or even lossless reduction while preserving the artist's original intention.

Furthermore, preparing digital models for physical simulation and fabrication often requires transforming surface representations into volumetric meshes. Generating volumetric meshes from a given shape is a fundamental subroutine within geometric computation and modeling, enabling critical downstream applications such as Constructive Solid Geometry (CSG) modeling and simulations. In practice, however, existing volumetric meshing algorithms frequently produce unexpected results, stemming from the assumption that the input triangle mesh defines a well-defined solid volume. This assumption often fails for artist-created assets or complex real-world geometries lacking a clear interior. Therefore, obtaining user-desired face labels—distinguishing between *solid* faces for tetrahedral meshing and *shell* faces for offset meshing—is paramount. Incorrect labeling can lead to significant artifacts in the resulting volume mesh, including distorted appearance, unintended deletion of invisible structures, and unpredictable physical behavior.

In this thesis, we focus on developing novel computational methods for digital shape creation, editing, repair, and optimization across diverse applications. In Chapter 2, we introduce a Bézier splines simplification framework that simplifies vector graphics as much as possible while maintaining exceptional accuracy to preserve the artist's intention. In Chapter 3, we present a method for volumetric meshing by classifying each face of a triangle mesh as belonging to either a *solid* or *shell* component, requiring only sparse user guidance. Further, in Chapters 4 and 5, by leveraging optimization frameworks, we develop pipelines to independently control the tactile properties and visual appearance of a texture, and automatically improves the radiofrequency (RF) coil configurations for better signal-to-noise ratio (SNR) performance in MRI.

2 | Bézier Spline Simplification Using Locally Integrated Error Metrics

This chapter is adapted from the publication [196] on SIGGRAPH Asia 2023, a joint work with Chenxi Liu, Daniele Panozzo, Denis Zorin, and Alec Jacobson.

ABSTRACT

Inspired by surface mesh simplification methods, we present a technique for reducing the number of Bézier curves in a vector graphics while maintaining high fidelity. We propose a curve-to-curve distance metric to repeatedly conduct local segment removal operations. By construction, we identify all possible *lossless* removal operations ensuring the smallest possible zero-error representation of a given design. Subsequent *lossy* operations are computed via local Gauss-Newton optimization and processed in a priority queue. We tested our method on the OpenClipArts dataset of 20,000 real-world vector graphics images and show significant improvements over representative previous methods. The generality of our method allows us to show results for curves with varying thickness and for vector graphics animations.

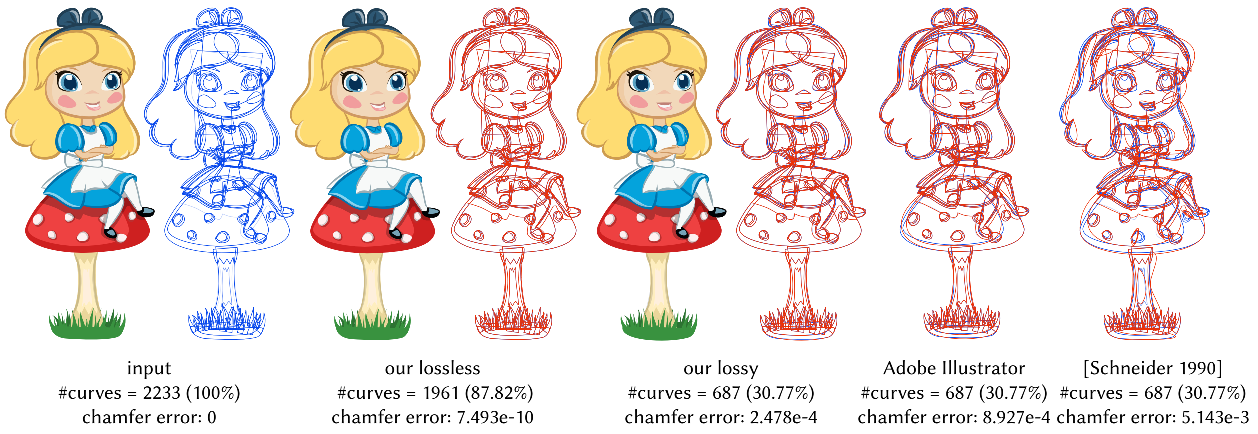


Figure 2.1: Left to right: Our method accepts as input collections of Bézier curves (left) and simplifies them *losslessly* to remove redundant control points. Our lossy extension produces much more simplified result without sacrificing visual quality. Simplifying using Adobe Illustrator and the method of Schneider [163] (in Inkscape) produce worse results for the same number of curves. Input image by Olga Bikmullina (CC0).

2.1 Introduction

Simplification is a core sub-routine found in any popular vector graphics editing tool (e.g., Inkscape, Adobe Illustrator, CorelDRAW). There are many ways that a design could end up with too many or too detailed curves: densely upsampling before applying a pointwise filter, adding anchors along curves but forgetting to delete them if they're never moved, scaling down a detailed design relative to its final display resolution, vectorizing raster art inefficiently, etc. With increasingly popular vector graphics *animation* formats (e.g., Lottie), dense vector graphics designed for static display or print may now expect to be served up to mobile devices at high framerates. Furthermore, vector graphics also function as path descriptions for CNC machines such as laser cutters, routers, and plotters. Overly dense designs cause fabrication defects or firmware failure.

In this paper, we demonstrate that existing simplification methods leave significant room for improvement. We are particularly interested at the high end of the simplification-accuracy curve: simplifying as much as possible while remaining exceptionally accurate to preserve the artist's intention. A critical unit test is recovering lossless simplification. For example, take a coarse

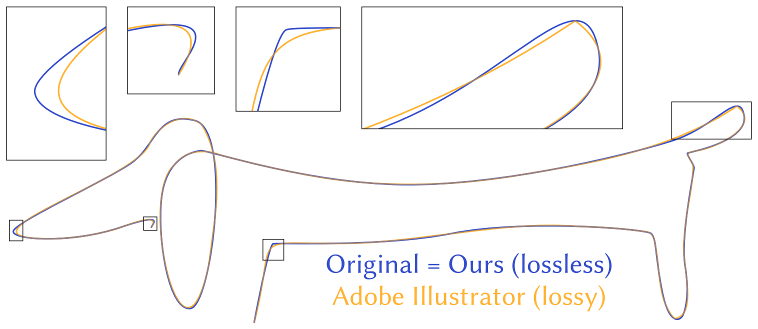


Figure 2.2: As a stress test, the blue curve is upsampled in place from 19 to 304 segments. Our method losslessly simplifies back to the original 19 segments. Adobe Illustrator's proprietary simplification to 19 segments produces noticeable error.

spline, subdivide it repeatedly, and then try to simplify back to the original number of curves (see Fig. 2.2). Existing methods found in the literature and in commercial software fail this seemingly simple test.

We propose a Bézier spline simplification technique inspired by progressive surface mesh simplification methods. We conduct local segment removal operations in a priority queue. These operations are based on a measure of curve-curve distance which is carefully constructed to efficiently identify *lossless* removals as zero-cost operations. Once lossless removals are exhausted, subsequent lossy operations based on local Gauss-Newton optimization are conducted in a greedy manner (see Fig. 2.3).

To evaluate our method, we conduct a large-scale — first of its kind — benchmark comparison on a large dataset of vector graphics images. Our method consistently outperforms representative state-of-the-art methods. Our method is agnostic to the dimension of the input curves' embedding space: we show results appending varying stroke thickness as a 3rd coordinate. We also demonstrate simplifying across an entire animation, implicitly ensuring temporally coherent control point distribution and movement.

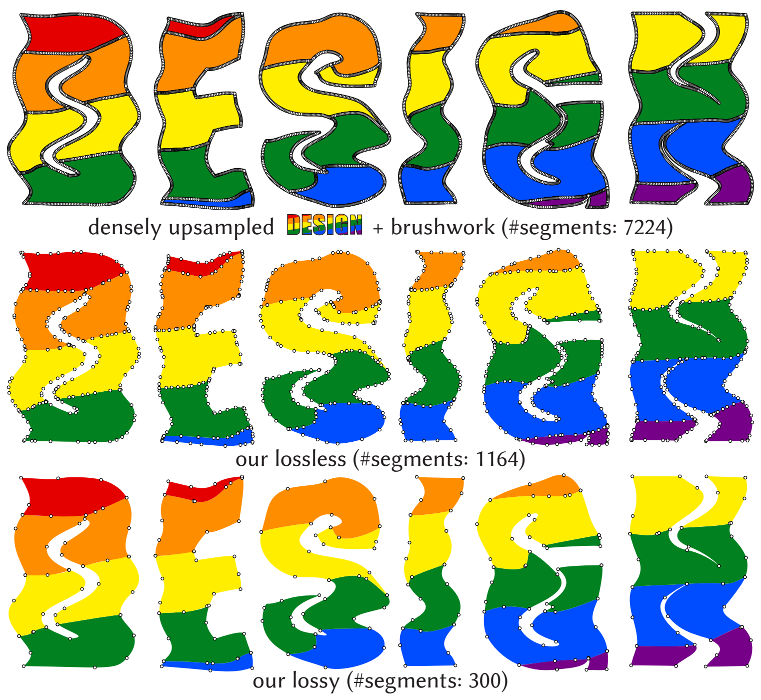

densely upsampled DESIGN + brushwork (#segments: 7224)
our lossless (#segments: 1164)
our lossy (#segments: 300)

Figure 2.3: Simplification is a core subroutine in an editing setting. Densely upsampling this input allows an artist's WARP brushwork in Illustrator to apply a pointwise deformation over the design. Our lossless simplification removes geometrically unnecessary segments, and further lossy simplification produces a coarse, yet visually accurate design.

2.2 Related Work

The literature on simplification for polylines is vast beginning with the error-bound method of Douglas and Peucker [38], which still enjoys practical use. We focus our attention on methods for Bézier curve simplification and related problems on smooth curves.

Sampling-and-refitting. Schneider [163] proposes a thorough and widely-used G^1-continuous solution to the general problem of fitting a cubic Bézier spline to a digitized curve, which can also be applied to Bézier curve simplification, as is used in the *Path Simplify* tool of Inkscape. Similar algorithms include [23, 130, 159, 160] based on recursive segment subdivision, characteristic points detection, and least-square fitting or other curve approximation approaches. Kolesnikov [97] introduces the inflection points with relaxed constraints of tangent continuity. Shao and

Zhou [170] proposes a global least-squares optimization fitting method with critical points identification to fit the input data points using block coordinate descend to do non-linear optimization. This global optimization technique struggles to find a global minimum for long complicated chains. Mokhtarian, Ung, and Wang [134] fits the digitized contours through curvature scale space techniques. Goethem et al. [59] takes polygon as input and allows topology-preserving Bézier curves fitting with Voronoi diagrams. The software Potrace can also transform bitmaps into vector graphics but takes a faster and simpler approach that only generates a subset of all possible Bézier curves. These methods are mostly aimed at fitting digitized curves or boundaries of bitmap images. However, in our setting, sampling and refitting the input introduces errors, cannot guarantee consistent tangents at endpoints, and cannot provide a lossless solution. Besides, the method has to pick a point on the closed loop to serve as the overlapping start and end points of the chain, which cannot be moved, and the selection is not optimized. These papers experimented on a small set of representative examples but were not tested on a large dataset. From the comparison of the latest paper [130] among them, the out-performance over [163] is insignificant. Thus, we use the widely-implemented method [163] as a representative comparison in our benchmark.

Cubic Bézier fitting. De Boor, Höllig, and Sabin [32] describes an interpolation scheme that matches position, tangents, and curvature at the endpoints and proves convexity preservation and $O(n^6)$ error scaling; but the solution is not guaranteed to exist for all input data. Penner [143] presents three criteria – fitting curvature at endpoints, least squares orthogonal distance fitting, and the fitting center of mass, with the former two relevant to our problem. However, the presented method for orthogonal distance fitting requires an initial setting of the parameters and is very slow. Levien [112] provides a highly efficient and accurate solution to cubic Bézier curve fitting by building an error metric to match the signed area of a cubic Bézier curve, which involves solving a quartic-polynomial system, and making careful decisions in the subdivision. However, this method sometimes generates cusps as one of the solutions. We included a comparison of the reference implementation of [112] (Kurbo) with our method in Section 2.4.

B-spline knot removal. Lyche and Mørken [125] proposes a knot removal algorithm for parametric B-spline curves, which aims to reduce the number of polynomial segments in a curve without exceeding a given error bound. Jupp [89] and Loach and Wathen [122] keep the number of knots fixed but optimize their positions. Kang et al. [90] solves a sparse optimization problem followed by additional fitting to determine the number and positions of knots. Dierckx [37] reviews a variety of methods for fitting curves, including variable knots, choosing number and initial knot placement, adding smoothing terms, and convexity preservation constraints. Some of these methods use a Gauss-Newton optimization akin to ours to minimize their respective distance metrics. Dung and Tjahjowidodo [40] presents a new strategy for fitting any forms of the curve by B-spline functions via a local algorithm by first splitting the data with bisection and then optimizing via the non-linear least-squares technique.

Vector graphics animation. Dalstein, Ronfard, and Van De Panne [31] allows features of a connected drawing to have topological changes via keyframes. Liu, Jacobson, and Gingold [121] provides a solution to animate an SVG with linear blend skinning interactively via least-squares fitting the control points of the input splines (whose density is defined by the user). Some 2D animation software, such as Cartoon Animation 5, supports the vector graphics format, but none pays attention to simplifying the SVG time series and keeping temporal coherency. In the animation software that has the functionality of adding spring bones and Free-Form Deformation (FFD), our work can also act as an adjoint tool to remove the redundant control points, add control points in the necessary regions, and export a smooth and clean SVG sequence. To maintain temporal coherency, a lot of work in mesh animation uses principle component analysis (PCA) or clustered principal component analysis (CPCA) to achieve animation compression, e.g. [3, 92, 111, 161, 188]. None of these methods can export a simplified sequence of vector graphic frames.

2.3 Method

Our method takes as input a vector graphics image and outputs a simplified vector graphics image. We operate directly on the underlying cubic Bézier splines representing the paths of standard vector graphics formats (e.g., `.svg` or `.ai` files). Consistent with common practice of vector graphics software (Inkscape, Illustrator, CorelDRAW) during editing, we homogenize all paths (e.g., circles, arcs, ellipses, quadratic Béziers) into cubic Bézier splines. These paths may define strokes, fill boundaries, clip-mask path boundaries, and other stylings. Our method is agnostic to stylings, which are reassociated with the simplified output paths to render the final image. We'll use the following terminology in this paper:

- **segment**: a single cubic Bézier curve,
- **chain**: G^1 continuous sequence of one or more segments,
- **component**: C^0 continuous sequence of one or more chains,

and unless context demands, Bézier always refers to cubic Bézier.

Thus, the *input* to our method is a collection of N segments in $\mathbb{R}^d$ and a target number of output segments K. The *output* is a collection of K segments in $\mathbb{R}^d$ whose components closely match corresponding input components.

2.3.1 Overview

Our method works in a greedy manner analogous to edge-collapse decimators for triangle mesh simplification in computer graphics [48, 74]. We define local "collapse" operations which remove a single segment and adjust neighboring control points on the component. We define a non-negative (and sometimes zero) cost function to measure the loss incurred by each operation. All possible operations are placed into a priority queue and processed in a greedy manner. When

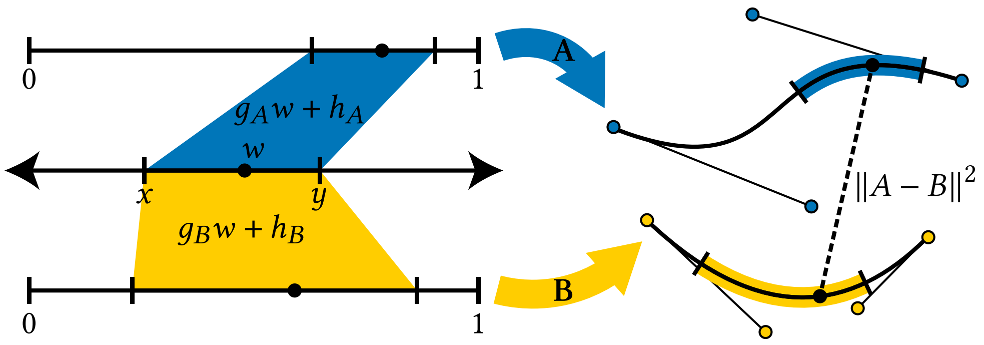


Figure 2.4: We measure the distance between partial parametric spans of two Bézier segments as an integral over a shared parametric domain that is linearly warped into each segment's Bézier parameter space.

a collapse operation is conducted, operations involving neighboring segments become out of date and are replaced in the queue with operations referencing the newly repositioned control points. All local operations are constructed to have $O(1)$ complexity so that total processing is asymptotically efficient: $O(N \log N)$. Let us now consider the definitions of operations and cost functions.

2.3.2 Distance between two segments

A fundamental expression in the following section will be a measure of distance between parametric spans of two Bézier segments.

There are many ways to measure distance between two curves. Hausdorff distance is difficult to compute precisely and sensitive to outliers [1]. Chamfer distance or integrated closest point is a good choice perceptually, but does not have a closed form expression and is sensitive to (near) overlaps and self-intersections. Meanwhile, Fréchet distance leverages that both curves may be parameterized over the real line segment [0,1]. Fréchet distance considers *all possible* re-parameterizations of the two curves. Imagine taking the max distance between your left fingertip as it runs over one curve and your right fingertip as it runs over the other. Fréchet distance takes the minimum across *all* possible parameterizations of each curve, or all possible speeds that our fingertips move relative to each other. Precise computation of Fréchet distance is somewhat tractable [157] but has similar outlier issues as Hausdorff: ultimately the measure is determined by a single pair of points [1].

Similar to Fréchet, we propose considering all possible pairs of piecewise-linear reparametrizations of each curve with parameter locations co-located at each segment boundary, but unlike Hausdorff or Fréchet we take the integrated squared distance along the curves. That is, the Bézier parameterization of each segment is only affected by a linear mapping. We will show this is both friendly to computation and a good model of perceived distance, as indicated by our qualitative results. Additionally, the distance is zero if and only if a lossless merge is possible and smoothly increases away from this case, enabling gradient-based minimization.

Consider a Bézier segment $\mathbf{A} : [0,1] \to \mathbb{R}^d$ defined[1] by control points $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \mathbf{a}_4 \in \mathbb{R}^d$. Similarly, define another curve $\mathbf{B} : [0,1] \to \mathbb{R}^d$ with control points $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_4 \in \mathbb{R}^d$. The parametric domains of $\mathbf{A}$ and $\mathbf{B}$ are both [0,1], but we will integrate distance over an arbitrary parametric subdomain:

$$\int_x^y \|\mathbf{A}(g_A w + h_A) - \mathbf{B}(g_B w + h_B)\|^2 \, dw, \tag{2.1}$$

where various parameters appear to control the measure: x, y control the stretch of a shared parametric domain from which the variable of integration w spans, g_A, h_A and g_B, h_B are parameters which define an *affine* map from this shared domain to the Bézier parameter domains of curves $\mathbf{A}$ and $\mathbf{B}$, respectively.

2.3.3 Distance between two chains

Building on segment-segment distance, we now consider distance between two chains. Consider a chain of n segments: $\mathcal{A} : [0,1] \to \mathbb{R}^d$ defined by control points $\mathbf{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_{3n+1}\}$ and a

[1] $\mathbf{A}(u) = (1-u)^3\, \mathbf{a}_1 + (1-u)^2 u\, \mathbf{a}_2 + (1-u)u^2\, \mathbf{a}_3 + u^3\, \mathbf{a}_4$.

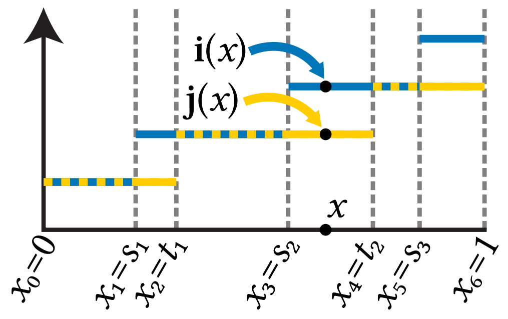


Figure 2.5: Piecewise-constant indexing functions for $\mathbf{i}(x)$ and $\mathbf{j}(x)$.

set of $n+1$ parameter locations $s_0, s_1, \ldots, s_n \in [0,1]$ where $s_0 = 0$ and $s_n = 1$:

$$\mathcal{A}(x) = \begin{cases} \mathbf{A}^1\left(\frac{x-s_0}{s_1-s_0}\right) & \text{if } x \leq s_1 \\ \mathbf{A}^2\left(\frac{x-s_1}{s_2-s_1}\right) & \text{else if } x \leq s_2 \\ & \vdots \\ \mathbf{A}^i\left(\frac{x-s_{i-1}}{s_i-s_{i-1}}\right) & \text{else if } x \leq s_i \\ & \vdots \\ \mathbf{A}^n\left(\frac{x-s_{n-1}}{s_n-s_{n-1}}\right) & \text{else if } x \leq s_n, \end{cases} \tag{2.2}$$

where points $\mathbf{p}_{3i-2}, \mathbf{p}_{3i-1}, \mathbf{p}_{3i}, \mathbf{p}_{3i+1}$ control each segment $\mathbf{A}^i$. Similarly, define a chain $\mathcal{B} : [0,1] \to \mathbb{R}^d$ with $n-1$ segments with control points $\mathbf{Q} = \{\mathbf{q}_1, \ldots, \mathbf{q}_{3n-2}\}$ and parameter locations t_j.

For each of these, let us define piecewise-constant indexing functions (see Fig. 2.5):

$$\mathbf{i} : [0,1] \to \{1, \ldots, n\}, \text{ where } \mathbf{i}(x) = i \text{ if } s_{i-1} < x \leq s_i, \text{ and} \tag{2.3}$$

$$\mathbf{j} : [0,1] \to \{1, \ldots, n-1\}, \text{ where } \mathbf{j}(x) = j \text{ if } t_{j-1} < x \leq t_j. \tag{2.4}$$

Without loss of generality, assume the parameter locations s_i and t_j are disjoint (outside of endpoints) so they cut up the domain [0,1] into $2n-2$ intervals. Since s_i and t_j may be in any order relative to each other, we introduce a joint sequence notation:

$$\{x_0, x_1, \ldots, x_{2n-2}\} = \{0, \mathrm{sort}(s_1, \ldots, s_{n-1}, t_1, \ldots, t_{n-2}), 1\}, \tag{2.5}$$

while this simplifies notation, it obscures that each parameter location x_k *depends* (when it comes to derivatives) on some s_i or t_j; this will be essential for correct gradient computation later.

Equipped with a way to refer to these intervals in order, we may define a distance between these chains as a sum over intervals:

$$\mathrm{E}(\mathbf{P}, \{s_i\}, \mathbf{Q}, \{t_j\}) = \sum_{k=1}^{2n-2} \omega_k \int_{x_{k-1}}^{x_k} \left\| \mathbf{A}^{\mathrm{i}(x)}\left(\frac{x - s_{\mathrm{i}(x)-1}}{s_{\mathrm{i}(x)} - s_{\mathrm{i}(x)-1}}\right) - \mathbf{B}^{\mathrm{j}(x)}\left(\frac{x - t_{\mathrm{j}(x)-1}}{t_{\mathrm{j}(x)} - t_{\mathrm{j}(x)-1}}\right) \right\|^2 dx, \tag{2.6}$$

where — despite the indexing hellscape — we observe that each term in the summation is an integral of the form in Eq. 2.1. The ω_k term *weighs* the impact of the kth integral on the total. One choice would be to use the arc-length of the corresponding pieces of the segments of $\mathcal{A}$ and $\mathcal{B}$. This will be problematic in our setting because only the longer curve $\mathcal{A}$ is known in advance and *approximating* arc-lengths of Bézier curves relies on computationally expensive operations. Instead, we propose to use $\omega_k = 1/(s_i - s_{i-1}) + 1/(t_j - t_{j-1})$, which can be seen as a first-order approximation of arc-length (i.e., assumes segments are straight) *and* discourages intervals from disappearing by inversely proportionally growing in scale. We draw special attention that ω_k therefore *depends* on some s_i and t_j values.

2.3.4 Segment removal operation

Analogous to an edge-collapse operation for triangle meshes, we now define a local segment remove operation for a chain of length n (see Fig. 2.6), defined by control points $\mathbf{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_{3n+1}\}$. For now, let us assume the input to this subroutine are n segments forming a chain within a possibly longer chain continuing at either end. The output are $n - 1$ segments

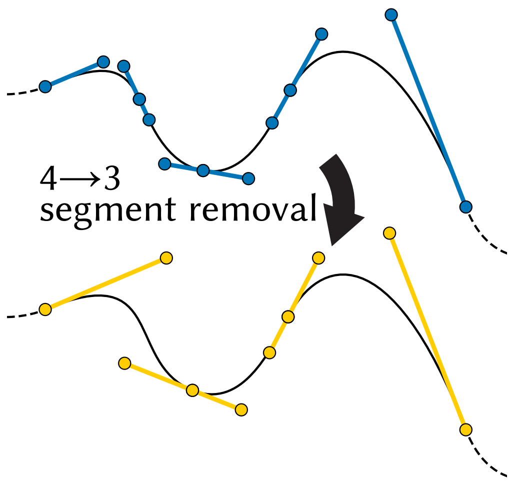


Figure 2.6: Local segment remove operation for a chain of length $n = 4$.

defined by control points $\mathbf{Q} = \{\mathbf{q}_1, \ldots, \mathbf{q}_{3n-2}\}$ and a measured *cost* of accepting this solution:

$$\text{cost} = \min_{\{s_i\},\mathbf{Q},\{t_j\}} E(\mathbf{P}, \{s_i\}, \mathbf{Q}, \{t_j\}). \tag{2.7}$$

2.3.4.1 General case ($n \geq 2$)

The cost in Eq. 2.7 is a minimization of a continuous function of the unknowns $\{s_i\}$, $\mathbf{Q}$, $\{t_j\}$. We will optimize this via a modified Gauss-Newton method.

By appending constraints to this opimization, we ensure C^0 and G^1 continuity at the endpoints (where the chain joins neighboring segments) and at internal interpolated control points in the output (for $n > 2$). Namely, we maintain C^0 continuity with:

$$\mathbf{q}_1 = \mathbf{p}_1 \quad \text{and} \quad \mathbf{q}_{3n-2} = \mathbf{p}_{3n+1} \tag{2.8}$$

and G^1 continuity with:

$$\exists\, \alpha > 0 \mid \mathbf{q}_2 - \mathbf{q}_1 = \alpha\,(\mathbf{p}_2 - \mathbf{p}_1)\,, \tag{2.9}$$

$$\exists\, \beta > 0 \mid \mathbf{q}_{3n-3} - \mathbf{q}_{3n-2} = \beta\,(\mathbf{p}_{3n} - \mathbf{p}_{3n+1})\,, \tag{2.10}$$

$$\exists\, \gamma_j > 0 \mid \mathbf{q}_{3j} - \mathbf{q}_{3j+1} = \gamma_j \left(\mathbf{q}_{3j+1} - \mathbf{q}_{3j+2}\right) \;\forall\, j = 1, \ldots, n-2. \tag{2.11}$$

The constraints in Eq. 2.8 can be substituted into Eqs. 2.9 & 2.10, revealing they are linear in $\alpha, \mathbf{q}_2$ and $\beta, \mathbf{q}_{3n-3}$, respectively. These are easily enforced via the null-space method: introducing α and β as variables.

That is, we build a matrix $\mathbf{N} \in \mathbb{R}^{d(3n-2)\times(3d(n-2)+2)}$ from relevant entries of $\mathbf{P}$ so that:

$$\mathbf{Q} = \mathbf{N}\,\mathbf{y} + \tilde{\mathbf{Q}} \text{ such that Eqs. 2.8-2.10 hold for any } \mathbf{y} \in \mathbb{R}^{3d(n-2)+2} \tag{2.12}$$

where $\mathbf{y}$ collects remaining free variables in $\mathbf{Q}$ and α, β.

Meanwhile, the constraints in Eq. 2.11 are *bi*-linear in the γ_is and $\mathbf{Q}$, respectively. Treating the s, γ, and t variables as *known*, then the cost function is a quadratic function in $\mathbf{Q}$. This implies it has a closed-form solution $\mathbf{Q}^\star$, which can be expressed as a non-linear function $f : \mathbb{R}^{3n-5} \to \mathbb{R}^{d(3n-2)}$ of γ, s, and t variables:

$$\mathbf{Q}^\star = f(\{s_i\}, \{\gamma_j\}, \{t_j\}). \tag{2.13}$$

We can now substitute $\mathbf{Q}^\star$ from Eq. 2.13 into the optimization problem in Eq. 2.7 to get a non-linear optimization problem over $\{s_i\}, \{\gamma_j\}, \{t_j\}$:

$$\min_{\{s_i\},\{\gamma_j\},\{t_j\}} E(\mathbf{P}, \{s_i\}, f(\{s_i\}, \{\gamma_j\}, \{t_j\}), \{t_j\}). \tag{2.14}$$

While we could apply simple gradient descent, we see significant improvements leveraging the sum-of-squares nature of the distance in Eq. 2.6 to use Gauss-Newton method.

Because the integrands in Eq. 2.1 are sixth-order polynomials over the integration variable (squared norm of a cubic polynomial), we may immediately apply sixth-order accurate quadrature rules to *exactly* evaluate these integrals for any input parameters.

$$\sum \int \|\ldots\|^2 \quad \xrightarrow{\text{exact quadrature}} \quad \sum \sum \|\ldots\|^2 = \sum \|\ldots\|^2$$

Our sum of integrals is now a simple sum of squared function evaluations: suitable for discrete Gauss-Newton method.

All that remains is to apply chain-rule and differentiate each term with respect to the free variables. Our implementation uses complex-step numerical differentiation [126] to easily (without hand-coding any derivatives), accurately (up to machine precision), and efficiently (with little more cost than a few forward evaluations) compute necessary Jacobian matrices. For small n (in our cases $n \leq 4$), this choice is appropriate. For much larger n (e.g., $n = O(N)$), sophisticated automatic-differentiation tools such as [2] may start to see performance improvements.

Given a Gauss-Newton search direction we conduct a back-tracking line-search [17] to find a suitable step length, while constraining α, β, γ parameters to stay positive. For our most common case of $n = 4$, we observe convergence typically in $\approx$10 iterations. For fixed n, this entire *local* optimization is $O(1)$ with respect to the complete input vector graphics image of N segments.

To initialize our optimization, we set $s_i = \sum_{k=1}^{i} \ell_k / \sum_{k=1}^{n} \ell_k$, where ℓ_i is the arc-length of the ith segment (approximated numerically). Then we pick the best $\{t_j\}$ initial values among all $n-1$ subsequences of $\{s_i\}$ and an additional uniform setting of $s_i = 1/(n-1)$. For small n, we observe that trying a few initial values often helps speed up convergence. Our specific heuristic ensures that we immediately identify input chains of $n-1$ straight segments where one segment can always be losslessly removed. Tangent length ratios α, β, γ are initialized to relative arc-lengths: again agreeing with constant speed straight segments.

2.3.4.2 Special lossless case $n = 2$

If $n = 2$ *and* we know that we are only interested in accepting the segment removal for numerically zero cost, then we can avoid the continuous optimization of the previous section and jump directly to the solution of the low-order polynomial root finding problem.

Key Observation: A lossless segment removal on a two-segment chain defined by control points $\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3, \mathbf{c}_4 \in \mathbb{R}^d$ and $\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3, \mathbf{d}_4 \in \mathbb{R}^d$ exists *if and only if* there exists a single curve with control points $\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3, \mathbf{q}_4 \in \mathbb{R}^d$ and a parameter value $t \in (0, 1)$ such that subdividing the $\mathbf{q}$ at t produces the segments $\mathbf{c}$ and $\mathbf{d}$.

The output segment $\mathbf{q}$ has constant third derivative, so to admit a lossless removal the inputs must have equal third derivatives that must be equal up to a scale factor. That scale factor can be written in terms of a parameter $t \in (0, 1)$ such that $\mathbf{c}$ and $\mathbf{d}$ segments are the result of splitting the output segment $\mathbf{q}$ at t:

$$-6\mathbf{q}_1 + 18\mathbf{q}_2 - 18\mathbf{q}_3 + 6\mathbf{q}_4 = \frac{-6\mathbf{c}_1 + 18\mathbf{c}_2 - 18\mathbf{c}_3 + 6\mathbf{c}_4}{t^3} = \frac{-6\mathbf{d}_1 + 18\mathbf{d}_2 - 18\mathbf{d}_3 + 6\mathbf{d}_4}{(1-t)^3}, \tag{2.15}$$

$$\frac{\dddot{\mathbf{c}}}{t^3} = \frac{\dddot{\mathbf{d}}}{(1-t)^3}, \tag{2.16}$$

where we further assume that third-derivatives $\dddot{\mathbf{c}}, \dddot{\mathbf{d}} \in \mathbb{R}^d$ are non-degenerate (if they both are, then the segments could be merged as quadratic or linear Bézier curves).

Now, we can solve this third-order polynomial equation for t:

$$\underbrace{\frac{\dddot{\mathbf{d}} \cdot \dddot{\mathbf{c}}}{\|\dddot{\mathbf{d}}\|^2}}_{r} (1-t)^3 = t^3 \tag{2.17}$$

$$-(1+r)t^3 + 3rt^2 - 3rt + r = 0. \tag{2.18}$$

For $r > 0$ (implying $\ddot{\mathbf{c}}$ and $\ddot{\mathbf{d}}$ point in the same direction), this equation has a single real root:

$$t = \left(1 + r^{-1/3}\right)^{-1}. \tag{2.19}$$

Finally, if this lossless removal is possible, then we can recover the corresponding control points from our continuity assumptions:

$$\mathbf{q}_1 = \mathbf{c}_1 \tag{2.20}$$

$$\mathbf{q}_4 = \mathbf{c}_4 \tag{2.21}$$

$$\mathbf{q}_2 - \mathbf{q}_1 = \frac{\mathbf{c}_2 - \mathbf{c}_1}{t} \tag{2.22}$$

$$\mathbf{q}_4 - \mathbf{q}_3 = \frac{\mathbf{d}_4 - \mathbf{d}_3}{1 - t}. \tag{2.23}$$

To determine if a lossless removal is actually possible, a practical approach is to compute t and $\mathbf{q}_i$ as above and then check if the integrated error between the original two segments and the new single segment (using Eq. 2.6) is (numerically close enough to) zero.

Assuming an infinite precision machine, a straightforward algorithm for lossless simplification of a sequence of n segments is to process all consecutive pairs of segments in a queue or stack. Upon each pop, if the pair is still valid and can be losslessly merged, then conduct it and insert any new consecutive pairs into the queue. The $2 \rightarrow 1$ merge operation is local and $O(1)$ cost, and the full algorithm is $O(n)$. The algorithm is also guaranteed to find all possible lossless mergers, resulting in the most compact lossless representation.

In practice, floating point operations introduce errors and these errors may accumulate. In the worst case, we could merge pairs *in order* along a chain of N segments all stemming from a single ($K = 1$) curve, accumulating error N times over before the final merge. Instead, we can process operations in a queue, which accumulates the worst case error at a rate of $\log N$.

Our cubic root finding relies on floating point operations which are not error-free. We observe that in the range $t \in [10^{-3}, 1 - 10^{-3}]$ the numeric loss for theoretically lossless merges is very low $<10^{-16}$. However, this numeric loss grows toward larger values $\approx 10^{-5}$ as the solution t gets closer to 0 or 1. We apply one iteration of the Gauss-Newton descent on E, initialized at our $\mathbf{c}$ and t values to "brush off" the last bit of numerical error.

2.3.5 Greedy Priority Queue Processing

Equipped with our local operations, we split the input into chains at corners, determined by a threshold on incident tangents. Every pair of consecutive segments is initially placed in a priority queue based on its lossless $2 \longrightarrow 1$ operation (see Section 2.3.4.2). These are processed — pushing neighboring lossless $2 \longrightarrow 1$ operations onto the queue any time an operation is accepted — until the queue is empty or only K segments remain.

If more than K segments remain, then for every consecutive four segments we push a (potentially lossy) $4 \longrightarrow 3$ operation on the queue. For chains of length $n < 4$, we also push an appropriate $n \longrightarrow (n-1)$ operation. These are processed — again pushing appropriately updated neighboring operations upon each acceptance — until K segments remain.

We use a priority queue with pop-min, update-key, and insert operations with cost $O(\log n)$, keyed on the cost of each possible $4 \longrightarrow 3$ or $2 \longrightarrow 1$ operation. Computing each cost is $O(1)$ and there are $O(n)$ operations: filling the queue has thus a $O(n \log n)$ cost. Finding the cheapest operation is $O(\log n)$, executing it $O(1)$, and updating the cost of the neighboring elements is $O(\log n)$ (note that the number of neighbors is constant): overall, it is $O(\log n)$ total work per pop. Since we have $K \leq n$ operations to pop, the final time complexity is $O(K \log n + n \log n) = O(n \log n)$.

Finally, our algorithm assumes all corners are preidentified. We can also simplify *across corners* (cost permitting) by relaxing the G^1 constraints in Eq. 2.11 based on a user-provided threshold. To support loops: during the local segment removal optimization for closed loop chains of length $\leq n$, G^1 continuity constraints are appropriately modified so that the choice of endpoints is unbiased.

2.4 Experiments and Discussion

We implemented our algorithm in Matlab, using gptoolbox [82] for vector graphic support and basic geometry processing. We implemented our Gauss-Newton solver with backtracking line search ($\alpha = 0.3, \beta = 0.5$) [18], with stopping criteria: relative energy $< 10^{-15}$, relative gradient $< 10^{-5}$, or max 30 iterations. Our algorithm and implementation is *by construction* $O(N \log N)$, but beyond ensuring correct asymptotic performance, we did not optimize our method and leave a high-performance implementation as incremental future work. We report the runtime of our implementation for the examples in the paper in Table 2.1, measured on an Intel i7 processor clocked at 2.6 GHz. Instead, we focus on the superior quality of our results in terms of error and the generality of our approach with respect to input. Fig. 2.7 contains a gallery and our supplemental data contains a more thorough `.html` explorer of results.

Table 2.1: Runtime performance across the examples in Chapter 2.

Example	*#segments before*	*#segments after*	*Runtime (s)*
Fig. 2.1	2,233	687	558
Fig. 2.2	304	19	1
Fig. 2.3	7,224	300	367
Fig. 2.11	2,280	400	432
Fig. 2.12	1,000	300	125
Fig. 2.13	1,130 × 100	(avg.) 89 × 100	1,349
Fig. 2.7 (1)	2,916	1,458	779
Fig. 2.7 (2)	5,001	1,500	1,595
Fig. 2.7 (3)	20,293	6,087	1,822
Fig. 2.7 (4)	16,628	6,651	2,094

Large-Scale Evaluation. We conducted a large-scale quantitative evaluation on 17,944 in-the-wild `.svg`s composed of 55 million segments in total from OpenClipArts dataset [77] (≈2k models were excluded because our `.svg` loader does not support esoteric path commands). Our loss-

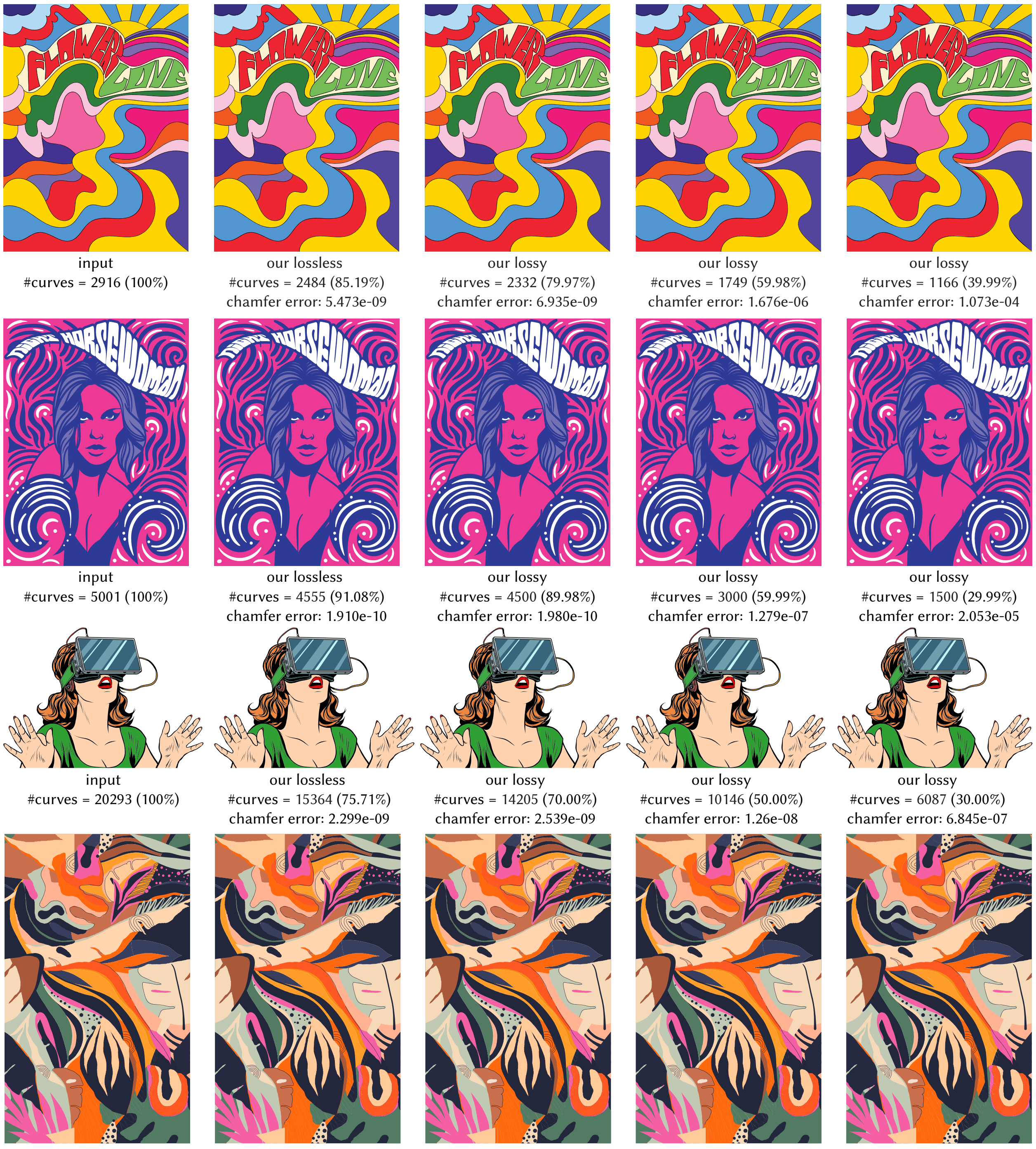


Figure 2.7: Our simplification results on stock images from the Internet. Our algorithm losslessly simplified the images to around 75-90% and managed to simplify the images further to 40% or less than 40% of its original number of segments without noticeable visual difference. The top-row input image © Aleksey. The second-row input image © Rudzhan. The third-row input image © studiostoks. The bottom-row input image © Irina.

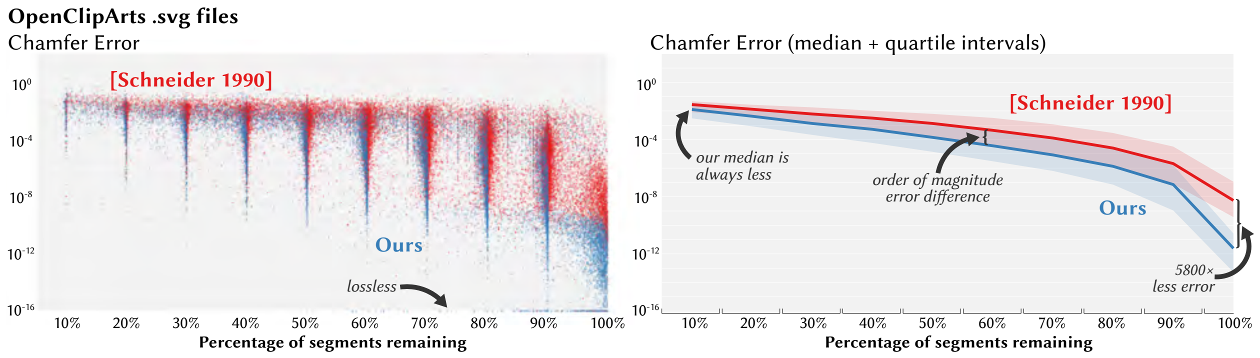


Figure 2.8: We conducted a large-scale benchmark of vector graphics simplification — as far as we know — the first of its kind. We compare our simplification method to [163] for a variety of target segment counts, recording the chamfer error to the original (left). Around 75% experience at least some lossless removals. On the right, we bin collected samples in deciles and report their medians and quartile intervals. Our method's median is always smaller though best improvements are observed for larger percentages. *For visualization clarity, we snap extremely small errors of our lossless results to axis bounds.*

less simplification benefited 74.2% of the models. This suggests that most vector graphics found in the wild could save on storage and bandwidth without any noticeable change: motivating our lossless contributions. Our method demonstrates favorable quality for the same simplification amount compared to [163] (Fig. 2.8). For each model, we consider a series of K values: we run our lossless process until completion $K = K_{\text{lossless}}$, then applying our lossy process to reach $K = 90\%, 80\%, \dots, 10\%$ of the original number of segments until no lossy removal is possible without exceeding a very large error value. We run [163] until achieving roughly the same K values. We measure bi-directional chamfer error between the input and output: densely, uniformly randomly sampling all curves of the input and taking the mean squared L^2 distance to the output curves, then *vice versa*, and averaging the two errors. Our method outperforms [163] across the entire range of the simplification amount: the median curve mostly staying around an order of magnitude lower, with a much large difference improvement in the 90-100% range.

We conducted a smaller quantitative comparison including Kurbo [112] on 5,000 especially numerically smooth chains of length $N \geq 20$ from randomly sampled models of the OpenClipArts dataset. We constructed this dataset out of fairness to Kurbo, as it is more sensitive to slight tangent mismatches at perceptually smooth control points than [163] and our algorithm. Kurbo

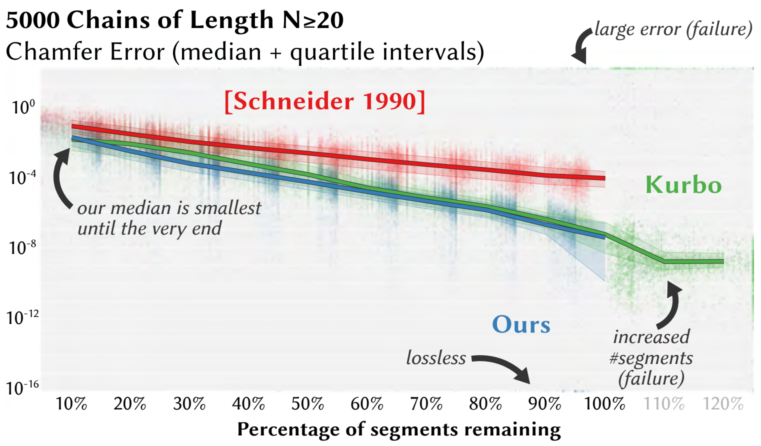


Figure 2.9: To include Kurbo [112] in our comparisons, we build a smaller dataset of long, smooth chains. When Kurbo works, it works reasonably well: its median is similar to ours. However, it often creates extreme error results or even adds more segments instead of removing them. *We snap extremely large/small errors and increased segment counts to axis bounds.*

does not provide a direct way of controlling the output number of segments, so we do a 13-value sweep over its simplifier's tolerance parameter. Fig. 2.9 visualizes the collected data and demonstrates our method again consistently outperforming [163]. We also observe that *when Kurbo succeeds* it works fairly well, but it often fails in one of three different ways: (1) getting no results after an extremely long time (>1 hour), (2) producing excessively high error results due to explosion (curves far outside the input's bounding box), and (3) outputting *more* segments than the input (see types (2) & (3) in Fig. 2.9). We ran Kurbo $65,000 = 5,000 \times 13$ times, of which Kurbo failed — in one of these three ways — 34.4% of the time.

Operation Type Ablation. We conducted an ablation experiment on the Night Horsewoman poster in Fig. 2.7. We compared performing only a single operation independently against our algorithm that combines them. We observe that $4 \longrightarrow 3$ only and $3 \longrightarrow 2$ only cannot simplify as much as our reference strategy, leading to higher errors. The $2 \longrightarrow 1$ only strategy is more flexible and can simplify as much as our reference but with higher errors. Overall, this experiment confirms that using all operations leads to superior simplification results for the same quality.

Editing Software. Our method qualitatively out-performs standard software: Adobe Illustrator

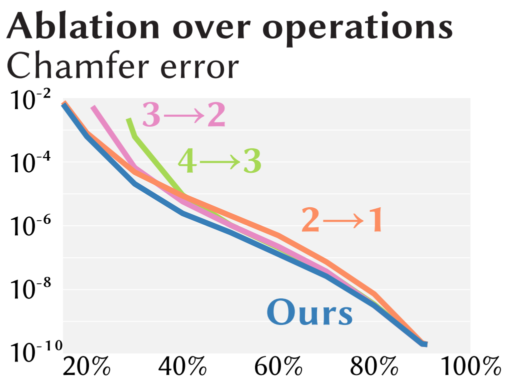


Figure 2.10: Ablation study on the Night Horsewoman poster in Fig. 2.7 by comparing only a single operation independently against our algorithm that combines them.

and Inkscape ([163]), as shown in Fig. 2.1 and Fig. 2.2. Additionally, it supports lossless simplification, which is not supported by these software. In Fig. 2.1, our lossless simplification down to 87.82% of the original segments. We further *lossily* simplify to 30.77% without perceptible visual change: at the same simplification level, Adobe Illustrator introduces noticeable visual differences over the entire image (especially on the mushroom) and Schneider [163] significantly alters the image. Kurbo [112] fails on this input.

Lossless simplification workflow. In Fig. 2.3, we emulate an artist's workflow with a black box pointwise brush tool. We densely upsample a graphic so the brush is applied directly to control points, then simplify to preserve only the necessary control points. The editing operation often leaves parts of the model with dense samples unmoved or moved in a locally affine way: this presents a significant opportunity for lossless simplification. We simplify down to 16.1% in this example. The artist can continue to repeat this workflow of upsampling, editing, lossless simplification for as many iterations as desired because no information is lost in the procedure, unlike in all previous methods. Once the artist is satisfied with the final design, the image can be processed by our lossy simplification into a much smaller file without a noticeable change to the design. This opens the door to a workflow similar to traditional photo editing, where lossless compression formats (`.png`) are used for editing, and lossy formats (`.jpg`) are used for

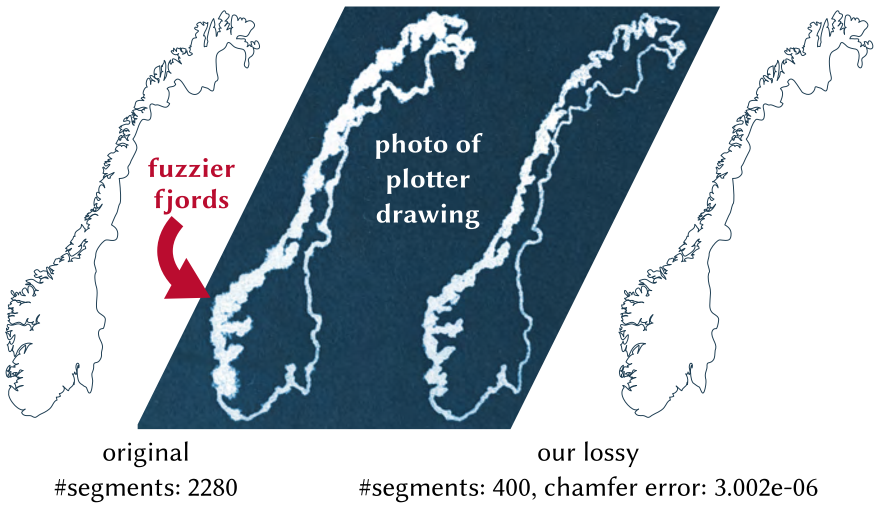


Figure 2.11: Attempting to draw a dense spline with a robotic plotting machine results more ink bleeding than plotting its simplification.

distribution.

Manufacturing. Our simplification can benefit manufacturing methods (plotting, laser cutting) that take vector paths as input. We drew the boundary of Norway using a robotic plotting machine (Fig. 2.11) and observe that the vector path simplified by our algorithm has less ink bleeding compared to the original one.

Higher dimensions. Simplification is a core subroutine for not only 2D vector images but also vector data of higher dimensions. We demonstrate that our method can be easily extended to higher-dimensional vector data, including vector images with stroke width and vector animation. The "John Hancock" signature in Fig. 2.12 consists of cubic Bézier curves whose control points contain an extra stroke width dimension. Our method yields a more compact image without losing noticeable accuracy. In Fig. 2.13, we densely upsampled the input `.svg` and applied an embedded mesh animation based on a full-space complementary dynamics [217] and barycentric coordinates interpolation: this procedure is necessary to faithfully capture the deformation but introduces many redundant control points. If the animation is applied directly on the `.svg` without upsampling, the final animation is noticeably defective: e.g., the dolphin's blowhole is in-

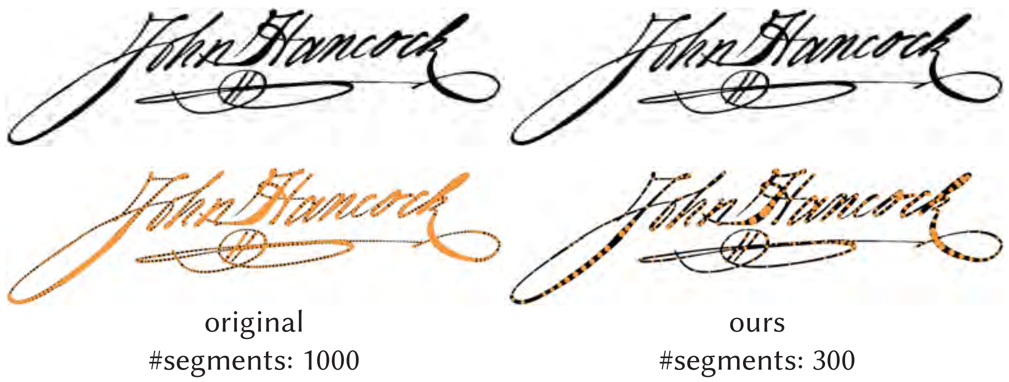


Figure 2.12: We are able to simplify a "3D" curve by treating the stroke width as an extra dimension. The bottom row visualizes stroke radii (3rd dimension) as an orange circle at each interpolated control point. We show that the hand-written "John Hancock" with 1000 segments can be represented by 300 curves without losing noticeable accuracy. Original input image sourced from John Hancock's signature (public domain).

correctly appears outside of the poorly animated silhouette. With our method, we can efficiently simplify the upsampled sequence with temporal coherency by (1) applying truncated principal component analysis (PCA) on the high dimensional animation data similar to [3] and then conducting lossy simplification on the PCA vectors and then (2) applying lossless simplification on each frame individually as a post process to remove redundant control points. Our simplification result is qualitatively better than a baseline of simplifying every frame independently, which leads to undesired temporal popping artifacts.

2.5 Conclusions and Limitations

We focused exclusively on asymptotic performance and implemented our prototype in Matlab where scripting overhead is high. We expect that a high-performance implementation in C++ is a straightforward extension, there are also advances in parallelism for surface mesh simplification in parallel which could be adapted to our analogous setting [91].

Our method does *not* conduct topological simplification; most notably, we simplify but never remove an entire input component. Thus, a valid input K should be greater than or equal to

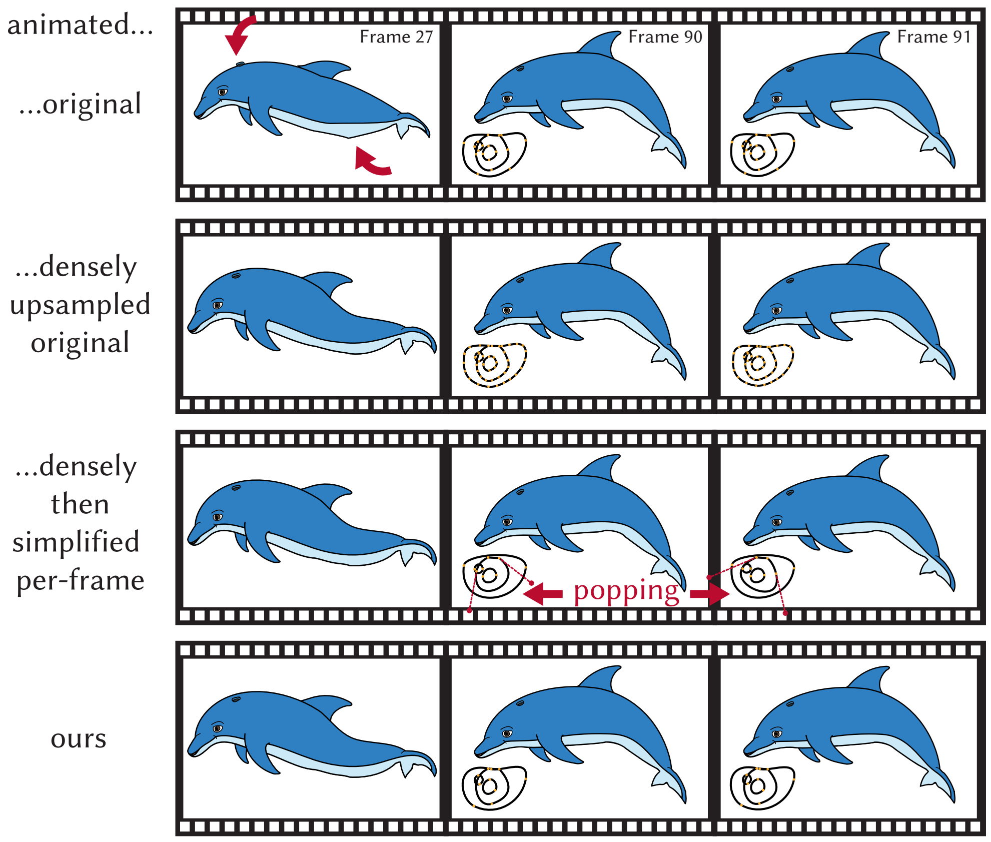

animated...
...original
...densely upsampled original
...densely then simplified per-frame
ours
Frame 27
Frame 90
Frame 91
popping

Figure 2.13: Naively applying animations to vector graphics either ends up with artifacts or too many segments. Simplifying per-frame produces popping artifacts, while our method is temporally coherent.

the number of input components. We conjecture that topological simplification requires more advanced perceptual metrics and leave for a future work which may benefit from our method as a subroutine. We also do not consider overdraw (we don't flatten or remove hidden paths). More perceptually accurate corner detection and persistence would aid all smooth simplification methods, including ours. Finally, we hope our emphasis on statistically meaningful large-scale testing inspires chasing down the remaining gains in vector graphics simplification.

3 Solid-Shell Labeling for Discrete Surfaces

This chapter is adapted from the submission [197] to SIGGRAPH Asia 2025, currently conditionally accepted, a joint work with Janos Meny, Izak Grguric, Mehdi Rahimzadeh, Denis Zorin, Daniele Panozzo and Hsueh-Ti Derek Liu.

ABSTRACT

Artist-created meshes in-the-wild often do not have a well defined interior. We observe that they typically consist of a mix of *solid* elements, faces that bound a volume, and *shell* elements that represent the medial surface of a thin shell. The lack of a well-defined interior prevents downstream applications, such as solid-modeling, simulation, and manufacturing. We present a method that takes as input a surface mesh and assigns to each face a label determining whether it belongs to a solid or shell. These labels reduce ambiguity by defining the interior for solid faces through thresholding the generalized winding number field, and for shell faces as the volume within an offset. We cast the labeling problem as an optimization that outputs a solid/shell label for each face, guided by a sparse set of user inputs. Once labeling is complete, we show how the shape can be volume meshed by passing the shell faces through an offset mesher and the solid faces to an off-the-shelf tetrahedral mesher, producing a final volumetric mesh by taking their

union. Experiments on diverse meshes with defects and multiple solid and shell components demonstrate that our approach delivers the desired labels, enabling modeling and simulation on wild meshes in a way that respects the user intent.

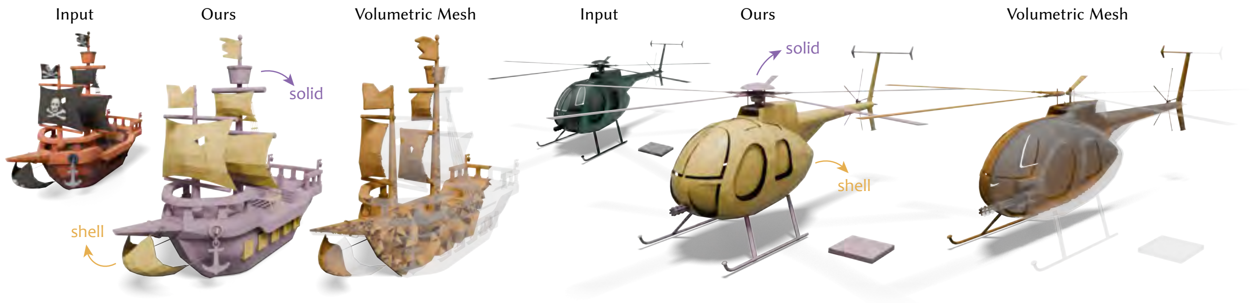


Figure 3.1: Given an input shape, our method labels each triangle as either the boundary of a *solid* (purple) or the medial surface of a *shell* (yellow), driven by sparse user guidance. These facet labels can be passed to off-the-shelf tetrahedral (for solid faces) and offset (for shell faces) meshers to produce volumetric meshes (orange) that are ready for downstream applications.

3.1 INTRODUCTION

Generating volumetric meshes from a given shape is a fundamental subroutine within geometric computation and modeling. Accordingly, extensive research has addressed key subproblems, such as tetrahedralizing a bounded volume and determining inside/outside relative to

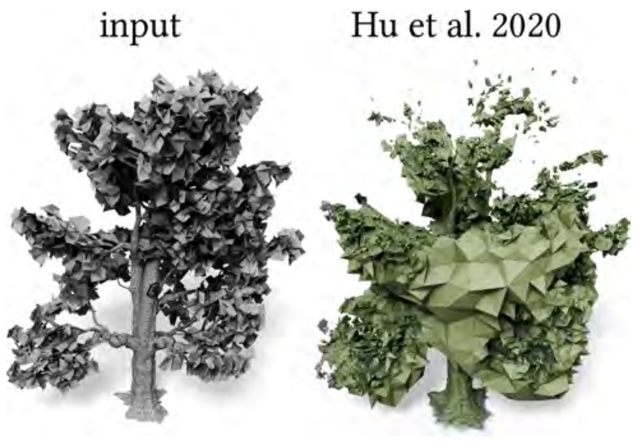


Figure 3.2: Unexpected results of exisiting volumetric meshing algorithms [75].

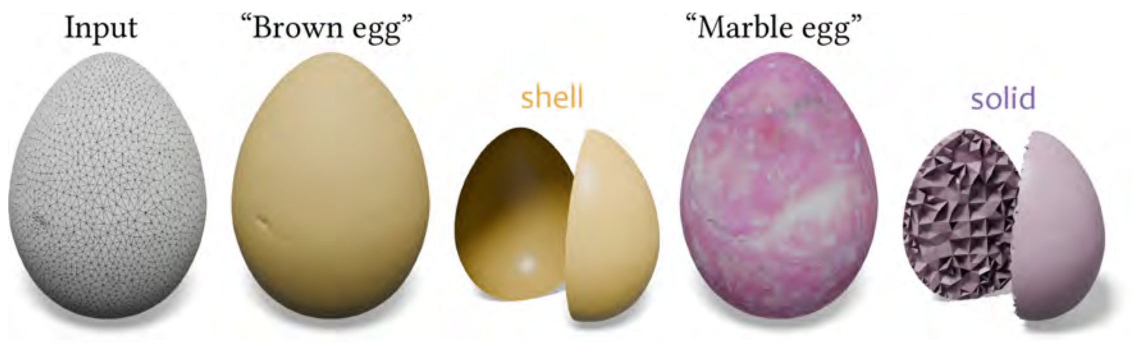


Figure 3.3: Given a triangle mesh (left), whether a shape is a shell or solid may require understanding the semantics, such as an eggshell (middle) or a marble (right). Our method incorporates user guidance and is able to produce user-desired face labels (yellow shell and purple solid) from the same input geometry.

a triangle mesh. Decades of research have yielded robust methods that reliably mesh volumetric domains.

Despite these advancements, existing volumetric meshing algorithms can sometimes produce unexpected results (see Fig. 3.2), especially for artist-created assets in online repositories. These failures stem from the assumption that the input triangle mesh "approximately" describes the boundary of a well-defined solid volume. In practice, however, this assumption does not always hold. Whether a shape is a volumetric thin *shell* or a *solid* depends on the semantics (see Fig. 3.3), instead of purely relying on geometric information. Moreover, many shapes in the wild consist of a mixture of solid and shell components (see Fig. 3.1), posing further challenges to meshing.

This ambiguity between shell and solid structures results in some heuristic strategies in practice, such as assuming everything is solid (or shell) or attempting to infer solids/shells based on visibility. However, these heuristics frequently prove inadequate (see Fig. 3.4), leading to meticulous, time-consuming manual intervention to correct inaccuracies at the individual polygon level. Obtaining user-desired face labels is critical to volumetrically meshing the object: tetrahedral meshing for solid faces and offset meshing for shell faces. Incorrect labeling can cause artifacts in the resulting volume mesh such as distorted appearance (Fig. 3.15), deletion of invisible structures (Fig. 3.4), and unexpected physical behavior (Fig. 3.12).

In lieu of this, we present a method for classifying each face of a triangle mesh as belong-

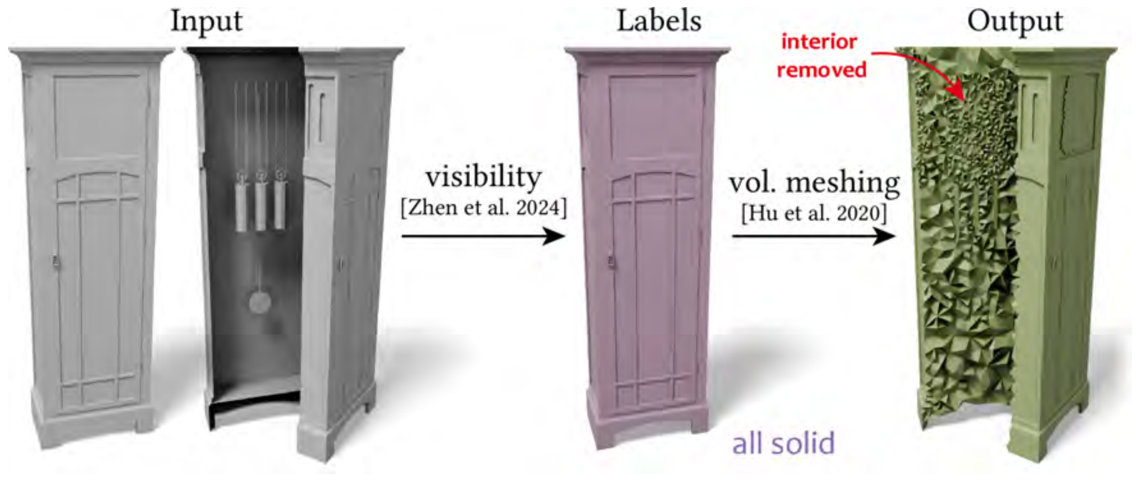


Figure 3.4: Existing heuristics, such as using triangle visibility [221], to classify solid/shell faces are prone to artifacts, such as deleting interior components.

ing to either a *solid* or a *shell* component, requiring only sparse user guidance. We formulate this labeling problem as an energy optimization, wherein the label of each face serves as the optimization variable, and the user-provided guidance is incorporated into the energy functional. After minimization, we take the resulting face labels to construct a volumetric mesh. Faces labeled as "solid" are processed through tetrahedral meshing tools to obtain a volumetric mesh. Faces that are identified as "shell" are passed through offset surfacing techniques, yielding a volumetric thin shell (see Fig. 3.5). After these two streams of volumetric meshing, we union the results and obtain a single mesh which is amenable to downstream applications, such as the Constructive Solid Geometry (CSG) modeling or physically-based simulation.

3.2 Related Work

Our method classifies each face in a mesh as either solid or shell to support volumetric meshing. We provide a brief overview of volumetric meshing and related topics that require determining the interior of an object.

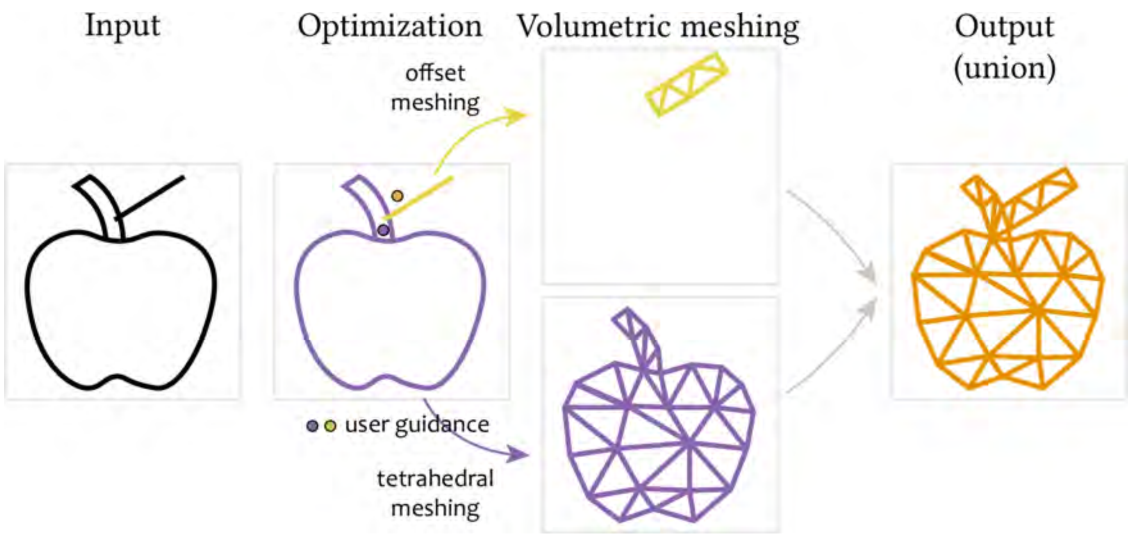


Figure 3.5: Given an input mesh (left), our method optimizes the label for each face to be either a solid (purple) or a shell (yellow) given sparse user guidance (purple/yellow dots). After the optimization, shell faces are passed through an offset mesher (top), solid faces are passed through a tetrahedral mesher (bottom). Then these two volumetric meshes are unioned to produce the final volumetric mesh (orange).

3.2.1 Volumetric Meshing and Offset Construction

One can compute a volumetric mesh from a surface mesh with either *volumetric* (e.g., tetrahedral meshing) or *offseting* meshing algorithms, which operate under opposing implicit assumptions. Solid-oriented volume meshers (e.g., [5, 35, 36, 75, 76, 174]) assume the input triangle mesh is an approximation of a *closed solid* and produce a volume mesh whose boundary faces approximate the input. This "closed-solid" assumption leads these methods to automatically close gaps and remove interior details, but this behavior can conflict with artistic intent (see Fig. 3.14). In contrast, shell-oriented meshers, such as [21, 22, 224], treat each triangle of a mesh as a thin shell element. These methods construct a volumetric mesh by offsetting the surface in both directions and tetrahedralizing the thin volume enclosed within the resulting shell. While offset surfacing methods preserve the visual appearance and maintain all interior components, they tend to produce unrealistic physical behaviors for objects that should behave as solid bodies. Due to conflicting assumptions between solid- and shell-oriented meshers, existing techniques are only applicable to *homogeneous* inputs where all faces in the mesh are consistently either solid or shell. In practice, however, artist-created meshes frequently contain mixed elements that violate these assumptions (see Fig. 3.1), causing both approaches to produce undesirable results. To address this limitation, our method automatically generates per-face labels that reconcile these two perspectives. Faces classified as solid are processed by a solid-oriented tetrahedralizer, while faces labeled as shell are handled by an offsetter that generates a thin volumetric mesh with appropriate physical thickness.

3.2.2 Solid/Shell Heuristics in Mesh Repairing

Mesh repairing aims at converting defective triangle soups into watertight manifolds by, for instance, filling gaps [87, 118] and removing self-intersections [6, 20, 63, 222]. Comprehensive surveys are provided by Attene, Campen, and Kobbelt [7] and Ju [86]. The question of whether a triangle belongs to a shell or solid often come up as a subproblem in a repairing pipeline. For instance, wrapping-based methods [27, 149] often impose the assumption that all faces should be labeled as solid, similar to tetrahedral meshing techniques, but similarly suffer from the problem of removing internal structures. The method by Huang, Zhou, and Guibas [78] indirectly determines face labels by checking whether each element connects to infinity through a voxelized representation of the shape. The closest method to ours is the method proposed by Zheng et al. [221]. Their pipeline uses ray casting to estimate the visibility of each face and converts this visibility information into an openness score. Faces judged open are offset, all geometry is inserted into a BSP tree, and a global graph cut is used to extract a watertight surface. Because the test hinges on visibility, shell structures hidden behind other geometry can be misclassified (see Fig. 3.4 and Fig. 3.14).

Our approach differs in two key ways. First, we rely on the visibility-independent generalized winding number (GWN), eliminating occlusion-related errors. Second, solid–shell segmentation can depend on semantics or be genuinely ambiguous, so a fully automatic method may choose an unintended interpretation with no easy recourse for the user. We let users add “inside” hints, treating them as soft constraints when optimizing the face labels and providing an intuitive mechanism for correcting ambiguous regions.

3.2.3 Winding Numbers for Inside/Outside Reasoning

The generalized winding number (GWN) introduced by Jacobson, Kavan, and Sorkine-Hornung [83] determines whether a point lies inside or outside an arbitrary triangle soup. Most applications assume the entire mesh encloses a uniform solid region, using the GWN for isosurface extraction via marching cubes [123] or dual contouring [88], or as the basis for “fuzzy” CSG operations [10]. We discard this uniform-solid assumption: our method removes faces not intended to enclose volume, yielding a cleaner set of solid faces whose induced GWN field is better suited to downstream contouring, CSG, and simulation. Fast-evaluation schemes for the GWN, such as that of Barill et al. [10], can be incorporated into our method for improved speed.

The winding number has also been employed in applications outside of meshing, such as coloring vector sketches [166], reconstruction [26] and has inspired work like computing generalized signed distances [44]. Another line of work [119, 131, 208] has used the GWN to orient pointclouds. Our method is similar in spirit, but has a different use case. We optimize a scalar variable attached to each face of a triangle mesh, determining how that face contributes to the GWN field. This optimization process results in a set of solid faces for which the GWN field more closely approximates that of an ideal closed solid object, so downstream volumetric meshing with homogeneous solid assumptions are more well behaved.

3.3 Method

The input to our method is a triangle mesh $\mathcal{M} = (\mathbf{V}, \mathbf{F})$, comprising a set of vertex positions $\mathbf{V}$ and a set of faces $\mathbf{F}$. Our method imposes no constraints on the input topology and is capable of handling non-manifold elements or even a soup of triangles. We do, however, assume that each face has the orientation intended by the artist. The output of our method is a set of (soft) labels $\boldsymbol{\ell} = \{\ell_f\}$ where each label $\ell_f \in [0, 1]$ indicates whether the face f is a *shell* (0) or a *solid* (1) component. These soft labels can then be thresholded to achieve the final binary labels.

We cast the labeling problem as an energy optimization, where the labels $\boldsymbol{\ell}$ are the optimization variable, and the energy can optionally incorporate user-controllable terms. Without user guidance, our method can serve as an automatic tool to provide solid/shell labels. But, in general, the label for a face can be ambiguous and can depend on the semantics of the object (see Fig. 3.3). We thus provide additional user-controllable terms to manipulate the outcome of the optimization.

3.3.1 Energy

We obtain face labels by solving an unconstrained optimization problem

$$\boldsymbol{\ell} = \operatorname*{arg\,min}_{\boldsymbol{\ell}} \; E_d(\boldsymbol{\ell}) + \alpha E_u(\boldsymbol{\ell}) + \beta E_s(\boldsymbol{\ell}), \tag{3.1}$$

where $\alpha, \beta \in \mathbb{R}^+$ are positive scalars to control the balance between these terms. E_d encourage the resulting face labels to induce a well-separated interior/exterior regions, E_u incorporates user controls into the optimization, and E_s encourage smoothness across the labels $\boldsymbol{\ell}$ within the same surface patch.

Double-Well Term. We observe that artist-created shapes often do not contain too many "irrelevant" components, meaning that most triangles in a mesh are created for a purpose. With this observation in mind, we design an energy which, by default, preserves nested interior structures (e.g., furniture inside a house, see Fig. 3.6). We draw inspiration from the globally injective parameterization [39] which detects nested structures using the *winding number* (see Fig. 3.7). Specifically, the winding number field of a set of closed, non-self-intersecting curves will be piece-

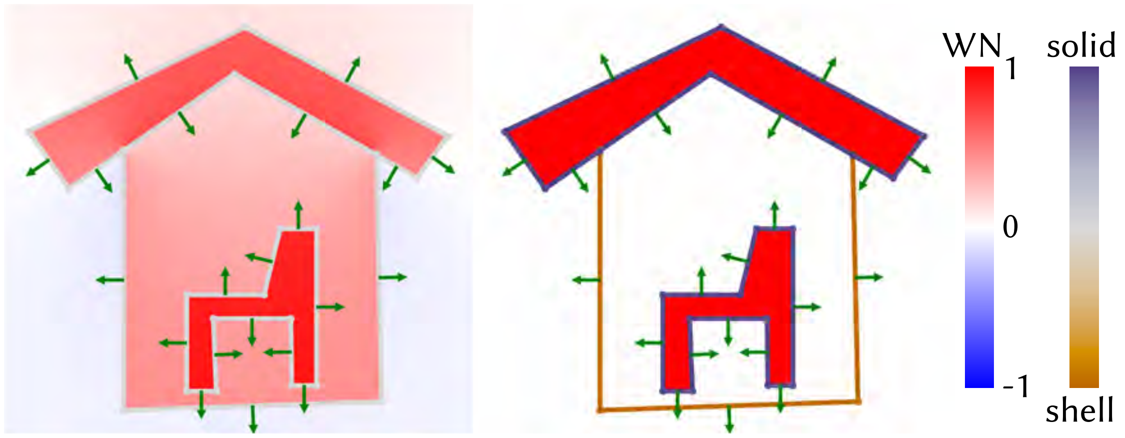


Figure 3.6: Generalized winding number field of a collection of segments before (left) and after (right) the optimization. Segments are colored by the tendency of being classified as solid or shell.

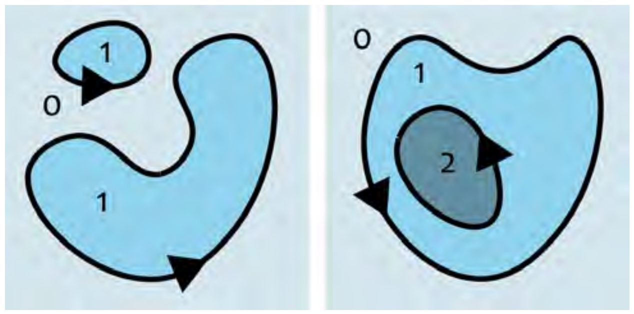


Figure 3.7: Winding number illustration for non-nested (left) and nested (right) structures.

wise constant 0/1 if the curves neither overlap nor have inverted orientations. This motivates our design of a double-well energy E_d which encourages the winding number field induced by the labeling to be either 0 or 1

$$E_d(\boldsymbol{\ell}) = \frac{1}{|\mathcal{P}|} \sum_{\mathbf{p} \in \mathcal{P}} \left(w(\mathbf{p}, \boldsymbol{\ell})(1 - w(\mathbf{p}, \boldsymbol{\ell})) \right)^2 \tag{3.2}$$

with

$$w(\mathbf{p}, \boldsymbol{\ell}) = \frac{1}{4\pi} \sum_{f} \Omega_f(\mathbf{p}) \ell_f, \tag{3.3}$$

where $\mathcal{P}$ denotes the set of sampled points we use to evaluate the energy, $|\mathcal{P}|$ denotes the number of samples, Ω_f is the solid angle of triangle f with respect to point $\mathbf{p}$, and ℓ_f is the face label for face f bounded between $0 \leq \ell_f \leq 1$. Intuitively, one can think of $w(\mathbf{p})$ as a *weighted* generalized winding number [83] with options to ignore shell faces ($\ell_f = 0$). The double well energy E_d has two global minima when $w(\mathbf{p}) = 0$ or 1, promoting a binary winding number field as the result of minimization.

User-Control Term. However, this energy formulation is motivated by the idealized scenario in [39] designed to promote global injectivity. When applied to 3D triangle soups that may contain various defects and non-manifold elements, minimizing E_d alone does not always produce the results users expect. Moreover, geometric cues by themselves (i.e., E_d alone) are insufficient to achieve the desired solid/shell classifications (see Fig. 3.3). To incorporate user preference in the labeling process, we introduce another energy term which allows users to specify the target winding number $t_\mathbf{q} \in \{0, 1\}$ for a sparse set of spatial locations $\mathbf{q} \in C$, indicating whether the point $\mathbf{q}$ should lie in the interior or exterior of the model. Specifically, we define the user control energy E_u as

$$E_u(\boldsymbol{\ell}) = \frac{1}{|C|} \sum_{\mathbf{q} \in C} \|w(\mathbf{q}, \boldsymbol{\ell}) - t_\mathbf{q}\|^\gamma, \tag{3.4}$$

where $|C|$ denotes the number of user-specified points and γ is a positive even integer, controlling how sensitive this term is to small deviations from the target winding number $t_\mathbf{q}$. The higher the power, the more tolerant this term becomes to small errors. Empirically, we set the default $\gamma = 4$ which gives us the easiest way to control the output labels (see Sec. 3.4.3).

Smoothness Term. As the control of E_u is local, using only E_u often requires more computation time to propagate label adjustments or specifying more constrained locations C. We therefore add a smoothness energy E_s on face labels $\boldsymbol{\ell}$ to encourage connected pieces to share the same label.

$$E_s(\boldsymbol{\ell}) = \frac{1}{A} \sum_{i,j \in \mathcal{N}_F} a_{ij} \|\ell_i - \ell_j\|^2, \tag{3.5}$$

where we use $\mathcal{N}_F$ to indicate neighboring faces which share an edge in-between, a_{ij} denote the barycentric edge area, and $A = \sum_{ij} a_{ij}$ is the sum of the areas. The combination and three terms forms our energy (see Eq. (3.1)). We set $\alpha = 10^3, \beta = 10$ as the default parameter throughout the experiments.

3.3.2 Implementation

We implement our method in Python, utilize PyTorch library [142] for auto-differentiation, and optimize our energy with the Adam optimizer [96]. We evaluate the double-well loss E_d on points $\mathcal{P}$ sampled near the surface. Specifically, we uniformly sample points (by default 64^3) that have distance less than ϵ away from the mesh, where ϵ is set to be 10% of the longest edge of the mesh bounding box.

As the winding number computation $\Omega_f(\mathbf{p})$ for $\mathbf{p} \in \mathcal{P}$ are constant throughout the optimization, we cache the result of $\{\Omega_f(\mathbf{p})\}$ as a single matrix to accelerate the optimization, leading to less than 1 millisecond runtime per optimization step with GPU acceleration on an Apple M4 Pro chip. The models presented in this paper converge in 5 to 30 seconds, with convergence determined by the gradient norm.

We notice that some meshes have big triangles slicing through multiple solid/shell regions (like a ground plane of a house scene). This issue prevents users from getting desired volumetric meshes because neither assigning solid nor shell to the problematic face yields a user-desired result. We thus apply a mesh arrangement [20, 63] algorithm as a preprocessing step to partition intersecting triangles into smaller fragments before running our algorithm.

We visualize the result of our method with Polyscope [171]. We allow users to add constraint points in Eq. (3.4) by clicking points in our preliminary interface and then specify whether this point belongs to an interior region or not. Due to the efficiency of our method, one can interactively see the result updated during optimization (please see the supplementary video for a demo usage of our implementation).

After obtaining the face labels $\boldsymbol{\ell}$, for the solid part, we extracted a closed surface mesh from the tetrahedral mesh computed with TetWild [76]. For the shell part, we extrude shells with a user-specified thickness τ using a robust topological offset algorithm [224], where τ is by default 1% of the longest edge of the mesh bounding box. Next, we take the union of the two closed surfaces with a robust mesh boolean code [28]. Finally, we use TetWild [76] to obtain our output tetrahedral mesh, that is ready for solid modeling and simulation applications as shown in Sec. 5.8. Moreover, one could apply the method by Attene et al. [8] to extract a closed manifold triangle mesh from our output to support surface computations.

3.4 Results

Our method outputs a set of face labels indicating solid and shell faces. We show that these labels can benefit off-the-shelf algorithms for constructing volumetric meshes, with unambiguous definitions of inside and outside based on users' specifications. With such user-defined interior/exterior definitions, one can ensure that the simulation behavior aligns with the semantic properties of an object (see Fig. 3.12) and can perform boolean operations without the need of relying on extra tools for "guessing" where the interior region is located (see Fig. 3.8).

Our method enables user control through point-based constraints that specify whether locations should be inside or outside the object. This approach can obtain a wide variety of user-desired segmentations by placing constraints appropriately. When all specified points are constrained to be outside, our energy function has a unique global minimum that labels every face as shell (provided there is at least one outside constraint). For instance, in Fig. 3.9, our optimization

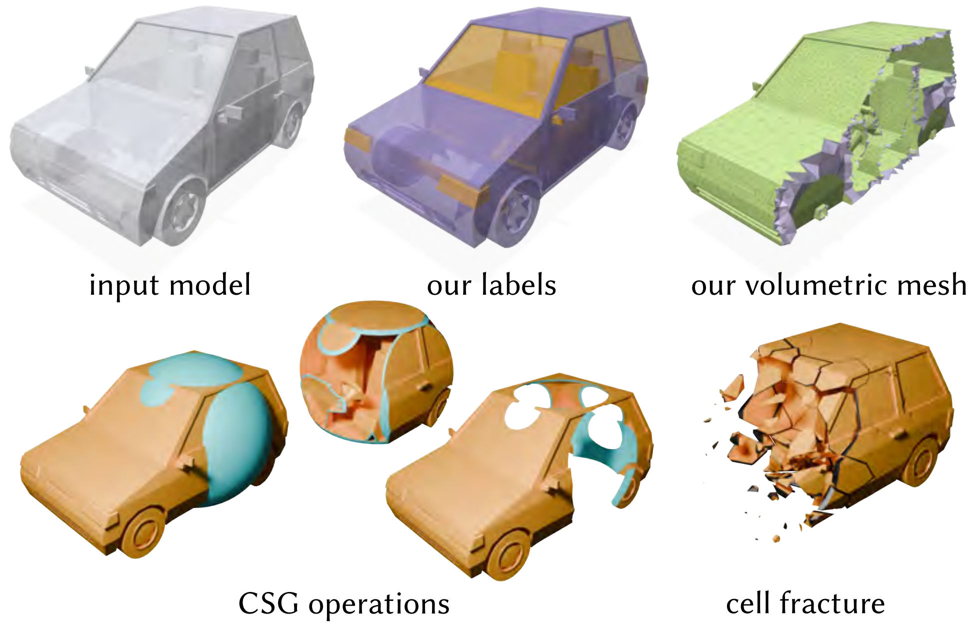


Figure 3.8: The face labels optimized by our method can assist volumetric meshing (green). This mesh is ready for boolean operations (e.g., intersection, union, subtraction) with the method by [28], and volumetric simulation such as the fractures.

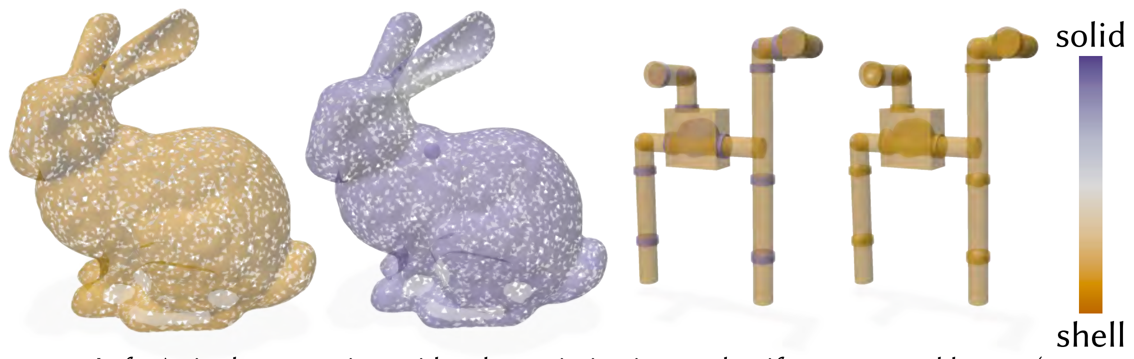


Figure 3.9: Left: A single constraint guides the optimization to classify a corrupted bunny (10 percent of faces removed) as fully solid. Right: A pipe modeled as a union of cylinders is classified as pure shell after placing some constraints.

produces an all-shell result for a watertight mesh representing pipes. When a constraint specifies that a point lies inside the object, the optimization encourage face labels to be solid when doing so helps achieve a winding number of 1 at that point. Fig. 3.9 shows a watertight mesh with a randomly deleted subset of faces, where a single inside constraint produces an all-solid segmentation. While classification is straightforward when meshes contain only solid or shell faces, in the wild meshes often combine faces that bound solids with others that represent thin shell surfaces. Our method provides flexibility to achieve diverse segmentations on wild meshes. Fig. 3.10 shows how adding constraints incrementally modifies face labeling until the desired result is achieved (bracket and light source as solid, light bulb as shell). These interactions allows one to place different constraint configurations to achieve different solid/shell segmentations (Fig. 3.11). Fig. 3.12 demonstrates the importance of user interaction in the labeling process. Given a closed manifold representing a coke can — which most heuristics would classify as solid (e.g., [221]) — our method allows users to specify a thin shell result, producing simulation behavior that better matches the expected physical properties of an actual coke can.

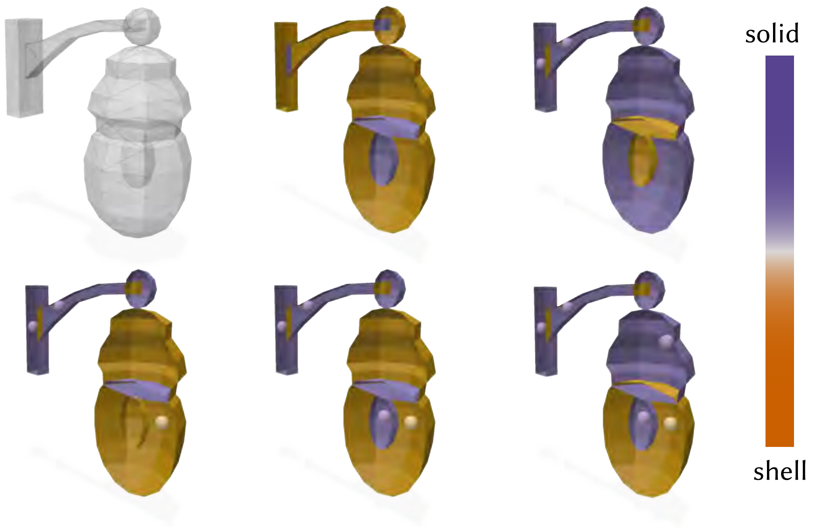


Figure 3.10: Incrementally adding point constraints can alter whether faces are labeled as solid or shell. Purple spheres denote interior constraints—points designated to lie within the solid—whereas yellow spheres denote exterior constraints—points designated to lie outside it.

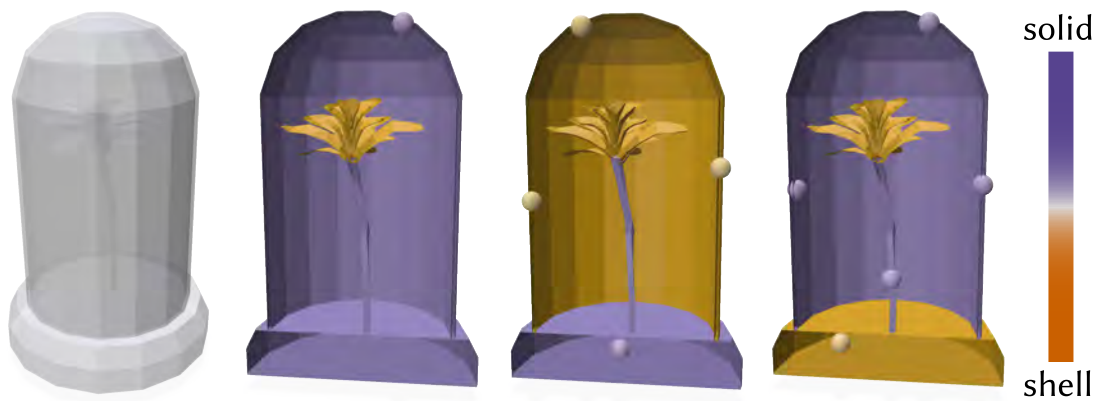


Figure 3.11: Our method allows one to control face labeling results by setting different configurations of the guidance point (colored spheres).

3.4.1 Comparison to Explicit Labeling

To evaluate our method, we build a preliminary interface that enables users to provide hints by specifying whether selected points should be classified as *inside* or *outside* the object. Table 3.1 quantifies the user effort required to obtain manually-created target labels for a set of five benchmark meshes (see Fig. 3.13) and compares our method against two baselines. The first baseline represents individual face labeling, where each face is selected separately. The second baseline approximates practical selection mechanisms such as face component selection or flood filling by counting the number of face components that contain at least one shell/solid face. For the second baseline, we pre-process the mesh by merging duplicate vertices to reduce the number of face components, because the meshes have too many face components to make selection mechanisms that rely on mesh connectivity practical. Note that this number represents a lower bound since connected face components do not necessarily correspond to actual shell/solid regions. These comparisons consistently show that our tools requires orders of magnitude less effort, in terms of the number of constraints needed, to achieve manually created target labels.

We further note that our labeling paradigm depends only on whether a point lies inside or outside the object and is independent of the object's discretization, requiring only that the discretization supports the definition of a GWN field. Users therefore need to understand only the

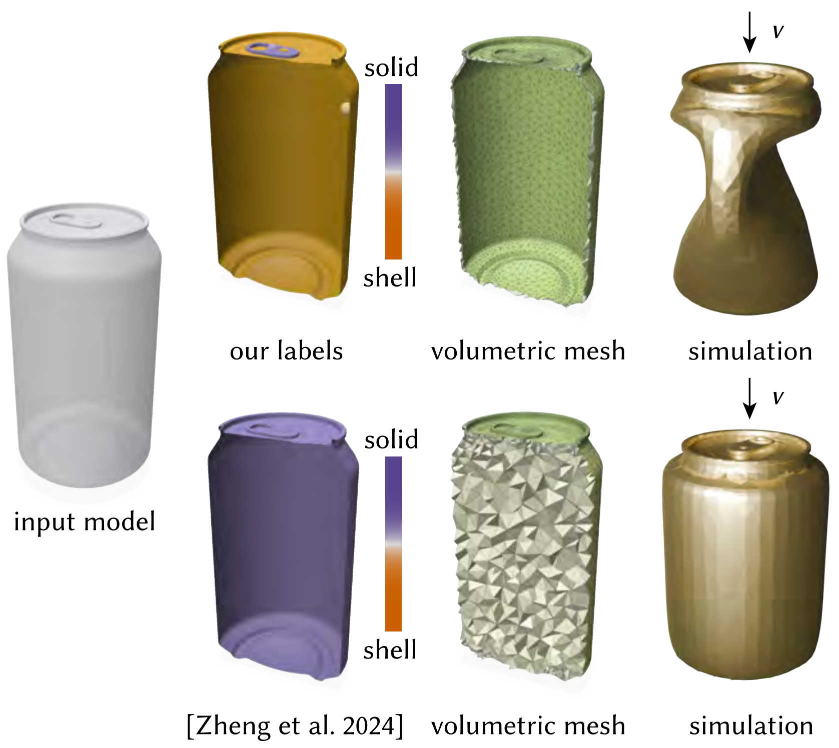


Figure 3.12: We present solid/shell labels (second column) and the corresponding volumetric mesh (third column) produced by our method (top) with two user-added constraints and the (all-solid) labeling from [221] (bottom). The last column shows a non-inverting, neo-Hookean elasticity simulation with contact [114] imposed on the two volumetric meshes, where we fix the bottom 5% and move the top 5% of the can downwards to compress the can. Making the can filled with solid yields unexpected simulation behavior (bottom right), contrasting our method which successfully produce the compression bending behavior (top right).

inside or outside concept, not details about the geometry representation. By contrast, both baseline methods require awareness of the triangle mesh, and the component-selection approach is especially fragile: it often requires an additional preprocessing step to merge vertices before it is practical.

3.4.2 Comparison to Prior Arts

Labeling all the faces as solid or shell does not always create a physically meaningful volumetric mesh. In Fig. 3.15 (left), we demonstrate a naive uniform solid labeling of the tree leads to artifacts when reconstructing a volumetric mesh based on the method by [75]. Labeling based on

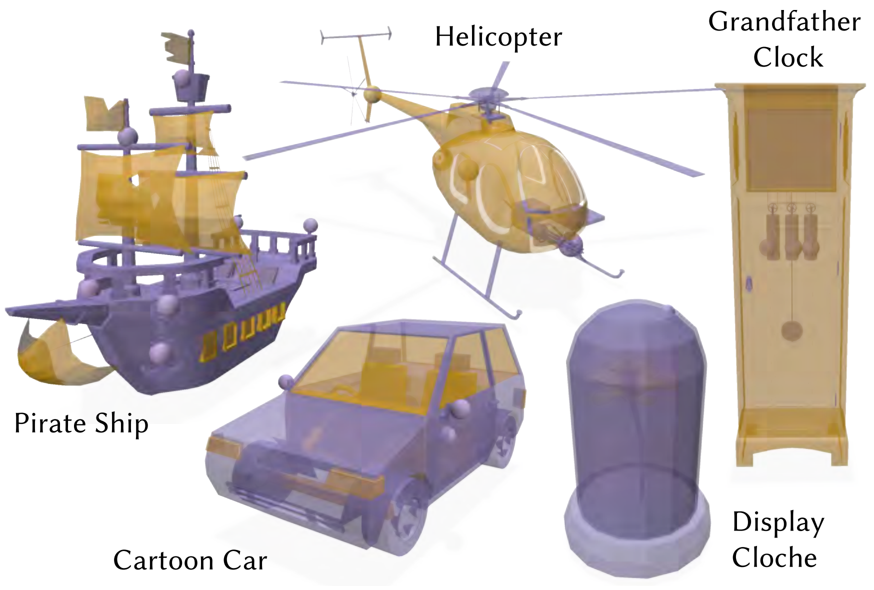


Figure 3.13: Five models benchmark for measuring the labeling effort. Face color represents the target labels: solid faces are colored purple, shell faces are colored yellow. The purple and yellow spheres in the scene represent the interior and exterior constraints in Eq. (3.4) one needs to obtain the target labels, respectively.

Table 3.1: Summary of labeling effort for each benchmark mesh in Fig. 3.13. For each model, we first obtain target face labels through manual face selection by a user. We report: (i) the number of constraints added to our optimization to achieve the target segmentation, (ii) the number of faces labeled as shell/solid, and (iii) the number of face components containing at least one shell/solid label. This table demonstrates that using our tool requires less effort compared to manual selection, by comparing the number of constraints needed to achieve target labels.

Model	*#Constraints*	*#Faces*	*#Components*
Cartoon Car	3	2764/480	14/18
Display Cloche	1	496/1610	6/32
Grandfather Clock	4	2331/1245	48/16
Helicopter	4	8347/5110	236/146
Pirate Ship	8	8118/1361	11/78

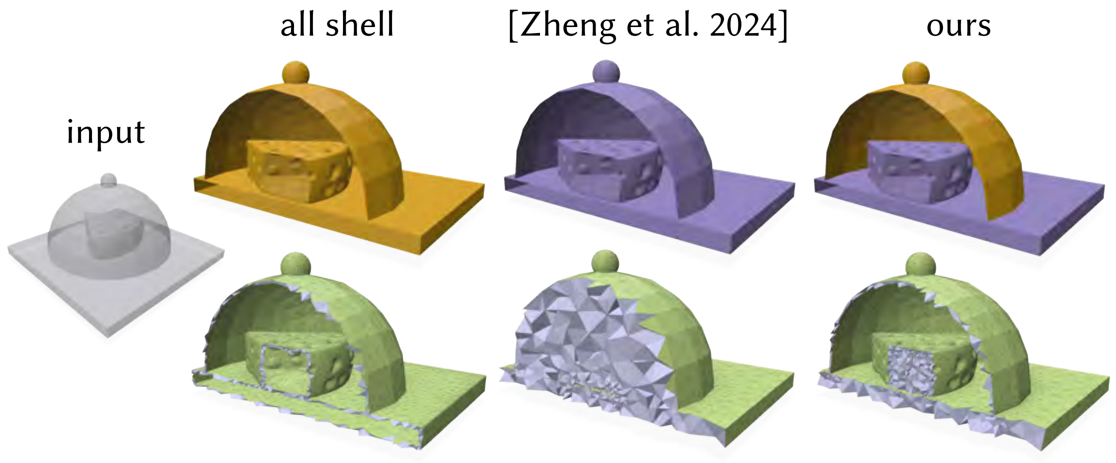


Figure 3.14: We show the volumetric mesh (bottom) created by our solid/shell face labels (no user constraint is needed in this case), which successfully labels the lid as shell and the cheese as solid (right), in contrast to all-shell labeling (left) and the visibility-driven labeling by [221] (middle).

heuristics, such as visibility [221], tends to label closed object as solid, which is prone to unexpected physical behavior or deleting invisible structures. In Fig. 3.12, solid/shell labeling method based on visibility [221] tends to label the can as all-solid, while with easy user-added constraints, our method gives the capability to label the body as shell and the zip as solid. These two different labels lead to different physical behavior when performing a compression simulation. For models with self-occlusion, the visibility-based labeling tends to remove important, but invisible, structures (e.g. Fig. 3.14) or assigning solid labels to invisible shell structures (see Fig. 3.15). In contrast, our method addresses the above issues by using a non-visibility-based winding number field as a prior, augmented with user guidance, to allow control over different inside/outside labelings (e.g., in Fig. 3.11) to be aligned with user's intent.

3.4.3 Ablation

We have an ablation study on how γ in Eq. (3.4) influences the convergence behavior of labels $\boldsymbol{\ell}$ in optimization. In Fig. 3.16, we demonstrate that for a given mesh with the same user-specified constraint, $\gamma = 4$ leads to an immediate convergence on $\boldsymbol{\ell}$ towards either 0 or 1, while $\gamma = 2$ fails to converge during the optimization with non-decreasing gradient. As even higher exponent can

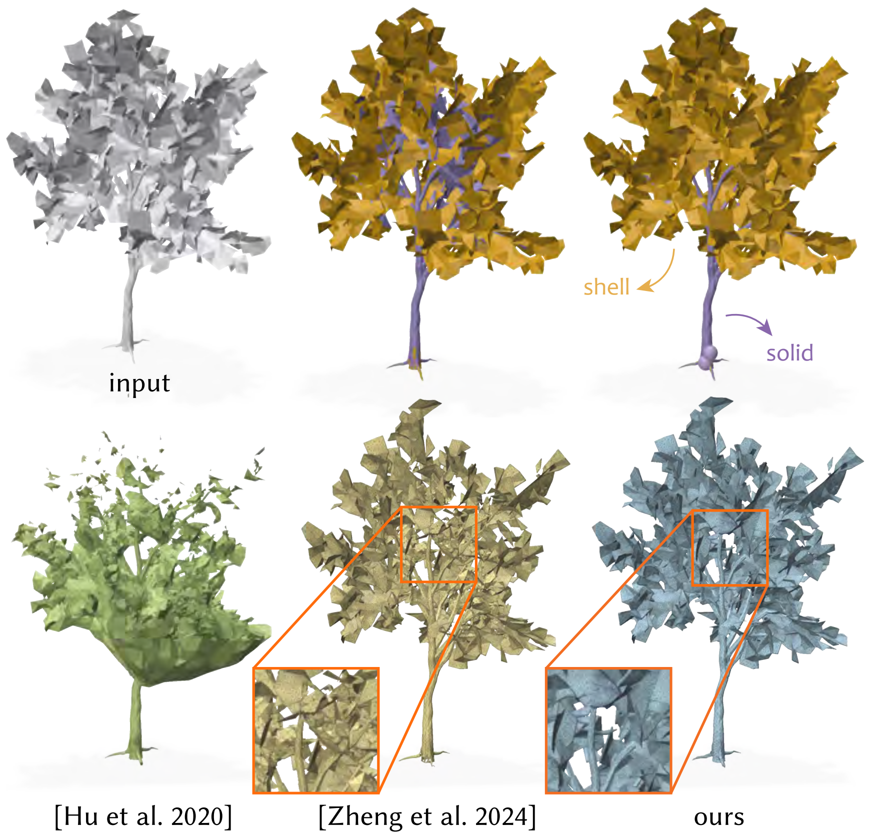


Figure 3.15: Given a tree model with a mixture of solid (e.g., trunk) and shell (e.g., leaves) elements (top left), using a visibility-driven classification by [221] is prone to label invisible leaves as solid elements (top middle), yielding unwanted volumes after tetrahedralization (bottom middle). In contrast, our method is able to label trunk faces to be solid, and leaf triangles to be shell (top right), yielding a user-desired volumetric mesh (bottom right). We also include the reconstruction from [75] if one assign every face to be solid.

lead to numerical instability, we choose $\gamma = 4$ in Eq. (3.4) for better optimization behavior and more effective solid/label classification.

3.5 Conclusions and Limitations

Our approach lays the groundwork towards an accessible tool that could help turning "in-the-wild" 3D assets into simulation-, solid-modeling-, or 3D-print-ready form. Our strategy requires far less manual efforts (often less than 10 user provided points C) to divide meshes into solid

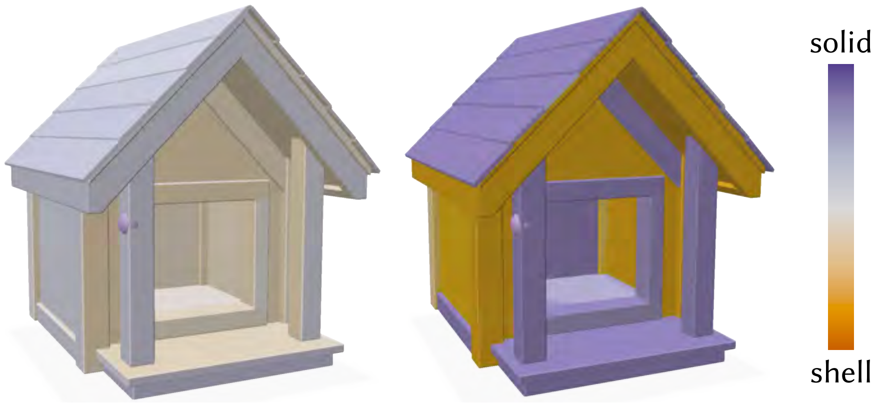


Figure 3.16: Comparison on the differences of the convergence behavior between $\gamma = 2$ (left) and $\gamma = 4$ (right) in Eq. (3.4).

and shell components, in comparison to explicit face selection, which typically requires selecting thousands of faces and familiarity with 3D modeling details such as triangulations, connectivity, selection, and flood-filling. Future work could involve developing a comprehensive and intuitive user interface that enables users to place spheres to designate interior/exterior regions, along with common utilities like mesh selection, undo, and slicing. Collecting a wide range of examples from such an interface could motivate future data-driven methods to learn how to automatically place those guiding spheres. Removing the assumption of requiring the input faces to be correctly oriented, possibly by incorporating normal orientation tools [131], could extend the reach of our method to more types of mesh defects. Allowing face orientations to change during optimization could achieve a broader range of user intents, such as turning a sphere inside out to create a volumetric mesh that extends towards infinity but is hollow inside. Building on top of the generalization of winding number to other representations [10], extending our methods to different representations (e.g., point clouds) could further impact other fields, such as robotics and 3D sensing/reconstruction.

4 | Appearance-Preserving Tactile Optimization

This chapter is adapted from the publication [184] on ACM Transactions on Graphics, a joint work with Chase Tymms and Denis Zorin.

ABSTRACT

Textures are encountered often on various common objects and surfaces. Many textures combine visual and tactile aspects, each serving important purposes; most obviously, a texture alters the object's appearance or tactile feeling as well as serving for visual or tactile identification and improving usability. The tactile feel and visual appearance of objects are often linked, but they may interact in unpredictable ways. Advances in high-resolution 3D printing enable highly flexible control of geometry to permit manipulation of both visual appearance and tactile properties. In this paper, we propose an optimization method to independently control the tactile properties and visual appearance of a texture. Our optimization is enabled by neural network-based models, and allows the creation of textures with a desired tactile feeling while preserving a desired visual appearance at a relatively low computational cost, for use in a variety of applications.

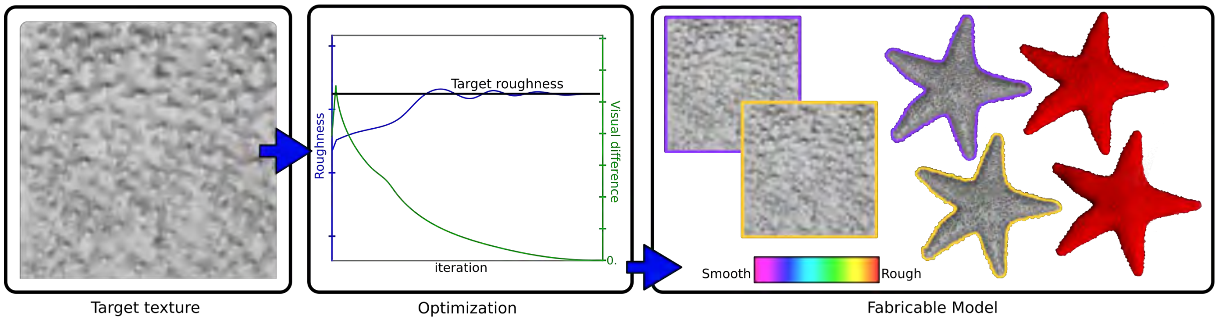


Figure 4.1: Our optimization procedure enables the control of a texture's tactile roughness while maintaining its visual appearance. Starting with a target texture (left), the procedure optimizes toward a desired tactile roughness while preserving the visual appearance (center). The resulting textures can be used to fabricate visually similar but tactually different objects, such as these 3D-printed starfish (right, photographed).

4.1 Introduction

Tactile textures are ubiquitous in everyday life. We encounter tactile textures on the surfaces of fruits and plants, skin, woven fabrics, and many manufactured surfaces. Tactile texture often serves a specific purpose, practical or aesthetic (an object should feel good, not just look good). Creating a particular tactile feeling is a common task which receives less attention than visual appearance, although is often just as important. Tactile feeling plays a particularly important role for people who are visually impaired, who rely on the sense of touch much more.

The tactile feeling and visual appearance of objects can interact in unpredictable ways; for example, the tactile texture may be a byproduct of creating a particular appearance (e.g., an etched pattern), or vice-versa (e.g., knurled grips have a particular look). The goals of achieving particular visual and tactile appearances may be conflicting: e.g., one may want a particular visual pattern on a tool handle, while achieving specific tactile properties optimal for usability. While in many cases, little can be done about the interaction of visual and tactile properties, advanced fabrication technologies like high-resolution 3D printing enable highly flexible control of both visual and tactile texture.

A characteristic feature of both visual and tactile textures is their statistical nature: that many distinct patterns and geometries may look or feel the same. We refer to distinct (in the sense of per-point equality) textures that are perceived in a similar way as perceptually equivalent. The large space of perceptually equivalent textures makes it possible to adjust one aspect of a texture (e.g., tactile) without affecting the other (visual). This type of adaptation makes it possible to separate the process of visual and tactile design.

In this paper, we propose an efficient optimization method for independent control of tactile feeling and visual appearance of a surface. More precisely, the problems we solve can be formulated as follows: given input texture geometry, how can we modify it to achieve certain target tactile properties while minimizing changes to its visual appearance? And conversely, how can we achieve specific visual appearance by modifying geometry, while preserving tactile properties? Our method builds on the previous work on quantitative modeling of perceptual roughness, as well as visual appearance perception. One of the main drawbacks of the highly accurate roughness model we use is the relative expense of its evaluation and the lack of differentiability, making it difficult to apply in the optimization context. One of the main contributions of our work is efficient neural network-based differentiable versions of models for tactile roughness, visual appearance and contact area. The roughness model is in close agreement with an accurate but expensive-to-evaluate model while it also does not require expensive 3D meshing and FEM simulation and can be evaluated directly on the input texture geometry. The speedups we obtain are on the order of 10,000 times for roughness evaluation (although the original FEM model we compare to was not fully optimized), making it possible to use this model in the inner optimization loop. In addition, the resulting neural-network model provides gradients, making it trivial to plug it into an efficient optimizer.

Using the same basic approach, we also constructed a similar neural network model for contact area and for a visual similarity measure for geometric textures involving advanced lighting effects, both with multiple-orders-of magnitude speedups.

Using these models, we developed an optimization method that allows for controlling the changes in visual appearance and tactile roughness. With the same approach, it can also control another aspect of tactile perception, temperature sensation. We demonstrate the behavior of our system for a variety of examples in different contexts and validate our approach with several visual and tactile experimental studies on flat and curved surfaces.

4.2 Related Work

Our work is related to previous work in several domains. Two of the most important works we build on are [183] (we use the roughness model described in that paper as a starting point), and [80], which describes an image-to-image CNN that we adapt to our purposes. Our work is connected to a spectrum of work in visual and tactile perception modeling, texture synthesis and applications of CNN to optimization.

Tactile perception. Research on the sense of touch has found that tactile perception consists of 4-5 dimensions ([178]), including large-scale and small-scale roughness; compliance; friction; and temperature. Here we focus on large-scale roughness, elicited by features larger than 0.1 mm in size and detected through strain; we also consider temperature, controlled here by mediating the area of contact between the skin and a surface. Most previous research in roughness perception has used different types of natural or artificial stimuli that are difficult to control, e.g. [127], [30]. We use 3D printing to allow creation of higher-resolution, more precisely controllable surfaces. We also gain insights from [179], who performed experiments on temperature perception based on the thermal diffusivity and found a relative threshold of discrimination of 43%.

Tactile fabrication. [145] developed a quantitative model for tactile *compliance* perception using stimuli fabricated from materials with different perceived tactile compliance, and demonstrated its applications to fabricating shapes with variable properties. Compared to roughness, compliance rarely affects the visual appearance of an object, so combining the two is relatively straightforward. In [43] a roughness model was obtained using *tactile* textures fabricated from a set of *visual* textures converted to shallow height maps, implicitly creating a close connection between visual and tactile appearance. In our work, we aim to decouple these.

Other recent work in the fabrication domain has aimed to facilitate the incorporation of tactile properties in 3D printed models. [180] provides an interface to fabricate objects with a user-specified weight, compliant infill, and rough displacement map. However, their roughness metric relies on texture feature size, which is not always definable and does not provide a comprehensive model for all textures. [25] develops methods to fabricate objects with specified deformation behavior and textured surface displacement, but does not allow direct perceptual control. [33] addresses a more specific question using 3D-printed hair structures to adequately simulate material roughness and softness for use in immersive virtual reality.

Thermal conductivity is of interest in fabrication but is typically controlled by altering the base material or creating a composite; [198] reviews several options to vary thermal conductivity and other material properties. We aim to control conductivity for tactile contexts by altering geometry. In a related application, [220] optimizes the tessellation pattern of 3D-printed orthopedic casts for thermal comfort.

Texture synthesis. [150] created a model for texture synthesis based on a set of image statistics. Their method performs well on some natural and artificial textures, but fails for others; it also requires a significant amount of time and is therefore poorly suited to optimization. [194] is based on CNN feature-based model (VGG-19) but similarly does not provide a close match for many textures. Classical non-parametric texture synthesis work, e.g. [41];[202], yield high-quality results for many textures, but are not readily adaptable for our optimization purposes. A recent survey of synthesis methods can be found in [11]. Works such as [49] and [186] present synthesis methods based on CNNs but are not robust enough for our optimization purposes. [223] presents a recent GAN-based texture synthesis method with impressive results, but it requires several hours of training for each image, and similarly [215] provides perceptually-based texture synthesis but requires days of training for a set of similar textures; neither is suitable for optimization in its current form. In contrast, we seek a method that is robust for all textures, and whose loss computation does not require a large amount of time.

Optimizing fabricated visual appearance. Several works use optimization to accomplish a similar goal of appearance preservation for 3D printing. [165] uses optimization to alter the geometry of 3D objects to maintain visual appearance subject to other geometric constraints, to produce bas-reliefs for fabrication. [158] designed a pipeline to optimize a 3D printed surface's microgeometry to replicate a desired BRDF. [42] employs optimization to correct for light scatter to more accurately reproduce color in 3D printing, and [173] uses optimization of the internal layer structure of color multimaterial 3D-printing to replicate the full spectrum of color of 2D art, invariant to illumination, more accurately than traditional 2D printing.

Visual similarity of images and textures. Visual similarity metrics are designed to quantify perceptual similarity, with consistency with perception measured by pairwise or three-way comparisons: if the numerical indicator of similarity for one pair of images is higher than for another, then we expect the first pair to be perceived as more different. Well-established visual metrics include those based on *structural similarity*: SSIM [201], FSIM [218], MSSIM [200]. A different metric designed primarily for evaluation of image compression quality, and based on a complex visual system model, is found in [128]. [219] presents a metric based on deep features learned for, e.g., a classification task and combined with a simple metric in the feature space. These metrics were demonstrated to be closer (on relevant datasets) to human perception compared to SSIM. We use a simple, tighter metric based on surface normals discussed in Section 4.3. We discuss our experiments with other measures there. This is consistent with some of the work on depth images, e.g., [68] a method for increasing resolution of depth images using an additional color channel, uses a metric including estimation of the normal difference. [129] develops a procedure to measure texture similarity by matching a localization task to texture statistics; but the current implementation was not shown successful for diverse textures.

Neural networks in model reduction. Model reduction is a well-established area which was using a variety of machine learning-related techniques to decrease the number of parameters needed to simulate a physical model, with the goal of reducing the cost of the simulation, which is particularly important in optimization context. We share this motivation, although we do not aim to achieve this goal through explicitly reducing the number of parameters of the model. Older methods are relatively well-covered in the survey [46]. Very recently, and concurrently to this work, neural networks were applied for reduced-order modeling of Poisson and fluids in 2D [70]. Other model examples are considered in [152].

Steganography. Steganography algorithms aim to hide watermarking or other types of information in data, with a few papers focusing on 3D data; see e.g., [195] for a survey, and more recently [210]. As we do in our work, these methods aim to preserve visual appearance, but the goal is to conceal the hidden information from the naive observer; in our case, we do not want to make the modification of tactile properties apparent.

4.3 Overview

The main goal of this work is to develop a process to allow the control of a texture's tactile roughness or tactile temperature while maintaining its visual appearance, which can produce a range of effects.

Summary. Given an input 2D height field and a desired tactile roughness value or contact area, the model uses learned functions – one for appearance based on rendered shading, and one either for tactile roughness, based on variation of strain in simulated skin, or for tactile temperature, based on a simulated skin contact area – to perform an optimization for roughness or contact area while minimizing visual distortion. We use psychophysical experiments to validate the results.

A general overview of the process is shown in Figure 4.1.

The development of our optimization process consists of the following steps:

- We create a set of 6300 height maps comprising a variety of textures and grayscale images. We run simulations estimating the human finger contacting these heightmaps, and find the resulting field of maximum compressive strain.

- We use a convolutional neural network to learn a function taking the input heightmap and outputting the maximum compressive strain field, and we compute tactile roughness on this field.

- We use a similar neural network to learn a function taking the input heightmap and outputting the contact area between the skin and the texture.

- We learn a function for the height field's visual appearance using a CNN to learn the rendering with shadow and lighting.

- We develop an optimization procedure taking the losses from the learned roughness or contact function and the learned rendering function to optimize for a target tactile roughness or temperature while minimizing change in appearance.

- We validate this procedure by testing several textures both as renderings and as 3D-printed textures and running human psychophysical experiments. We compare against the simpler method of altering tactile feeling using linear scaling.

4.4 Optimization

The optimization procedure acts to alter the geometry of the input texture height field, in order to modify the tactile feeling of the input while minimizing its change in visual appearance.

4.4.1 Optimization Overview

We use three functions in our optimization process to compute tactile and visual difference estimates:

- Roughness: $\phi_r : \mathbb{R}^n \longrightarrow \mathbb{R}^n$, where n is the number of pixels in the height and stress maps, mapping the height field to stress magnitudes at a plane inside the skin where tactile sensors are located. The stresses are sampled at the same resolution as the input height field.

- Visual appearance: $\phi_v : \mathbb{R}^n \longrightarrow \mathbb{R}^{kn}$, mapping the height field to the pixel values of k rendered images with different lighting.

- Contact area: $\phi_c : \mathbb{R}^{2n} \longrightarrow \mathbb{R}^n$, where n is the number of pixels in the height and contact maps, mapping the height field and corresponding strain field to the distance between the skin and the surface at each point.

In addition, we use a function $V : \mathbb{R}^n \longrightarrow \mathbb{R}$, to evaluate the perceptual roughness estimate from the stress field $\sigma = \phi_r(H)$ of height field H.

Using these functions, which we define precisely below, our target functional is defined as follows. For a given input texture height field H_0, and target perceptual roughness r_{trg}, target contact area c_{trg} and target height range $[0, H_{trg}]$ we define the following energy terms:

1. $E_{rough}(H) = |r_{trg} - V(\phi_r(H))|$: the difference between the current roughness and the target roughness, with the strain variation function V defined in Section 4.4.2.

2. $E_{contact}(H) = |c_{trg} - A(H))|$: the difference between the current contact and the target contact, where A is the weighted contact distance function defined in section 4.4.3.

3. $E_{vis}(H, H_0) = \sum \frac{1}{n} \|\phi_v(H_0) - \phi_{v_k}(H)\|_2$: the visual difference, computed as the L2-norm of the pixel-wise difference between the current rendered image and target rendered image, summed over the three different rendering conditions used.

4. $E_{reg}(H, H_0) = \sum \frac{1}{n}(\|\Delta_x(\phi_v(H_0) - \phi_{v_k}(H))\|_2 + \|\Delta_y(\phi_v(H_0) - \phi_{v_k}(H))\|_2)$: the sum of difference variation regularization energies for all rendering conditions, where Δ_x and Δ_y are finite difference matrix operators for horizontal and vertical directions; i.e., an approximation of $\int \|\nabla(\phi_v(H_0) - \phi_v(H)\|_2 dA$.

5. $E_{clamp}(H) = \|H - \text{clamp}_{[0,H_{trg}]}(H)\|_2^2$: the clamping energy to keep the result in the $[0, H_{trg}]$ range.

The total energy we minimize is defined as

$$E(H, H_0) = E_{rough}(H) + w_1 E_{vis}(H, H_0) + w_2 E_{reg}(H, H_0) + w_3 E_{clamp}(H) \tag{4.1}$$

To make the optimization of this function practical, we need to compute $E(H, H_0)$ as well as $\nabla_H E(H, H_0)$ efficiently. However, computation of E_{rough} involves a 3D finite element simulation, including 3D domain meshing and contact resolution; computation of E_{vis} requires rendering of textures with some global illumination effects.

We address both of these problems by approximating ϕ_r, ϕ_c and ϕ_v using neural networks, as these provide (a) fast evaluation of function values (b) evaluation of derivatives with respect to the input parameters. The details of the approximations are discussed below.

Convergence criteria and weight choices. The main parameter of the optimization is w_1, controlled by the user, which represents the trade-off between visual fidelity and closeness to the target roughness.

The weight w_3 is chosen to be relatively high, 10^5, so that the last term operates as constraint. The weight w_2 is chosen to be lower compared to w_1, as E_{reg} acts as a regularizing term, minimizing small-scale noise by picking smoother solutions among those with low values of the first two terms. We use $w_2 = 0.06$.

For contact area, which has values on the order of $100\,\text{mm}^2$, approximately 1000 times the

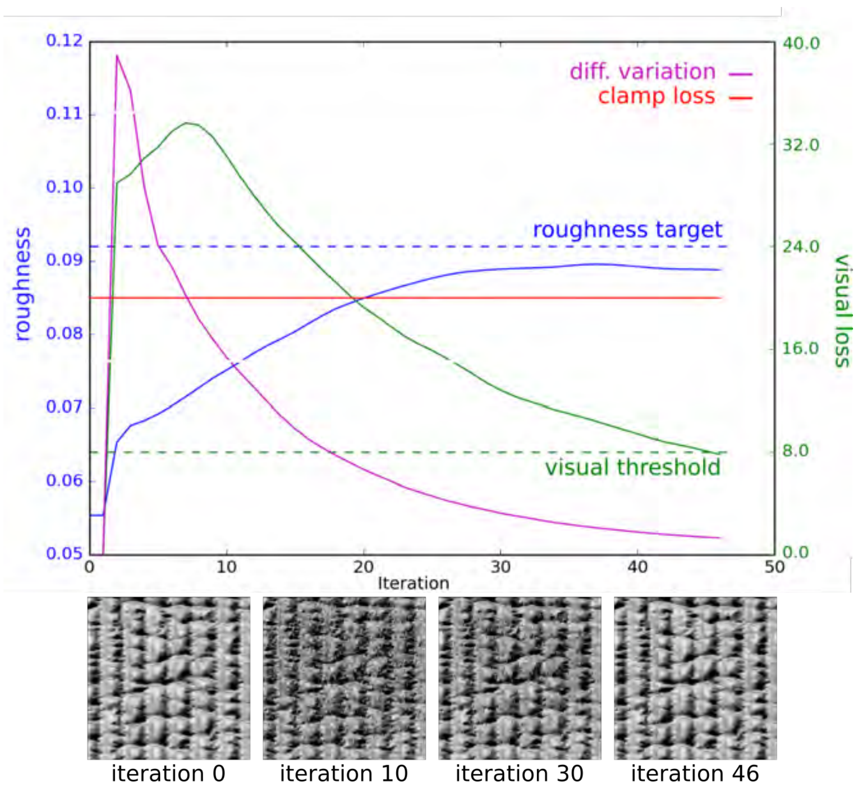


Figure 4.2: Parameter convergence during optimization for roughness and visual appearance. The goal is to alter the roughness of the input texture (iteration 0) while preserving its visual appearance, which is done by the final iteration.

typical roughness values, these weights were scaled up by 1000.

We use a stopping criteria for optimization that places bounds on three of the energy components: For roughness, $E_{rough} < \varepsilon_r r_{trg}$, with $\varepsilon_r = 0.1$, about half of the 19% threshold for tactile roughness discrimination described in [183]. For visual difference, $E_{vis} < \varepsilon_v \|\phi_v(H_0)\|_2$, with $\varepsilon_v = 8$; this is proportional to image resolution, and was experimentally found as a conservative goal to avoid visible changes, corresponding to a 2% change in pixel values.

The height constraint is expected to be satisfied nearly precisely: $E_{clamp} < \varepsilon_c$, with $\varepsilon_c = 10^{-4}$. We used $H_{trg} = 3$, to ensure the height remain below 3 mm, selected as a reasonable maximum height for a fabricable tactile texture.

An example of the optimization process for a texture is shown in 4.2. The effect on convergence of using altering the weights is shown in Figure 4.3.

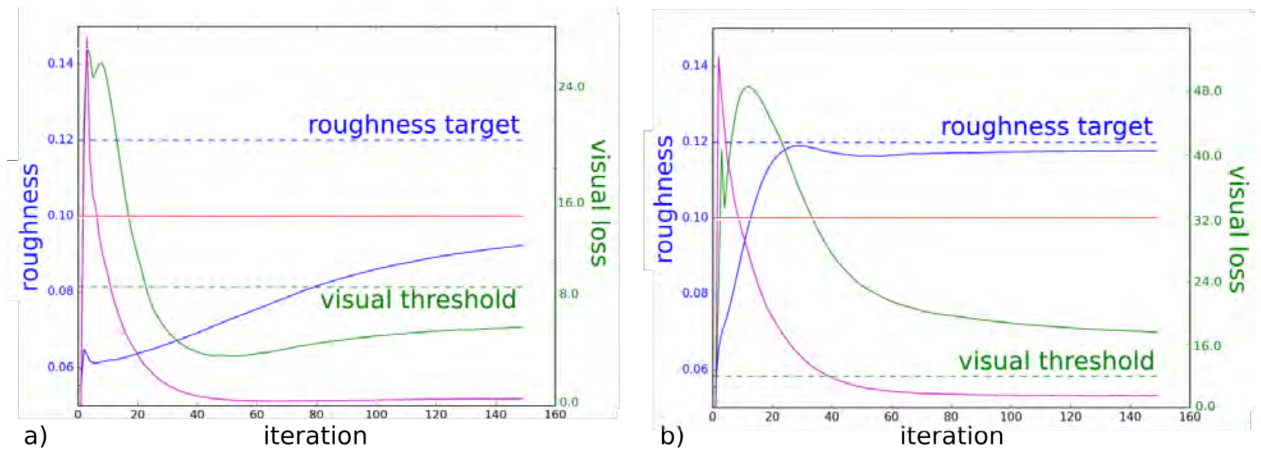


Figure 4.3: a) When a significantly (10x) lower weight is used for w_1, convergence of roughness to the target may not occur. b) A significantly higher (10x) weight for w_1 causes the visual energy to converge more slowly, and it may not reach the target threshold.

The Adam optimizer ([95]) implemented in Pytorch is used for optimization. A learning rate of 0.027 was chosen through trials with single parameters to permit convergence of the parameters but avoid excessive oscillation. In the next sections, we explain how the roughness, contact and visual functions, respectively ϕ_r, ϕ_c and ϕ_v, are defined.

4.4.2 Tactile roughness

We use a modified version of the model developed in [183], which computes the tactile roughness of a surface by simulating the strain variation field resulting from skin contact on the surface. The computation of the model is relatively expensive; we briefly summarize the model here for completeness. The main step of the model is a finite element method simulation of the contact of the skin with the tactile texture defined by $H(x, y)$, to obtain a corresponding displacement field $\mathbf{u}_H(u, v, w)$, where u, v, w are 3D coordinates in the undeformed layer of skin, with $w = 0$ corresponding to the surface, and $w_0 = 0.75\,\text{mm}$ corresponds to the approximate depth of the tactile receptors.

To approximate the skin, we use the same two-layer block model as [183]. The block is 1 cm^2 in surface area and 0.5 cm in height, with a rigid upper half and soft lower half, and a force of 10

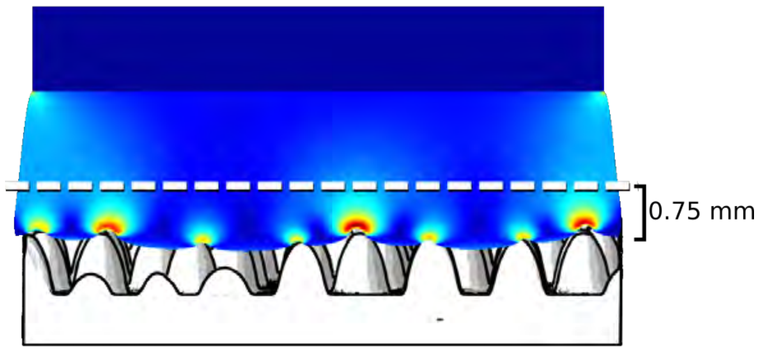


Figure 4.4: In the original roughness model, a 3D FEM simulation was used to simulate the skin touching a textured surface, and the maximum compressive strain field was sampled from a depth of 0.75 mm.

N is used. A model of the simulation is shown in Figure 4.4.

For a displacement field $\mathbf{u}$, $\epsilon[\mathbf{u}] = \frac{1}{2}(\nabla\mathbf{u} + \nabla^T\mathbf{u})$ is small-deformation strain tensor. If $\lambda_3(\epsilon)$ is the largest-magnitude negative (compressive) eigenvalue of the strain tensor, our perceptual roughness estimate $f(H)$ can be written as

$$f(H) = \mathrm{V}(\lambda_3(\epsilon(\mathbf{u}_H(\cdot,\cdot,w_0)))) \tag{4.2}$$

where V is the strain variation function on the plane $w = w_0$. We replace a stochastic function defined in the original model with a deterministic function described in more detail below.

The expensive step is the computation of displacements $\mathbf{u}_H$ for a given H: it requires sufficiently fine 3D meshing to resolve the detail at the scale of smaller texture features, and solving a nonlinear (due to contact) constrained elastic deformation problem, which in our current implementation has a computation time of 20-40 minutes and uses 15GB of memory when using the required highly-refined mesh. In addition to the cost of evaluation, it is difficult to obtain an approximation of the derivative of this function other than by extremely expensive finite differences, so optimizing a functional depending on the $\mathbf{u}_H$ can only be done with gradient-free methods.

This is the step that we replace with a direct map

$$\phi_r(H)(u,v) \approx \lambda_3(\epsilon(\mathbf{u}_H(u,v,w_0))), \tag{4.3}$$

represented with a neural network.

Strain variation function. In [183], the strain variation function $V(\sigma)$ was computed using a large set S of N randomized pairs of samples (p_1,p_2), $p_i = (u_i,v_i)$, separated, on average, by a distance d, are computed. Denoting $\sigma(u,v) = \lambda_3(\epsilon[\mathbf{u}_H](u,v,w_0))$,

$$V(H) = \frac{1}{N}\sum_{(p_1,p_2)\in S} |\sigma(u_1,v_1) - \sigma(u_2,v_2)| \tag{4.4}$$

$N = 8000$ sample pairs were used, sampled from disks of radius 0.8 mm placed at the endpoints of randomly selected segments of length 2.2 mm.

Instead of using a random sampling of points, here we use a deterministic evaluation of variation between each point and its neighbors within the desired distance, in order to derive a strain variation field (Fig. 4.5):

$$V(H) = \frac{1}{2rl}\int_{x=0}^{l}\int_{y=0}^{l}\int_{\Delta=d-r}^{d+r}\int_{\theta=0}^{\pi} |\sigma(x,y) - \sigma(x+\Delta\cos\theta, y+\Delta\cos\theta)| dxdyd\Delta d\theta \tag{4.5}$$

This function is smooth, so the gradient of the complete roughness estimates can be computed.

Learning the strain field. The FEM simulation used to compute $\sigma(u,v)$ in [183] is used solely to find a single 2D strain field; that is, the simulation takes as input a 2D grid (the heightmap defining the boundary conditions for the contact area), and returns as output a 2D grid (the maximum compressive strain at a depth of 0.75 mm). Image-to-image translation problems have been studied extensively in machine learning, and here we adapt a convolutional neural network

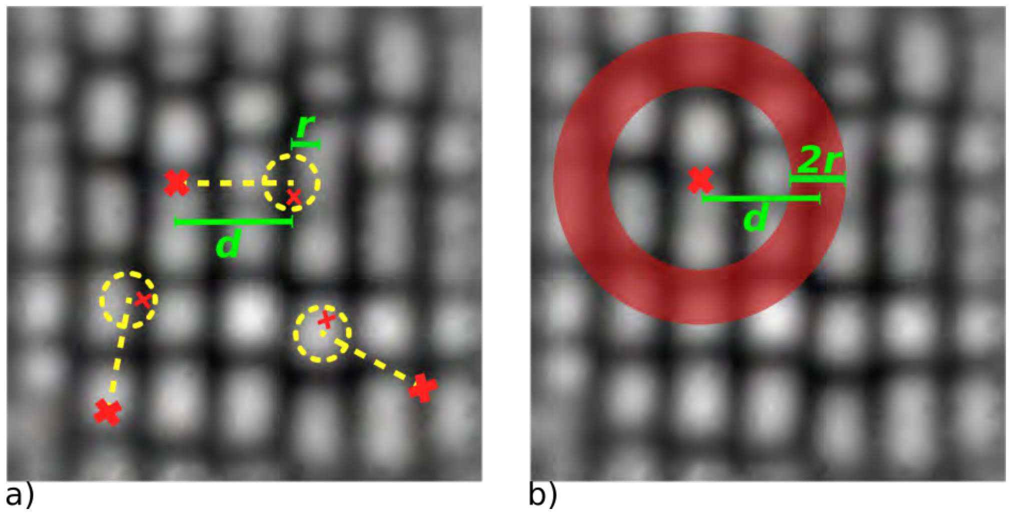


Figure 4.5: a) Stochastic sampling; b) Equivalent deterministic sampling

described in [80] to learn a relationship ϕ_r between the input height map and the output maximum compressive strain.

To acquire ground truth simulation data, we ran the FEM simulation for the 3D skin model using a heightmap dataset of with 6307 image pairs, similar to the size of several datasets successfully trained with this neural network structure. We use a set of black and white images and textures (including the Describable Textures Dataset [29], VisTeX [132], and Brodatz texture database [19]) and procedural textures to enrich the dataset. In some cases, images were randomly cropped and/or scaled, and in some cases procedural noise was added. Heightmaps had a maximum vertical height of 3 mm and represented a texture of size 100 mm^2. As suggested in [183], for each simulation we found the maximum compressive strain field at a depth 0.75 mm, and the strain field of a flat texture simulation was subtracted to discount any effect from edges.

Inputs and outputs were scaled to 128×128px images. The set was split randomly into three sets: testing (312 images); training (4918), and validation (1077). We used the convolutional neural network used as the generator in [80], with no dropout and using BCE loss, and trained for 200 epochs with batch size 1.

The learned strain field and its resulting tactile roughness value were computed from an un-

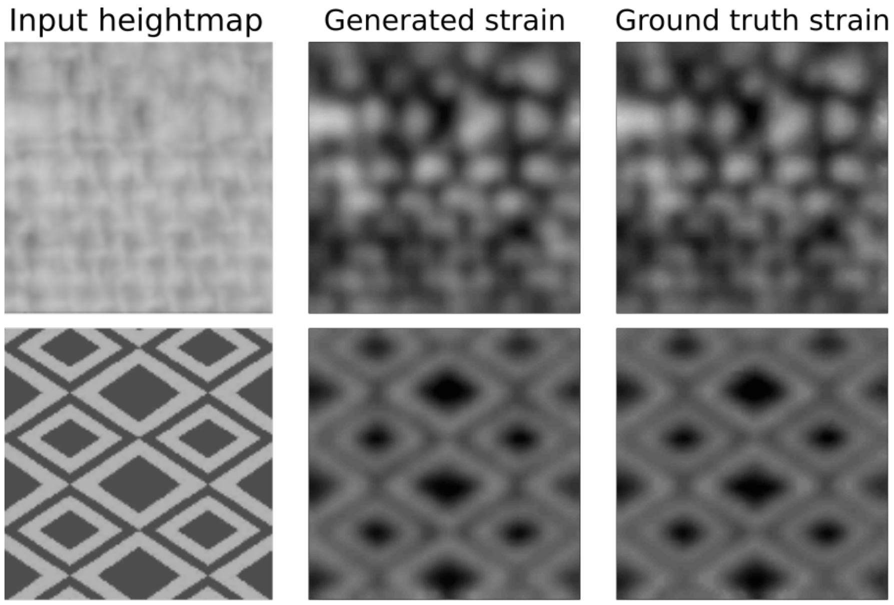


Figure 4.6: Two examples from the learned CNN test set show the learned and ground-truth maximum compressive strain fields from the input heightmaps. Strain fields are shown with increased contrast for visual clarity.

seen testing set, and the learned value was compared against the actual value. The median error in roughness was 5.3%, and the average error was 8.0%, well below the perceptual threshold of 19%. These values are well below the threshold of discrimination of 19% described in [183]. The error distribution is shown in Figure 4.7.

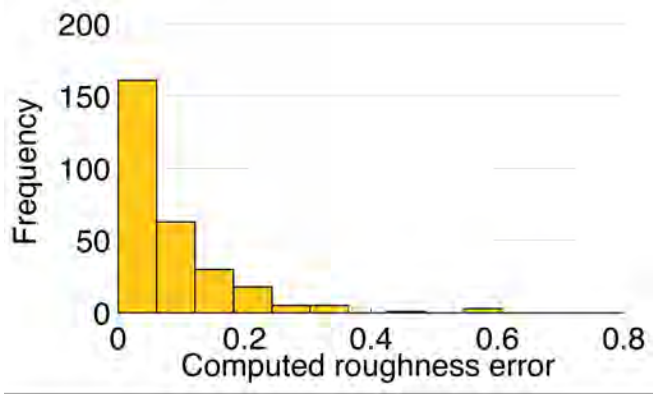


Figure 4.7: The difference in computed roughness between the learned strain field and the real strain field is typically very low, with a median of 5.3%.

The network allows the roughness to be computed in an average of 0.05 seconds, a significant speedup compared to the 20-40 minutes required to run the full FEM simulation.

The learned function and its gradient are used in optimization for a texture to converge toward a desired tactile roughness, as shown in Figure 4.2.

4.4.3 Contact area

Computing the contact area requires the same time-intensive FEM simulation as computing the roughness field. To compute the contact area, we use a function taking as input the height field and outputting the field of distances between the surface and the simulated skin at each point. The computation of this distance field is expensive and requires an FEM simulation as described in section 4.4.2. Therefore, we replace this step with a neural network.

Learning the contact distance field. The FEM simulation takes in the input height field H and outputs a mesh displacement field $\mathbf{u}_H$ describing the displacement of the skin when in contact with height field H. From this displacement field and the height field, we can acquire the field of the distance $\mathbf{d}$ between the skin and the input texture at each point, where a distance of 0 indicates skin contact with the texture surface.

We adapt a similar convolutional neural network to learn the relationship between the input height map H and the output distance field $\mathbf{d} = \phi_c(H)$. We used the same height field training set as used previously in Section 4.4.2, which had about 6300 pairs. To improve the accuracy of the learned function, we also provided the strain field as input, so that the input to the function has 2 channels of input: the height field and the strain field. An example of the function's input and output is shown in Figure 4.8.

To compute the error for the testing set of size 250, the learned distance field was computed for each input heightmap with its learned strain fields, and the contact area was computed and compared to the actual contact area derived from simulation. The errors in computed contact area for this set had a mean of 2.7%, as shown in Figure 4.9.

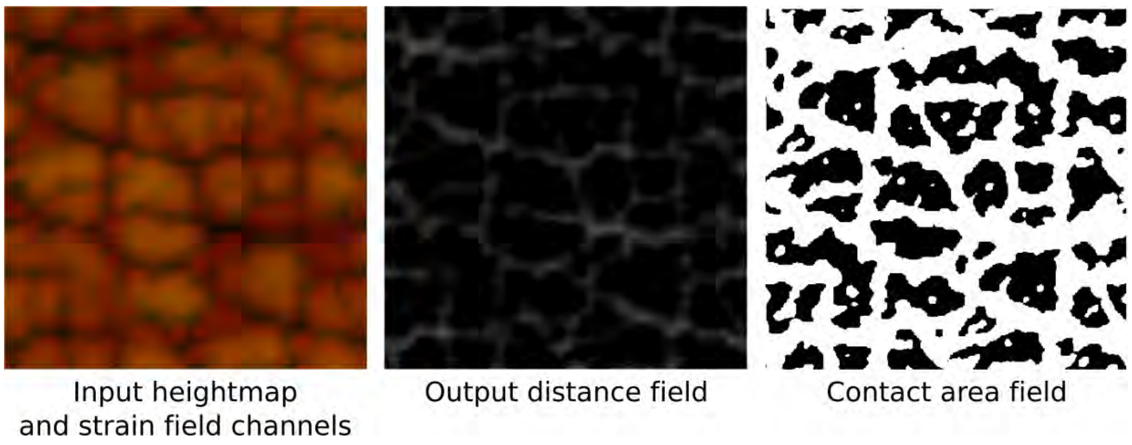


Figure 4.8: The contact area function takes as input the input heightmap (left, red channel) and the strain field (left, green channel) and outputs the distance field (center, where black indicates a distance of 0). The distance field can be used to compute the contact area (right, where the contact area is black)

.

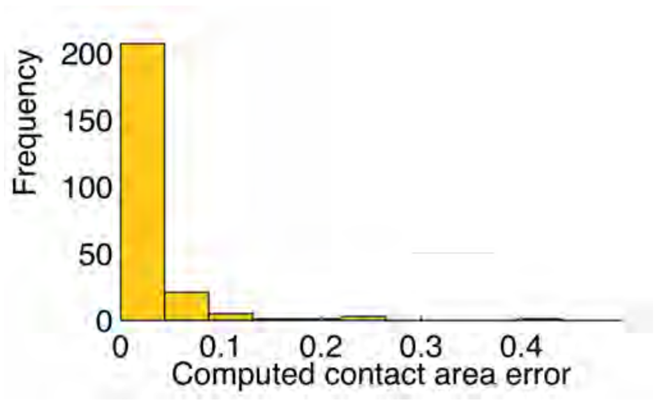


Figure 4.9: The learned contact area matches the actual contact area very closely, with an error of 2.7%.

.

Contact optimization. The optimization aims to modify a texture so that its total contact area moves to a particular target. Because contact area itself is a discontinuous function, the optimization process was often unable to converge. Therefore, we use a smooth function weighting the contact area at each point proportionally to the inverse of its distance. That is, for contact distance field **d**, contact area is approximated by:

$$A(H) = \int_{x=0}^{l} \int_{y=0}^{l} \frac{1}{1 + 80 * \mathbf{d}(x, y)} \tag{4.6}$$

This function provides a smooth weighted contact distance, so that a distance of 0 has a weight of 1; weights decay rapidly so that a distance of 0.01 mm has a weight of 0.5 and a distance of 0.1 mm has a weight of 0.1.

4.4.4 Visual appearance

To preserve a texture's visual appearance during optimization, we use a custom function based on visual similarity of the original height field and the optimized one. Ideally, to measure visual similarity, we would consider all possible views of a pair of textures under different lighting conditions, apply a visual difference metric between each pair, and compute an aggregate metric. We follow these steps, but use a restricted set of lighting conditions and use the simplest visual metric to compare the images. In Section 5.8, we validate the setup we use comparing it with a more expensive multiview optimization.

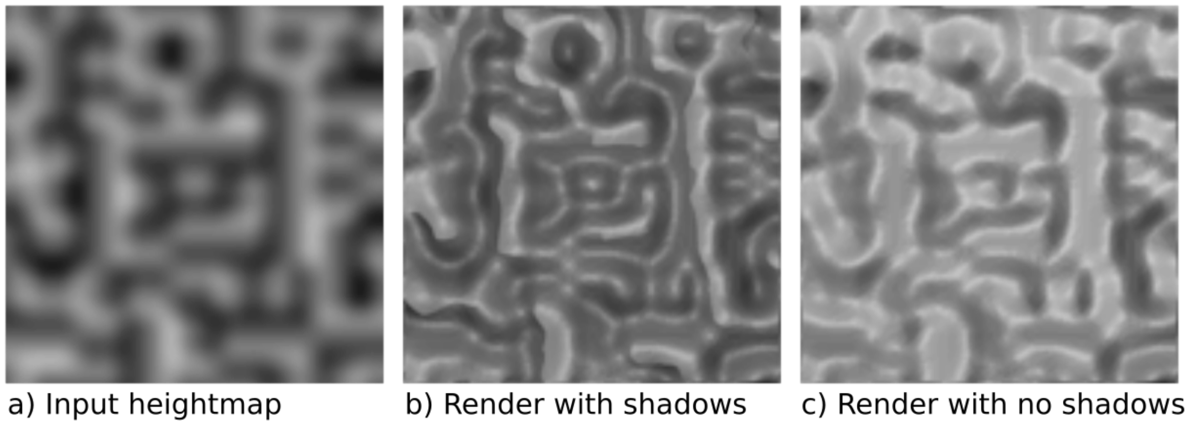


Figure 4.10: A texture heightmap rendered with (center) or without (right) shadowing and ambient occlusion. Shadowing in small regions of lowered height is critical to a texture's visual appearance.

We found that to ensure realistic results some features of images used to evaluate visual similarity are critical. Specifically, we have observed that *shadows*, *ambient occlusion* and *gloss* affect visual texture perception in a critical way (Figure 4.10), as a texture comprises many small elements that cast shadows over the surface. For this reason, we must opt for a rendering pipeline

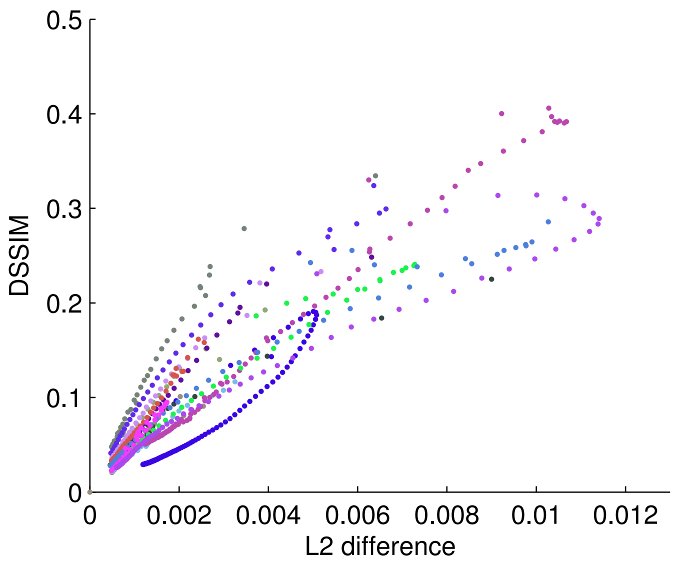


Figure 4.11: Plot showing render DSSIM and L2 difference errors for a set of textures in optimization steps.

supporting these features to generate views of the texture, rather than, e.g., approximating the texture image with the dot products of the normal with the light direction.

As discussed in Section 4.2, a variety of measures of visual similarity exist and are widely used. Most could be used in our context in a way similar to the function V above used for roughness; e.g., [219] describes a perceptual measure of visual similarity represented with a neural network, that can be easily applied in our context. However, we found that in the optimization context, these measures tend to be too "permissive": while these metrics are good for measuring distance between real images, synthetic images can be far from a given image perceptually, but close in the sense of these metrics. For this reason, we opt for a relatively conservative $L2$ norm of the difference between images. Figure 4.11 shows a scatter plot exhibiting that $L2$ has a correlation with DSSIM.

Rendering. Heightmaps were rendered in a gray material with low specularity, similar to matte plastic, using a Phong shader. Objects were rendered with three different lighting conditions, with a single constant-direction parallel-ray light sources at an angle of 35° from the x-y plane

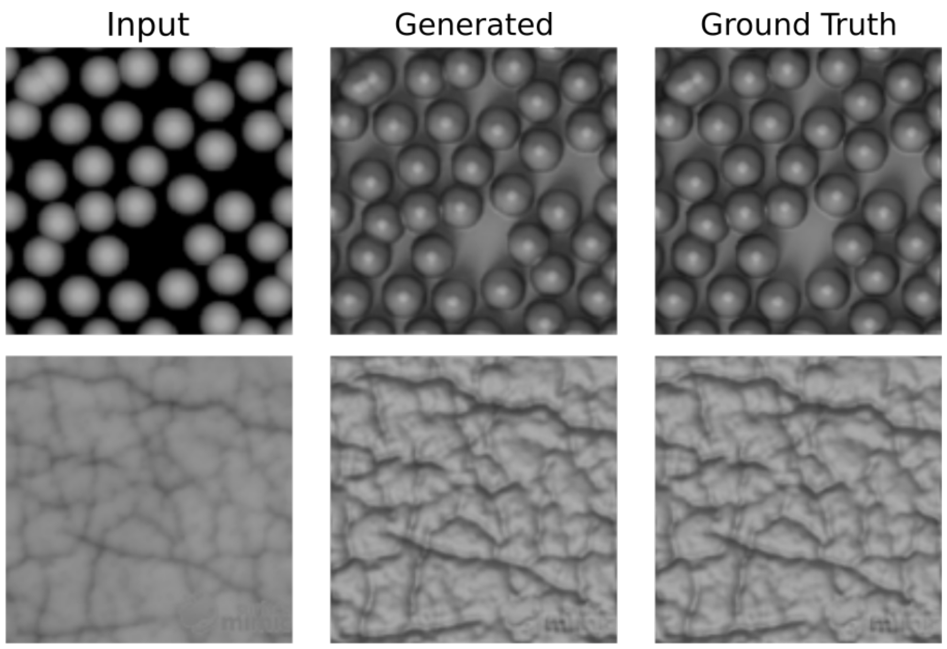


Figure 4.12: Two examples from the test set for visual rendering. The learned function for the rendering of heightmaps was learned with high accuracy: in most cases generated and real renderings are visually indistinguishable.

and rotated on the z-axis 10°, 130°, or 250°. Images were rendered at 128 × 128px using Blender.

While differentiable renders have recently appeared [115], given the highly restricted nature of the renderings that we need to compute (square texture samples), we opted for a similar approach as we use for the stress maps for the roughness measure. As an additional benefit, this approach also provides a gradient of the rendered image with respect to the heightfield.

For each lighting condition, we trained the generative adversarial network of [80] on a set of 4764 images, with a validation set of size 1059. The network was trained for 200 iterations. Results showed high accuracy, as seen in Figures 4.12 and 4.13. All three lighting conditions had similarly high accuracy (with mean pixel errors of 2.2%, 2.3%, and 2.4%).

The neural network offers a significant speedup to rendering: the neural network render computation time is only 0.05 seconds after the network is loaded; while traditional rendering time is approximately 15 seconds with ray-tracing shadows using Adaptive QMC and 20 samples for the lighting source, and ambient lighting and occlusion.

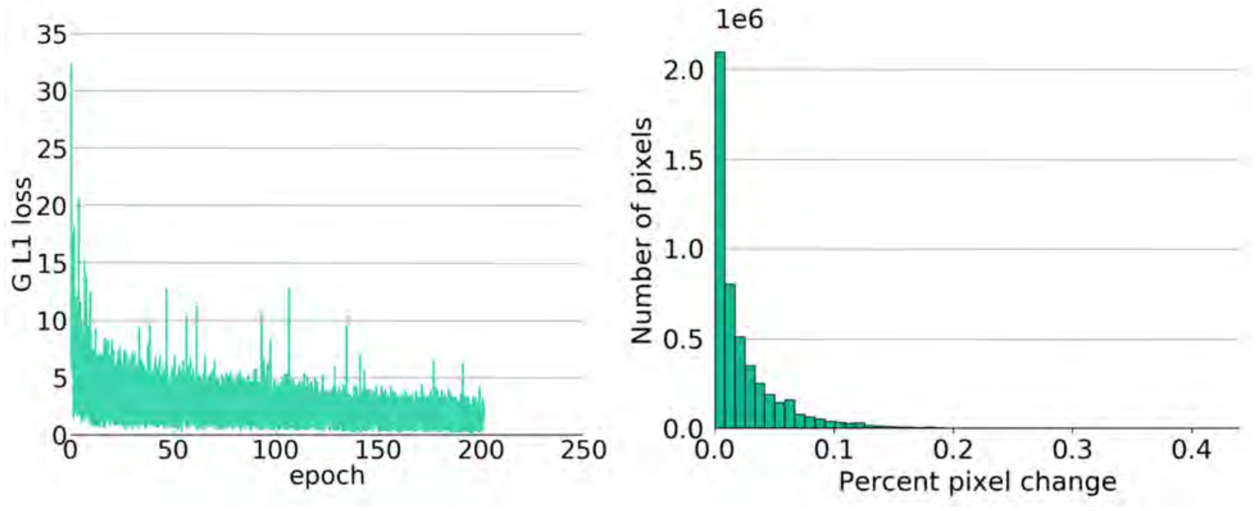


Figure 4.13: Left: L1 loss convergence of the generator during training of the GAN on texture rendering for one lighting condition. Right: Histogram of the percent differences of all rendered pixel values across 300 real and generated texture pairs in the test set. Real pixel values are approximated very closely by the network, with most pixels changing by less than 5%. The mean difference is 2.3%.

4.5 Results

4.5.1 Optimization results

Figure 4.15 shows the results of altering the roughness of a selection of textures using our optimization for a desired tactile roughness maintaining visual appearance. Textures are rendered here using a different lighting setup than the ones used for learning. Textures shown represent 10 mm × 10 mm in size.

Choosing a ground truth to compare to in our experiments is somewhat difficult, as we are not aware of any previous work on optimizing tactile properties for complex textures. We have chosen *linear scaling* as one obvious way to change geometry to increase texture roughness, while maintaining similarity to the original texture; this method was used in [183].

On the righthand side of Figure 4.15, we show the results of using linear scaling of textures to achieve the desired roughness. Textures are first scaled in height up to a limit of 3 mm; then, if necessary, they are scaled in the x-y direction. The optimization-generated textures are nearly

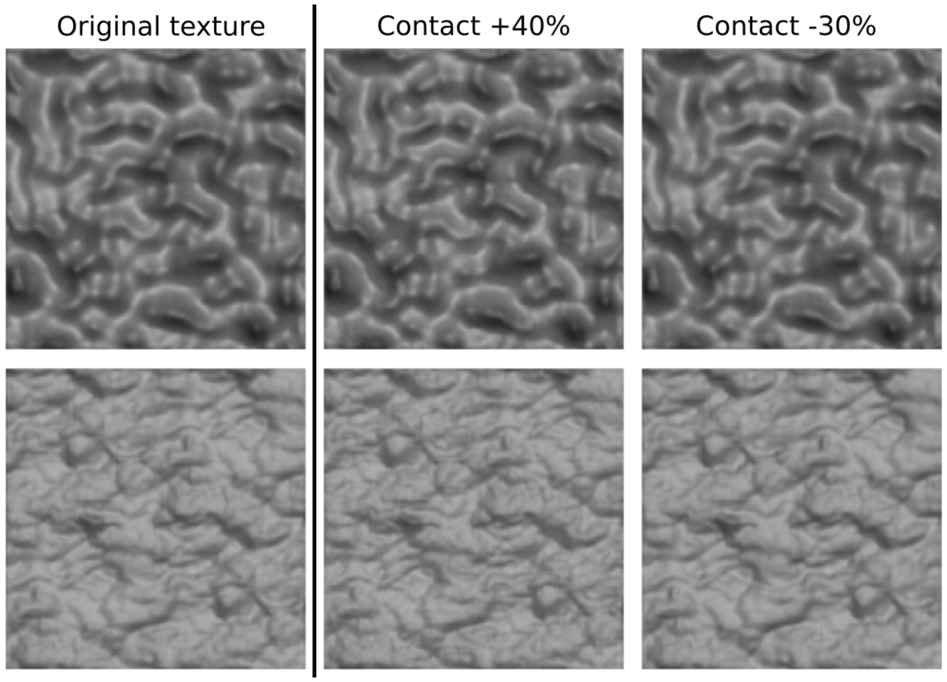


Figure 4.14: Renderings of two textures optimized for different contact areas.

indistinguishable from the original textures, while the textures modified with linear scaling are almost always noticeably different, except in cases where the desired roughness is very close to the original roughness. Making the texture flatter or scaling it upwards results in obvious differences. Additionally, for some textures, a sufficient change in roughness is not achievable through linear scaling alone.

4.5.1.1 Errors

Roughness. To ensure that the learned functions were robust to the types of textures generated with optimization, the errors in computed roughness were also computed for a set of 150 optimized textures, with three different target roughness values. The average error between the simulated and learning-computed roughness for the optimized texture was 8.4% with a median of 6.4%, compared to an average of 8.0% and mean of 5.3% for the overall test set.

Contact area. The same test was performed for a set of 100 textures optimized to have signifi-

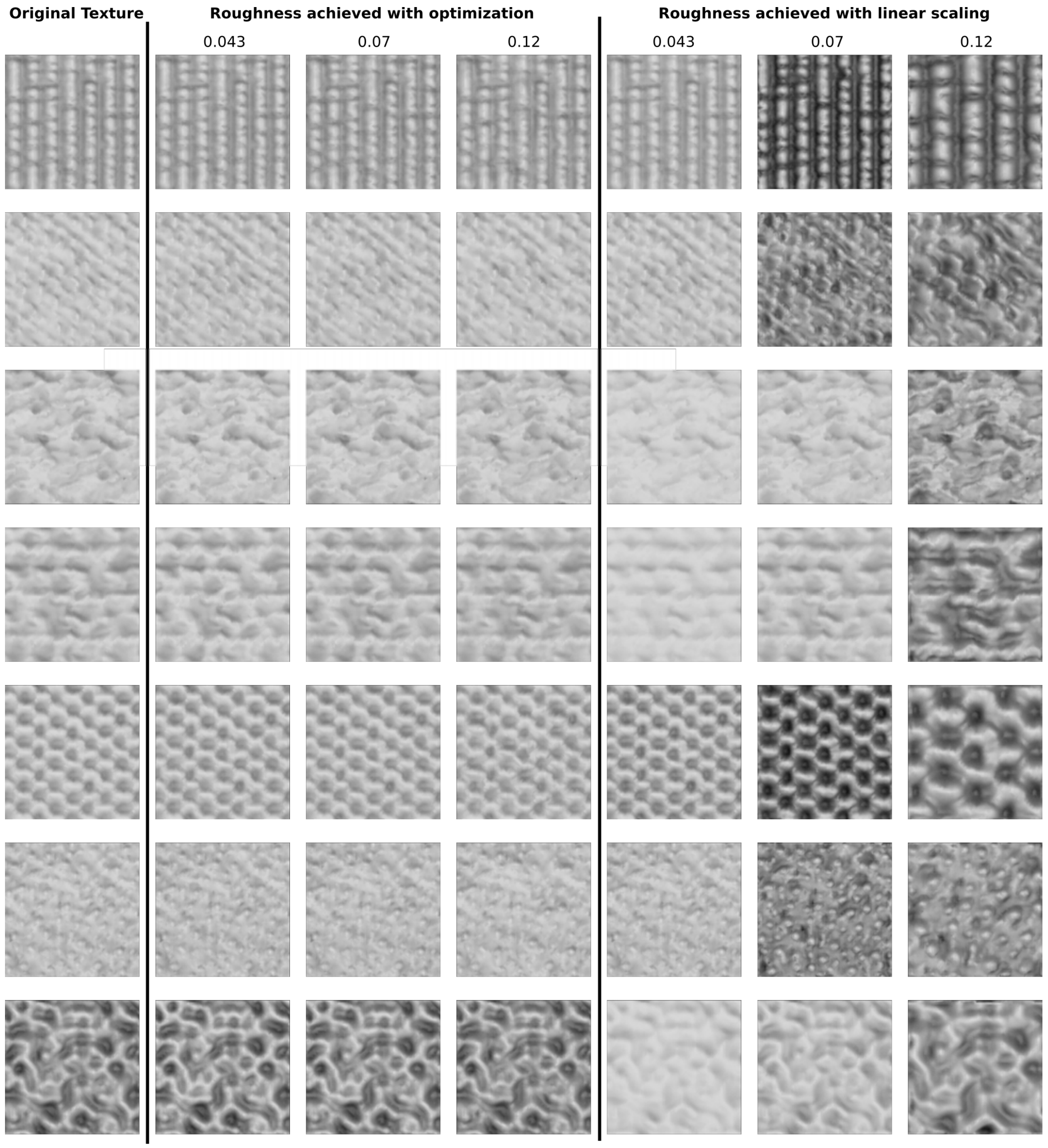

Original Texture
Roughness achieved with optimization
Roughness achieved with linear scaling
0.043
0.07
0.12
0.043
0.07
0.12

Figure 4.15: Seven example textures optimized for a desired roughness. The leftmost column shows the original, target visual texture; the next three columns show the results when the roughness is achieved through our optimization process; the final three columns show the results when the same roughness is achieved through linear scaling in the z and/or xy directions. The optimization process achieves the desired roughness with nearly-imperceptible changes to the visual appearance.

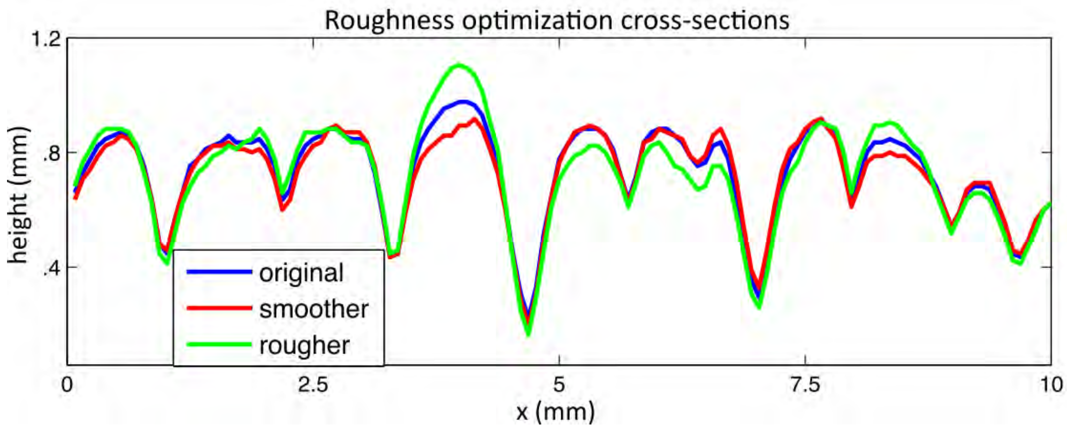


Figure 4.16: Example of a texture cross-section for textures optimized for roughness. Changes to peaks and troughs are not easily predictable.

cantly different target contact area. Here the average error in contact area between the simulation and the learned data was 9% with a median of 4.1%. The average error for the non-optimized input set was 7.9%, with a median of 2.4%.

4.5.2 Evaluation and comparisons

The relationship between a surface's geometry and its tactile properties is intricate, as it depends on the difficult-to-predict way the elastic skin contacts the texture geometry. The tactile roughness is dependent on the uneven distribution of pressure resulting from that contact area. Optimization for these tactile properties while maintaining a similar appearance results in subtle changes to the texture geometry, as shown in Figure 4.16. It typically is not as simple as, for example, using height modification or frequency filtering: the target may be impossible to achieve, and the visual appearance may not be preserved well, as discussed below.

4.5.2.1 Comparison with other methods

Height modification. Linear scaling is a simple method of altering a texture's roughness or contact area. If a texture is scaled up vertically, the contact area will decrease and the roughness

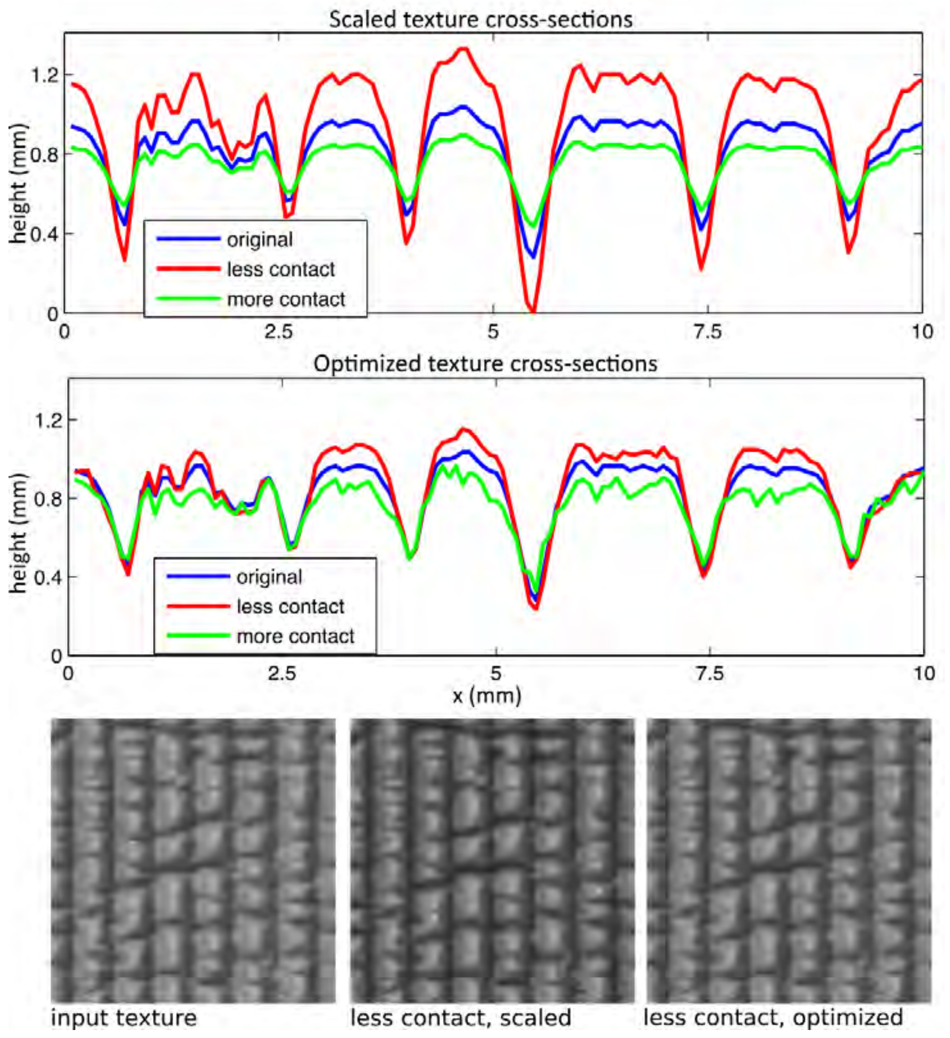


Figure 4.17: Comparison of textures optimized or vertically scaled to alter contact area. The optimization results in smaller changes to the geometry and better preservation of visual appearance.

will increase. The relative geometry is preserved, which suggests the appearance is also preserved to an extent. However, as seen in Figures 4.17 and 4.15, the appearance often cannot be preserved. In contrast, our contact and roughness optimizations alter the geometry in precise and small ways to change the contact while preserving appearance.

Frequency modification. Another intuitive method of altering a texture's roughness is to use bandpass filtering, altering the texture in the frequency domain to reduce or amplify certain frequencies. Literature and psychophysics studies suggest that roughness perception is highest

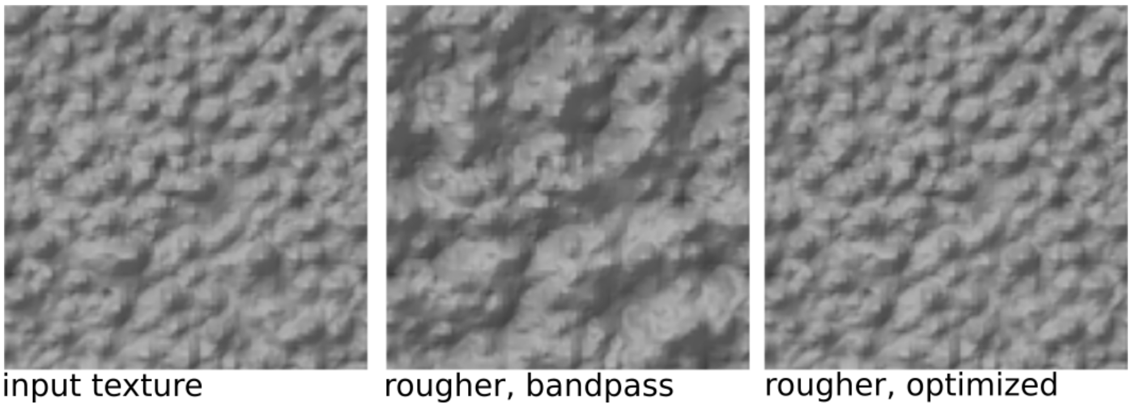
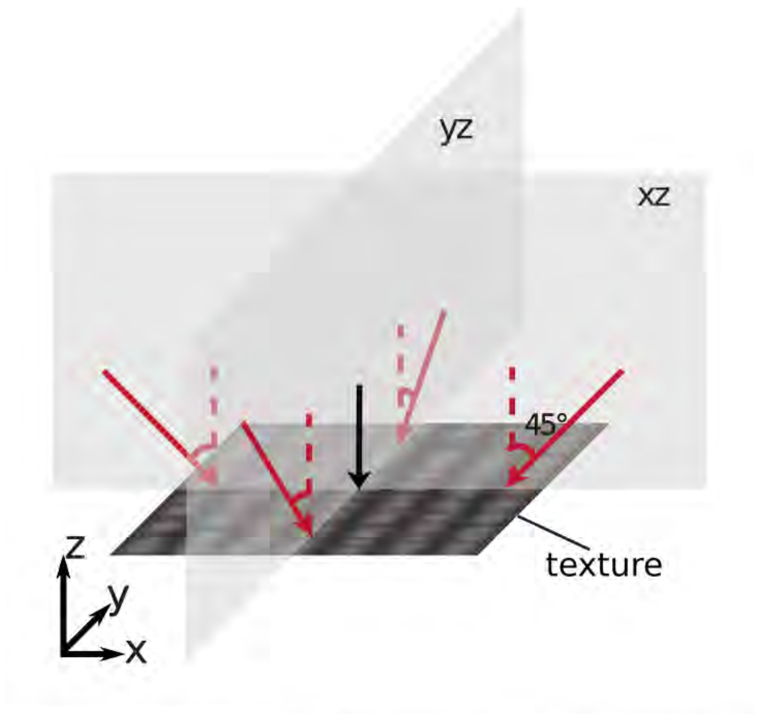


Figure 4.18: Comparison of modifying a texture's roughness by modifying the dominant frequency of the texture, and using the optimization process. Modifying the frequency adds large-scale noise to the geometry, which is clearly visible on the texture.

when features are spaced at a wavelength of 2-3 mm apart [72].

However, we have observed that modifying a texture to alter the frequencies in that range does not alter the roughness is a reliable manner for all textures. For example, if the frequency is increased but the amplified areas are not contacted by the skin, the roughness will not be affected. More importantly, modifying the frequencies does not guarantee preservation of visual appearance. Figure 4.18 shows an example of modifying a texture's roughness by 4 just-noticeable-difference (JND) thresholds by increasing the geometry's frequencies in the 2.5mm wavelength range. The large-scale noise added to the geometry to achieve the target roughness is visible in the texture. Our optimization produces smaller changes in the geometry that are not easily apparent.

Other filtering methods. Contact area and tactile roughness depend on the way the elastic skin conforms around the geometry, which changes in nontrivial ways when the geometry is e.g. smoothed using a filter. For example, smoothing a sharp peak results in increased contact area and decreased height, which decreases roughness; but smoothing a round peak results in decreased contact area and therefore may increase perceived roughness.



	top only	all five	top & xz 45°	top & yz 45°
top only				
all five	2.09%			
top & xz 45°	1.88%	1.43%		
top & yz 45°	1.79%	1.29%	1.71%	

Figure 4.19: Top: We tested the optimization with the addition of more points of view, 45° from vertical on the yz and xz planes, along with the original single top point of view. Bottom: The table shows the percent pixel difference between the optimization height map results, for the original top point of view, one additional point of view, or all five points of view. Only small changes occur when one or more additional points of view are added.

4.5.2.2 Alternate points of view

Our visual optimization used a single, top point of view and multiple lighting conditions. To determine the value of utilizing additional view directions, we ran the optimization with four additional points of view, 45° from the vertical direction at four angles (Figure 4.19, top) and with the same three lighting conditions. As with the top view, additional neural networks (one per view) were successfully trained to produce the rendered image from the specified angle and light source. Rendered images were rescaled to 128×128 pixels. The roughness optimization was run with visual weight assigned to the new points of view: the single additional point of view was weighted 40%; and when using all five points of view, 40% was given to the four new points of view. The remainder of the visual weight was given to the top view direction. We choose a higher weight for the top view to reflect higher importance of direct viewing in surface perception. As the sensitivity of the results to the addition of new points of view was shown to be low, we did not explore other options for weight allocation further.

We evaluated the results of the new optimizations against each other and the results from the previous optimization, using pairwise comparisons. As shown in the table in Figure 4.19, the optimized heightmap did not change significantly when new points of view were incorporated: the image pixel difference between the results differed by less than 2.1%. For this reason, we determined that using a single top viewpoint in the visual difference functional is an adequate choice: this agrees with the intuition that views of a texture from different directions are highly correlated from a broad range of angles.

4.5.3 Visual Experiments

In a set of visual psychophysics user studies, we tested the accuracy of our visual optimization by comparing the source texture appearance to the optimized texture appearance and a simple baseline method. For the baseline, we used a version of the texture scaled linearly in the direction perpendicular to the surface to achieve the same roughness. We also tested the validity of our single-view formulation by comparing it with a more expensive multiple-view formulation.

4.5.3.1 Stimuli

Six source textures were tested (shown in Figure 4.20). Source textures comprised different types of natural and manufactured textures and had different base tactile roughness values.

For each source texture, four target roughness values were selected, and textures were optimized to achieve those four roughness values. Additionally, successive linear scaling was used to

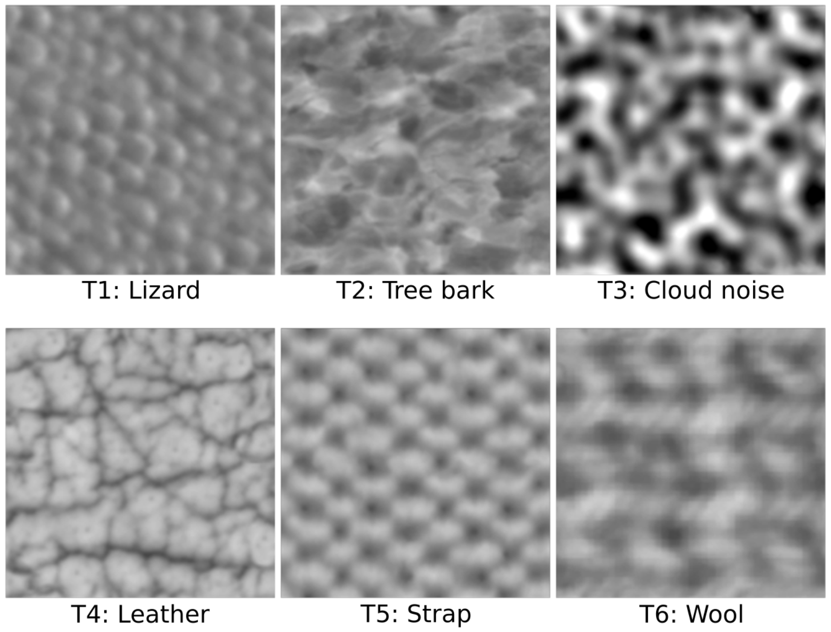


Figure 4.20: Heightmaps of the six textures used in visual experiments.

create alternate textures with the same four target roughnesses.

For each source textures, a set of eleven 25 mm square textured stimuli plates were 3D-printed using a B9Creator DLP stereolithography printer, at 50 μm resolution. Three of the textures were derived from different patches of the source textures; four were the optimized textures; and four were linearly scaled textures.

As we have used B9 Black resin to yield the most accurate geometric results, to improve visibility, textures were spray-painted with matte gray primer (Rust-Oleum Flat Gray Primer) and a coat of clear matte varnish.

4.5.3.2 Experiments

In each trial, two textures were placed in a case that slides beneath a circular window, through which one of the textures could be seen. Observers viewed the textures overhead at a distance of 40 cm, viewing through a mirror placed at an angle of 45° as seen in Figure 4.21.

During each trial, observers were presented with two different textured surfaces sequentially.

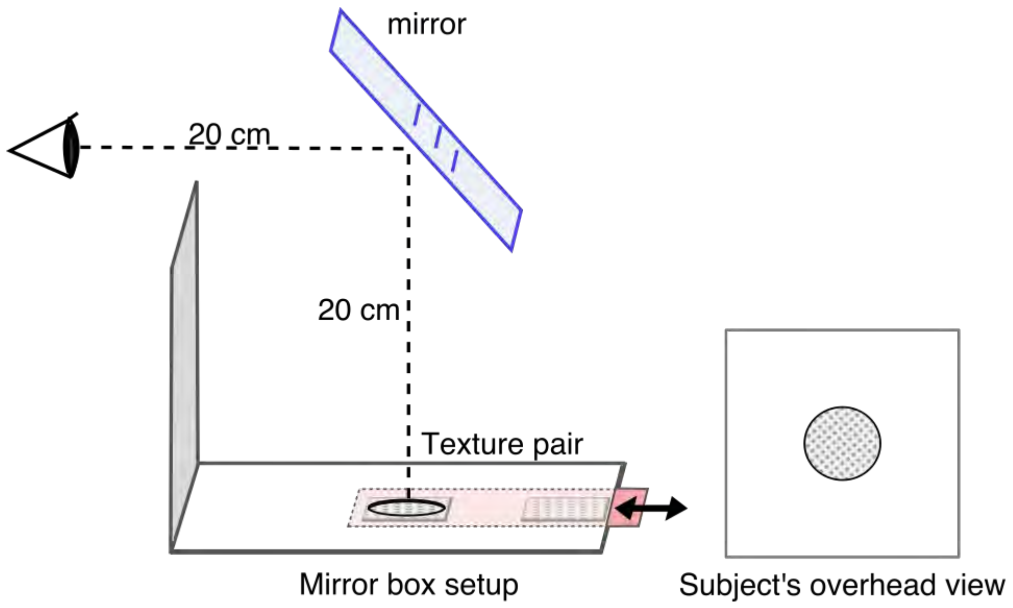


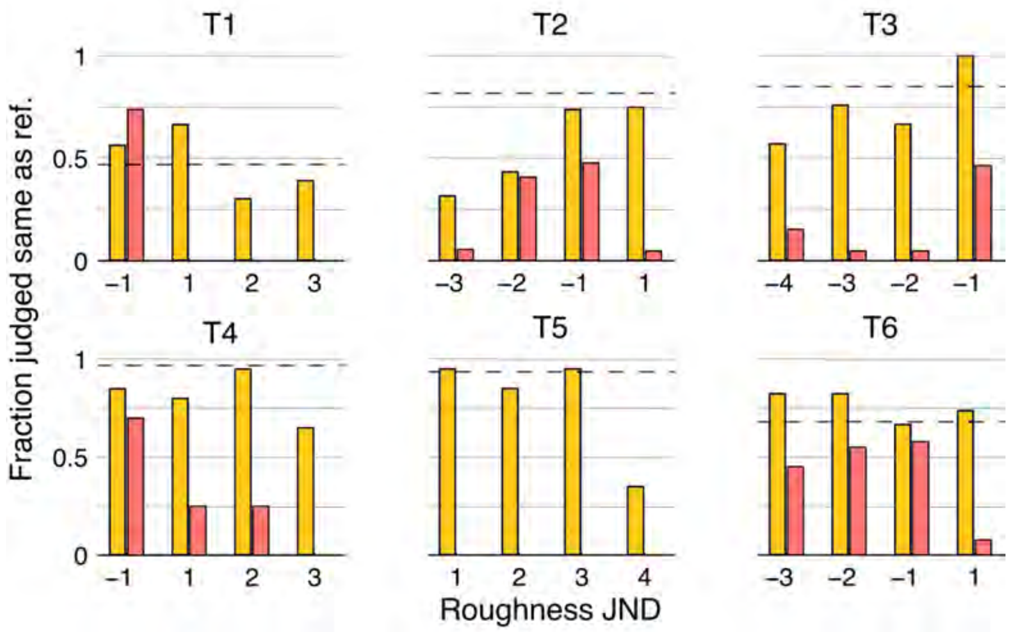
Figure 4.21: The experimental setup for visual experiments, which allows the subject to comfortably view the three trial textures from an overhead view.

One of the pair was derived from the original source texture, and the other could be either another patch of the source texture; a version scaled to a different roughness using linear scaling; or a version scaled to a different roughness using our optimization process. Observers were tasked to choose whether the two textures appeared the same (i.e., derived from the same texture source) or different. Locations of the pair of textures were switched with equal probability. Observers were given 4 seconds to view the textures two times each.

Trials were presented in a pseudorandom ordering, with the constraint that trials using the same source texture were separated by at least two trials.

Six subjects took part in the experiments and performed four repetitions per texture pair. Experiments took place in an office setting with ambient fluorescent lighting.

Experiment results. Our experiment results showed that the optimization process performed substantially better than linear scaling. Figure 4.22 shows the proportions for the 48 test textures. The dotted black line on each graph shows the threshold at which subjects judged the reference textures from the same patch as similar to each other. Of the 24 optimized textures tested, 20



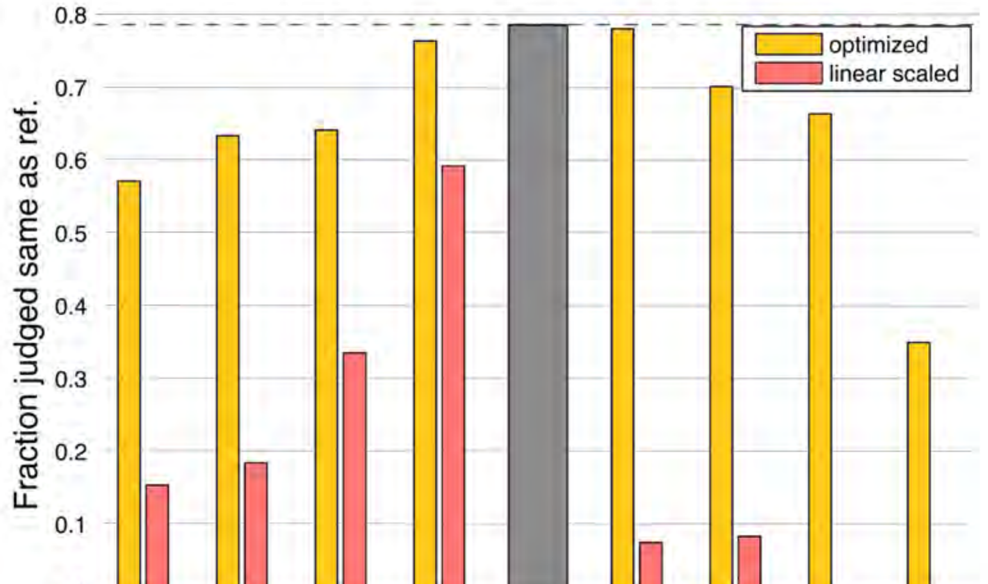


Figure 4.22: Top panel: results from the experiments for each of our six textures. Bottom panel: Proportions for all textures accumulated by JND distance from the reference texture. The x-axis for each graph shows the distance in just-noticeable-differences in roughness values between the test texture and the reference, and the y-axis shows the proportion judged the same. The dotted black line shows the reference threshold at which the reference textures were judged the same as each other.

of them were judged the same as the source at least 50% of the time. In contrast, only 4 of the linearly scaled textures were judged the same as the source at least 50% of the time. In fact, half of the linearly scaled textures were judged different from the reference textures over 90% of the time throughout all trials. 23 of 24 of the optimized textures were judged more similar than the non-optimized version. The other one was derived from T1, a texture which was had high sensitivity to small changes, as shown by the fact that only 50% of textures from the same source were judged the same.

The bottom panel of Figure 4.22 shows textures accumulated by JND threshold distance from the reference texture, according to the difference threshold of 19% found in [185]. In all cases, the optimized textures match the references better than the linearly scaled textures. In general, linear scaling tends to perform more successfully for small decreases in roughness, but performs poorly for larger decreases or increases in roughness. Optimization creates textures that appear very similar for small differences in roughness; the visual difference is only visible when the target roughness is much larger.

4.5.3.3 Alternate points of view

To validate the visual similarity for different points of view and complex geometry, a subset of three textures was chosen for a less restricted version of the experiments. In these experiments, three texture heightmaps (T4, T5, T6) were used to fabricate a new set of textured objects, whose curved geometry includes a local maximum and saddle (Figure 4.23). As in the previous experiment, the reference stimulus was the source texture, and the test stimuli included a source texture along with three optimized textures and three linearly-scaled textures of different roughness values. The textures on the test stimuli were shifted by 50% so as to not appear identical to the reference source texture. Protocols were similar to the previous experiment: the reference and test stimulus were displayed sequentially for one second each, and the participant was asked whether the two plates had the same or different textures. The participant sat around 30 cm from

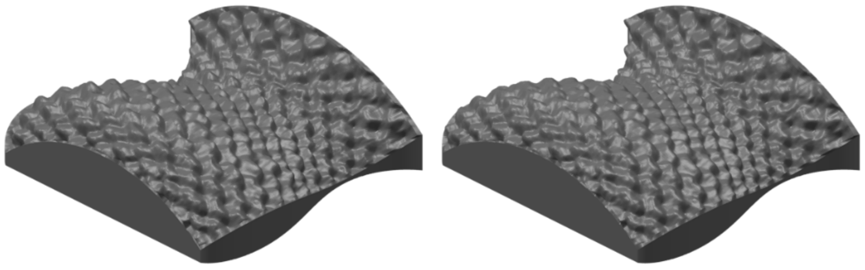


Figure 4.23: A shape with curvature, including a local maximum and a saddle, was used for a less-restricted visual experiment. A rendering of the textured shape with the T5 input texture and a two-JND rougher optimized output texture is shown here.

the textures on the table, and was free to move their head and rotate the textures; in combination with the shape's curvature, the viewer was able to see the texture geometry from many directions.

Eight people participated in the study, and each performed two evaluations of each texture against the source. As shown in the results in Figure 4.24, participants judged the optimized textures the same as the reference a majority of the time, but the linearly-scaled textures were almost never judged the same. The linearly-scaled textures were judged the same as the reference at a lower rate than the previous experiment as a result of the new viewing angles, suggesting differences are more apparent when many view directions are allowed; in contrast, the optimized textures were judged the same at a rate similar to the previous experiments, showing that our optimization is robust to different viewing directions.

The bottom panel of Figure 4.22 shows textures accumulated by just-noticeable-difference (JND) threshold distance from the reference texture, according to the difference threshold of 19% found in [185]. In all cases, the optimized textures match the references better than the linearly scaled textures. In general, linear scaling tends to perform more successfully for small decreases in roughness, but performs poorly for larger decreases or increases in roughness. Optimization creates textures that appear very similar for small differences in roughness; the visual difference is only visible when the target roughness is much larger.

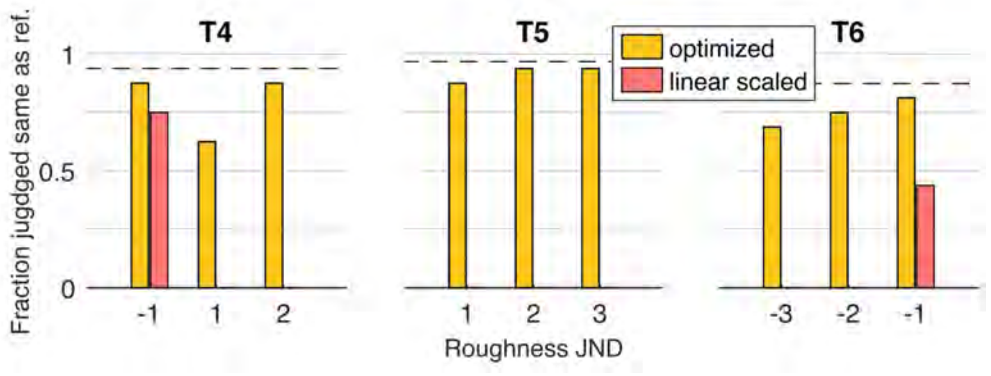


Figure 4.24: Texture similarity results from the experiments on a subset of three textured shapes with curved geometry. The x-axis shows the difference in roughness just-noticeable-difference intervals between the test texture and the reference, and the y-axis shows the proportion judged the same. The dotted line shows the reference threshold a which the reference textures were judged the same as each other.

4.5.4 Tactile roughness experiments

Tactile roughness experiments were used to validate the tactile roughness optimization. Stimuli for this experiment were the same six sets of five 3D-printed texture plates used in the first visual experiments.

In the tactile experiments, ten participants were asked to sort groups of five texture plates by touch from smoothest to roughest. In each trial, the five plates were placed in a random order beneath a translucent panel that obscured the textures' fine-scale appearance. Participants used their dominant index finger to press each plate and determine a sorted order. They were free to feel the plates multiple times and to use as much time as needed.

4.5.4.1 Results

The heat map in Figure 4.25 shows the mean proportion with which each texture plate was judged rougher than each other of the same texture source. Textures are numbered from 1 to 5 according to the designed JND level from smoothest to roughest. Participants were able to reliably sort the plates, including pairs differing by only one threshold, a majority of the time.

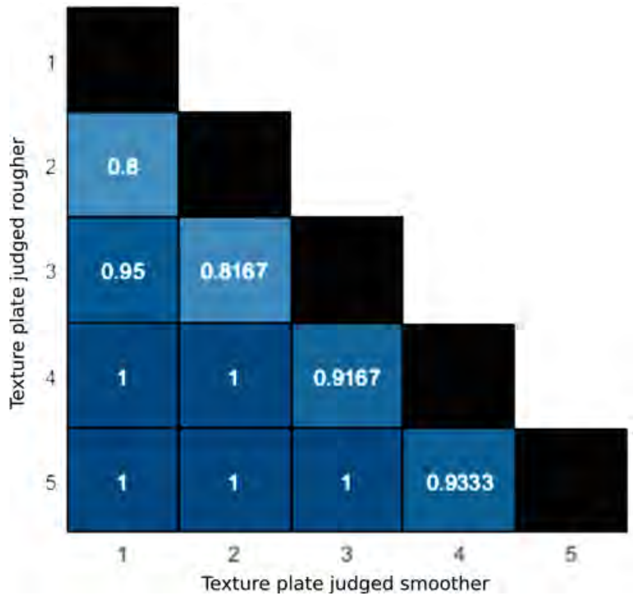


Figure 4.25: This map shows the average proportion of trials in which each texture (vertical) was sorted as tactually rougher than each other texture (horizontal). Almost all discrepancies were between textures designed to differ by one threshold, and the error rate is close to the expected 84%.

Participants sorted these most-similar pairs according to the designed ordering 85% of the time (across-subject standard deviation 4%), which is nearly the expected threshold of 84% with which consecutive plates were designed. Only three of the 60 total comparisons resulted in an ordering discrepancy between a pair of textures differing by more than one threshold step.

4.5.5 Contact area experiments

Twelve textures were fabricated to experimentally verify the change in contact area. For each texture, three versions were fabricated: the original texture, a texture optimized to have 70% the contact area, and a texture optimized to 140% the contact area. These 36 textures were fabricated as 10mm squares, using B9Creator V1.2 with a resolution of 50 µm in B9 black resin.

To compute the contact area, the fabricated texture surfaces were coated in ink using a compliant sponge and an ink-pad. The thumb or second finger of each participant was covered with Tegaderm (3M), a thin layer of transparent plastic with a thickness of 0.1 mm. The tegaderm was

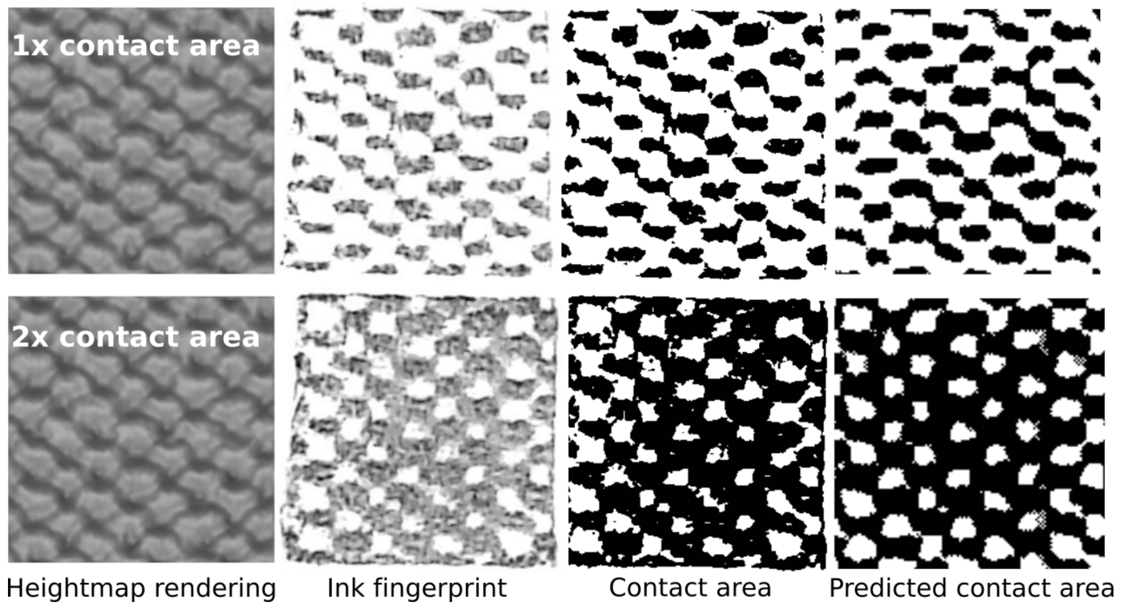


Figure 4.26: The results of one texture optimized for a lower (top) or higher (bottom) contact area. From left to right: texture rendering, texture fingerprint, thresholded finger contact, and simulated contact.

used to avoid discrepancies due to the fingerprint ridges, and to provide easier cleaning of the finger surface between trials to avoid ink residue. The finger was pressed against the texture with a weight of 8.8 N placed on the finger to ensure uniform force. Then the finger was pressed to a sheet of paper to derive an inkprint of the contact surface.

Nine participants provided texture finger prints in this manner. The prints were scanned at 600dpi in 8-bit grayscale, and the contact areas were computed and averaged over all subjects. The pipeline is shown in Figure 4.26. All optimized contact surfaces fell within 20% of their target contact area, with an average difference of 9.2%. Additionally, the contact areas of the 36 printed textures were compared against the simulated contact areas. Figure 4.27, shows the result of this comparison, with a close linear correlation with an approximate slope of 1.

4.5.5.1 Temperature Experiments

An additional set of experiments was used to determine whether the printed textures felt different from one another in tactile temperature.

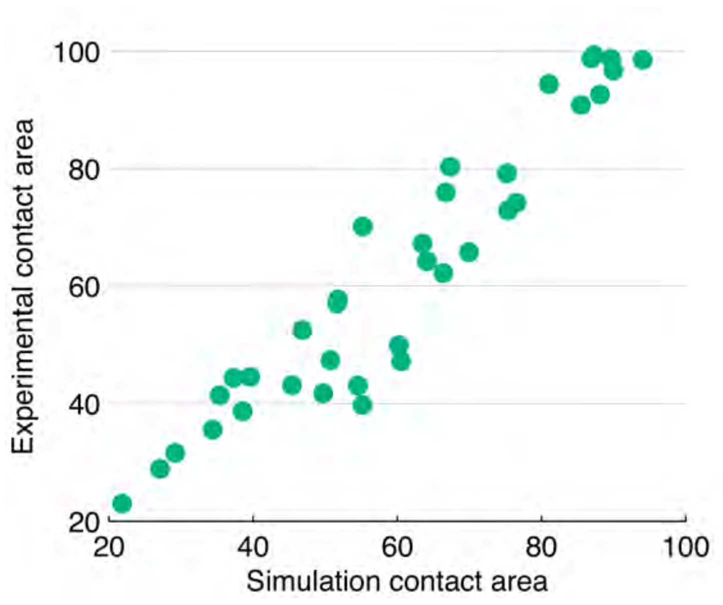


Figure 4.27: Comparison of the experimental and simulated contact area of 36 textures.

Stimuli. The stimuli for this experiment were twelve textures: four base textures (T2, T3, T4, T5) each optimized with three different contact areas differing by 40%. Each texture surface was applied to the top of a flat plate 1.2 mm in height. To enable tactile discriminability at room temperature for the purposes of the experiment, we used metal rather than plastic, due to its higher thermal conductivity: texture models were 3D printed and cast in bronze. A photograph is shown in Figure 4.28.

Setup and protocols. In the experiments, textures placed on a flat cast-iron plate over ice, which maintained a temperature at the top surface of approximately 16°C as measured by a laser thermometer.

In each trial, the participant was presented with two textures of the same class with different optimized contact area (either 40% smaller or 40% greater, where 40% is approximately the JND threshold for thermal discrimination described by [179]). A cover was placed over the experiment area to hide the textures from view.

In experiments, eight participants were asked to feel the two textures using static pressure with the index finger and to answer which texture felt colder. Participants were allowed as much

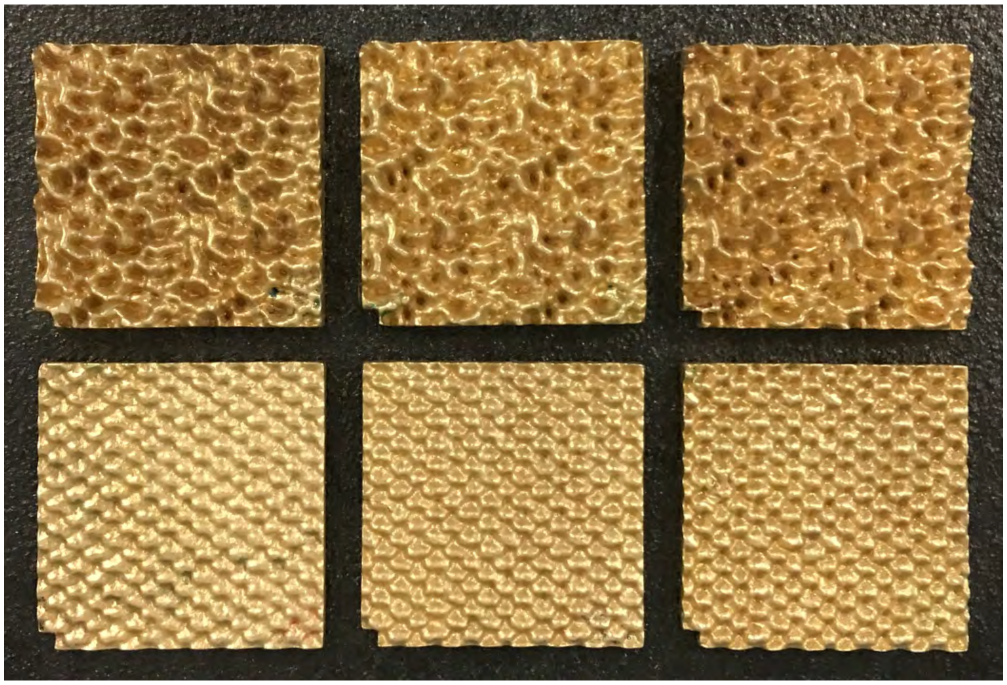


Figure 4.28: Photograph of sets of bronze-cast textures used for tactile temperature experiments (T3 and T5). Textures are ordered from less to more contact area. Inconsistencies in appearance may be due to the manufacturing process and polishing.

time as needed to feel the textures, and were given time between trials to ensure the finger itself was not too cold.

Results. Throughout the trials, participants responded that the texture with more contact area felt colder 83.1% of the time, which suggests that the threshold of discrimination is indeed approximately 40%, as found by previous research. For pairs that differed by two JND, the texture with more contact area was judged as colder 91% of the time.

4.6 Applications

Applying tactile textures to fabricated objects is useful for both aesthetic and practical purposes. We have formulated several examples and have fabricated a subset as textured 3D models (Figure 4.29).

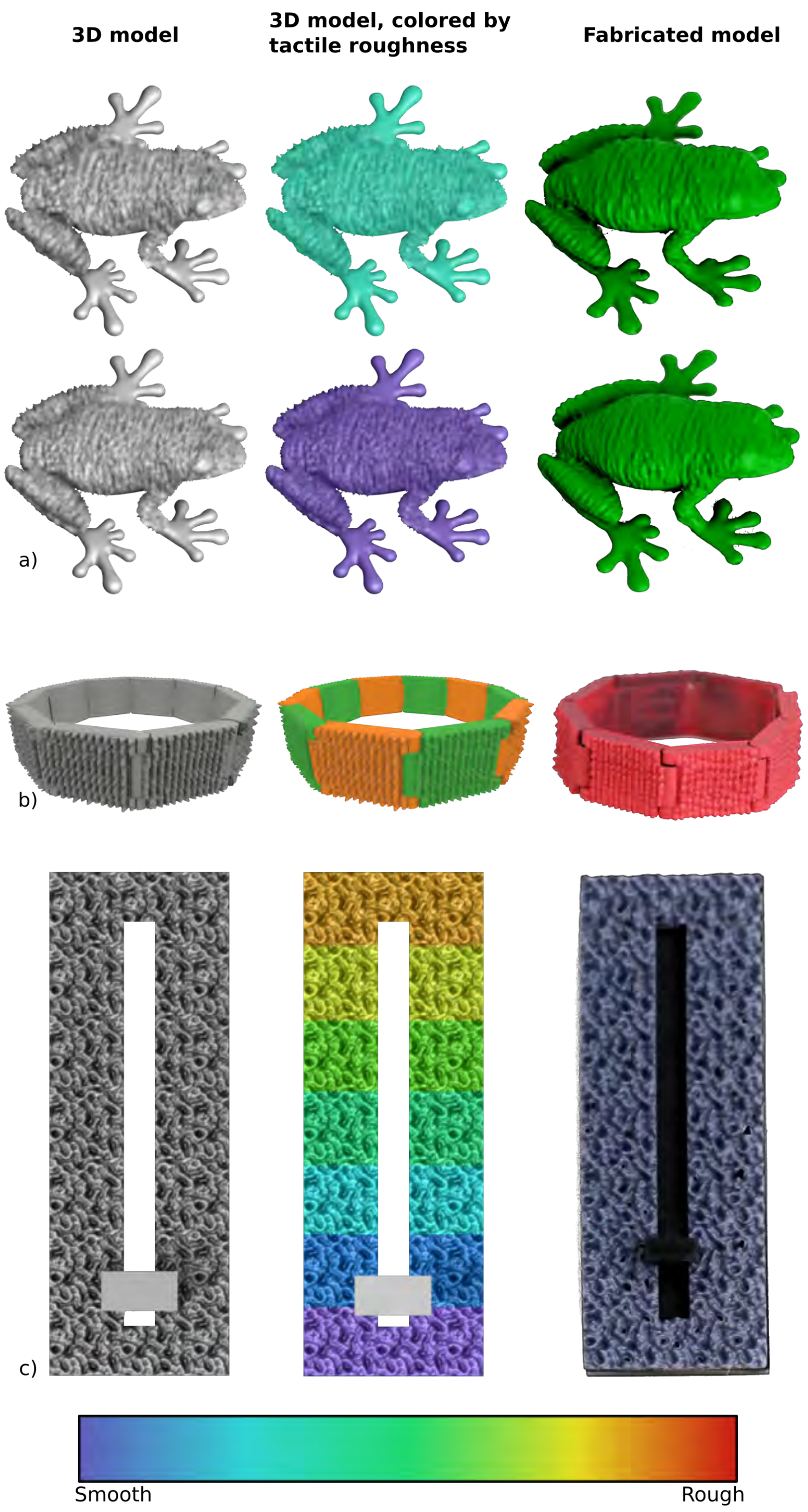


Figure 4.29: From left to right: textured models; models colored by roughness; photographs of 3D-printed models. a) Two frogs textured with a tactile wood texture optimized for different roughnesses. (Modified from [209]). b) A bracelet with textured links that have the same visual texture but have alternating roughnesses. c) A procedural texture slider for a light switch, where the tactile roughness corresponds to light intensity.

Modeling. Often, one might prefer a particular visual texture for an object while preferring a distinct tactile feeling. For example, imitation plastics are often used to match a specific material's appearance, and our model could help match the material's desired feeling. Our model could enable the creation of multiple surfaces that look similar but feel different, either for aesthetic purposes or to serve as a tactile signifier of another characteristic. It could also be used to make surfaces that feel similar but look different, which could be combined in a visual pattern or logo, for example on a mat, that feels uniform when touched.

We manufactured two different animal models as examples. First, a starfish model was textured with a relatively smooth surface texture (roughness 0.05). The texture was altered to feel rougher (roughness 0.092), and was applied to produce another, rougher fabricated starfish with the same appearance (shown in Figure 4.1). We also fabricated a textured model of a tree frog with a keeled wood pattern. The initial texture had a roughness of 0.045, and was modified to produce a smoother texture of roughness 0.03 and used to fabricate a smoother frog with the same appearance (Figure 4.29a).

Wearables. Tactile and visual aesthetics are common to clothing, jewelry, and other wearables, which often touch the skin. Tactile properties may also serve as functional. Some wearable devices, such as headphones with buttons, have areas that the user finds and uses by touch rather than sight; these areas could be hidden visually for aesthetic appearance or for more discreet use. Wearables could also use roughness actively to convey haptic signals that are unobtrusive to the user and invisible to others: a watchband with a high-resolution pin array could produce different tactile textures that could be felt by the user to convey different signals; similarly, altering the contact area could allow different rates of thermal transfer between a wearable and the user's skin.

As an example of a wearable with tactile aesthetics, we fabricated a bracelet band with links having the same texture appearance, but alternating smoother and rougher tactile feelings (Figure 4.29b). Smoother links had a roughness of 0.06, and the rougher links had roughness 0.09.

Accessibility. Tactile items and textures are particularly useful for people with visual impairments. If a designer creates two objects that look different, our model could be used to tune the textures so that they also feel different, while preserving the designed visual appearance. Visual textured objects are commonly used in pieces for board games, puzzles, and household items, where colors or visual labels are often used to distinguish between otherwise similar objects or regions. A variation in tactile feeling can provide similar cues for a person unable to see the differences. As an example, we produced a model for a dimmer light switch slider. The texture gradient looks the same throughout, but the roughness increases such that it will correspond with the light intensity as the slider is moved (Figure 4.29c).

4.7 Conclusion

We have presented an optimization procedure to preserve texture appearance while altering tactile roughness or temperature. We used neural networks to enable computation of tactile roughness, contact area, and visual appearance at speeds several orders of magnitude faster than the standard methods, providing differentiable functions usable in optimization for a target appearance and feeling. We used psychophysical experiments to demonstrate that our method provides a significant improvement over simple linear scaling in controlling tactile roughness, and we provided several examples of how our procedure can be used to produce interesting and useful textured objects.

4.8 Limitations and Future Work

While the tactile model has been tested on objects with moderate curvature ([183]), it may not be usable for high-curvature 3D objects. Furthermore it was tuned for hard materials, and it has a minimum feature resolution; using it for soft or fine materials may require changes. Our static touch simulation is adequate for dynamic touch up to a certain resolution, as static touch receptors dominate perception for features over 100μm ([73]); nevertheless it may be improved by dynamic simulation. The model is also based on a simulation of a simplified model of human skin, which, while found to be robust, may be improved by a more complex model. Our procedure could used for a different material or more physically complex skin structure by retraining the neural network on a new set of simulation field outputs.

Similarly, our procedure describing visual appearance was tuned for the shading of the our material (diffuse plastic resin) and may not be directly usable for surfaces that are much more glossy, translucent, or non-smooth. In these cases, our method could be modified to learn the rendered appearance for a particular desired material given a suitable training set. However, as seen in Figure 4.28, our current visual model can still work to preserve visual appearance fairly well even for non-matte materials.

Our model presents a tradeoff between preserving exact visual appearance and achieving an exact tactile roughness. Very high changes in tactile roughness may not be achievable while fully preserving visual appearance. We found that similarly-appearing textures can typically be produced within a range of 3-4 JND thresholds in each direction. Our metric for visual appearance similarity is likely a lower bound for perceptual similarity, so a fast perceptually-based method for texture similarity could be used instead in the optimization and could improve texture generation. Our model was evaluated to target either a tactile roughness or contact area; optimizing for both or more quantities is future work. Other optimization parameters could also be used: for example, we could enforce printability constraints depending on the printer used to manufacture a model.

Our visual model uses shading from an overhead view with ambient lighting. At severe angles or severe lighting conditions, the differences may be more apparent. Our model could be tuned to a particular lighting condition or viewpoint if it were used in the training set, but any optimized texture likely will not appear exactly the same under all lighting and viewing conditions, as some geometric changes will always be present near the surface. However, as we found in both optimization tests and human user studies with curved objects, our optimization using a single viewpoint is robust, and the results look similar to the target even when viewed at other angles.

Our model limits texture height to 3 mm as a manufacturing constraint. We have observed that due to the limited elasticity of the finger, textures deeper than this are not different tactually from those with the lower-depth truncated to 3 mm. However, we could easily optimize a taller texture by optimizing the top 3mm of it, and preserving the remainder.

To aid in the fabrication process, our method could be integrated into a 3D modeling tool to provide precise control of tactile feeling when modeling a textured fabricable object. Using our model and existing models for compliance using compliant microstructures, it would be possible to control three of major dimensions of touch: compliance, temperature, and roughness, and to study the unknown interactions between these properties. Creating objects with differing tactile properties as separate from appearance may also be of interest to the fields of neuroscience and neurophysiology in future studies of psychophysics and multi-modal perception.

5 | Rational MRI Coil Design: An Optimization Framework for the Design of Radiofrequency Coils for Magnetic Resonance Imaging

This chapter is adapted from a preprint [167], a joint work with José E. Cruz Serrallés, Ilias I. Giannakopoulos, Damien Chen, Daniel Zint, Daniele Panozzo, Denis Zorin and Riccardo Lattanzi.

ABSTRACT

The radiative characteristics of the radiofrequency receive coils dictate the signal-to-noise ratio (SNR) of magnetic resonance images. Despite the crucial importance of RF coils, the practical coil design process has remained a largely empirical one. This work introduces a novel optimization framework for rational coil design, which relies on a fully automated pipeline that combines rapid electromagnetic simulations, shape optimization and coil meshing. The objective function iteratively maximizes SNR performance in a target region of interest with respect to the ultimate intrinsic SNR, which is the theoretically highest SNR independent from any particular coil design. The forward simulation employs a fast electromagnetic solver based on coupled surface

and volume integral equations. The coils are represented as B-spline curves with an associated width, and automatically meshed for EM simulation. We implemented a new method to tune and decouple coils at each iteration without manual user intervention. The algorithm optimizes the size and position of a given number of coils with a combination of grid search and a line search. We demonstrated the framework by designing receive arrays of increasing complexity that yield optimal SNR for different target regions inside a numerical head model. SNR simulation time ranged from 15 s for a 3-coil configuration to 32 s for a 12-coil array, constrained to a helmet-like surface, including tuning and decoupling. The optimized 12-coil geometry yielded 9% higher average SNR performance in the brain at 3 T. This work represents the first automated coil optimization framework that uses full-wave electromagnetic simulations and ultimate performance benchmarks. This novel approach enables the systematic design of coils for magnetic resonance imaging with significantly improved SNR performance, potentially transforming coil development from empirical design to physics-driven optimization.

5.1 Introduction

Radiofrequency (RF) coils are at the heart of any magnetic resonance (MR) imaging application. They are the source of the RF fields that generate MR signals and the mechanism by which RF fields generated by excited nuclear spins are detected. RF coil design is therefore a critical determinant of the performance of all MR imaging (MRI) systems. As the number of channels available in MR systems has increased to enable faster acquisitions with parallel MRI [93, 162, 205], building prototypes of coil arrays has become more difficult and expensive; as a consequence, coil design has relied ever more on electromagnetic (EM) simulations. While careful coil design is important at all field strengths, appropriate coil designs are truly essential for high-field MRI, both for the preservation/improvement of image quality and for the avoidance of adverse effects in patients. This work aims to introduce a novel optimization framework for rational RF coil design.

For receive arrays, the number, geometry, and arrangement of each coil element, as well as interelement decoupling, coil loading, and parallel imaging performance, are important design parameters that can impact the overall signal-to-noise ratio (SNR) [93, 162, 205]. However, current approaches to coil design (Section 5.2.1 are limited by time-consuming simulations and a lack of systematic, automated tools to explore complex design spaces.

This previous work does not address another major limitation of the current approach to coil design: the quality of a coil is typically judged in comparison to other available coils, giving no indication of whether there is room for further improvement beyond the best-performing design tested. To address this, theoretical coil performance limits [50, 100, 102, 109], such as the ultimate intrinsic SNR (UISNR), could be used as absolute references during coil design [57, 100, 101, 103, 187]. The UISNR is the highest possible SNR compatible with electrodynamics and independent from any particular coil design [109, 138, 139, 204]. The UISNR represents the highest SNR allowed by electrodynamics, independent of any specific coil configuration. Prior studies (Section **??** showed that even state-of-the-art coils capture only a fraction of this limit in cortical regions, especially at ultra-high field strengths, leaving substantial room for improvement. This suggests the need for alternative design strategies that aim to approach ultimate performance.

UISNR integration into the coil design process has remained an open problem. To address this, we employed the MRGF concept to develop a novel shape optimization technique that automatically enhances coil configurations based on ultimate performance benchmarks. Given an existing coil configuration, we compute its variations for a set of design parameters to find a design that improves coil performance. This approach has been highly successful in material design, structural mechanics, and fluid dynamics, but has yet to be effectively applied to MRI coil design [66, 107, 108, 136, 212].

A key challenge is the need for an unconditionally robust simulation and modeling pipeline. At each step of the optimization, a new coil geometry must be generated, meshed, tuned, and simulated with no human intervention – a task currently infeasible with standard MRI simulation tools, in which users routinely spend days setting up a single simulation [216].

We propose an integrated approach to tackle this problem, which jointly considers forward simulation, shape computation, and coil meshing. Together with a new framework for rational coil design based on shape optimization, this work introduces several innovations: an accurate and fast solver for the surface integral equation, a method to automatically tune RF coils, an approach to mimic preamplifier decoupling in coil simulations, and a method for automatic meshing of coil geometries.

We demonstrate the use of our optimization framework through numerical examples of increasing complexity, from single-coil positioning, to shape optimization of small arrays, to large-array performance optimization toward the UISNR. We also show that the optimization results are consistent for different anatomical models.

The rest of the manuscript is organized as follows. In Section 5.2, we present a brief overview of previous work that is relevant to this paper. In Section 5.3, we specify the constraints of the design space for our proposed coil optimization. In Section 5.4, we describe our approach to efficiently model RF coils, whereas in Section 5.5, we describe the new SIE solver and the proposed method for automatic coil tuning and ideal decoupling. In Section 5.6, we introduce the coil design optimization algorithm and in Section 5.7, we provide details of the numerical experiments performed in this study to demonstrate it. The results are presented in Section 5.8 and discussed in Section 5.9, whereas Section 5.10 summarizes the main points of this work.

5.2 Related Work

In this section we briefly review related work in MRI coil optimization and shape optimization. Some more in-depth discussion of most closely related numerical methods can be found in Section 5.5.1.4.

5.2.1 Coil design optimization

Initial work on coil design optimization was performed in the quasi-static regime due to lower field strengths and hence lower operating frequencies [47, 98, 156, 182, 206]. Analytical methods, such as spherical harmonics [156] and cylindrical harmonics (Fourier-Bessel series) [182], were employed for the design of receive coils, gradient coils, and solenoidal coils. Analytical methods were popular because their corresponding basis and testing functions yield diagonalized systems, which are easily invertible. Later on, the quasi-static Biot-Savart Law was used explicitly to design gradient and RF coils, along with an error function that was minimized using conjugate gradient descent [206]. Further work involved variations of these basic approaches, such as randomized Monte Carlo sampling to match desired spherical harmonics [98], applications of the Biot-Savart Law to obtain desired field patterns (inverse design), and full-wave simulation using thin-wire approximations [47], among others. Quasi-static coil design methods have remained popular tools, especially for gradient coils, since these operate at much lower frequencies than RF coils. More recent examples of quasi-static coil optimization include parametric optimization of 1- and 2-loop arrays using the Biot-Savart law [67], boundary element method-based field profile matching with ohmic loss penalties for gradient coil design [147], and multi-objective field matching for gradient and shim coil design [148], among others [207].

Biot-Savart and quasi-static harmonic solutions are no longer accurate at MRI field strengths higher than 1.5 T. While Jefimenko's Equation [81] offers a full-wave equivalent of the Biot-Savart Law, modeling using this equation can be difficult due to the strong $\frac{1}{R^3}$ singularity in the Green's function and requires making approximations that might not be valid, such as uniform current distribution in a loop. Early attempts at coil optimization in the full-wave regime include the application of the Helmholtz Green's function with so-called stream functions to design a loop at 4.5 T [106], the design of a body coil by simulating first with Biot-Savart and then with Method of Moments discretization of the EFIE formulation while optimizing with a genetic algorithms metrics such as slice B_1 homogeneity [211], and the application of a genetic algorithm to body coil design using a method-of-moments approach [155]. More recent examples in the full-wave regime include an inverse design approach for the design of a uniplanar RF coil while optimizing for RF field homogeneity [69] and parametric coil optimization for the design of RF transmit arrays while optimizing for slice homogeneity [168].

Another work solved an optimization problem to find the current density that maximized SNR for parallel imaging applications [135]. All these works focused on the design of one- or two-element arrays, rather than dense arrays, and used simple geometries to mimic the anatomy. Genetic algorithms have also been proposed to simultaneously optimize multiple design parameters for birdcage coils or two-element arrays, also in this case using uniform objects with simple geometries [66, 136, 212]. The main limitation of these previous studies was the time-consuming EM simulations that prevented the implementation of iterative optimization algorithms to design many-element arrays using realistic anatomical models.

The Magnetic Resonance Green Function (MRGF) method, based on a fast EM solver tailored to MRI applications, was more recently proposed to simulate in minutes the EM field inside a realistic anatomical model for any RF coil designed over a specified substrate for which the MRGF had been precomputed [190]. A proof-of-concept study suggested that the MRGF method could be employed to iteratively optimize size and position of two transmit coils to maximize magnetic field homogeneity inside a numerical human head model [168]. No other attempts have been made at using MRGF to automatically optimize the design of arrays with a larger number of coil elements.

The UISNR mentioned in the intro provides an absolute benchmark for coil design, representing the highest possible SNR compatible with electrodynamics, and was considered in a few previous works [100, 102, 109, 138, 139, 204]. It is computed using a complete EM basis to simulate idealized, infinite arrays. Prior studies have shown that conventional coils achieve only a fraction of the UISNR, particularly in superficial cortical regions and at higher field strengths [50, 57, 101, 103, 187]. This underscores the significant potential for improvement. More recently, UISNR has been used to directly inform coil design and optimization pipelines [99, 104, 192, 216], motivating the approach taken in this work.

5.2.2 Shape optimization

There is an extensive literature on shape optimization in a variety of contexts, ranging from fluid dynamics to electromagnetic modeling and metamaterial design. The general mathematical foundations of PDE-constrained shape/topology optimization can be found in classical references [4, 12, 13, 133].

Most of these work focus on optimization of surfaces bounding 3D domains or planar curves bounding 2D domains in the context of fluid or heat flows or elasticity. A recent survey [124] focuses on shape optimization in the context of electromagnetic modeling. POE-constrained shape and topology is widely used for metamaterial design in the context of elasticity [15, 141], electromagnetics and optics [34, 116]. PDE-constrained optimization of curves *on surfaces* of the type we perform in our work is less common. Most of the work is related to either curve smoothing/-fitting to data e.g., adaptation of active contours to surfaces [16] and other smoothing methods [71, 85, 105]. Our approach to avoiding overlaps in coils, based on the contact potential introduced in [114] and extended to codimensional objects in [113] is related to the potential-based repulsive curves approach considered in [214]. A constrained-optimization based approach to self-avoiding curves is applied to polymer adsorption on surfaces in [193].

5.3 Coil Design Constraints

The coil optimization pipeline must satisfy specific constraints to ensure that the array can be constructed as designed and utilized effectively for MRI. The first constraint is that a coil cannot intersect itself, and no two coils should touch each other. Otherwise, the current patterns would

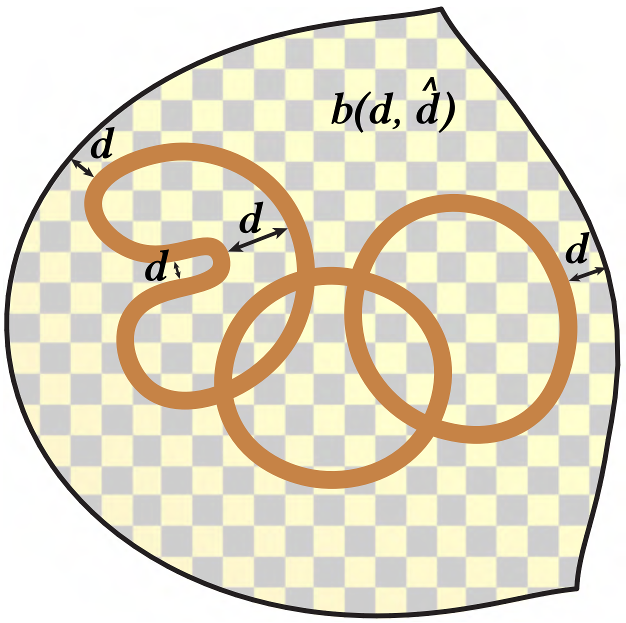


Figure 5.1: Coils are constrained so that they do not self intersect nor go outside the domain of the substrate. We implemented this by adding a barrier energy term $b(d, \hat{d})$, which vanishes if the distance d between contact pairs is larger than a predefined $\hat{d}$ threshold.

be modified, and for example, two touching coils would effectively become a single, larger coil. In addition, these topology changes could result in sharp changes in the objective function with a negative effect on the optimization algorithm. The second constraint is that the coils cannot be too small, otherwise the SNR would be dominated by electronic noise and the loading of the sample would be poor, resulting in low performance [61]. The third constraint is that the optimized values for lumped elements (capacitance, inductance, and resistance) should fall within a user-defined range that reflects commercially available capacitors. The fourth constraint is that the coils must belong to the parametrization domain of the surface of the substrate (Section 5.4.2).

During the automated design process, the first constraint is guaranteed by selecting a design space that satisfies it by construction. While optimizing the coil geometry, this is achieved by enforcing a barrier potential term, where we consider (1) a coil edge and the parameterization domain boundary edge; (2) edges within one individual coil to be the set of possible contact pairs (see Figure 5.1 and Section 5.6.1). We also introduced bridges (Section 5.4.4) to avoid coils touching each other. Instead of specifying a minimum value for the coil radius, the second constraint is automatically enforced by adding coil noise into the SNR calculation. In this work, we used only capacitors as lumped elements, and we satisfied the third constraint by specifying the minimum and maximum capacitance to be 1 pF and 200 pF, respectively, in the input JSON. These values are read and enforced during tuning and ideal decoupling (Sections 5.5.4.1 and 5.5.4.2). The fourth constraint is enforced by construction during the meshing process (Section 5.4.4).

5.4 Modeling

We represent coils in a parametrized way, suitable for optimization. The coil parametrization is established in two steps. First, we construct a map from a 2D domain to the 3D surface of the coil former (Section 5.4.2). Second, we represent coils as parametric curves with width in the 2D domain (Section 5.4.3). Finally, we discretize the parametrized coil design (Section 5.4.4). To facilitate interactive coil design, a graphical user interface (GUI) was developed (Section 5.4.1).

5.4.1 User interface

The GUI was implemented in TypeScript and based on the Babylon.js framework (Figure 5.2). The GUI allows users to place and adjust control points within a 2D domain, where coil geometries are rendered in real-time as a collection of B-splines. The interface supports intuitive translation, scaling, and duplication of entire coils, individual strips, or single control points. Lumped element components (Section 5.4.3) can be both visualized and parametrically adjusted through dedicated property panels. All modifications to the coil geometries are automatically reflected within the integrated 3D scene, which also allows users to overlay visualizations of volumetric body models that can be loaded as isosurfaces. A 3D cursor tool enables precise selection and transformation of coil elements within the 3D view, ensuring that users can configure coil geometries and their associated lumped element properties in a single, closed environment.

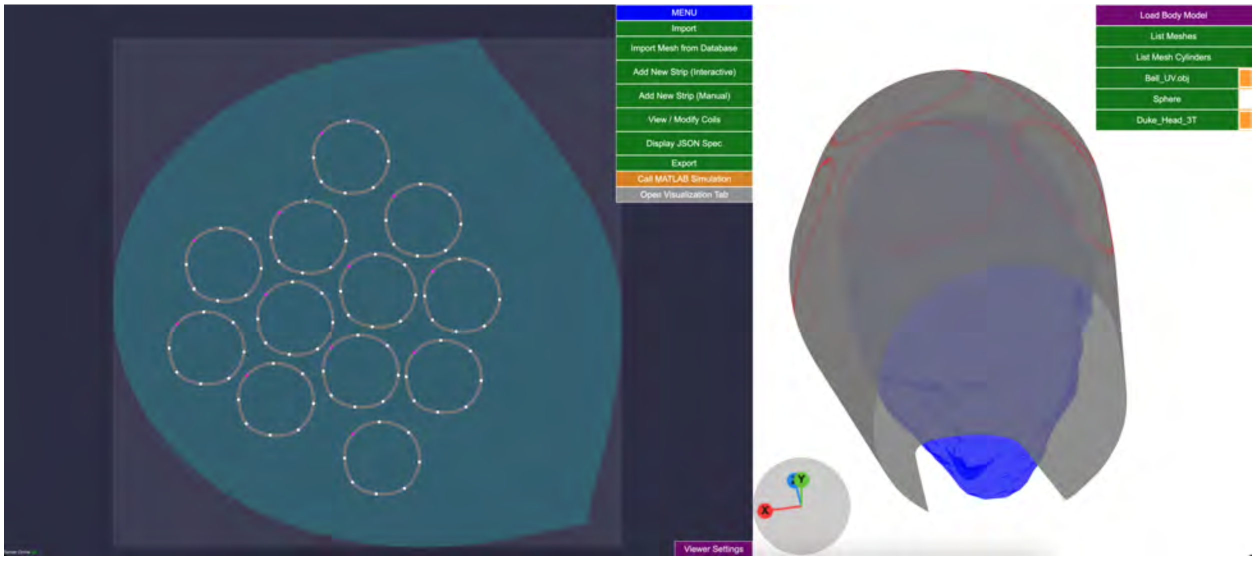


Figure 5.2: The graphical user interface (GUI) to manually design or display coils. The GUI visualizes coils and the substrate using both 2D and 3D interconnected views.

The parametrized coil design (control points, width, lumped elements information etc.) can be exported in a human-readable JSON file. This file is also the input for the optimization pipeline (Section 5.6.2).

5.4.2 Substrate parametrization

We assume that the substrate surface, i.e., the coil former, is given as a triangular surface mesh, which is a collection of triangles embedded in 3 dimensions. The triangular surface mesh is flattened in 2D with the method of Scalable Locally Injective Maps (SLIM) [151]. SLIM minimizes distortion energies with a re-weighting scheme and prevents flipped geometry. Any position $\mathbf{u}$ in 2D can be mapped to a position on the 3D substrate surface with $\Phi(\mathbf{u}) : \mathbb{R}^2 \rightarrow \mathbb{R}^3$, by finding the nearest 2D triangle, computing the position's barycentric coordinates for that triangle, and evaluating the coordinates in the corresponding 3D triangle (Figure 5.3). For efficient evaluation of $\Phi(\mathbf{u})$, we use an axis-aligned bounding box (AABB) tree data structure [14] to find the nearest triangle.

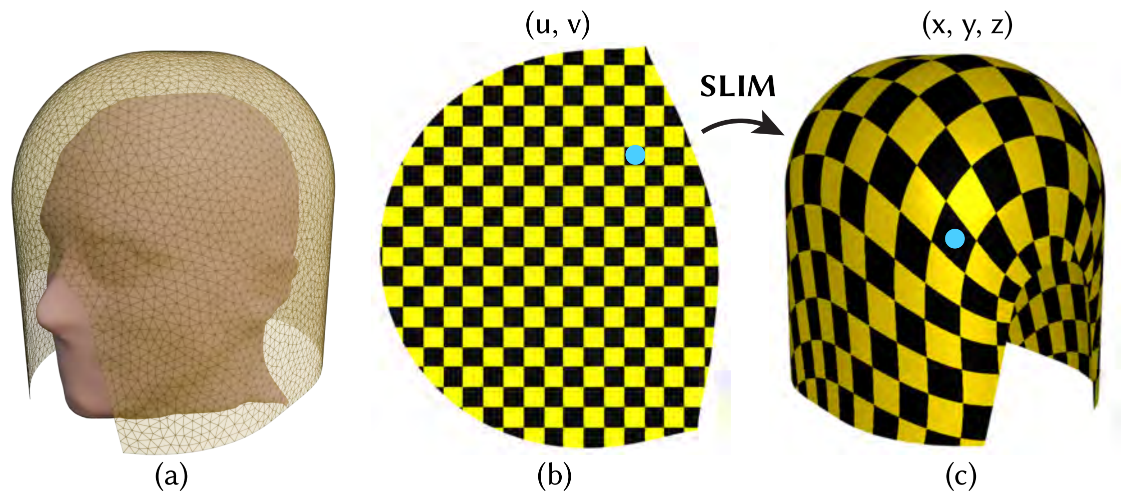


Figure 5.3: (a) Geometry of an example substrate surface mesh outside the head model. (b,c) Scalable Locally Injective Maps (SLIM) [Rabinovich 2016] with symmetric Dirichlet to minimize arbitrary distortion energies.

5.4.3 Coil parametrization

We use B-splines $B(t), t \in (0, 1)$ as parametric curves to represent coils in 2D. Every coil has a width w, such that the entire coil domain in 2D is described as $B(t, s), t \in (0, 1), s \in (-w/2, w/2)$, where ∇s is orthogonal to ∇t. The 3D coil geometry is described by $\Phi(B(t, s))$.

RF coils have electrical components (capacitors, inductors, resistors, or feeding ports) attached to them that also need to be part of the coil parametrization. We represent those components as lumped elements and store their parametric position $t \in [0, 1)$ within the B-spline to which they are attached. Note that lumped elements always span the entire coil width, i.e., $s \in (-w/2, w/2)$.

5.4.4 Meshing

We discretize the coils in 2D with triangle meshes and then map those triangles to the 3D surface (Figure 5.4). The entire meshing pipeline consists of: (1) converting the B-splines into polylines, (2) generating strips from the polylines by offsetting them by $w/2$ in both directions, (3) inserting a line for each lumped element, (4) triangulating, and (5) mapping to 3D.

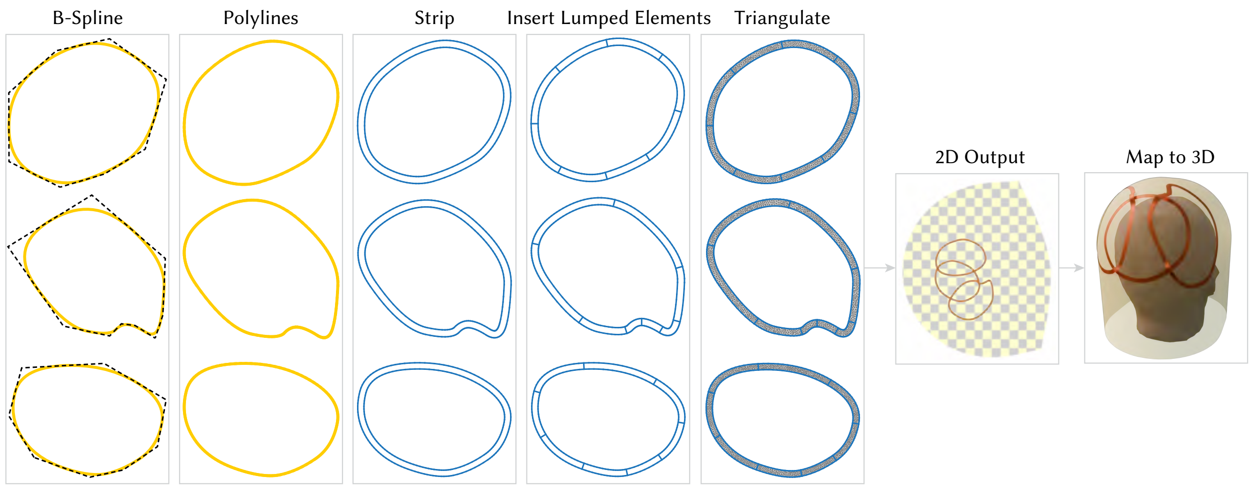


Figure 5.4: Overview of the meshing pipeline.

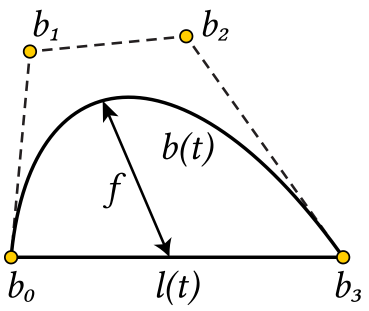


Figure 5.5: Illustration for the flatness evaluation in Equation (5.1), where the flatness f of a cubic Bézier curve $b(t)$ is defined as the maximum distance between $b(t)$ and $l(t)$.

Conversion into polylines. For simpler handling, we convert the B-splines into Bézier curves. We recursively subdivide the Bézier curves at $t = 0.5$ until the flatness of each curve is below a certain threshold τ [45]. For a cubic Bézier curve with four control points $[\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3]$, the flatness of a curve $b(t)$ is defined as

$$f = \max_{0 \le t \le 1} ||b(t) - l(t)||, \tag{5.1}$$

where $l(t)$ is the line segment between the start point $\mathbf{b}_0$ and end point $\mathbf{b}_3$ (see Figure 5.5), which can also be represented as a cubic Bézier curve with control points $[\mathbf{b}_0, (2\mathbf{b}_0 + \mathbf{b}_3)/3, (\mathbf{b}_0 +$

$2\mathbf{b_3})/3, \mathbf{b_3}]$. Hence we have

$$b(t) - l(t) = (1-t)^2 t(3\mathbf{b_1} - 2\mathbf{b_0} - \mathbf{b_3}) + (1-t)t^2(3\mathbf{b_2} - \mathbf{b_0} - 2\mathbf{b_3}). \tag{5.2}$$

For $\xi = (3\mathbf{b_1} - 2\mathbf{b_0} - \mathbf{b_3})$ and $\eta = (3\mathbf{b_2} - \mathbf{b_0} - 2\mathbf{b_3})$, we obtain an upper bound for the flatness,

$$\begin{aligned} f^2 &= \max_{0\le t\le 1} \|b(t) - l(t)\|^2 \\ &= \max_{0\le t\le 1} (1-t)^2 t^2 [((1-t)\xi_x + t\eta_x)^2 + ((1-t)\xi_y + t\eta_y)^2] \\ &\le \frac{1}{16}(\max\{\xi_x^2, \eta_x^2\} + \max\{\xi_y^2, \eta_y^2\}). \end{aligned} \tag{5.3}$$

By recursively subdividing the curves until $\max\{\xi_x^2, \eta_x^2\} + \max\{\xi_y^2, \eta_y^2\} \le 16\tau^2$ is satisfied, and replacing all curves with line segments, we guarantee to introduce a discretization error of at most τ.

Strip generation. We generate a strip contour from the polylines by offsetting each line segment by $w/2$ in the positive and negative normal direction and computing their intersections (Figure 5.6). The vertices in the interior of a strip are the intersections between two adjacent offset polylines; for example, in Figure 5.6, A and B are computed by intersecting lines l_1 and l_2, and l_2 and l_3, respectively. The polyline's boundary vertices are offset in the normal direction of the incident line segment. In this way, we obtain the Planar Straight Line Graph (PSLG) of the strips. To avoid introducing small discretized elements due to short lines in the PSLG, we collapse edges shorter than a tolerance of $\epsilon = 10^{-3}$, starting with the shortest edge (Algorithm 1).

Lumped elements insertion. The lumped elements are modeled with the delta-gap method [84], which requires the lumped elements to be attached to edges that are aligned. We store the lumped element position as a parametric position $t \in [0, 1)$ on the B-Spline. While converting B-splines into polylines, we first split the B-splines at these parametric positions, so that the lumped element position exists among the converted polyline vertices. We treat all the lumped

Algorithm 1 PSLG decimation for strip generation

```
Input: PSLG with vertices and edges, tolerance $\epsilon$
Output: Simplified PSLG with vertices and edges
for all edges $(v_1, v_2)$ do
    Compute edge length $L_{v_1 v_2}$
    Insert $(v_1, v_2)$ into priority queue $Q$ with cost $L_{v_1 v_2}$
end for
while the minimum cost $> \epsilon$ do
    Extract edge $(v_1, v_2)$ with minimum cost from priority queue
    if $v_1$ is an endpoint or $L_{v_1 v_3} < L_{v_0 v_2}$ then    ▹ $(v_0, v_1)$ and $(v_2, v_3)$ are the adjacent edges
        $v' \leftarrow v_1$
    else
        $v' \leftarrow v_2$
    end if
    Collapse $(v_1, v_2)$ to $v'$
    Remove all edges adjacent to $v'$ from queue
    for all edges $(v', v_i)$ adjacent to $v'$ do
        Recompute edge length $L_{v' v_i}$ as cost
        Reinsert edge $(v', v_i)$ into priority queue $Q$
    end for
end while
```

element vertices as endpoints, so that they still exist after collapsing the small edges during strip generation. Finally, for each lumped element, we insert a line segment (the "delta gap") that connects the two endpoints of the lumped element and attaches it to the strip PSLG. We also enforce that each lumped element is attached to a collection of edges forming a straight line, which is a requirement of MARIE.

Triangulation. We compute the Delaunay triangulation of the PSLG using the software Triangle [172]. During triangulation, we also save all the edge correspondences for the lumped elements by adding edge masks.

Mapping to 3D. Once the coils are discretized in 2D, the vertex positions of the triangulation are mapped to 3D using $\Phi(\mathbf{u})$. However, while coils may overlap in the 2D representation, they must not touch in 3D. We ensure this by adjusting our mapping function $\Phi(\mathbf{u})$ for each coil. The first coil C_0 is mapped without any modifications, i.e., $\Phi_0(\mathbf{u}) = \Phi(\mathbf{u})$. For every new coil we ensure

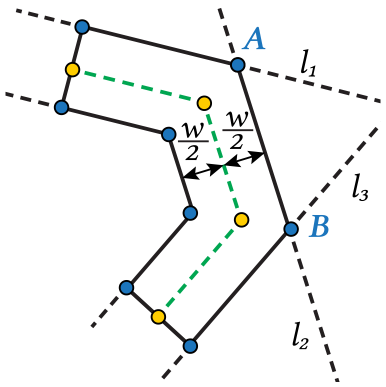


Figure 5.6: Strip generation (solid black) by offsetting polylines (dashed green).

that it does not intersect with the other coils already placed on the substrate. We achieve this by applying a displacement to the mapping,

$$\Phi_k(\mathbf{u}) = \Phi(\mathbf{u}) + D_k(\mathbf{u}) \tag{5.4}$$

$$D_k(\mathbf{u}) = \lambda \sum_{i=0}^{k} W(d_i(\mathbf{u}), w_i + \epsilon) \tag{5.5}$$

$$W(r,h) = \begin{cases} \left(1 - \frac{r}{h}\right)^4 \left(4\frac{r}{h} + 1\right) & \text{if } r < h \\ 0 & \text{otherwise} \end{cases}, \tag{5.6}$$

where $d_i(\mathbf{u})$ is the distance to the i^{th} coil, and $W(r,h)$ is the Wendland function [203]. The parameters λ and ϵ control the height and the steepness of the displacement. The Wendland function has compact support and therefore influences the coil geometry only locally, creating bridges on the coil C_k (Figure 5.7).

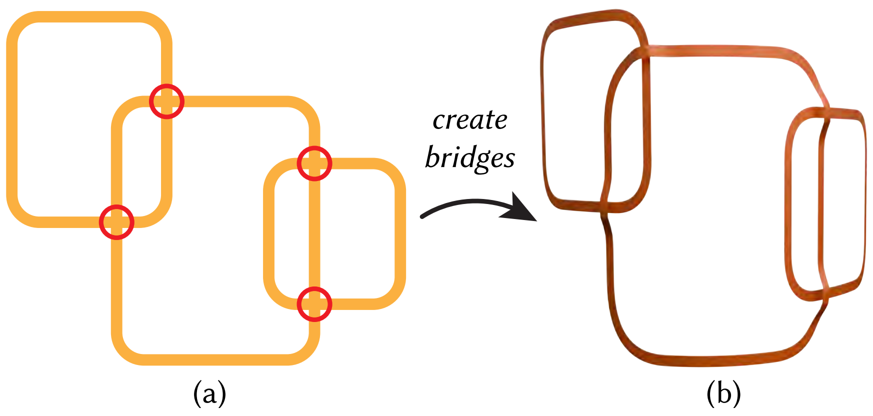


Figure 5.7: Coils can overlap in the 2D representation (a), but they must not intersect in the 3D model (b). We ensure this by embedding a displacement mapping that automatically creates bridges at positions where two coils cross each other.

5.5 Simulation

5.5.1 Background

Integral equation (IE) techniques offer distinct advantages for electromagnetic (EM) modeling in MRI applications. Unlike finite-difference time-domain and finite element methods, they are inherently free from grid dispersion artifacts [110, 177]. This is because IE formulations rely on Green's functions, which serve as exact propagators of EM fields from sources to observation points. Furthermore, for time-harmonic (single-frequency) simulations, like MRI simulations, IE-based solvers give rise to dense matrices with properties such as symmetry, low-rank, hidden low-rank, or smooth spectral decay, which can be leveraged for fast and memory-efficient solutions using numerical linear algebra algorithms [213].

5.5.1.1 Definitions

All EM simulations operate in periodic steady-state, at frequency f with units Hz. We define the angular frequency as $\omega = 2\pi f$ with units $^{\mathrm{rad}}/_{\mathrm{s}}$. In MRI, the frequency of operation is given by $f = \bar{\gamma} B_0$, where $\bar{\gamma} \approx 42.58\,^{\mathrm{MHz}}/_{\mathrm{T}}$ is the gyromagnetic ratio of ^{1}H hydrogen atoms in water and B_0

is the static magnetic field strength of the scanner.

Given real-valued relative permittivity ϵ_R, electrical conductivity σ, absolute permittivity in vacuum ϵ_0, and imaginary unit i, we define the complex-valued relative permittivity as:

$$\epsilon = \epsilon_\mathrm{R} + \frac{\sigma}{i\omega\epsilon_0}. \tag{5.7}$$

In the IE formulation, we also use the complex-valued electric susceptibility $\chi = \epsilon - 1$. In EM modeling, ϵ, χ, ϵ_R, and σ are scalar fields, which we denote as $\epsilon(\vec{r})$, $\chi(\vec{r})$, $\epsilon_\mathrm{R}(\vec{r})$, and $\sigma(\vec{r})$, respectively. We discretize these quantities and the EM fields onto a uniform 3D grid of N_v voxels, and we denote them with a subscripted index to refer to their value at the voxel with the same index.

To describe coil tuning and decoupling, we extensively use the Schur complement, which is defined as follows: Given a block matrix,

$$\mathbf{M} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}, \tag{5.8}$$

the Schur complement of block $\mathbf{A}$ in $\mathbf{M}$ is denoted as $\mathbf{M}/\mathbf{A}$ and defined as:

$$\mathbf{M}/\mathbf{A} = \mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B}. \tag{5.9}$$

Similarly, the Schur complement of block $\mathbf{D}$ in $\mathbf{M}$ is defined as:

$$\mathbf{M}/\mathbf{D} = \mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C}. \tag{5.10}$$

5.5.1.2 Volume Integral Equation (VIE) solver

The Volume Integral Equation (VIE) formulation that we use involves calculating the volumetric current density $\vec{J}_b(\vec{r})$ in the body subject to an incident electric field excitation at a single frequency f. The body is characterized by a distribution of relative permittivity $\epsilon_{\mathrm{R}}(\vec{r})$ and electrical conductivity $\sigma(\vec{r})$, which we group into a distribution of complex relative permittivities $\epsilon(\vec{r})$.

To discretize the system, we can employ piecewise constant (PWC) basis functions and PWC testing functions, which results in a Galerkin discretization. We first consider a basis function centered at the origin $f_0 : \mathbb{R}^3 \longrightarrow \{0, 1\}$ in a grid of voxel dimensions $\Delta_x \times \Delta_y \times \Delta_z$. We define f_0 evaluated at point $\vec{r} = [x\ y\ z]$ as:

$$f_0(\vec{r}) = \begin{cases} 1 & \text{if } |x| < \frac{\Delta_x}{2} \wedge |y| < \frac{\Delta_y}{2} \wedge |z| < \frac{\Delta_z}{2} \\ 0 & \text{otherwise} \end{cases}. \tag{5.11}$$

We then define the basis function $f_i : \mathbb{R}^3 \longrightarrow \{0, 1\}$, at voxel i with center $\vec{r}_i$:

$$f_i(\vec{r}) = f_0(\vec{r} - \vec{r}_i). \tag{5.12}$$

Note that each Cartesian component of the current density at a voxel has its own basis function, which means that the total number of PWC basis functions needed to discretize the current density is equal to $3N_v$.

We lump all possible body current contributions into density $\vec{J}_b$ and consider this density to consist of equivalent current distributions in free space, for simplicity. The discretized system is given by:

$$\left(\boldsymbol{\Delta} - \mathbf{P}_{\chi/\epsilon}\mathbf{N}\right)\mathbf{J}_b = i\omega\epsilon_0\mathbf{P}_{\chi/\epsilon}\mathbf{E}_{\mathrm{i}} \tag{5.13}$$

The term $\mathbf{E}_{\mathrm{i}}$ refers to the discretized incident electric field distribution. The operator $\mathbf{P}_{\chi/\epsilon}$ multiplies each Cartesian component at voxel i by the corresponding ratio χ_i/ϵ_i. The operator $\boldsymbol{\Delta}$ refers to the self-testing term or Gram matrix and for PWC basis functions is equal to $\Delta_x\Delta_y\Delta_z\mathbf{I}$, where $\mathbf{I}$ is the identity matrix of appropriate dimensions.

With as few as $30 \times 30 \times 30$ voxels for a 5 mm discretization, the full system would occupy approximately 100 GB of RAM when using double-precision arithmetic. To address this, we exploit the translation invariance of the Green's function underlying the problem, and diagonalize the block Toeplitz operator $\mathbf{N}$ using the 3D Fast Fourier Transform (FFT), lowering the memory footprint from $O(N_v^2)$ to $O(N_v)$. To invert the system, we can employ an iterative solver for non-symmetric systems, such as the Generalized Minimal Residual (GMRES) method.

5.5.1.3 Surface Integral Equation (SIE) solver

The Surface Integral Equation formulation that we employ discretizes the single-frequency Electric Field Integral Equation (EFIE) using Rao-Wilton-Glisson (RWG) basis functions and RWG testing functions, resulting in a Galerkin discretization of the underlying system. Each RWG basis function is defined over a different pair of edge-adjacent triangles in the corresponding mesh.

Given a vector of basis coefficients $\mathbf{J}_c \in \mathbb{C}^N$, where N is the number of triangles in the mesh that discretizes the conductors, and a vector of applied voltages $\mathbf{V}_c \in \mathbb{C}^{N_t}$ at the terminals, where N_t is the number of terminals (e.g., one port for each element of a coil array), the SIE system is given by:

$$\mathbf{Z}_{cc}\mathbf{J}_c = -\mathbf{F}\mathbf{V}_c \tag{5.14}$$

The complex symmetric matrix $\mathbf{Z}_{cc} \in \mathbb{C}^{N \times N}$ is the Galerkin discretized system, and the operator $\mathbf{F} \in \mathbb{C}^{N \times N_t}$ maps voltages to electric fields at the terminals. Assembling the $\mathbf{Z}_{cc}$ system involves computing $O(N)$ singular and $O(N^2)$ non-singular integrals. We adapt 1D Gauss-Legendre quadrature rules to compute these integrals, using a simple nesting scheme to approximate integrals over triangles. We compute the singular integrals using case-specific coordinate system transformations that remove the singularity from the integrand, at the expense of heavy use of trigonometric and inverse trigonometric functions.

Because the system $\mathbf{Z}_{cc}$ is complex symmetric and not Hermitian, we use direct inversion instead of conjugate gradients to invert the system in (5.14). Note that our SIE system results from the discretization of a Fredholm equation of the First Kind, which yields eigenvalues clustering at 0, further compromising the speed and applicability of iterative solving methods such as the conjugate gradients.

5.5.1.4 Magnetic Resonance Integral Equation (MARIE) suite

In the Magnetic Resonance Integral Equation (MARIE) suite [189], SIE and VIE are jointly solved to compute the EM fields generated by RF coils within a dielectric sample (e.g., a numerical body model). The SIE models the coils by discretizing their geometry with triangular surface meshes and representing surface currents using RWG basis functions [153]. The VIE models the EM phenomena within the body, discretized on a uniform voxel grid, with polarization currents approximated via PWC basis functions [146]. The combined volume-surface integral equation (VSIE) framework exploits the multilevel Toeplitz structure of the Green's function operators that map volumetric currents to fields, enabling fast matrix-vector multiplications through FFT [225].

We combine the SIE system with the VIE system by calculating the body-coil coupling operator $\mathbf{Z}_{cb} \in \mathbb{C}^{N \times N_v}$, which maps from body current PWC basis coefficients $\mathbf{J}_b$ to an incident electric field for each RWG basis function. Similarly, we calculate the operator $\mathbf{Z}_{bc} = \mathbf{Z}_{cb}^{\mathrm{T}}$ that maps RWG currents to incident electric field at each voxel. We introduce the term $\mathbf{Z}_{bb}$:

$$\mathbf{Z}_{bb} = \left(i\omega\epsilon_0 \mathbf{P}_{\chi/\epsilon}\right)^{-1}\left(\boldsymbol{\Delta} - \mathbf{P}_{\chi/\epsilon}\mathbf{N}\right). \tag{5.15}$$

Finally, the coupled VSIE system is given by:

$$\begin{bmatrix} \mathbf{Z}_{\mathrm{cc}} & \mathbf{Z}_{\mathrm{cb}} \\ -\mathbf{Z}_{\mathrm{cb}}^{\mathrm{T}} & \mathbf{Z}_{\mathrm{bb}} \end{bmatrix} \begin{bmatrix} \mathbf{J}_{\mathrm{c}} \\ \mathbf{J}_{\mathrm{b}} \end{bmatrix} = \begin{bmatrix} -\mathbf{F}\mathbf{V}_{\mathrm{t}} \\ 0 \end{bmatrix}. \tag{5.16}$$

In the past decade, a series of algorithms have been implemented to further improve MARIE's accuracy and efficiency. In terms of accuracy, it was shown that piecewise linear (PWL) basis functions yield more accurate simulations than PWC basis functions [51]. MARIE's solution time can considerably increase for simulations involving fine voxel resolutions. To address this, since the discretized Green's function tensors of the VIE sub-problem exhibit low multilinear ranks, it was shown that they can be compressed by at least two thousand times using the Tucker model, without sacrificing numerical precision [53]. As a result, the compressed operators can more easily fit in the limited memory of graphical processing units, leading to faster simulations. It was later shown that also the full VSIE system [56] can be compressed, using the precorrected FFT (pFFT) [64, 65] for cases when the coils are close to the body model, tensor train (TT) [55] for cases when the coils (or shields) are far from the body. A hybrid formulation of the VSIE that combines pFFT, cross-TT, and the adaptive cross approximation can also be used [54]. Finally, the SIE formulation has been adapted to explicitly model the coils' lumped elements in the simulation [84] rather than considering each lumped element a separate port [190], which can considerably increase the number of required MARIE solves.

These advances have enabled a broader class of applications that involve iterative evaluations of MARIE or its constituent modules. In particular, the increased speed and accuracy of MARIE's solver have allowed for the efficient use of body model-specific EM field bases to compute theoretical ultimate performance limits and the associated ideal current patterns (ICP) [57]. Tissue-dependent EM bases can be combined with the discrete empirical interpolation method [24] to construct the Magnetic Resonance Green Functions (MRGF) [191], which is a reduced-order model of MARIE's VIE and VSIE operators.

5.5.1.5 The Magnetic Resonance Green Function (MRGF)

EM simulation tools typically require an initial preprocessing step to assemble the necessary geometrical matrices involved in the simulation [191]. This preprocessing step can become computationally intensive, particularly for multi-element coil arrays discretized with thousands of triangles. For coil design optimization, where the coil geometry changes in each iteration, this pre-processing step becomes a significant bottleneck. The MRGF method addresses this high computational cost by allowing the incident fields of arbitrary RF coils to be expressed as linear combinations of precomputed fields of an EM basis [191]. This allows considerably faster predictions of the resulting electromagnetic fields in the body with less than 1% average accuracy loss when compared with the full MARIE solver [191]. Since MRGF transforms MARIE into a powerful framework for inverse problems, it was employed to implement the RF coil design optimization framework in this work.

Next, we briefly describe the steps associated with the MRGF method. First, one generates a basis of incident electric fields for a given body model by computing the fields generated by current sources external to the body. These current sources may consist of either a cloud of voxelized volumetric currents surrounding the body or of RWG surface currents whose support is a closed surface enclosing the body model [52]. One assembles the basis by exciting with one current element at a time and storing the vectorized incident field as a column of the incident field matrix $\mathbf{E}_\mathrm{i} \in \mathbb{C}^{3N_s \times N_J}$, where N_s is the number of non-air voxels in the grid and N_J is the number of source current elements. Next, one applies the singular value decomposition (SVD) to $\mathbf{E}_\mathrm{i}$, yielding the following factorization.

$$\mathbf{E}_\mathrm{i} = \mathbf{U}_\mathrm{i}\Sigma_\mathrm{i}\mathbf{V}_\mathrm{i}^\mathrm{H}. \tag{5.17}$$

The orthonormal columns of the $\mathbf{U}_\mathrm{i}$ matrix contain the dominant incident field modes, which one would then truncate up to a desired tolerance in the matrix L_2 norm sense, or equivalently one would keep all vectors whose singular values σ_k satisfy $\sigma_k > \sigma_1\varepsilon$, where σ_1 is the largest singular value and ε is the prescribed tolerance.

After the calculation of the incident field basis, one applies the Discrete Empirical Interpolation Method (DEIM) [24] to select a subset of voxels for which to calculate the incident fields, which can then be interpolated to derive the incident fields for all tissue voxels using the operator $\mathbf{X}$ that DEIM provides. After DEIM, one solves the VIE system $\mathbf{Z}_{\mathrm{bb}}$ for all incident fields in $\mathbf{U}_{\mathrm{i}}$, yielding a set of volumetric currents stored in matrix $\mathbf{M}$. One then forms the matrix $\mathbf{M}_{\mathrm{m}}$:

$$\mathbf{M}_{\mathrm{m}} = \mathbf{X}^{\mathrm{T}}\left(\mathbf{U}_{\mathrm{i}}^{\mathrm{T}}\mathbf{M}\right)\mathbf{X}. \tag{5.18}$$

When simulating a coil geometry, one would calculate the matrix $\mathbf{Z}_{\mathrm{dc}}$, which is equal to the incident electric field at the DEIM interpolation points over all of the RWG coil currents. The product $\mathbf{Z}_{\mathrm{dc}}^{\mathrm{T}}\mathbf{M}_{\mathrm{m}}\mathbf{Z}_{\mathrm{dc}}$ is a close approximation of the term $\mathbf{Z}_{\mathrm{cb}}\mathbf{Z}_{\mathrm{bb}}^{-1}\mathbf{Z}_{\mathrm{cb}}^{\mathrm{T}}$ of the full VSIE system. One then applies the SVD to $\mathbf{M}_{\mathrm{m}}$, resulting in the approximate factorization $\mathbf{U}_{\mathrm{m}}\Sigma_{\mathrm{m}}\mathbf{V}_{\mathrm{m}}^{\mathrm{H}}$, which one would then truncate to achieve a specified tolerance, as was done with the incident fields in $\mathbf{U}_{\mathrm{i}}$, concluding the MRGF assembly process.

The MRGF only needs to be assembled one time per body model, and can be used to simulate any coil configuration constrained to surfaces that lie or extend beyond the substrate where the current sources used to generate the basis were defined. In summary, using the MRGF method, we can precompute the inverse of the VIE operators and reduced the number of points needed to compute the electric and magnetic field over all voxels, drastically accelerating the solution of (5.16), which we invert directly following the procedure outlined in [191]. With this precomputed solution, we then calculate the total electric and magnetic fields, which are required to determine the SNR during coil design optimization.

5.5.2 Simulation improvements

The previous versions of MARIE [65, 190] assembled the SIE operators using fixed quadrature orders that were too low for precise computation of the coil-to-coil interaction matrix $\mathbf{Z}_{\mathrm{cc}}$. A lack of precision in these operators means that gradient calculations via finite differences will be too

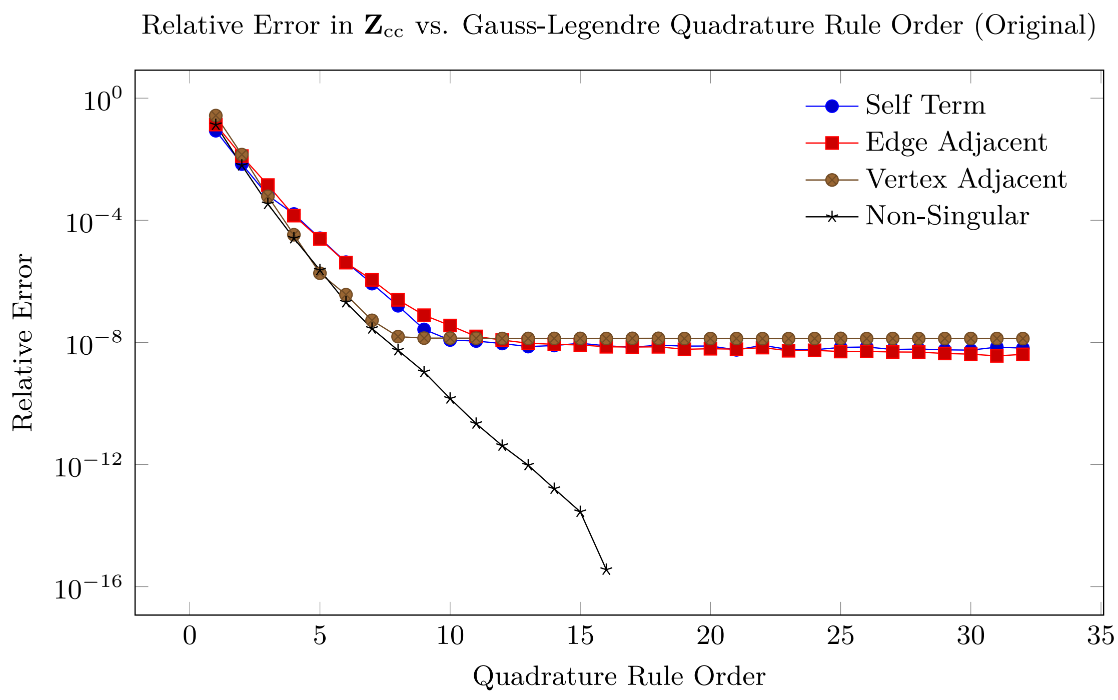


Figure 5.8: Convergence of error in SIE integrals as a function of Gauss-Legendre quadrature order, when assembling $\mathbf{Z}_{cc}$ using the original implementation.

inaccurate for optimization, unless the step size is increased substantially. This code was additionally unoptimized and implemented in the MATLAB scripting language, resulting in assembly times for $\mathbf{Z}_{cc}$ that would render the optimization intractable. Additionally, the previous version lacked algorithms for tuning, matching, and decoupling, which are essential components of any electromagnetic simulations involving coils and antennas. In this work, we developed optimized routines for the assembly of $\mathbf{Z}_{cc}$ that are not only faster but also considerably more precise. We also developed algorithms for tuning and decoupling of coil arrays that allow us to simulate the coil-to-coil interactions more realistically.

5.5.2.1 Improving the precision of the SIE operator assembly

The SIE assembly process for computing coil-to-coil interactions $\mathbf{Z}_{cc}$ involves numerically integrating complex-valued singular functions for four different scenarios: vertex adjacent pairs, edge adjacent pairs, overlapping or self-term pairs, and non-singular pairs. Since the non-singular

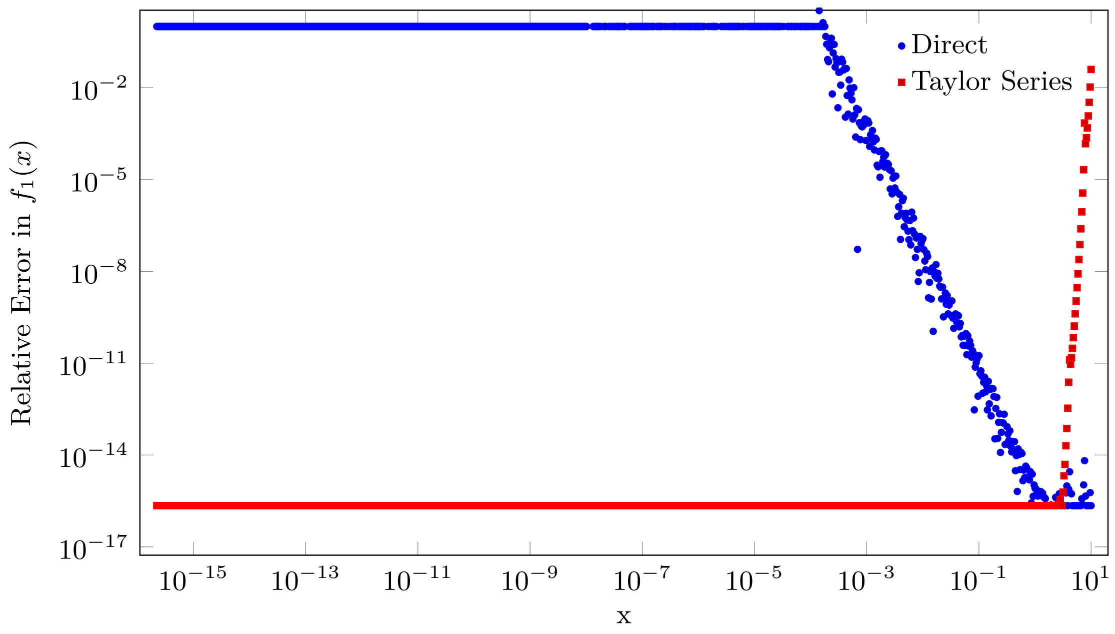


Figure 5.9: Evaluation of $f_1(x)$ using direct evaluation (blue) and Taylor Series approximation (red). The precision of direct evaluation suffers from a total loss of precision for arguments of magnitude 10^{-3} or smaller. The threshold is set to the point in the graph where the two sets of data meet ($x \approx 3$).

pairs are analytic over the 4D integration domains, we apply simple Gauss-Legendre quadrature to integrate these terms. However, for the other three cases, we apply coordinate transformations that then eliminate these singularities at the expense of greater numerical complexity in the integrands. We start by considering the real part of an integrand used in the vertex-adjacent integrals,

$$f_1(x) = \left(\frac{x^2}{2} - 1\right)\cos(x) - x\sin(x) + 1. \tag{5.19}$$

When evaluating these integrands, x is equal to $k_0 r$, where k_0 is the wavenumber in free space and r is the distance between observation and source quadrature points. Such a function, when evaluated for small arguments as defined, suffers from catastrophic loss of numerical precision. We can understand why by expressing this function in terms of its Taylor series about $x = 0$, which has an infinite radius of convergence in x:

$$\begin{aligned} f_1(x) = {} & 1 \\ & -1 + \frac{x^2}{2!} - \frac{x^4}{4!} + \frac{x^6}{6!} - \dots \\ & + \frac{x^2}{2} - \frac{x^4}{2 \cdot 2!} + \frac{x^6}{2 \cdot 4!} - \dots \\ & - \frac{x^2}{1!} + \frac{x^4}{3!} - \frac{x^6}{6!} - \dots \\ = {} & - \frac{x^4}{8} + \frac{x^6}{72} - \dots \end{aligned} \tag{5.20}$$

As is evident, the constant and quadratic terms in the Taylor series cancel out. For small arguments, the quartic terms are considerably smaller than the quadratic terms. When these terms cancel out, most of the precision in the floating point representation is lost, resulting in a catastrophic loss of numerical precision. Figure 5.9 shows the relative error in the direct evaluation of $f_1(x)$ when using double-precision floating point numbers with respect to the direct evaluation when using 512-bit precision floating point numbers, as well as the relative error of a Taylor Series approximation with 12 non-zero coefficients (polynomial order 26). Figure 5.8 shows the convergence of the relative error in $\mathbf{Z}_{\mathrm{cc}}$ as a function of Gauss-Legendre quadrature order, for each different adjacency type. For the quadrature orders in the original implementation, the relative error in $\mathbf{Z}_{\mathrm{cc}}$ was on the order of 10^{-4}, which is insufficient for coil optimization. Additionally, even when the quadrature order was increased to the maximum possible order, the relative error in $\mathbf{Z}_{\mathrm{cc}}$ plateaued at approximately 10^{-8}, which was still significant, potentially impacting the estimation of gradients and the line search procedure when optimizing coil array geometries.

We addressed this loss of numerical precision by dividing the evaluation of each singular integrand into two cases depending on whether the absolute value of the argument is smaller or larger than a threshold. The threshold depends on the approximated function but is usually either $|x| = 1$ or $|x| = 3$. For this work, it was determined empirically for every integrand by comparing the evaluation in double-precision with the evaluation in either 512-bit floating point representa-

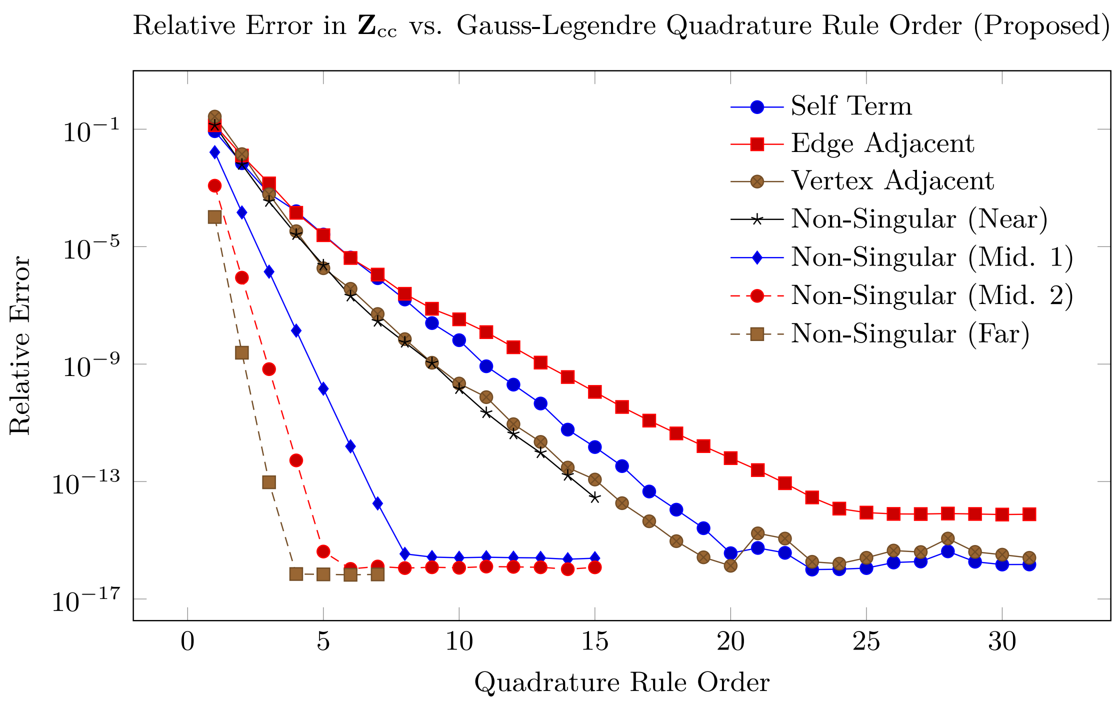


Figure 5.10: Convergence of error in SIE integrals as a function of Gauss-Legendre quadrature order, when assembling $\mathbf{Z}_{cc}$ using the proposed implementation.

tions for logarithmically spaced samples. When the input exceeds the threshold, we evaluate the integrands using their functional forms, as in (5.19). When the input is smaller than the threshold, we use the Taylor series of the integrand, evaluated using Horner's method for polynomial evaluation, because direct polynomial evaluation can also suffer from loss of precision. We also subdivided the non-singular interactions into four categories, based on the ratio of the distance between the triangle pairs to the free-space wavelength $\lambda = c/f$. We used a high quadrature order for the near terms and progressively lowered the quadrature order as for greater distances.

We summarize all of the replaced integrands in the SIE assembly and their Taylor series approximations in Table 5.1. We include the leading term of each Taylor series about 0, as the order of the leading term correlates with the severity of the loss of numerical precision. Fig. 5.10 demonstrates the same convergence analysis as in Fig. 5.8 but with the proposed implementation. We observe that with these improvements, the relative error can drop to 10^{-17}, with the largest relative error equaling 10^{-15} for edge adjacent interactions. The quadrature order for each category was then set to values for which the relative error was minimized.

Table 5.1: Singular integrals needed for the SIE assembly and associated Taylor expansion when the input x is lower than the specified threshold.

Function	Leading Term	Threshold
$\cos(x) - 1 + \frac{x^2}{2}$	$+\frac{x^4}{4!}$	1
$\cos(x) - 1 + \frac{x^2}{2} - \frac{x^4}{24}$	$-\frac{x^6}{6!}$	1
$x - \sin(x)$	$+\frac{x^3}{3!}$	1
$x - \frac{x^3}{3!} - \sin(x)$	$-\frac{x^5}{5!}$	3
$\cos(x) + \frac{x}{2}\sin(x) - 1$	$-\frac{x^4}{4!}$	3
$\cos(x) + \frac{x}{2}\sin(x) - 1 + \frac{x^4}{4!}$	$+\frac{2x^6}{6!}$	3
$\frac{x}{2}(\cos(x) + 1) - \sin(x) + \frac{x^3}{12}$	$+\frac{x^4}{80}$	3
$\frac{x}{2}(\cos(x) + 1) - \sin(x)$	$-\frac{x^3}{12}$	1
$\left(1 - \frac{x^2}{8}\right)\cos(x) + \frac{5x}{8}\sin(x) - 1$	$-\frac{x^6}{6!}$	3
$\frac{3x}{8} + \frac{x^3}{48} + \frac{5x}{8}\cos(x) - \left(1 - \frac{x^2}{8}\right)\sin(x)$	$-\frac{x^5}{320}$	3
$\left(3 - \frac{x^2}{2}\right)\cos(x) + 2x\sin(x) - 3 - \frac{x^4}{4!}$	$-\frac{x^6}{5!}$	3

Function	**Leading Term**	**Threshold**
$2x\cos(x) + \left(\frac{x^2}{2} - 3\right)\sin(x) + x$	$-\frac{3x^5}{5!}$	3
$\left(1 - \frac{x^2}{8}\right)\cos(x) + \frac{3x}{4}\sin(x) - 1 - \frac{x^2}{8} + \frac{x^4}{48}$	$-\frac{x^6}{4\cdot 6!}$	3
$\left(\frac{x^2}{8} - 1\right)\sin(x) + \frac{3x}{4}\cos(x) + \frac{x}{4} + \frac{x^3}{12}$	$+\frac{x^5}{4\cdot 5!}$	3
$\frac{x}{2}\cos(x) + \left(\frac{x^2}{8} - 1\right)\sin(x) + \frac{x}{2} - \frac{x^3}{4!}$	$-\frac{x^5}{5!}$	3
$\cos(x) - 1 + \frac{x}{4}\sin(x) + \frac{x^2}{4}$	$+\frac{x^6}{2\cdot 6!}$	3
$\frac{x}{4}\cos(x) - \sin(x) + \frac{3x}{4} - \frac{x^3}{4!}$	$+\frac{x^5}{4\cdot 5!}$	3
$\cos(x) + x\sin(x) - 1$	$+\frac{x^2}{2}$	1.6818
$\cos(x) + x\sin(x) - 1 - \frac{x^2}{2}$	$-\frac{x^4}{8}$	3
$x\cos(x) - \sin(x)$	$-\frac{x^3}{3}$	3
$\left(1 - \frac{x^2}{2}\right)\cos(x) + x\sin(x) - 1$	$+\frac{x^4}{8}$	3
$x\cos(x) + \left(\frac{x^2}{2} - 1\right)\sin(x)$	$+\frac{x^3}{6}$	1.6818
$\left(1 - \frac{x^2}{2}\right)\cos(x) + \left(x - \frac{x^3}{6}\right)\sin(x) - 1$	$-\frac{x^4}{4!}$	1.6818
$\left(x - \frac{x^3}{6}\right)\cos(x) + \left(\frac{x^2}{2} - 1\right)\sin(x)$	$+\frac{4x^5}{5!}$	3
$\cos(x) + \frac{x}{3}\sin(x) + \frac{x^2}{6} - 1$	$-\frac{x^4}{3\cdot 4!}$	3

Function	Leading Term	Threshold
$\frac{x}{3}\cos(x) - \sin(x) + \frac{2x}{3}$	$+\frac{2x^5}{3\cdot 5!}$	3
$\left(1 - \frac{x^2}{6}\right)\cos(x) + \frac{2x}{3}\sin(x) - 1$	$+\frac{x^4}{3\cdot 4!}$	2
$\left(\frac{x^2}{6} - 1\right)\sin(x) + \frac{2x}{3}\cos(x) + \frac{x}{3}$	$-\frac{x^5}{5!}$	3
$\left(\frac{x^2}{2} - 1\right)\cos(x) - x\sin(x) + 1$	$-\frac{x^4}{8}$	3
$\left(1 - \frac{x^2}{2}\right)\sin(x) - x\cos(x)$	$-\frac{x^3}{6}$	3
$\left(\frac{x^2}{2} - 1\right)\cos(x) - \left(\frac{x^3}{6} - x\right)\sin(x) + 1$	$+\frac{x^4}{4!}$	1.6818
$\left(1 - \frac{x^2}{2}\right)\sin(x) + \left(\frac{x^3}{6} - x\right)\cos(x)$	$-\frac{x^5}{30}$	3
$\left(-\frac{x^4}{24} + \frac{x^2}{2} - 1\right)\cos(x) + \left(\frac{x^3}{6} - x\right)\sin(x) + 1$	$+\frac{x^6}{144}$	3
$\left(+\frac{x^4}{24} - \frac{x^2}{2} + 1\right)\sin(x) + \left(\frac{x^3}{6} - x\right)\cos(x)$	$+\frac{x^5}{5!}$	3

5.5.2.2 Accelerating the assembly of SIE Operators

The original SIE assembly implementation was written in the Matlab scripting language and was unoptimized. As we saw in the previous section, many of the integrands in the calculation of singular integrals also suffered from catastrophic loss of numerical precision, necessitating custom implementations of these functions for small arguments that are also fast to compute. We addressed this requirement and the need for optimized code by rewriting the SIE assembly code in C++. We introduced a number of optimizations meant to both decrease assembly times and to lighten the load on CPU caches.

Each singular integrand type uses a number of trigonometric expansions that are independent of the dimensions of the pairs of triangles, and depend only on the quadrature order. We calculated these geometry-independent quantities at compile time using the C++11 standard's `constexpr` variable and function modifier, which forces evaluation at compile time. Additionally, the original implementation kept a sorted list of non-singular interaction pairs over the N_t triangles of the mesh that grows as $O\left(N_t^2\right)$, as well as sorted lists of vertex-adjacent, edge-adjacent, and self term pairs that grow as $O(N_t)$. The self term pairs consist of pairs $\{(i, i) \text{ for } i \in \{1, 2, \ldots, N_t\}\}$, which can be inferred automatically during assembly. The non-singular pairs consist of all pairs that are not vertex-adjacent, edge-adjacent, or self terms. This allows us to discard entirely the list of non-singular pairs, which we replace with a loop that iterates over all possible pairs, using a binary search algorithm over the entries in the vertex-adjacent and edge-adjacent lists to determine whether to carry out the computation. The binary search introduces a negligible $O(\log(N_t))$ term during assembly, but because the input data needed to assemble the operator grows as $O(N_t)$ instead of $O\left(N_t^2\right)$, the assembly is typically considerably faster due to improved cache coherence.

5.5.3 Admittance and impedance parameters over terminals

When we apply voltages $\mathbf{V}_\mathrm{t}$ to the coil array, we obtain a corresponding set of currents $\mathbf{J}_\mathrm{t}$, with positive currents flowing into the corresponding positive side of each terminal. The linear operator describing this transfer function is called the admittance parameter matrix, which we denote as $\mathbf{Y}_\mathrm{t}$. The voltage-current relationship can thus be summarized as

$$\mathbf{J}_\mathrm{t} = \mathbf{Y}_\mathrm{t}\mathbf{V}_\mathrm{t}. \tag{5.21}$$

In order to calculate the admittance parameters $\mathbf{Y}_\mathrm{t}$, we use the Schur complement of block $\mathbf{Z}_\mathrm{bb}$ in (5.16) to obtain the loaded coil-to-coil interaction matrix $\mathbf{Z}_\mathrm{cbc}$,

$$\mathbf{Z}_\mathrm{cbc} = \mathbf{Z}_\mathrm{cc} + \mathbf{Z}_\mathrm{cb}\mathbf{Z}_\mathrm{bb}^{-1}\mathbf{Z}_\mathrm{cb}^\mathrm{T}. \tag{5.22}$$

We can thus express the solution for $\mathbf{J}_\mathrm{t}$ in (5.16) as

$$\mathbf{Z}_\mathrm{cbc}\mathbf{J}_\mathrm{c} = -\mathbf{F}\mathbf{V}_\mathrm{t}. \tag{5.23}$$

Finally, we use the identity $\mathbf{J}_\mathrm{t} = -\mathbf{F}^\mathrm{T}\mathbf{J}_\mathrm{c}$ to obtain the following relationship between applied terminal voltages $\mathbf{V}_\mathrm{t}$ and induced terminal currents $\mathbf{J}_\mathrm{t}$. The minus sign stems from $\mathbf{F}^\mathrm{T}\mathbf{J}_\mathrm{c}$ resulting in the current flowing out of the positive side of the terminal,

$$\mathbf{J}_\mathrm{t} = \mathbf{F}^\mathrm{T}\mathbf{Z}_\mathrm{cbc}^{-1}\mathbf{F}\mathbf{V}_\mathrm{t} = \mathbf{Y}_\mathrm{t}\mathbf{V}_\mathrm{t}. \tag{5.24}$$

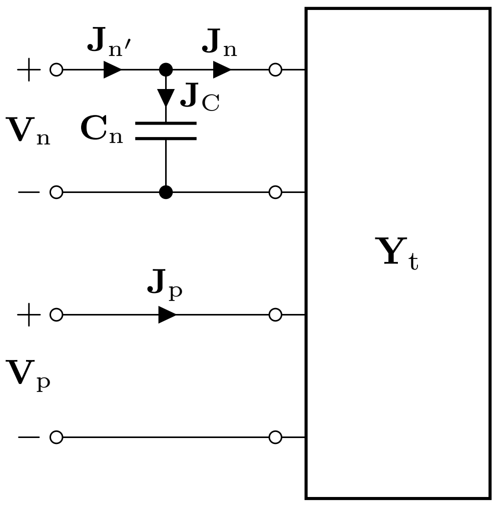
The operator acting on $\mathbf{V}_\mathrm{t}$ is precisely the admittance parameter matrix of the system.

5.5.3.1 Accelerating SVIE using the MRGF

As discussed in Sec. 5.5.1.5, we used the MRGF method to drastically accelerate the computation of $\mathbf{Z}_\mathrm{cbc}$ in (5.22). More specifically, for each coil geometry, we calculated the incident electric field over the DEIM interpolation points $\mathbf{Z}_\mathrm{dc}$ and computed the product with

$$\left(\mathbf{Z}_\mathrm{dc}^\mathrm{T}\mathbf{U}_\mathrm{m}\right)\mathbf{M}_\mathrm{m}\left(\mathbf{V}_\mathrm{m}^\mathrm{H}\mathbf{Z}_\mathrm{dc}\right) \approx \mathbf{Z}_\mathrm{cb}\mathbf{Z}_\mathrm{bb}^{-1}\mathbf{Z}_\mathrm{cb}^\mathrm{T}. \tag{5.25}$$

The right-hand side is precisely the SVIE term appearing in (5.22), which we replace with the left-hand side to calculate the SVIE term much more efficiently with the precomputed VIE inverse operator $\mathbf{M}_\mathrm{m}$. In this work, we required fewer than $5,000$ interpolation points when assembling the MRGF operators for all numerical experiments at 5 mm isotropic resolution.



$\mathbf{J}_{\mathrm{n}'}$
$\mathbf{J}_{\mathrm{n}}$
$\mathbf{J}_{\mathrm{C}}$
$+$
$\mathbf{V}_{\mathrm{n}}$
$\mathbf{C}_{\mathrm{n}}$
$-$
$\mathbf{Y}_{\mathrm{t}}$
$\mathbf{J}_{\mathrm{p}}$
$+$
$\mathbf{V}_{\mathrm{p}}$
$-$

Figure 5.11: Circuit schematic demonstrating the tuning approach.

5.5.4 Adding tuning capacitors to coils

The terminals can be subdivided into two categories: non-port terminals and port terminals. We denote each type using subscripts n and p, respectively. As such, we can subdivide the terminal voltages, terminal currents, and admittance (or impedance) parameters using this convention, allowing us to rewrite (5.24) as follows:

$$\begin{bmatrix} \mathrm{J_p} \\ \mathrm{J_n} \end{bmatrix} = \begin{bmatrix} \mathrm{Y_{pp}} & \mathrm{Y_{pn}} \\ \mathrm{Y_{pn}^T} & \mathrm{Y_{nn}} \end{bmatrix} \begin{bmatrix} \mathrm{V_p} \\ \mathrm{V_n} \end{bmatrix}. \tag{5.26}$$

In order to tune the coil array, we attach capacitors $\mathbf{C}_\mathrm{n}$ at the non-port terminals in parallel. Consequently, when we express the admittance parameters of (5.26) with a new non-port terminal current $\mathrm{J_n'} = \mathrm{J_C} + \mathrm{J_n}$ flowing into the parallel combination, with $\mathrm{J_C}$ corresponding to the vector of currents flowing into the capacitors. The Voltage-Current (V-I) relationships of the capacitors are

$$\mathrm{J_C} = \mathrm{D}(j\omega \mathrm{C_n})\mathrm{V_n}. \tag{5.27}$$

We can thus write the current-voltage (I-V) relations with respect to $\mathbf{J}_{\mathrm{n}'}$,

$$\begin{bmatrix} \mathbf{J}_\mathrm{p} \\ \mathbf{J}_{\mathrm{n}'} \end{bmatrix} = \begin{bmatrix} \mathbf{Y}_\mathrm{pp} & \mathbf{Y}_\mathrm{pn} \\ \mathbf{Y}_\mathrm{pn}^\mathrm{T} & \mathbf{Y}_\mathrm{nn} + \mathrm{D}(j\omega\mathbf{C}_\mathrm{n}) \end{bmatrix} \begin{bmatrix} \mathbf{V}_\mathrm{p} \\ \mathbf{V}_\mathrm{n} \end{bmatrix}. \tag{5.28}$$

We introduce $\mathbf{Y}_{\mathrm{nn}'} = \mathbf{Y}_\mathrm{nn} + \mathrm{D}(j\omega\mathbf{C}_\mathrm{n})$ and obtain the loaded impedance parameters by applying the Schur complements to invert the loaded admittance matrix over the terminals,

$$\mathbf{Z}_\mathrm{t} = \begin{bmatrix} \mathbf{Z}_\mathrm{pp} & \mathbf{Z}_\mathrm{pn} \\ \mathbf{Z}_\mathrm{pn}^\mathrm{T} & \mathbf{Z}_\mathrm{nn} \end{bmatrix} = \begin{bmatrix} (\mathbf{Y}_\mathrm{t}/\mathbf{Y}_{\mathrm{nn}'})^{-1} & -\mathbf{Y}_\mathrm{pp}^{-1}\mathbf{Y}_\mathrm{pn}\left(\mathbf{Y}_\mathrm{t}/\mathbf{Y}_\mathrm{pp}\right)^{-1} \\ -\mathbf{Y}_{\mathrm{nn}'}^{-1}\mathbf{Y}_\mathrm{np}^\mathrm{T}(\mathbf{Y}_\mathrm{t}/\mathbf{Y}_{\mathrm{nn}'})^{-1} & \left(\mathbf{Y}_\mathrm{t}/\mathbf{Y}_\mathrm{pp}\right)^{-1} \end{bmatrix}. \tag{5.29}$$

This square matrix dictates the Voltage-Current (V-I) relationships of the loaded system. However, because only the capacitors at the non-port terminals load the coil array, no current will flow into the primed terminals, meaning that $\mathbf{J}_{\mathrm{n}'} = 0$, which allows us to ignore the second block column,

$$\begin{bmatrix} \mathbf{V}_\mathrm{p} \\ \mathbf{V}_\mathrm{n} \end{bmatrix} = \begin{bmatrix} \mathbf{I} \\ -\mathbf{Y}_{\mathrm{nn}'}^{-1}\mathbf{Y}_\mathrm{np}^\mathrm{T} \end{bmatrix} (\mathbf{Y}_\mathrm{t}/\mathbf{Y}_{\mathrm{nn}'})^{-1}\mathbf{J}_\mathrm{p}. \tag{5.30}$$

The top block dictates the Voltage-Current relationships over the ports and is equal to the impedance parameter matrix over the ports $\mathbf{Z}_\mathrm{p}$. The admittance parameters over the ports, which we use in the following section when tuning, are simply the inverse of $\mathbf{Z}_\mathrm{p}$,

$$\mathbf{Y}_\mathrm{p}(\mathbf{C}_\mathrm{n}) = \mathbf{Y}_\mathrm{t}/\mathbf{Y}_{\mathrm{nn}'} = \mathbf{Y}_\mathrm{pp} - \mathbf{Y}_\mathrm{pn}(\mathbf{Y}_\mathrm{nn} + \mathrm{D}(j\omega\mathbf{C}_\mathrm{n}))^{-1}\mathbf{Y}_\mathrm{pn}^\mathrm{T}. \tag{5.31}$$

While not relevant when tuning, the lower block contains the transfer function from the port currents $\mathbf{J}_\mathrm{p}$ to the voltages at the non-port terminals, which we will use when discussing our decoupling strategy.

5.5.4.1 Tuning a coil array

We apply Lorentz reciprocity to our problem and argue that decoupling preamplifiers behave like current sources in parallel with source resistances under reciprocity. We set these resistances to ∞ and refer to this process as *ideal decoupling*. Note that this could be approximated in practice for receive arrays with preamplifier decoupling. As a result, when a port is driven, all of the other port currents are identically 0. Therefore, we only consider the self-interactions when tuning the coil array, i.e., we consider only the diagonal of loaded impedance parameters $\mathbf{Y}_\mathrm{p}$ in (5.31). We tune the coil array by solving the following optimization problem:

$$\begin{aligned} \min_{\mathbf{C}_\mathrm{n} \in \mathbb{R}^{N_c}} \quad & \left\| \mathrm{Im}\left\{ \mathrm{D}\left(\mathbf{Y}_\mathrm{p}\right) \right\} \right\|_2^2 \\ \text{s.t.} \quad & (\mathbf{C}_l)_k \leq (\mathbf{C}_\mathrm{n})_k \leq (\mathbf{C}_u)_k \, \forall\, k \in \{1, \ldots, N_c\}. \end{aligned} \tag{5.32}$$

The vectors $\mathbf{C}_l$ and $\mathbf{C}_u$ denote lower and upper bounds, respectively, for the tuning capacitors. This optimization problem is highly non-convex and features sharp peaks near the optima with flat regions between these optima. When the cost function exceeds a pre-determined threshold, we solve this problem using a Particle Swarm optimization. Once that optimization completes, we refine the solution by solving the optimization problem using Newton's Method.

5.5.4.2 Decoupling a coil array

When we solve the coupled SVIE system, we simply invert the operator on the left-hand side of (5.23) while setting $\mathbf{V}_\mathrm{t}$ to the identity matrix. This is equivalent to setting the voltage at a port to 1 V and the rest to 0 V using voltage sources, and repeating this for all ports. We denote this solution as $\hat{\mathbf{J}}_\mathrm{c}$, which we further subdivide,

$$\hat{\mathbf{J}}_\mathrm{c} = \begin{bmatrix} \hat{\mathbf{J}}_\mathrm{cp} & \hat{\mathbf{J}}_\mathrm{cn} \end{bmatrix},$$

based on whether the voltage was applied at a port or a non-port terminal. We then perform tuning, resulting in a set of capacitors $\mathbf{C}_\mathrm{n}$. Given these capacitors, we can obtain the voltage at each terminal using the transfer function in (5.30) that maps from port currents to terminal voltages. We then multiply our original solution $\hat{\mathbf{J}}_\mathrm{c}$ by this transfer function to obtain the RWG basis coefficients corresponding to driving a tuned, decoupled coil using current sources,

$$\mathbf{J}_\mathrm{c} = \begin{bmatrix} \hat{\mathbf{J}}_\mathrm{cp} & \hat{\mathbf{J}}_\mathrm{cn} \end{bmatrix} \begin{bmatrix} \mathbf{I} \\ -\mathbf{Y}_{\mathrm{nn}'}^{-1} \mathbf{Y}_\mathrm{np}^\mathrm{T} \end{bmatrix} \mathbf{Z}_\mathrm{p}. \tag{5.33}$$

5.6 Optimization

We cast the optimization of an RF coil as a constrained optimization problem. We define an objective function $f(x)$, which evaluates the coil's absolute performance in an ROI, along with a constraint function $g(x)$ that ensures that the coil geometry is valid (Section 5.3). Whenever a coil is considered invalid, $g(x)$ is negative, e.g., for the case of a coil intersecting itself or other coils. Our coil shape optimization problem can therefore be written as:

$$\min_x f(x), \ \text{s.t. } g(x) \geq 0. \tag{5.34}$$

We convert this into an unconstrained optimization problem where the constraints are enforced by a barrier potential term $B_g(x)$ that increases to infinity if $g(x)$ approaches 0, and vanishes when $g(x) > 0$ is sufficiently large.

$$\min_x f(x) + B_g(x). \tag{5.35}$$

The conversion to the unconstrained problem using a potential has several advantages: First, while inequality constraints may introduce non-smoothness to the solution, replacing these with a smooth barrier in general leads to approximating smooth solutions of similar accuracy, as long as the underlying PDEs allow for this. Second, converting the problem to an unconstrained optimization problem allows one to use simpler and more reliable optimization algorithms. On the downside, barrier potentials introduce additional non-physical forces in configurations close to contact. However, if the extent of barrier potential is chosen to be sufficiently small, the overall solution error is dominated by the geometry discretization error.

We solve the optimization problem in (5.35) using L-BFGS [120]. The gradients for the barrier term are computed analytically. The gradients for the objective are estimated using finite differences. Although an analytic derivation of the gradient of $f(x)$ is possible, one would need to compute multiple derivatives of multidimensional singular integrals appearing in the MRGF's geometrical Green's function operators [191]. For simplicity, this work employs the finite difference method. In future work, we will consider analytic gradients to make searching for a larger design parameter space computationally manageable. Finally, we use explicit line-search checks to ensure that the optimization remains within the feasible space $g(x) > 0$ (Section 5.6.2).

5.6.1 Objective function

Our goal is to optimize receive coils so that the associated SNR approaches the UISNR within a target region of interest (ROI). Once the electric ($\mathbf{e}$) and magnetic ($\mathbf{h}$) fields of each coil element are computed using an EM simulator (e.g., MARIE or MRGF described in 5.5), the optimal SNR at a position of interest $\mathbf{r}_0 \in \text{ROI}$ can be computed as [109]:

$$\text{SNR}(\mathbf{r}_0) = \frac{\omega M_0}{\sqrt{4k_B T \left(\mathsf{S}^{\mathrm{H}} \left(\boldsymbol{\Psi}_b + \boldsymbol{\Psi}_a\right) \mathsf{S}\right)^{-1}}}, \tag{5.36}$$

with ω the angular operating frequency, M_0 the equilibrium magnetization, $\mathsf{S} \in \mathbb{C}^{n\times p}$ the receive coil sensitivity $\left(= \mathbf{h}_{\mathrm{x}} - \mathrm{i}\mathbf{h}_{\mathrm{y}}\right)$, k_B Boltzmann's constant, T the temperature of the sample, n the number of coils in the array, and p the number of positions of the voxels discretizing the sample.

The noise covariance matrix $\Psi_b \in \mathbb{R}^{n\times n}$ [94] accounts for intrinsic thermal losses due to the sample's conductivity and its elements can be computed for each coil pair n_1, n_2 as:

$$\mathbf{\Psi}_b^{n_1,n_2} = \iiint \sigma_e\left(\mathbf{r}\right)\mathbf{e}_{n_2}\left(\mathbf{r}\right)\mathbf{e}_{n_1}^{*}\left(\mathbf{r}\right)d\mathbf{r}^3, \tag{5.37}$$

where the integral is computed over the entire sample and $\sigma_e \in \mathbb{R}^{p\times 1}$ is the electric conductivity. $\Psi_a \in \mathbb{R}^{n\times n}$ is a diagonal matrix computed using the equivalent-power formulation presented in [154], which models the noise equivalent resistance associated with Joule heating of the coil conductors. The RF coils are modeled as non-perfect electric conductors by incorporating the finite surface resistivity of copper directly into the surface integral equation (SIE) matrix, following the formulation in [84], to account for ohmic losses in the conductors.

The UISNR is computed using Equation 5.36, without Ψ_a, combining the elements of an EM basis as if they were coils in a hypothetical infinite array. One can construct a basis of EM fields following the Huygens–Fresnel principle [9] and the approach in [52]. This basis includes all possible field distributions within the sample generated by RF sources placed outside an enclosing surface. As the number of modes included in Equation 5.36 increases, the resulting SNR converges to the UISNR [137, 140, 164], which represents the theoretical maximum achievable SNR for the given sample, independent from coil geometry. The UISNR must be computed once as the benchmark of the optimization problem, while the SNR is updated in every iteration of the optimization for each new coil configuration.

For our problem, the objective function (Equation 5.35) quantifies the difference between the simulated SNR and the UISNR within a target ROI indicated by $\mathbf{r}$,

$$f(\mathbf{x}, \mathbf{c}, \mathbf{r}) = 1 - \frac{\|\mathrm{SNR}(\mathbf{x}, \mathbf{c}, \mathbf{r})\|}{\|\mathrm{UISNR}(\mathbf{r})\|}. \tag{5.38}$$

Here, $\mathbf{x}$ represents the geometric parameters of the coils, and $\mathbf{c}$ are the variable capacitors distributed around the coil conductors, which are optimized for coil tuning (see Section 5.5.4.1).

Note that, depending on the sample and frequency of operation, the UISNR values could diverge at voxels near the surface of the sample due to numerical instability [144]. To ensure the validity of our results, the ROIs considered in this work do not include voxels located within 2.5 cm of the surface of the sample.

For a meshed coil, we define the constraint $g(x)$ as

$$g(\mathbf{x}) = \min_{k \in C^a} d_k(\mathbf{x})$$

where C^a is the set of all possible pairs (e, v) of edges e and vertices v on the boundaries of coils, and d_k is the signed distance between a vertex and an edge in the pair k, where the sign is determined by the dot product with edge perpendicular pointing outside the coil.

Note that we do not compute $g(x)$ explicitly; rather, we define a potential B_g that ensures that $g(\mathbf{x})$ remains positive, and much less expensive to compute, as it uses only nearby pairs.

We constrain the objective function with a barrier potential term $B_g(x)$ that forces the coils to stay within the feasible domain, and prevent individual coils from self-intersecting (Section 5.3):

$$B_g(\mathbf{x}) = \kappa \sum_{k \in C} b(d_k(\mathbf{x}), \hat{d}). \tag{5.39}$$

$B_g(x)$ measures the sum of these barriers over each contact pair $k \in C \subset C^a$, which are sufficiently close. The parameter $\kappa > 0$ controls the barrier stiffness, which primarily affects the optimization efficiency. Following [114], we construct a continuous barrier energy b that creates a localized repulsion force when primitives are closer than a distance $\hat{d}$, and that vanishes otherwise (see Figure 5.1).

$$b(d, \hat{d}) = \begin{cases} -(d - \hat{d})^2 \ln\left(\frac{d}{\hat{d}}\right), & 0 < d < \hat{d} \\ 0 & d \geq \hat{d}. \end{cases} \tag{5.40}$$

In the definition of B_g, the set C can be restricted to pairs which are closer than $\hat{d}$. We use the logarithmic potential following [114]; this choice is far from unique, as long as the essential barrier properties are satisfied (infinite barrier at contact, zero value at a small distance to contact and smoothness w.r.t., geometric degrees of freedom). The logarithmic potential was demonstrated to work well in [114] and follow-up works, and is also commonly used in interior-point solvers for constrained optimization.

5.6.2 Optimization process and line search

The objective function (5.38) and the barrier potential term (5.39) are both smooth, allowing gradient-based optimization to be employed. The optimization algorithm consists of two steps. First, we optimize the size and location of the coils with only affine transformations by defining a scaling factor s and translation distances t_u and t_v for each coil. Second, we further optimize individual B-spline control point coordinates (p_u, p_v). The first step has $3 \times n$ parameters, and the second step has $2 \times m$ parameters to optimize, where n is the number of coils and m is the total number of control points.

In each step, we start with a grid search of the parameters, using q samples on each dimension, which requires $O(q^n)$ simulations. To reduce computational load, we use coordinate descent, i.e., we search for the optimal solution in one dimension at a time, and terminate when the objective can no longer decrease in any dimension. This reduces the complexity to $O(nq)$ simulations.

After the grid search, we continue with gradient-based optimization with gradient descent or L-BFGS. We compute the gradient of the objective from (5.35) by adding $\nabla f(x)$, computed with finite differences, and the gradient of the barrier potential $\nabla B_g(x)$. $\nabla B_g(x)$ is derived with a chain rule by multiplying the analytic derivative $\mathbf{D}B_g$ of the barrier potential term [114] and the shape velocity $\mathbf{D}v$ defined on mesh vertices with respect to the optimization parameters:

$$\mathbf{D}(B_g \circ v) = \mathbf{D}B_g\mathbf{D}v. \tag{5.41}$$

Next, we propose a continuous, intersection-aware line search for the optimization parameters. In each line search, we first apply an intersection-aware continuous collision detection (CCD) [114, 176] (StepSizeUpperBound in Algorithm 2) to conservatively compute the largest feasible step size along the gradient descent direction. We then apply a backtracking line search, bound by the largest feasible step size, to obtain an energy decrease. We terminate the optimization when the norm of gradient is smaller than $\epsilon_d = 10^{-4}$ or the line search fails to find an energy decrease with a maximum number of iterations $n_d = 30$ (Algorithm 2).

Algorithm 2 Gradient-based Optimization. ComputeConstraintSet determines the set of pairs of segments for which the contact potential is not zero.

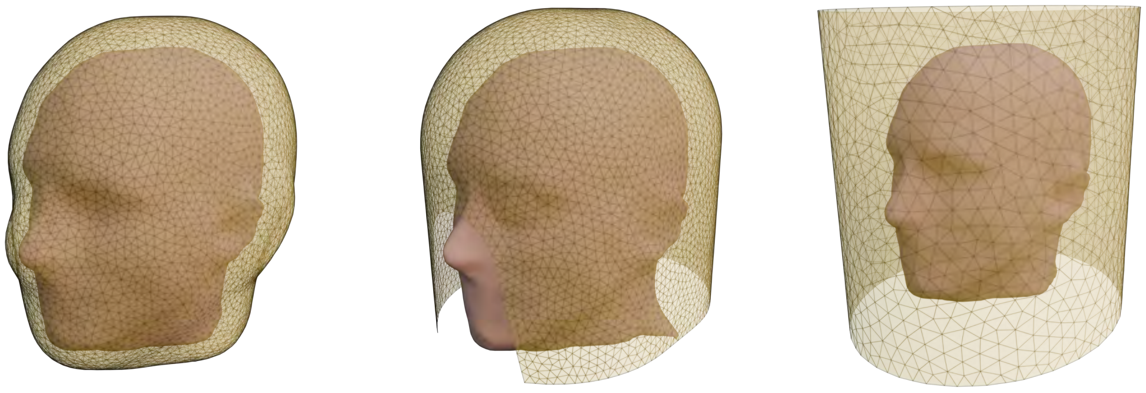
```
1: Input: Geometric parameters to be optimized $x_0$, tolerance $\epsilon_d$, maximum number of iterations for line search $n_d$
2: Output: Optimal geometric parameters $x$
3: $x \leftarrow x_0$
4: $C \leftarrow \text{ComputeConstraintSet}(x, \hat{d})$
5: $E_{\text{prev}} \leftarrow f(x) + B_g(x, \hat{d}, C)$
6: $x_{\text{prev}} \leftarrow x$
7: do
8:     $p \leftarrow -\nabla f(x) - \nabla_x B_g(x, \hat{d}, C)$
9:     $\alpha \leftarrow \min(1, \text{StepSizeUpperBound}(x, p, C))$          ▷ CCD line search
10:    $i \leftarrow 1$
11:    do
12:        if $i > n_d$ then                                                  ▷ Line search failed
13:            return $x$
14:        end if
15:        $x \leftarrow x_{\text{prev}} + \alpha p$
16:        $C \leftarrow \text{ComputeConstraintSet}(x, \hat{d})$
17:        $\alpha \leftarrow \alpha/2$
18:        $i \leftarrow i + 1$
19:    while $f(x) + B_g(x, \hat{d}, C) \geq E_{\text{prev}}$
20:    $E_{\text{prev}} \leftarrow f(x) + B_g(x, \hat{d}, C)$
21:    $x_{\text{prev}} \leftarrow x$
22: while $\|p\| > \epsilon_d$
23: return $x$
```

5.7 Methods

We first demonstrated the numerical issues that we addressed with the improved precision in the SIE assembly code (Section 5.5.2.1) by estimating gradients of the SNR cost function (Eq. (5.38)) of a simple parametrization of a single loop. We then conducted numerical experiments of increasing complexity to demonstrate the coil design optimization pipeline. We started by optimizing a single parameter – the horizontal placement of a single electric dipole (Section 5.7.3) – as we moved the target ROI. We then optimized the position and shape of three coils to maximize the average SNR within a spherical ROI at different locations inside a head model (Section 5.7.4). To evaluate our framework for different cost functions, we optimized four coils to maximize SNR on two decoupled ROIs (Section 5.7.5). Finally, we optimized a 12-coil head array for different ROIs in the brain (Section 5.7.6) and tested the generalization capacity of our optimization approach by evaluating the consistency of the results for a different head model. All computations were executed on a server running the Ubuntu 24.04.2 LTS operating system, equipped with an AMD Ryzen Threadripper PRO 3995WX CPU at 2.70 GHz, 64 cores, 2 threads per core and an NVIDIA GeForce RTX 3080 Ti GPU with 12 GB of memory.

5.7.1 Numerical samples

The experiments in Sections 5.7.3, 5.7.4, 5.7.5, 5.7.6 used the realistic Duke human model of the Virtual Family Population [60]. The computational domain enclosing the Duke's head model was $18.5 \times 23 \times 22.5$ cm^3 and was discretized over a uniform grid of 5 mm^3 voxel resolution, corresponding to $38 \times 47 \times 46$ voxels. For one experiment in Section 5.7.6, we used the Ella human model of the Virtual Family with a computational domain of $17.5 \times 21 \times 24$ cm^3, which was discretized over a uniform grid of 5 mm^3 voxel resolution, corresponding to $35 \times 42 \times 48$ voxels.



(a) Ultimate **(b)** Bell-Shaped **(c)** Elliptical Cylinder

Figure 5.12: Geometry of the ultimate (a), the bell-shaped (b), the cylindrical elliptical (c) current-bearing substrate surfaces. The ultimate surface closely surrounded the realistic head model (Duke).

5.7.2 Coil formers

To calculate the UISNR, we expanded the isosurface of Duke by approximately 2 cm and constructed a Hugyens' surface that fully surrounded the sample (Figure 5.12(a)) [57]. This basis support was discretized with 11 618 triangular elements, and a basis of incident fields was generated using RWG surface currents (see Section 5.5.1.5). For the coil SNR calculations, we modeled two realistic coil formers. The first former resembled a bell-shaped structure with its front region carved out (Figure 5.12(b)). The former was 24 cm long and spanned 25 cm in the x-direction and 20 cm in the y-direction. We used $9,205$ triangular elements for its discretization. The second former was an open elliptic cylinder of height $h = 28.5$ cm, semi-major axis length 14.7 cm, and semi-minor axis length 12.8 cm. We used 2 643 triangular elements for its discretization (Figure 5.12(c)). The SVD operations required for the basis generation method were performed using a threshold $\varepsilon = 10^{-3}$ (see Section 5.5.1.5).

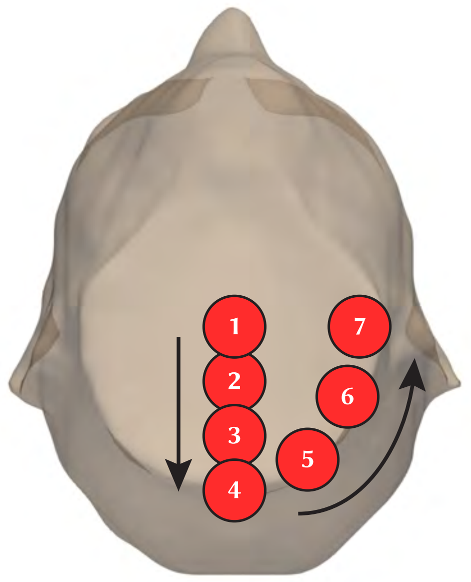


Figure 5.13: Schematic visualization of the seven locations of the spherical ROI for the three-coil optimization. All locations were centered on an axial plane at $z = 5\,\text{cm}$. A manual initial guess was defined before optimizing for the ROI at location 1. For each subsequent ROI location, the coil configuration in the last iteration of the optimization for the previous location was used as initial guess.

5.7.3 Optimization of the position of a single dipole

We optimized the position of a single dipole with length $l = 25\,\text{cm}$ operating at 14 T, mapped on the elliptic cylinder former (Figure 5.12(c)). The dipole was segmented with two capacitors positioned at one-quarter and three-quarters of the dipole length, which were adjusted at every step of the optimization to ensure tuning. We uniformly sampled seven voxels as the target optimization ROIs, along an elliptical trajectory on the transverse plane at $z = 5\,\text{cm}$. The dipole was placed vertically to the X-Y plane. Starting from a position behind the head, we optimized the horizontal placement of the dipole on the elliptic cylinder substrate for each ROI independently. This was performed as a two-step optimization: grid search followed by a gradient-based optimization.

5.7.4 Three-coil optimization with a moving spherical ROI

We optimized a three-element array at 3 T. The loops were mapped to the bell-shaped former (Figure 5.12(b)), discretized with triangular elements of average edge length 6 mm, and segmented with seven capacitors for tuning. The values of the capacitors were adjusted to ensure tuning and ideal decoupling at every iteration of the optimization. The optimization target was the average SNR within a spherical ROI defined on a transverse plane $z = 5$ cm with a radius $r = 2$ cm. The ROI was moved to different locations within the head (Figure 5.13). For location 1, we initialized the optimization with an arbitrary initial guess for the shape and position of the three coils. We then performed a four-step optimization: grid search on the coil size and position with only affine transformation, gradient-based optimization on the coil size and position, grid search on the control point position, and gradient-based optimization on the control point position (see Section 5.6.2). For locations 2 to 7, the initial guess was the last iteration of the second-step optimization (after affine transformation) for the previous ROI location.

5.7.5 Four-coil optimization with two regions-of-interest

We optimized a four-element loop array at 3 T with two separate voxels as the optimization target, one in the frontal part and the other in the rear part of the head. We also introduced a weighted formulation of the cost function in Equation (5.38):

$$f'(\mathbf{x}, \mathbf{l}, \mathbf{r}) = w_1 f(\mathbf{x}, \mathbf{l}, \mathbf{r}_1) + w_2 f(\mathbf{x}, \mathbf{l}, \mathbf{r}_2), \tag{5.42}$$

where w_1 and w_2 are weights that prioritize the relative importance of voxel $\mathbf{r}_1$ or $\mathbf{r}_2$ for the SNR optimization.

5.7.6 12-coil optimization

We performed three experiments aimed at optimizing a 12-coil receive array for 3 T brain imaging. We defined two ROIs: one for the cerebrum and one that combined the white matter (WM), gray matter (GM), and cerebrospinal fluid (CSF) of the Duke's head model. We used the cost-function of equation (5.42). In the first experiment, we optimized the average SNR performance with equal weighting for the two ROIs; in the second experiment, we assigned a larger weight to the cerebrum; in the last experiment, we optimized the array to maximize SNR performance solely over the cerebellum. Finally, to evaluate the generalizability of the optimization process, we loaded the optimized coil configuration of the first experiment with Ella's head model and calculated the SNR performance.

5.8 Results

5.8.1 Numerical precision of the VSIE

We used a simple loop geometry with initial node coordinates $\mathbf{r}_i$, one port, and three tuning capacitors. As the parametrization for the coordinates, we use a simple affine transformation

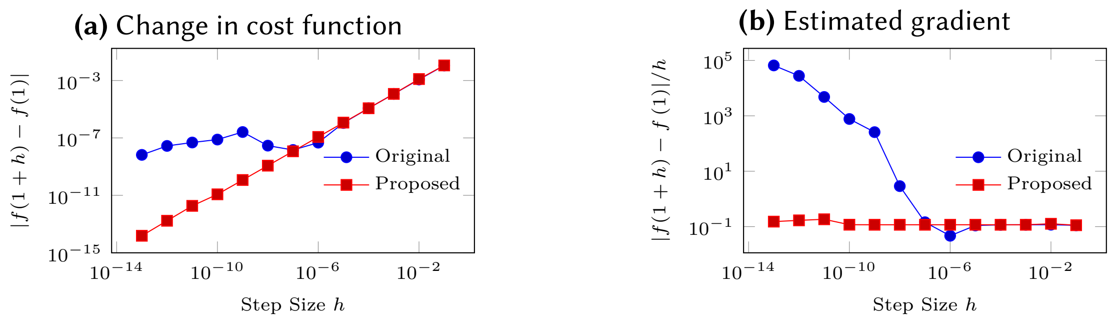


Figure 5.14: Change in cost function and estimated gradient of cost function about $\alpha = 1$ as a function of Forward Euler finite difference step size using original implementation and proposed implementation.

with parameter α. The parametrized coordinates $\mathbf{r}(\alpha)$ are defined as

$$\mathbf{r}(\alpha) = \alpha(\mathbf{r}_i - \langle \mathbf{r}_i \rangle) + \langle \mathbf{r}_i \rangle \tag{5.43}$$

with $\mathbf{r}(1) = \mathbf{r}_i$ and $\langle \mathbf{r}_i \rangle$ denoting the mean of the initial coordinates.

We estimated the gradient using the original implementation with limited precision and with the proposed implementation with enhanced precision for a number of step sizes. We estimated the gradient of the cost function about $\alpha = 1$ using the Forward Euler approximation of the derivative, which is given by

$$\left.\frac{\partial f}{\partial \alpha}\right|_{\alpha=1} \approx \frac{f(1+h) - f(1)}{h} \tag{5.44}$$

for step size h.

Fig. 5.14 shows the change in the cost function and the estimated gradient magnitude as a function of step size h, using the existing and proposed approaches. As can be seen in Fig. 5.14(b), estimating the gradient of the SNR cost function fails catastrophically when the finite-difference step size is too small, even for a simple affine transformation that results in all nodes changing position. Fig. 5.14(a) shows that below $h = 10^{-5}$, the change in the cost function becomes independent of the step size, explaining why estimating the gradient fails catastrophically.

5.8.2 Coil design optimization experiments

Without the MRGF, the iterative optimization process would have been intractable. For example, the MRGF approach resulted in a solution time, including tuning and decoupling, of 15 s for each three-coil design, as opposed to a few minute when assembling the full system in Equation (5.16). The full solution would take several minutes for the 12-coil array, whereas it required 32 s with MRGF. The optimization of the dipole position required seven minutes for each ROI location. The length of the full coil optimization ranged between 5 and 21 hours for the seven three-coil optimization experiments (Section 5.7.4), and between a week and two weeks for the

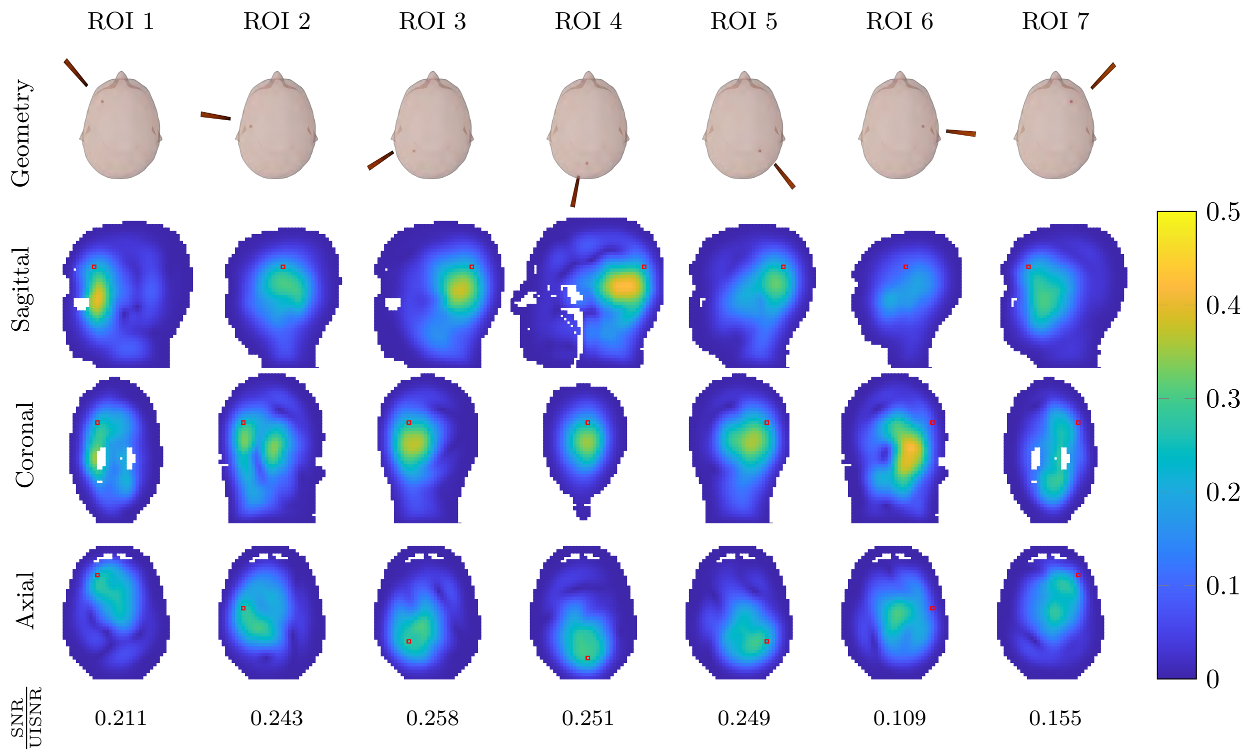


Figure 5.15: Optimization of the absolute SNR performance of a dipole at 600 MHz (14 T), as the target ROI is moved around the head. The dipole was constrained to move azimuthally on a cylindrical former surrounding the head. From top to bottom row: position of the dipole resulting from the optimization, with the red voxel indicating the target ROI; performance maps showing the percentage of the UISNR achieved by the dipole along sagittal, coronal, and axial sections cutting through the target voxel (in red). The absolute performance for each configuration is reported at the bottom.

four 12-coil optimization experiments (Section 5.7.6). Note that the exact duration of the individual iterations is not reported since it varied during the optimization based on the convergence of each step of the grid search and gradient-based optimization.

Figure 5.15 shows that the optimization algorithm adapts the position of the dipole as the position of the target ROI moves around the head. Since in this experiment we did not optimize the shape of the coil but only its azimuthal position, the relative SNR performance of the dipole was lower for ROI 6 and ROI 7 because they are closer to the surface of the head, where the UISNR grows exponentially.

Figures 5.17–5.30 show the results of the three-coil optimization for the seven positions of the spherical ROI.

For each experiment, there are two figures. The first figure shows the evolution of the cost function during the four steps of the optimization: grid search on the coil size and position with only affine transformation (Grid Search 1), gradient-based optimization on the coil size and position (Gradient-based optimization 1), grid search on the control point position (Grid Search 2), and gradient-based optimization on the control point position (Gradient-based optimization 2). The second figure shows the evolution of the coil design for different iterations, together with the associated performance maps for three orthogonal section of the head cutting through the center of the spherical ROI. In these and subsequent figures, the coils were constrained to be on a bell-shaped former surrounding the head. In each figure, from top to bottom, the rows show: geometry and ROI (in red); performance maps showing the percentage of the UISNR achieved by each configuration along sagittal, coronal, and axial sections cutting through the center of the ROI (contour shown in red). The mean and maximum performance within the ROI is reported at the bottom.

For all cases, the average ratio of SNR over the UISNR within the ROI increased monotonically with the number of iterations. The performance was lower when the ROI was closer to the surface of the head, where it is more difficult to approach the UISNR [187]. Since the three coils were constrained to the bell-shaped former, the performance difference among ROI positions was also affected by the distance between the former and the surface of the head. Table 5.2 summarizes the results of the three-coil experiments. Except for ROI 1, for which the optimization was initialized to a random geometry, in all other cases the initial guess was the coil geometry optimized for the previous ROI position. The values in Table 5.2 show that the cost function decreased in each case. The optimization algorithm converged to a different coil geometry as the ROI moved to a different position. Figure 5.16 shows that the three-coil geometry optimized for ROI 4 is consistent with the corresponding ideal current patterns associated with the UISNR.

Using Equation (5.38), the four-coil optimization starting from an arbitrary initial guess con-

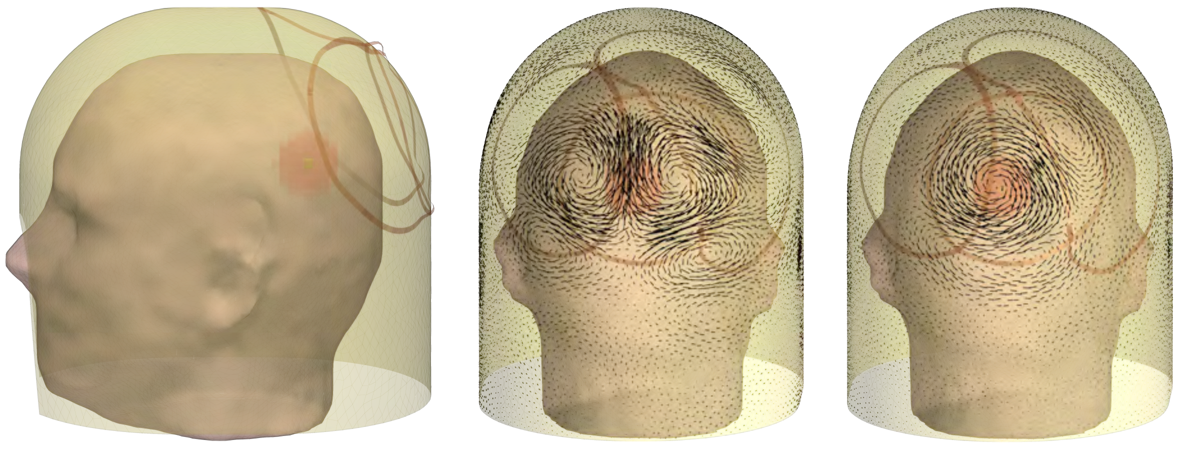

(a) Geometry **(b)** ICP ($\omega t = 0$) **(c)** ICP ($\omega t = \frac{\pi}{2}$)

Figure 5.16: Realistic coil former surrounding the head model along with optimal coil configuration for spherical ROI 4 ((a)). The former also served as a current-bearing surface for the estimation of the ideal current patterns (ICP). Temporal snapshots of the ICP yielding optimal signal-to-noise ratio at the center of the spherical ROI 4 for two time-points are shown in ((b)) and ((c)).

verged to a geometry in which three coils were clustered near the target voxel at the back of the head and one coil was located near the target voxel in the front (Figures 5.31 and 5.32). When the target voxels were separately considered, using Equation (5.42) with $w_1 = w_2 = 0.5$, the optimized configuration yielded two coils in the front of the head and two coils at the back (Figures 5.33 and 5.34). The optimization with the weighted cost function achieved higher SNR performance than the one that considered both voxels as a single target ROI.

For all 3 T 12-coil array experiments, we used an arbitrary initial guess with loop elements surrounding the head. Figures 5.35 and 5.36 show the results using equal weights $w_1 = w_2 = 0.5$ for the two ROIs. The optimized coil design achieved 7% higher performance at the center of the head and improved the averaged SNR performance over the entire brain (both ROIs) by 9%. The average performance in the cerebrum and WM-GM-CSF was 57.8% and 47.4%, respectively. Figures 5.37 and 5.38 show the optimization results when weights $w_1 = 0.6$ and $w_2 = 0.4$ are used for the cerebellum and WM-GM-CSF, respectively. The optimized SNR performance in the center of the head was 6% higher than the initial guess, and the averaged SNR performance over the entire brain improved by 8%. The average performance in the cerebrum and WM-GM-CSF

Table 5.2: Summary of three-coil optimization results. For each position of the spherical ROI (Figure 5.13), we report: the cost function of the initial guess, the cost function at the end of the optimization, the average and maximum performance within the ROI after the optimization.

ID	Cost Function (first iteration) %	Cost Function (final iteration) %	Average SNR/UISNR %	Maximum SNR/UISNR %
ROI 1	64.4	26.4	74.7	83.9
ROI 2	35.7	23.2	77.9	88.0
ROI 3	38.5	29.8	73.2	86.3
ROI 4	60.9	54.8	54.5	79.7
ROI 5	63.8	57.7	52.8	79.0
ROI 6	72.5	65.9	46.7	75.2
ROI 7	84.2	80.2	34.1	66.2

was 57.5% and 46.5%, respectively. Figures 5.39 and 5.40 show the results when optimizing for the cerebellum alone ($w_1 = 1$, $w_2 = 0$). The region with the highest performance shifted to the center of the cerebellum, where the performance improved on average by 14%. The average SNR performance in the cerebrum and WM-GM-CSF was 59.2% and 27.2%, respectively. Note that this example was included only for completeness, because it would neither be practical nor useful to construct a 12-coil array to image exclusively the cerebellum. Figures 5.41 and 5.42 show the results when the cerebellum ROI was excluded from the optimization target ($w_1 = 0$, $w_2 = 1$). In this case, the optimized coil design achieved 8.5% higher performance at the center of the head and improved the averaged SNR performance over the entire brain by 10%. The average SNR performance in the cerebrum and WM-GM-CSF was 42.2% and 48.9%, respectively.

Figures 5.43 and 5.44 show the performance of the coil designs found at the completion of each optimization step (when $w_1 = 0.5$, $w_2 = 0.5$) in Figures 5.35 and 5.36 when Ella's head model is used instead of Duke's one. These results show that the optimized coil geometries at different optimization steps can also monotonically improve the SNR performance on Ella's brain. The final optimized coil configuration increased the SNR performance in the center of Ella's brain by 8% and improved the average SNR performance over the entire brain by 9%.

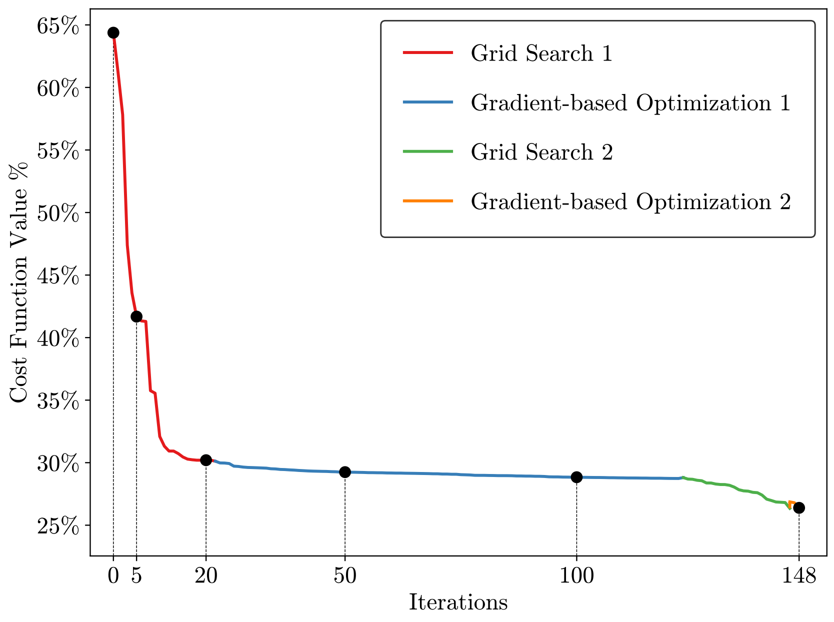


Figure 5.17: The cost function evolution for three-coil optimization with the spherical ROI in region 1.

5.9 Discussion

The goal of this work was to develop an optimization framework for rational coil design for MRI applications. We developed an automatic tool that combines geometry processing techniques with state-of-the-art EM simulations to optimize coils using the UISNR as the reference. We eliminated user interactions, whereas setting up a single simulation could take days with the current coil design software. To achieve automation and computational efficiency, several innovations were introduced in this work. The new approach to solving the singular integrals during the assembly of the SIE operators considerably increases numerical precision, which is critical when these operators are used in iterative optimizations. We also introduced a novel C++ implementation that dramatically accelerates the assembly of the SIE operators. To perform iterative shape optimization of coil configurations, we needed to develop a rapid coil meshing method, including the automatic implementation of bridges to avoid contact between overlapping coil conductors, a feature not available in commercial coil design software. To efficiently simulate

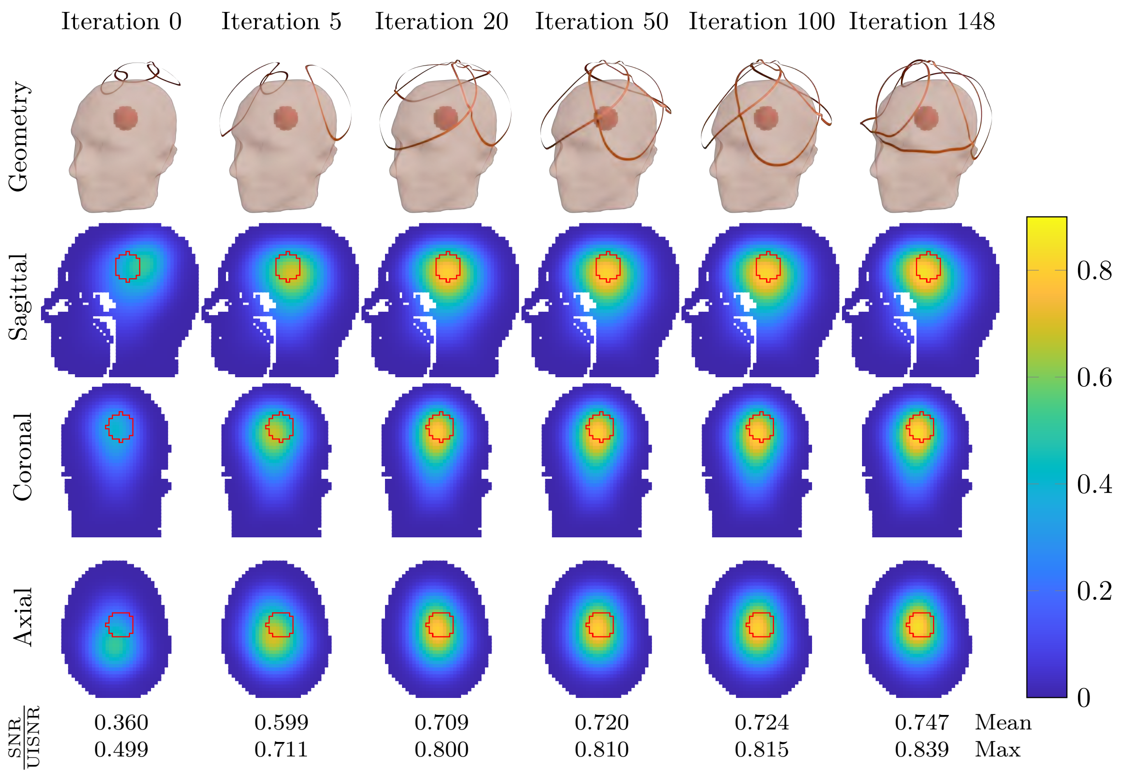


Figure 5.18: Evolution of the three-coil geometry during the iterative optimization of the average SNR performance within the spherical ROI in region 1.

coils at each iteration, we developed an algorithm for automatic coil tuning and a method for ideal coil decoupling.

Previous work showed that ideal current patterns associated with the UISNR can provide a qualitative target to design coils that approach the optimum performance [57, 100]. Here, we have shown that certain coil configurations found by the optimization algorithm resemble the ideal current patterns, confirming graphically the near-optimality of common surface quadrature coil designs. Previous work on coil optimization cannot be directly compared with our results, since it was limited to one or two coils, used different design targets, and was performed in the quasi-static regime or using simplified analytical models. This is the first approach to use full-wave EM simulations and ultimate performance limits in the cost function to optimize coils for high-field MRI. As a proof of principle, in this work, we showed that it is computationally feasible to auto-

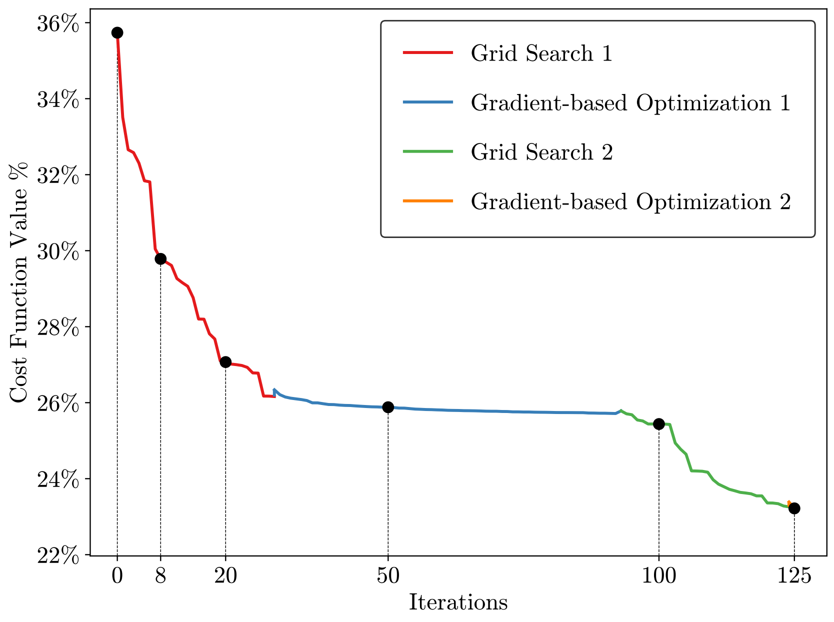


Figure 5.19: The cost function evolution for three-coil optimization with the spherical ROI in region 2.

matically optimize a 12-coil receive head array at 3 T using only grid search and gradient descent, achieving an absolute SNR performance in the brain region comparable to that of state-of-the-art 32-coil arrays [103]. The objective function that iteratively evaluates new coil geometries against the UISNR is smooth, which is favorable for gradient descent. However, optimizing denser arrays can become increasingly inefficient, so in future work we plan to calculate the gradients analytically using the Hermitian adjoint solution of the VSIE as in other optimization problems that utilize it [58, 169]. This will also allow us to broaden the design space parametrization, which in this work was limited to the size and position of the coils. We included various design constraints to ensure that the optimization would converge to coil configurations that could be practically constructed. Additional constraints could be added in the future to accommodate design requirements for particular clinical applications. Different constraints would also be needed to optimize transmit and transceive coils. For example, the ideal decoupling strategy developed for this work would not be feasible for transmitters since preamplifiers are not used during transmission. The objective function would also need to be modified to account for different metrics, such as the

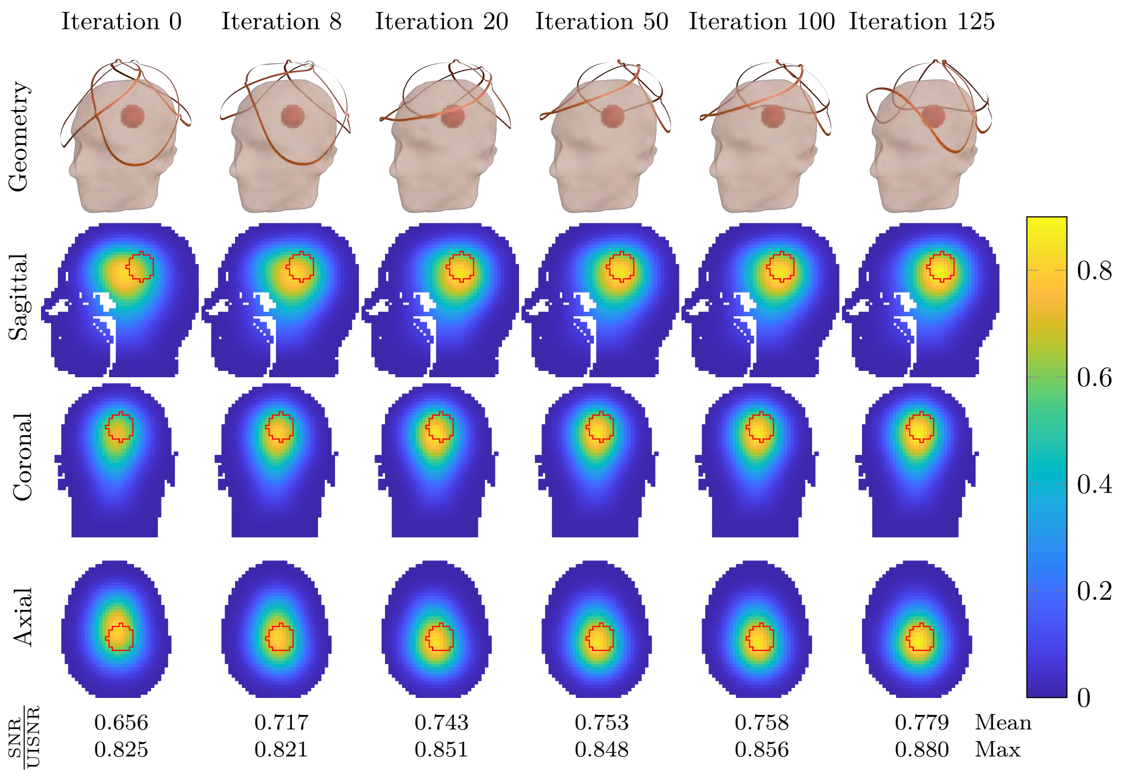


Figure 5.20: Evolution of the three-coil geometry during the iterative optimization of the average SNR performance within the spherical ROI in region 2.

ultimate intrinsic SAR [62, 102] or the optimal transmit efficiency [50]. Note that the proposed optimization framework is flexible and generalizable enough to enable the incorporation of these changes in a straightforward manner.

The proposed optimization framework affects the entire flow of coil design and evaluation, and we expect that it will result in coils that yield higher performance than currently available coils. Since previous studies have shown that traditional coil designs cannot approach the ultimate SNR performance at ultra-high field MRI [101, 192, 216], this project could lead to the discovery of novel coil types that yield superior signal encoding capabilities. The coil designs generated by the automated pipeline would be directly importable into computer-aided design (CAD) environments for rapid prototyping, making application-specific, tailored coils a practical possibility.

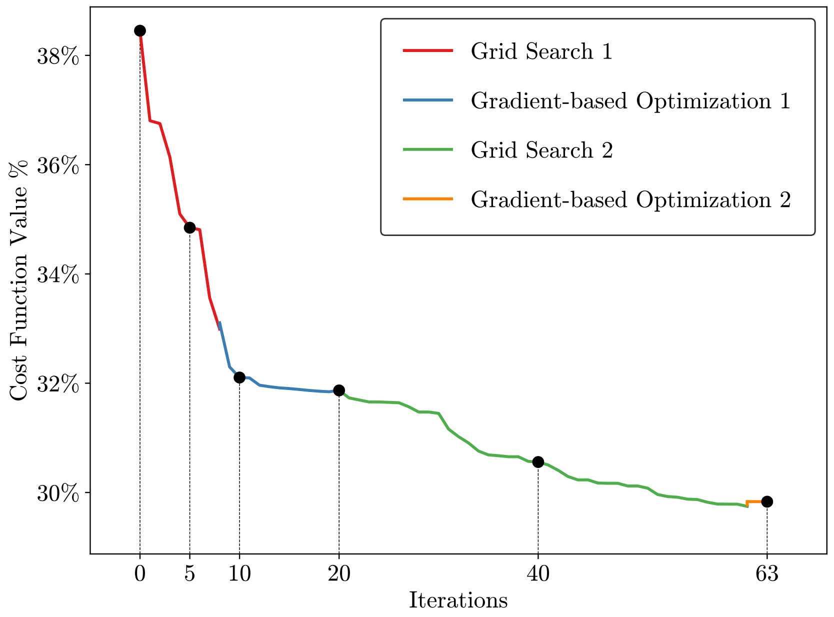


Figure 5.21: The cost function evolution for three-coil optimization with the spherical ROI in region 3.

This work aimed to introduce the new optimization framework. Several extensions will be pursued in future work, in addition to the already mentioned adjoint formulation and the implementation of other target optimization metrics. For example, replacing PWC with PWL basis functions would improve the accuracy of the UISNR [52]. However, the memory footprint and computational complexity would increase considerably, requiring the incorporation of novel strategies for tensor compression and efficient computation. We also plan to incorporate a wire integral equation solver to efficiently optimize coils with wire conductors, which are often preferred over flat copper strips to reduce coupling in arrays with many coil elements.

5.10 Conclusion

This work introduced the first fully automated software pipeline for optimizing coil design using ultimate performance limits as the reference benchmark. The pipeline integrates novel approaches for rapid EM simulations, shape optimization, and coil meshing. Several improvements

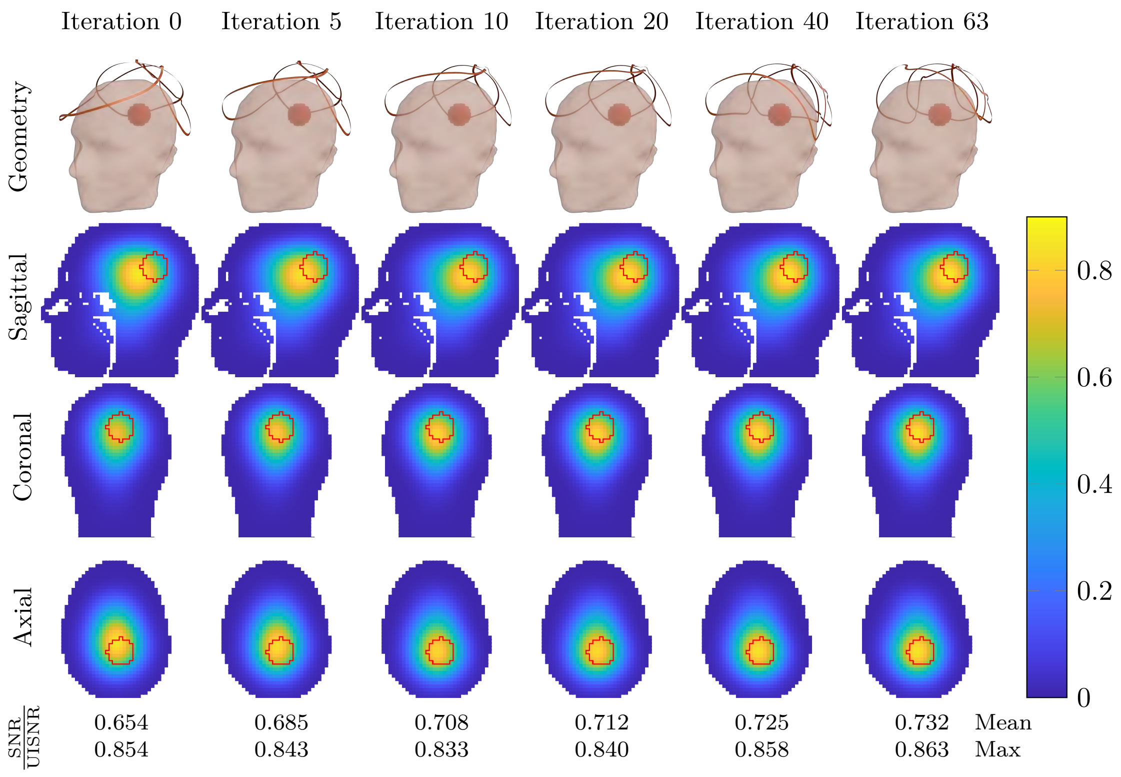


Figure 5.22: Evolution of the three-coil geometry during the iterative optimization of the average SNR performance within the spherical ROI in region 3.

and new features were incorporated into MARIE, an open-source VSIE solver tailored to MRI simulations. These include orders of magnitude higher precision in the assembly of the SIE operators, automatic coil tuning, and ideal decoupling of array elements. The coil design pipeline was demonstrated for optimizing head receive arrays with respect to the UISNR, for an increasing number of elements and different optimization target regions. The optimizations monotonically converged in all cases. The optimized 12-coil design at 3 T yielded approximately 10% higher SNR performance over an extended region of the brain compared to an arbitrary designed array.

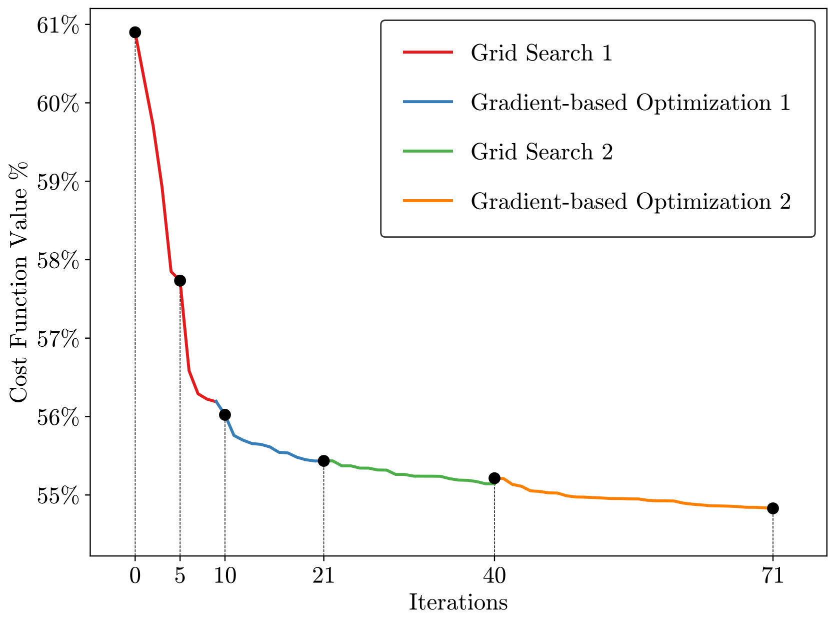


Figure 5.23: The cost function evolution for three-coil optimization with the spherical ROI in region 4.

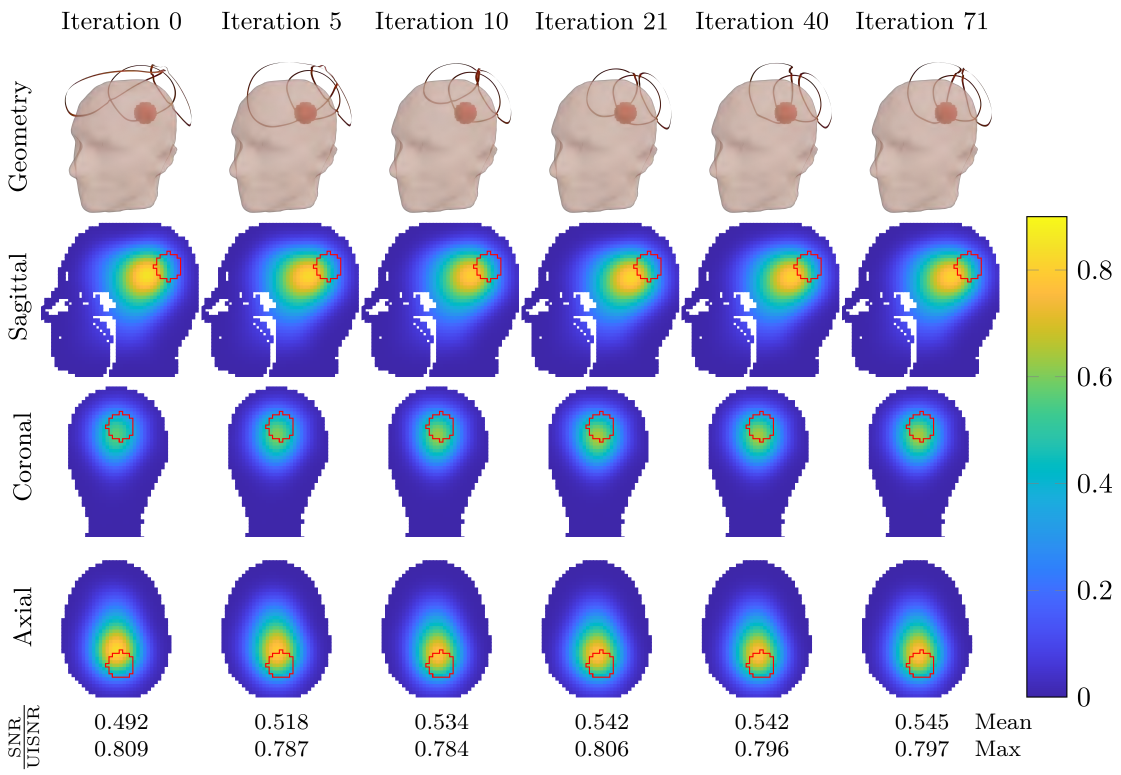


Figure 5.24: Evolution of the three-coil geometry during the iterative optimization of the average SNR performance within the spherical ROI in region 4.

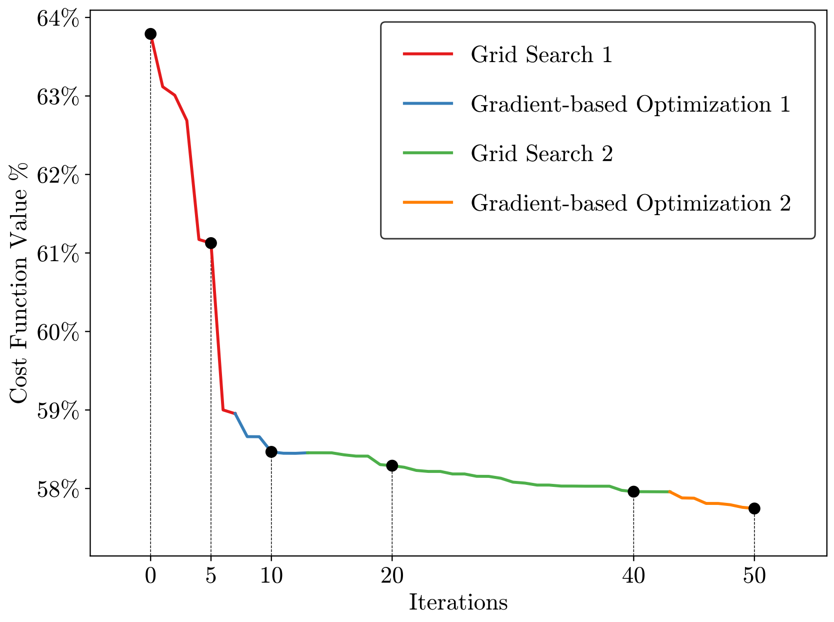


Figure 5.25: The cost function evolution for three-coil optimization with the spherical ROI in region 5.

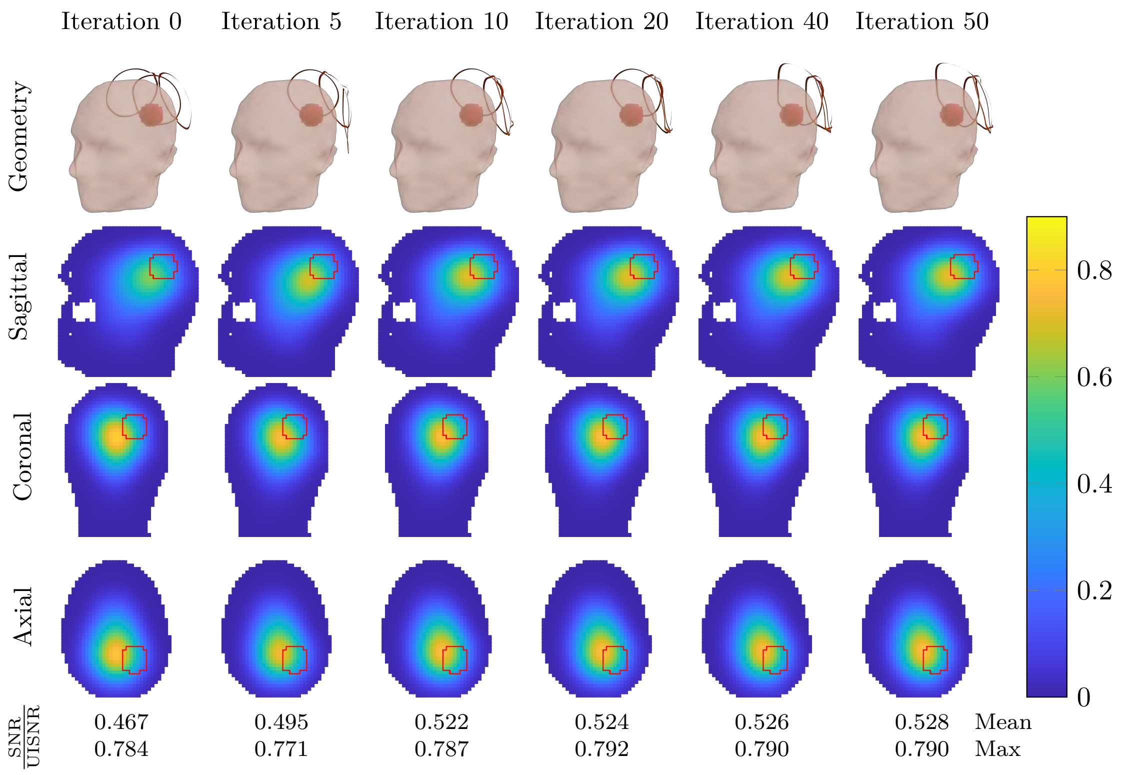


Figure 5.26: Evolution of the three-coil geometry during the iterative optimization of the average SNR performance within the spherical ROI in region 5.

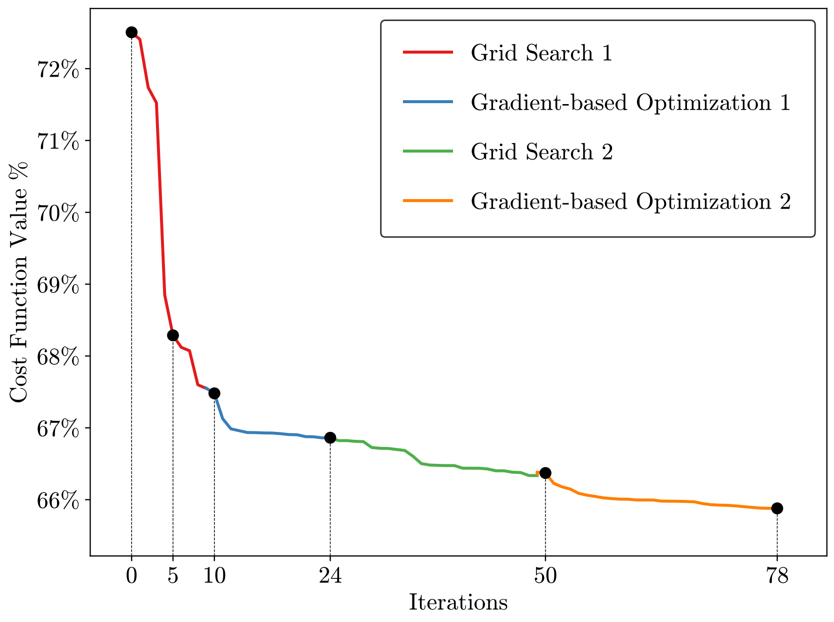


Figure 5.27: The cost function evolution for three-coil optimization with the spherical ROI in region 6.

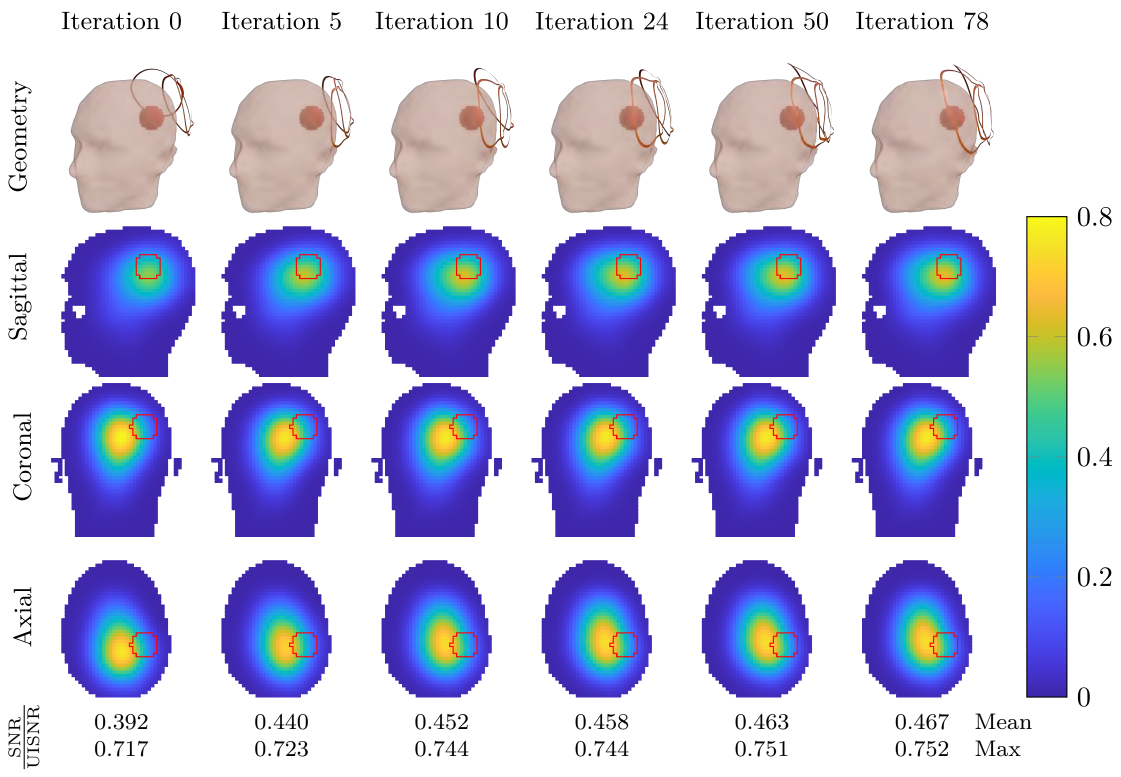


Figure 5.28: Evolution of the three-coil geometry during the iterative optimization of the average SNR performance within the spherical ROI in region 6.

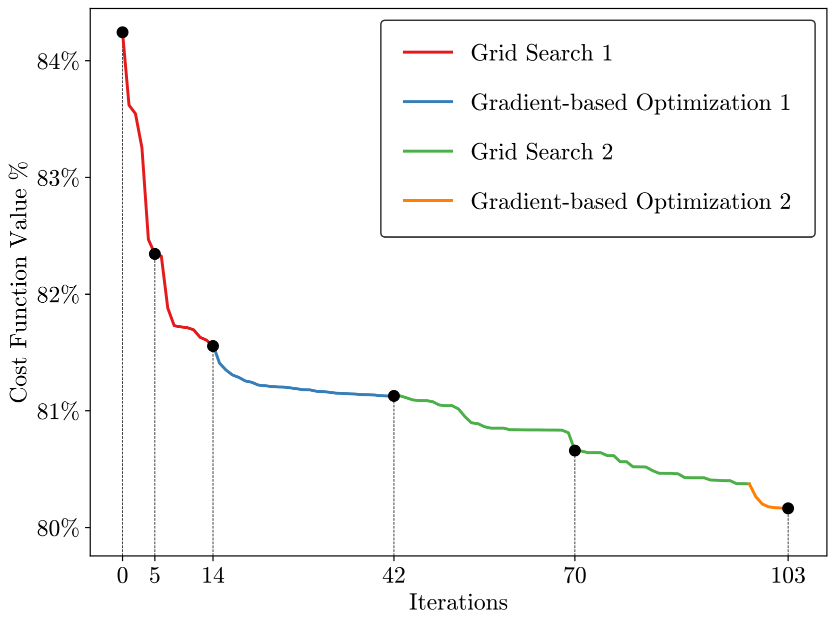


Figure 5.29: The cost function evolution for three-coil optimization with the spherical ROI in region 7.

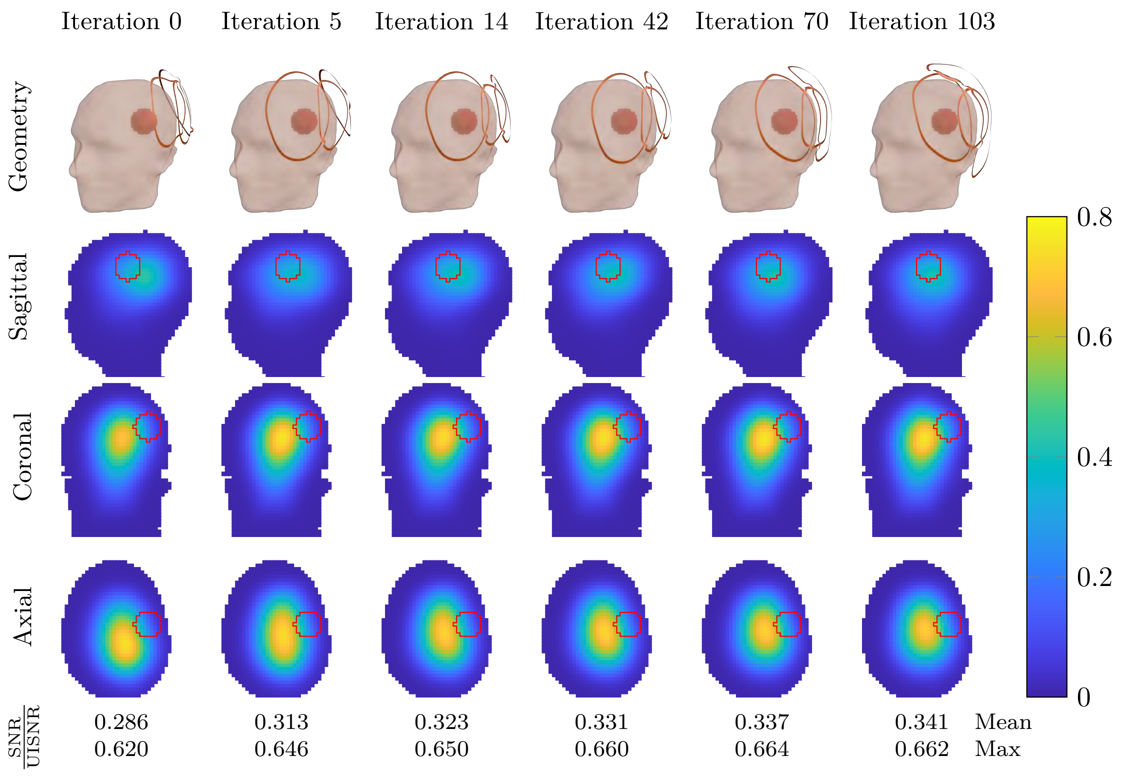


Figure 5.30: Evolution of the three-coil geometry during the iterative optimization of the average SNR performance within the spherical ROI in region 7.

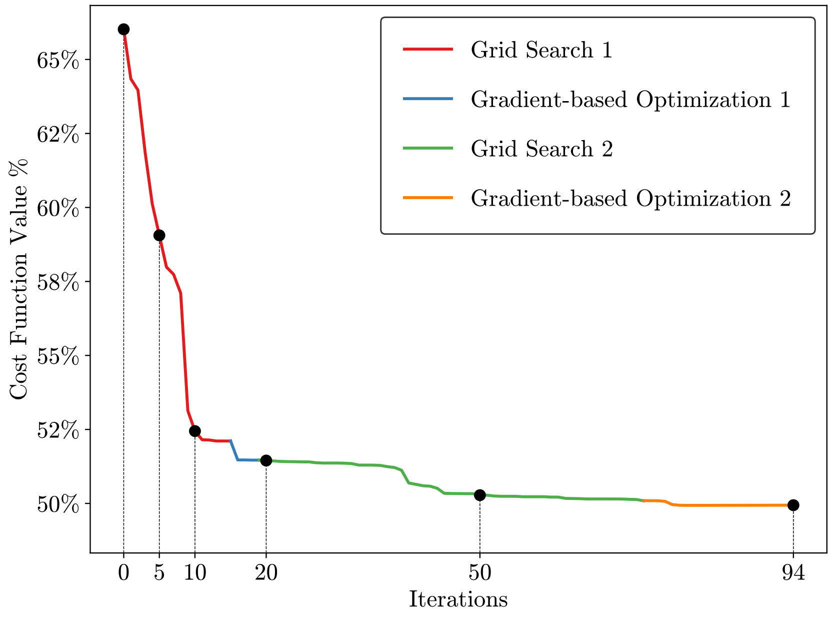


Figure 5.31: The cost function evolution for four-coil optimization with a two-voxel ROI.

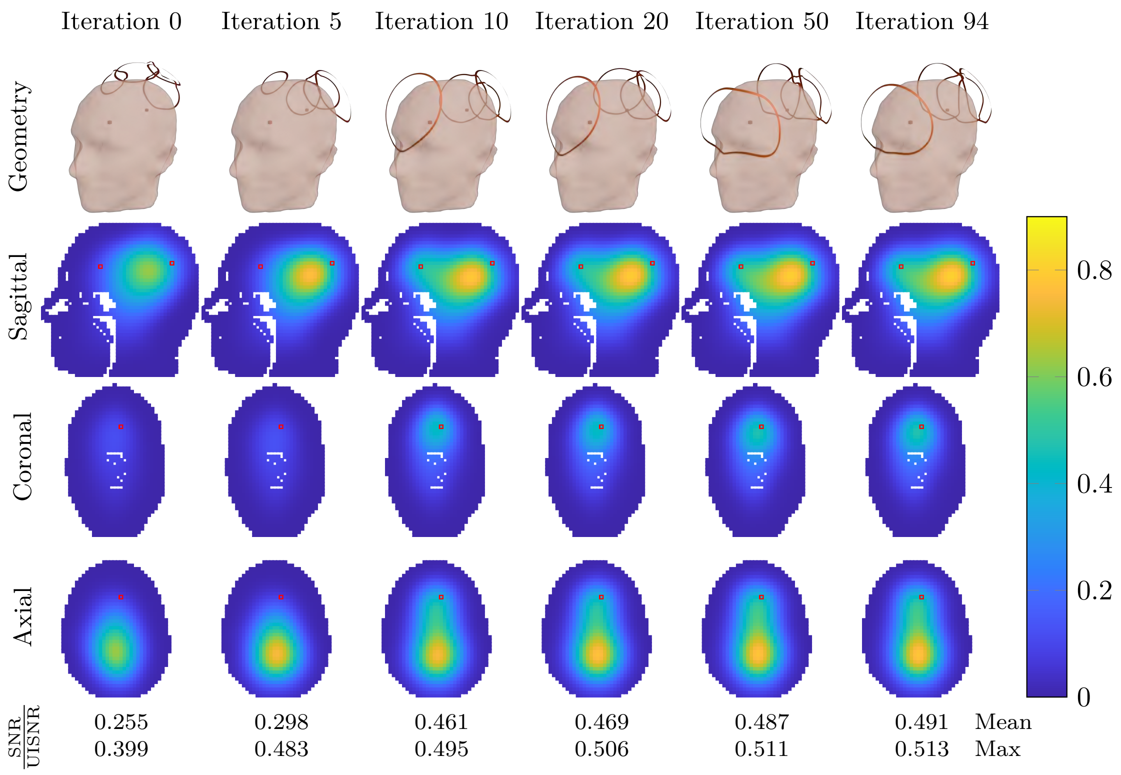


Figure 5.32: Evolution of the four-coil geometry during the iterative optimization of the average SNR performance within a two-voxel ROI.

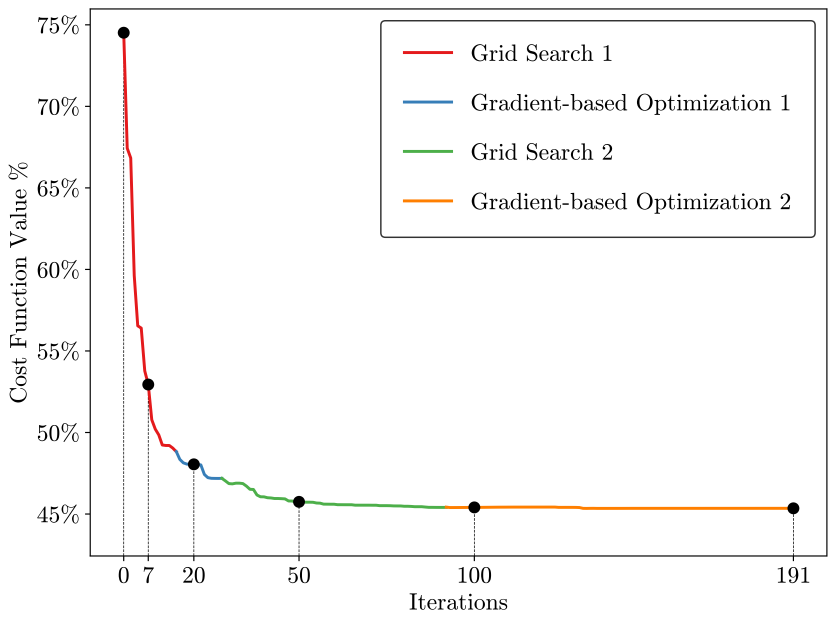


Figure 5.33: The cost function evolution for four-coil optimization with two target voxels, equally weighted and considered as separate ROIs.

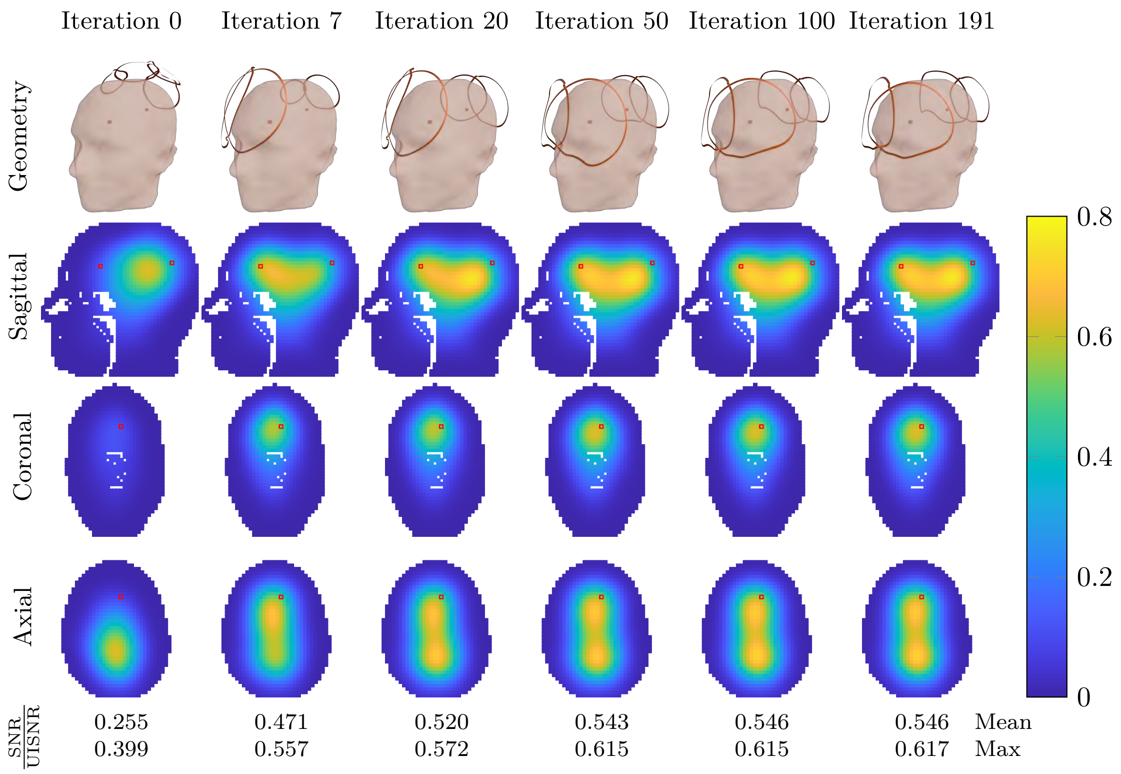


Figure 5.34: Evolution of the four-coil geometry during the iterative optimization of the average SNR performance within two target voxels, equally weighted and considered as separate ROIs.

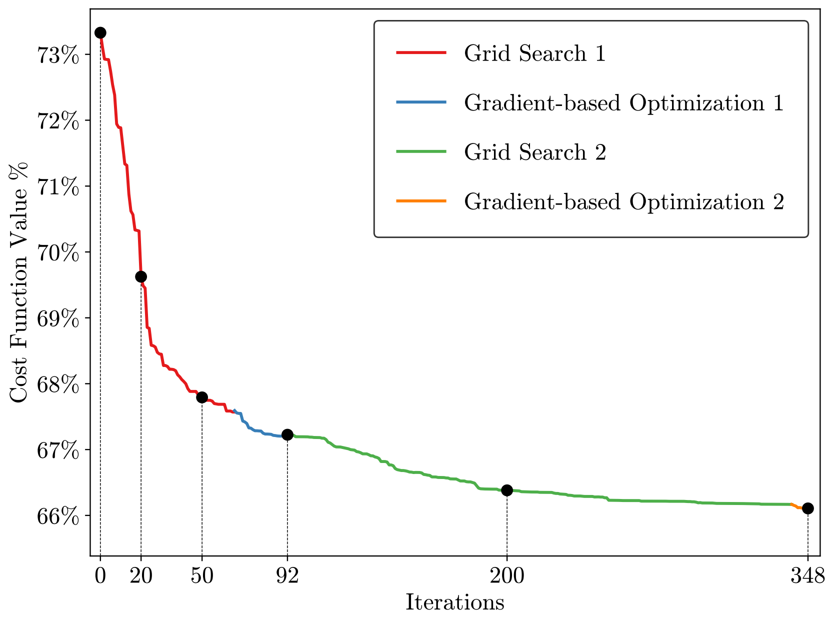


Figure 5.35: The cost function evolution for 12-coil optimization with the cerebellum and cerebrum ROIs, equally weighted ($w_1 = w_2 = 0.5$).

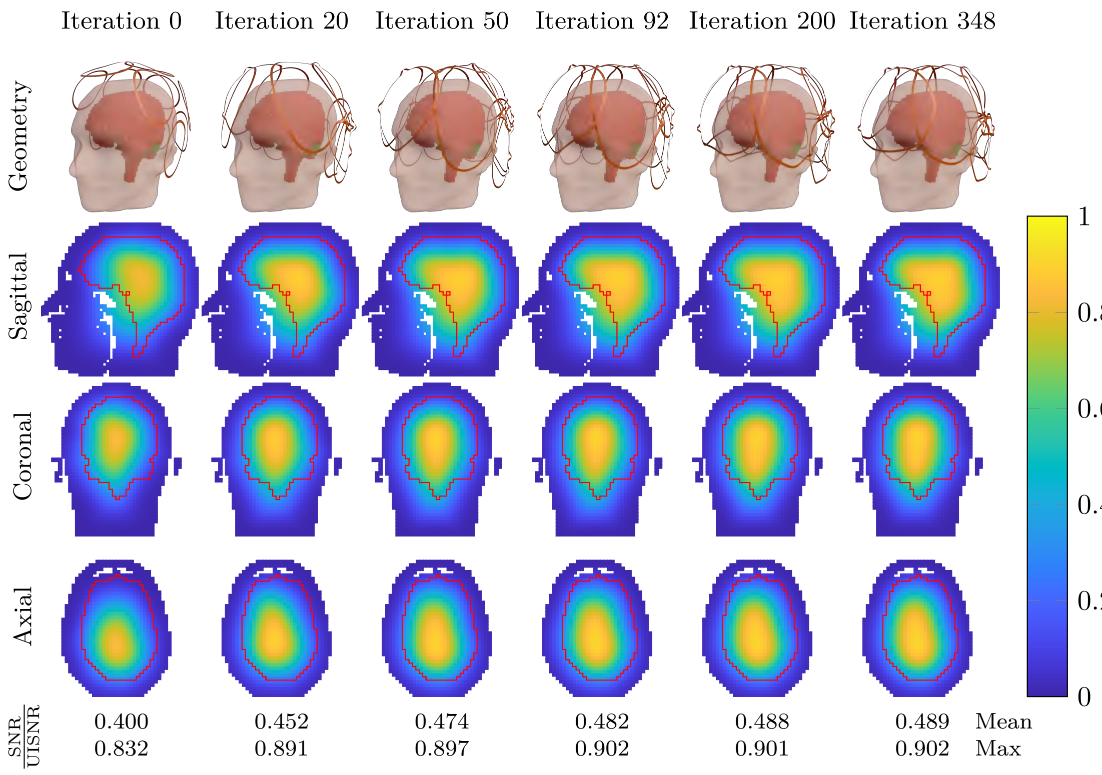


Figure 5.36: Evolution of the 12-coil geometry during the iterative optimization of the average SNR performance within the cerebellum and cerebrum ROIs, equally weighted ($w_1 = w_2 = 0.5$).

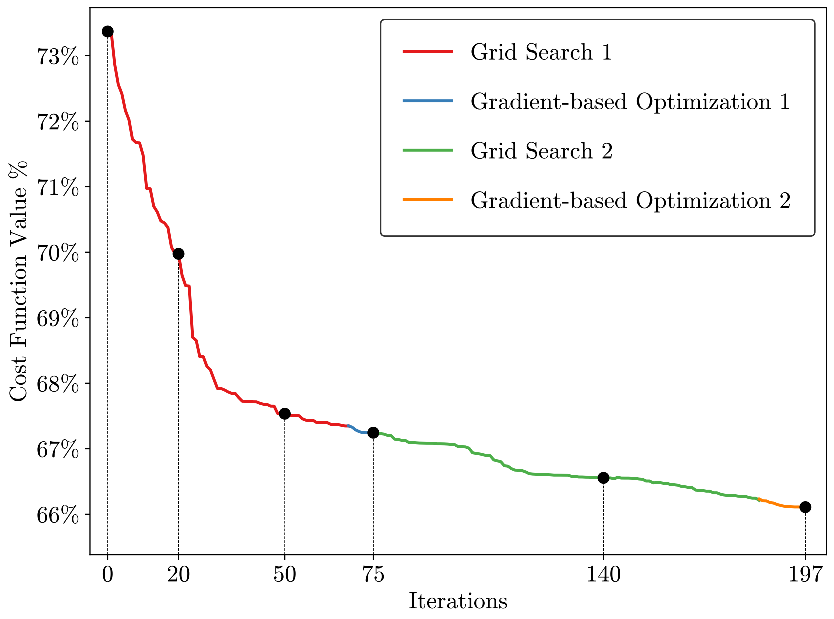


Figure 5.37: The cost function evolution for 12-coil optimization with the cerebellum and cerebrum ROIs, with weights $w_1 = 0.6$, $w_2 = 0.4$.

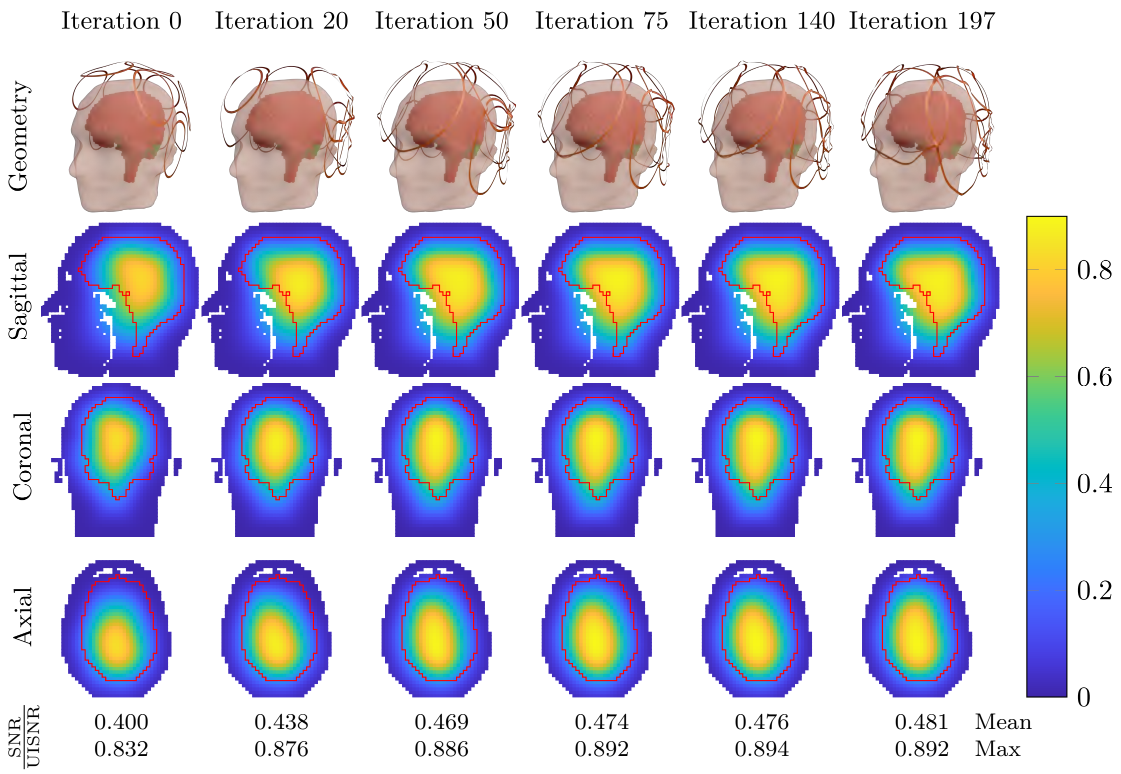


Figure 5.38: Evolution of the 12-coil geometry during the iterative optimization of the average SNR performance within the cerebellum and cerebrum ROIs, with weights $w_1 = 0.6$, $w_2 = 0.4$.

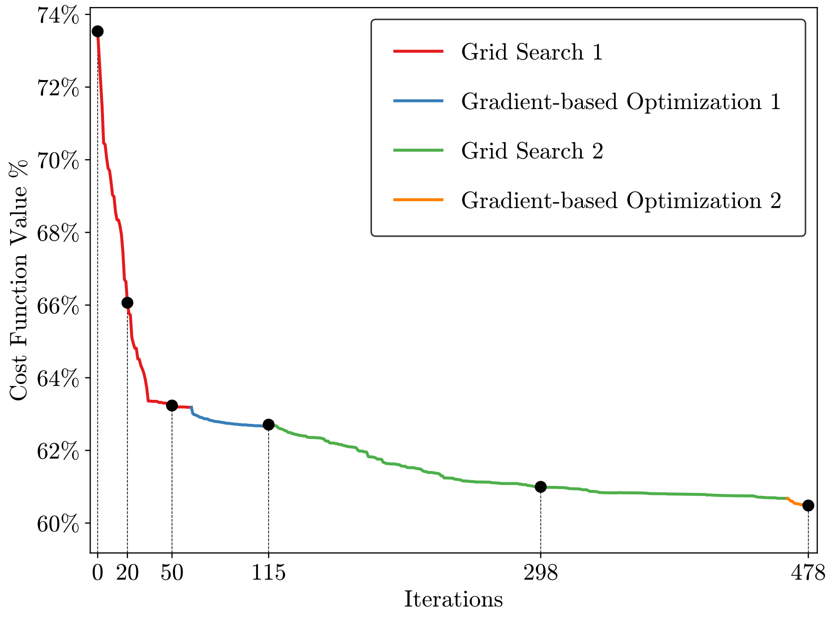


Figure 5.39: The cost function evolution for 12-coil optimization with the cerebellum ROI.

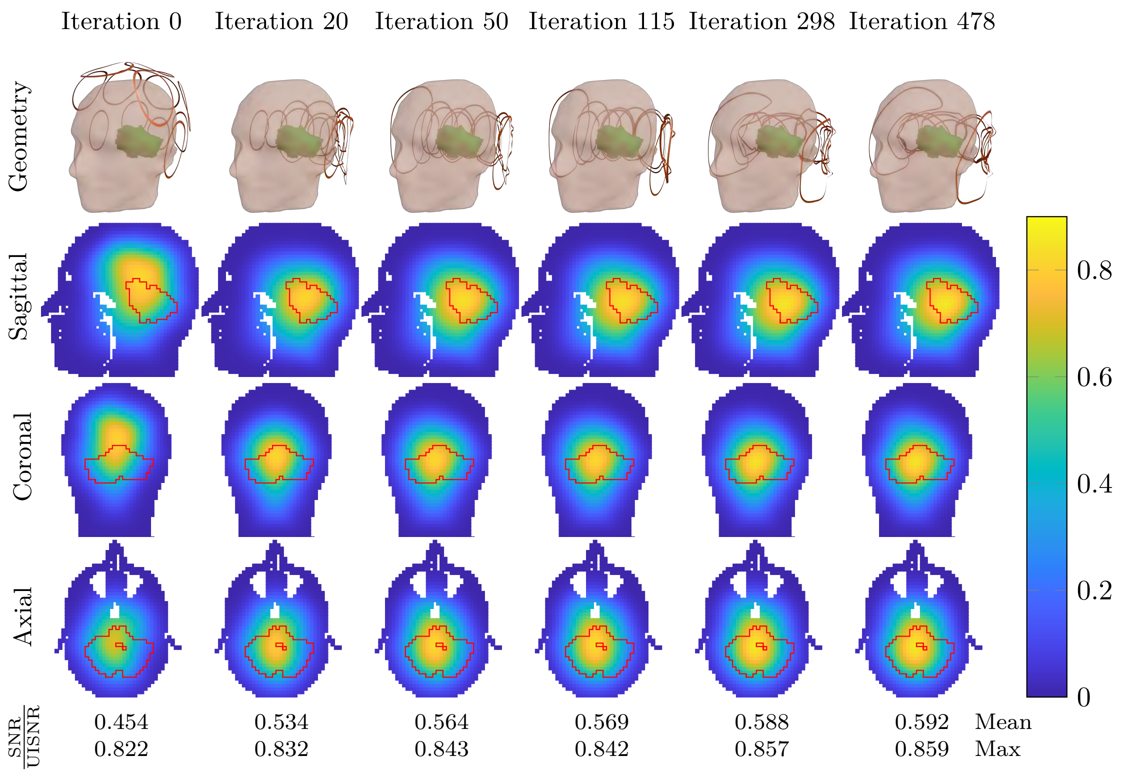


Figure 5.40: Evolution of the 12-coil geometry during the iterative optimization of the average SNR performance within the cerebellum ROI.

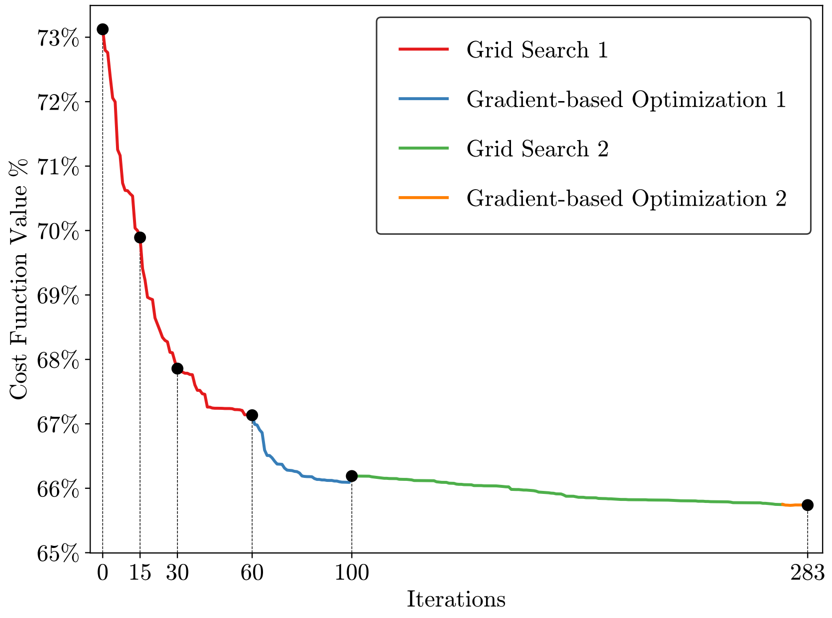
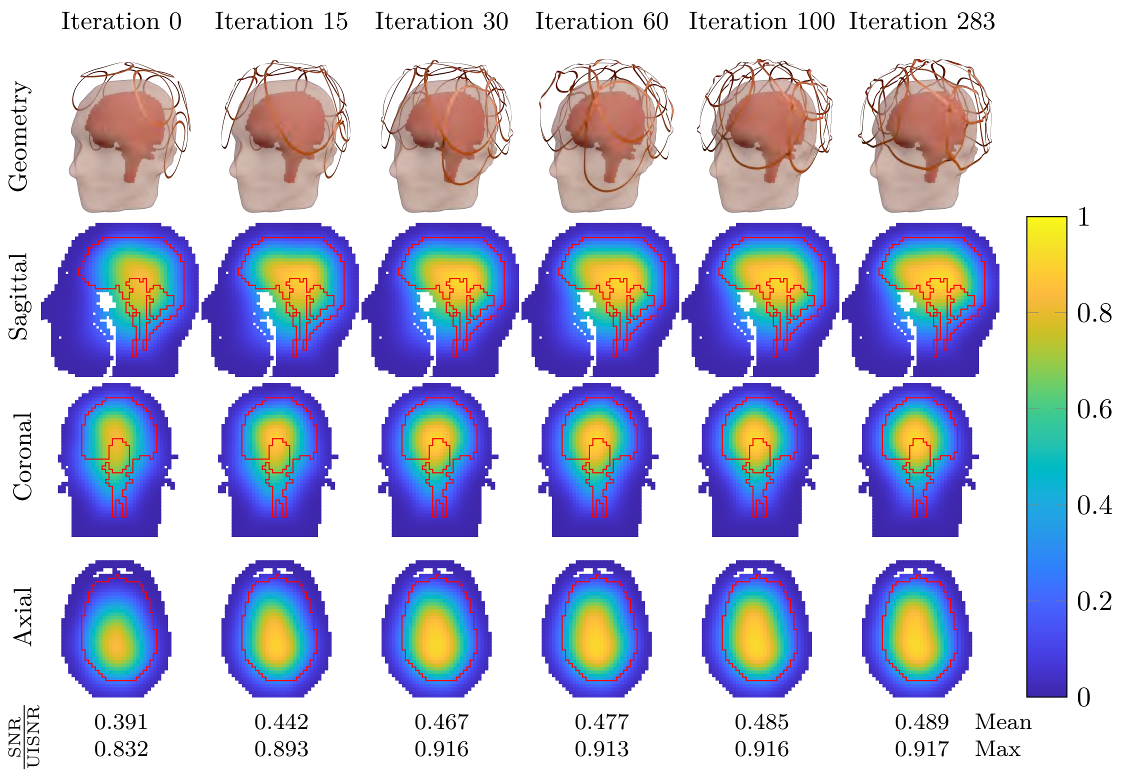


Figure 5.41: The cost function evolution for 12-coil optimization with the cerebrum ROI.



	Iteration 0	Iteration 15	Iteration 30	Iteration 60	Iteration 100	Iteration 283	
$\frac{\text{SNR}}{\text{UISNR}}$	0.391	0.442	0.467	0.477	0.485	0.489	Mean
	0.832	0.893	0.916	0.913	0.916	0.917	Max

Figure 5.42: Evolution of the 12-coil geometry during the iterative optimization of the average SNR performance within the cerebrum ROI.

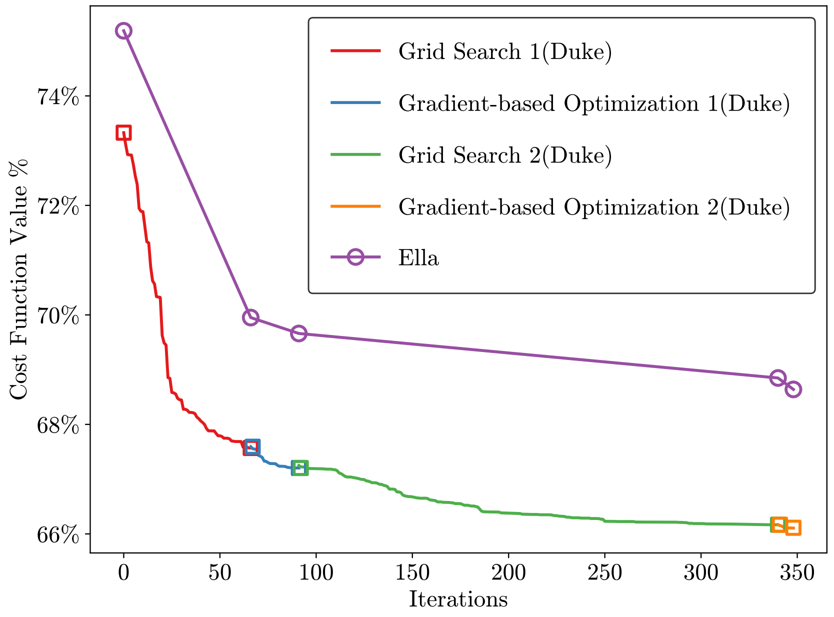


Figure 5.43: The cost function evolution for 12-coil optimization with the cerebellum and cerebrum ROIs using the Ella head model.

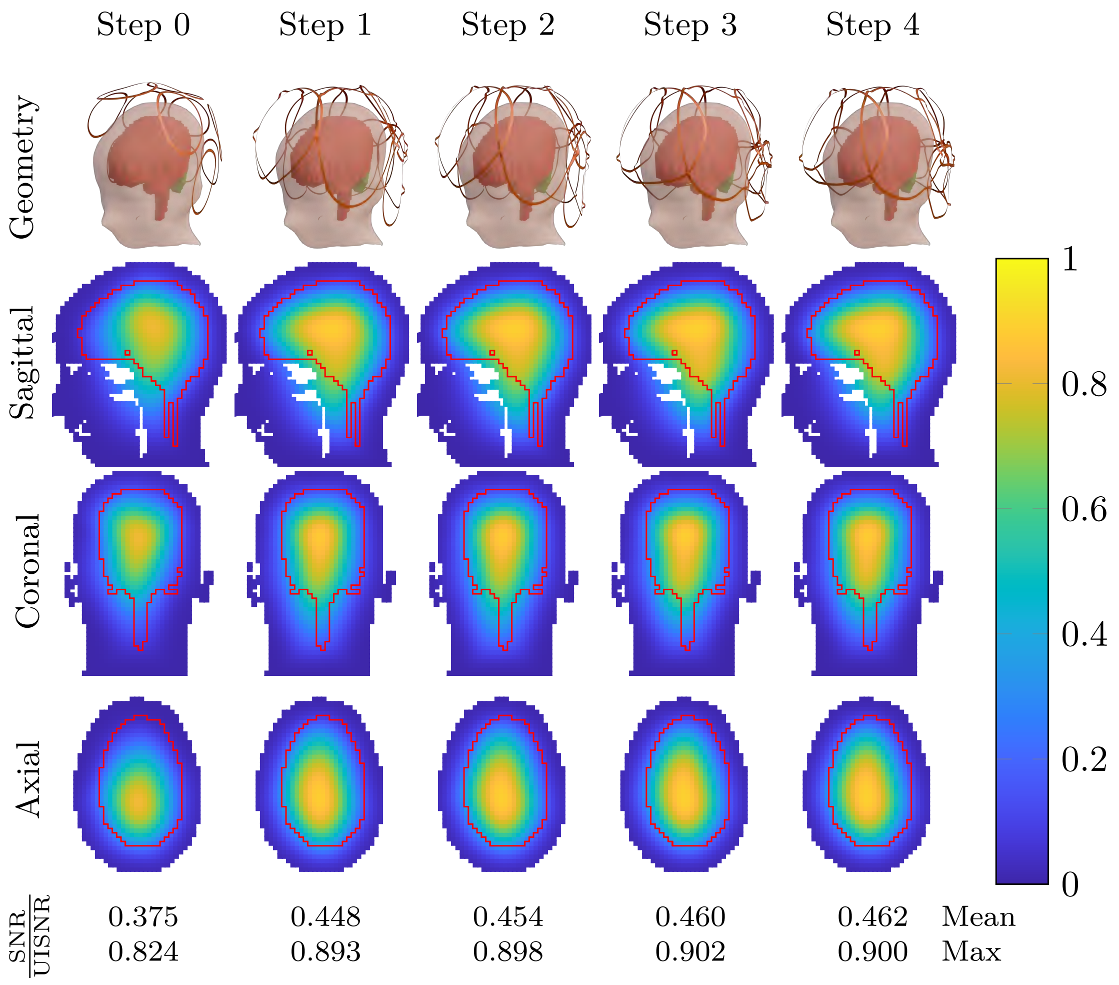


Figure 5.44: Evolution of the 12-coil geometry during the iterative optimization of the average SNR performance within the cerebellum and cerebrum ROIs using the Ella head model.

6 Conclusion

To summarize, this thesis advances shape design, repair, and optimization through contributions spanning fidelity-preserving vector graphics simplification, minimal-input solid/shell mesh labeling, neural-driven texture synthesis, and physics-based MRI coil optimization. First, our simplification method reduces vector artwork to the minimal number of curves while maintaining exceptional accuracy, preserving the artist's intent. Its effectiveness has been demonstrated through large-scale evaluations, and it has been extended to handle curves of varying thickness as well as vector graphics animations. Second, our repair tool streamlines the conversion of "in-the-wild" assets into formats suitable for simulation, solid modeling, or 3D printing with orders-of-magnitude less manual effort. Finally, our optimization frameworks preserve texture appearance while altering tactile roughness or temperature with neural acceleration, and deliver the first fully automated software pipeline for optimizing coil design using ultimate performance limits as the reference benchmark. Collectively, these techniques provide a unified, practical framework for high-fidelity, application-aware geometry processing, enabling efficient design workflows in computer graphics, manufacturing, and biomedical imaging.

Bibliography

[1] Pankaj K. Agarwal. *Lecture 23: Hausdorff and Frechet distance*. 2007.

[2] Sameer Agarwal, Keir Mierle, and The Ceres Solver Team. *Ceres Solver*. Version 2.1. Mar. 2022.

[3] Marc Alexa and Wolfgang Müller. "Representing animations by principal components". In: *Computer Graphics Forum*. Vol. 19. 3. Wiley Online Library. 2000, pp. 411–418.

[4] G. Allaire and M. Qatu. "Shape Optimization by the Homogenization Method. Applied Mathematical Sciences, Vol 146". In: *Applied Mechanics Reviews* 56 (2003). DOI: 10.1115/1.1553443.

[5] Pierre Alliez et al. "3D Mesh Generation". In: *CGAL User and Reference Manual*. 6.0.1. CGAL Editorial Board, 2024.

[6] Marco Attene. "Direct repair of self-intersecting meshes". In: *Graph. Model.* 76.6 (2014), pp. 658–668.

[7] Marco Attene, Marcel Campen, and Leif Kobbelt. "Polygon mesh repairing: An application perspective". In: *ACM Computing Surveys (CSUR)* 45.2 (2013), pp. 1–33.

[8] Marco Attene et al. "On converting sets of tetrahedra to combinatorial and PL manifolds". In: *Computer Aided Geometric Design* 26.8 (2009), pp. 850–864.

[9] Constantine A Balanis. *Advanced engineering electromagnetics*. John Wiley & Sons, 2012.

[10] Gavin Barill et al. “Fast winding numbers for soups and clouds”. In: *ACM Transactions on Graphics (TOG)* 37.4 (2018), pp. 1–12.

[11] Connelly Barnes and Fang-Lue Zhang. “A survey of the state-of-the-art in patch-based synthesis”. In: *Computational Visual Media* (2016), pp. 1–18. ISSN: 2096-0662. DOI: 10.1007/s41095-016-0064-2.

[12] Martin P. Bendsøe. “Topology Optimization”. In: *Encyclopedia of Optimization*. Ed. by Christodoulos A. Floudas and Panos M. Pardalos. 2nd. Boston, MA: Springer, 2008, pp. 3928–3929.

[13] Martin P. Bendsøe and Ole Sigmund. *Topology Optimization: Theory, Methods, and Applications*. Berlin, Germany: Springer, 2004.

[14] Gino van den Bergen. “Efficient collision detection of complex deformable models using AABB trees”. In: *Journal of graphics tools* 2.4 (1997), pp. 1–13.

[15] K. Bertoldi et al. “Flexible mechanical metamaterials”. In: *Nature Reviews Materials* 2 (2017), p. 17066. DOI: 10.1038/NATREVMATS.2017.66.

[16] Stephan Bischoff, Tobias Weyand, and Leif Kobbelt. “Snakes on triangle meshes”. In: *Bildverarbeitung für die Medizin 2005: Algorithmen—Systeme—Anwendungen Proceedings des Workshops vom 13.–15. März 2005 in Heidelberg*. Springer. 2005, pp. 208–212.

[17] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. 2006.

[18] Stephen P Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[19] Phil Brodatz. *Testures: A Photographic Album for Artists and Designers*. Dover, 1966.

[20] Marcel Campen and Leif Kobbelt. “Exact and Robust (Self-)Intersections for Polygonal Meshes”. In: *Comput. Graph. Forum* 29.2 (2010), pp. 397–406.

[21] Hongyi Cao et al. “A Parallel Feature-preserving Mesh Variable Offsetting Method with Dynamic Programming”. In: *CoRR* abs/2310.08997 (2023). DOI: 10.48550/ARXIV.2310.08997.

[22] Hongyi Cao et al. “Robust and Feature-Preserving Offset Meshing”. In: *CoRR* abs/2412.15564 (2024). DOI: 10.48550/ARXIV.2412.15564.

[23] Hung-Hsin Chang and Hong Yan. “Vectorization of hand-drawn image using piecewise cubic Bezier curves fitting”. In: *Pattern recognition* 31.11 (1998), pp. 1747–1755.

[24] Saifon Chaturantabut and Danny C Sorensen. “Nonlinear model reduction via discrete empirical interpolation”. In: *SIAM Journal on Scientific Computing* 32.5 (2010), pp. 2737–2764.

[25] Desai Chen et al. “Spec2Fab: a reducer-tuner model for translating specifications to 3D prints”. In: *ACM Transactions on Graphics (TOG)* 32.4 (2013), p. 135.

[26] Hanyu Chen, Bailey Miller, and Ioannis Gkioulekas. “3D Reconstruction with Fast Dipole Sums”. In: *ACM Trans. Graph.* 43.6 (2024), 192:1–192:19. DOI: 10.1145/3687914.

[27] Zhen Chen et al. “Robust Low-Poly Meshing for General 3D Models”. In: *ACM Trans. Graph.* 42.4 (2023), 119:1–119:20.

[28] Gianmarco Cherchi et al. “Interactive and Robust Mesh Booleans”. In: *ACM Trans. Graph.* 41.6 (2022), 248:1–248:14.

[29] Mircea Cimpoi et al. “Describing textures in the wild”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 3606–3613.

[30] Charles E Connor et al. “Tactile roughness: neural codes that account for psychophysical magnitude estimates”. In: *The Journal of neuroscience* 10.12 (1990), pp. 3823–3836.

[31] Boris Dalstein, Rémi Ronfard, and Michiel Van De Panne. “Vector graphics animation with time-varying topology”. In: *ACM Transactions on Graphics (TOG)* 34.4 (2015), pp. 1–12.

[32] Carl De Boor, Klaus Höllig, and Malcolm Sabin. "High accuracy geometric Hermite interpolation". In: *Computer Aided Geometric Design* 4.4 (1987), pp. 269–278.

[33] Donald Degraen, André Zenner, and Antonio Krüger. "Enhancing Texture Perception in Virtual Reality Using 3D-Printed Hair Structures". In: (2019).

[34] Alejandro R Diaz and Ole Sigmund. "A topology optimization method for design of negative permeability metamaterials". In: *Structural and Multidisciplinary Optimization* 41.2 (2010), pp. 163–177.

[35] Lorenzo Diazzi and Marco Attene. "Convex polyhedral meshing for robust solid modeling". In: *ACM Transactions on Graphics (TOG)* 40.6 (2021), pp. 1–16.

[36] Lorenzo Diazzi et al. "Constrained Delaunay Tetrahedrization: A Robust and Practical Approach". In: *ACM Transactions on Graphics (TOG)* 42.6 (2023), pp. 1–15.

[37] Paul Dierckx. *Curve and surface fitting with splines.* Oxford University Press, 1995.

[38] David H Douglas and Thomas K Peucker. "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature". In: *Cartographica: the international journal for geographic information and geovisualization* 10.2 (1973), pp. 112–122.

[39] Xingyi Du et al. "Optimizing global injectivity for constrained parameterization". In: *ACM Trans. Graph.* 40.6 (2021), 260:1–260:18.

[40] Van Than Dung and Tegoeh Tjahjowidodo. "A direct method to solve optimal knots of B-spline curves: An application for non-uniform B-spline curves fitting". In: *PloS one* 12.3 (2017), e0173857.

[41] Alexei A Efros and Thomas K Leung. "Texture synthesis by non-parametric sampling". In: *iccv*. IEEE. 1999, p. 1033.

[42] Oskar Elek et al. “Scattering-aware texture reproduction for 3D printing”. In: *ACM Transactions on Graphics (TOG)* 36.6 (2017), p. 241.

[43] Galal Elkharraz et al. “Making tactile textures with predefined affective properties”. In: *Affective Computing, IEEE Transactions on* 5.1 (2014), pp. 57–70.

[44] Nicole Feng, Mark Gillespie, and Keenan Crane. “Winding Numbers on Discrete Surfaces”. In: *ACM Trans. Graph.* 42.4 (2023), 36:1–36:17. DOI: 10.1145/3592401.

[45] Kaspar Fischer. *Piecewise linear approximation of Bézier curves*. 2000.

[46] Alexander IJ Forrester and Andy J Keane. “Recent advances in surrogate-based optimization”. In: *Progress in aerospace sciences* 45.1-3 (2009), pp. 50–79.

[47] Hiroyuki Fujita et al. “A hybrid inverse approach applied to the design of lumped-element RF coils”. In: *IEEE transactions on biomedical engineering* 46.3 (1999), pp. 353–361.

[48] Michael Garland and Paul S Heckbert. “Surface simplification using quadric error metrics”. In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. 1997, pp. 209–216.

[49] Leon Gatys, Alexander S Ecker, and Matthias Bethge. “Texture synthesis using convolutional neural networks”. In: *Advances in neural information processing systems*. 2015, pp. 262–270.

[50] Ioannis P Georgakis, Athanasios G Polimeridis, and Riccardo Lattanzi. “A formalism to investigate the optimal transmit efficiency in radiofrequency shimming”. en. In: *NMR Biomed.* 33.11 (Nov. 2020), e4383.

[51] Ioannis P Georgakis et al. “A fast volume integral equation solver with linear basis functions for the accurate computation of EM fields in MRI”. In: *IEEE Transactions on Antennas and Propagation* 69.7 (2020), pp. 4020–4032.

[52] Ioannis P Georgakis et al. “Novel numerical basis sets for electromagnetic field expansion in arbitrary inhomogeneous objects”. In: *IEEE transactions on antennas and propagation* 70.9 (2022), pp. 8227–8241.

[53] Ilias I Giannakopoulos, Mikhail S Litsarev, and Athanasios G Polimeridis. “Memory footprint reduction for the FFT-based volume integral equation method via tensor decompositions”. In: *IEEE Transactions on Antennas and Propagation* 67.12 (2019), pp. 7476–7486.

[54] Ilias I Giannakopoulos et al. “A hybrid volume-surface integral equation method for rapid electromagnetic simulations in MRI”. In: *IEEE Transactions on Biomedical Engineering* 70.1 (2022), pp. 105–114.

[55] Ilias I Giannakopoulos et al. “A tensor train compression scheme for remote volume-surface integral equation interactions”. In: *2021 International Applied Computational Electromagnetics Society Symposium (ACES)*. IEEE. 2021, pp. 1–4.

[56] Ilias I Giannakopoulos et al. “Compression of volume-surface integral equation matrices via Tucker decomposition for magnetic resonance applications”. In: *IEEE transactions on antennas and propagation* 70.1 (2021), pp. 459–471.

[57] Ilias I Giannakopoulos et al. “Computational methods for the estimation of ideal current patterns in realistic human models”. en. In: *Magn. Reson. Med.* 91.2 (Feb. 2024), pp. 760–772.

[58] Ilias I Giannakopoulos et al. “Global Maxwell Tomography Using the Volume-Surface Integral Equation for Improved Estimation of Electrical Properties”. In: *IEEE Transactions on Biomedical Engineering* (2025).

[59] Arthur van Goethem et al. “Topologically safe curved schematization”. In: *The Cartographic Journal* 50.3 (2013), pp. 276–285.

[60] Marie-Christine Gosselin et al. "Development of a new generation of high-resolution anatomical models for medical device evaluation: the Virtual Population 3.0". In: *Physics in Medicine & Biology* 59.18 (2014), p. 5287.

[61] Bernhard Gruber et al. "RF coils: A practical guide for nonphysicists". In: *Journal of Magnetic Resonance Imaging* 48.3 (2018), pp. 590–604. DOI: https://doi.org/10.1002/jmri.26187.

[62] Bastien Guérin et al. "Computation of ultimate SAR amplification factors for radiofrequency hyperthermia in non-uniform body models: impact of frequency and tumour location". In: *International Journal of Hyperthermia* 34.1 (2018), pp. 87–100.

[63] Jia-Peng Guo and Xiao-Ming Fu. "Exact and Efficient Intersection Resolution for Mesh Arrangements". In: *ACM Trans. Graph.* 43.6 (2024), 165:1–165:14.

[64] Georgy D Guryev et al. "Fast field analysis for complex coils and metal implants in MARIE 2.0". In: *Proc. ISMRM.* 2019, p. 1035.

[65] Georgy D Guryev et al. "MARIE 2.0: a perturbation matrix based patient-specific MRI field simulator". In: *IEEE Transactions on Biomedical Engineering* 70.5 (2022), pp. 1575–1586.

[66] J Rock Hadley, Dennis Parker, and Cynthia M Furse. "RF coil design for MRI using a genetic algorithm". In: *The Applied Computational Electromagnetics Society Journal (ACES)* (2007), pp. 277–286.

[67] J Rock Hadley, Dennis Parker, and Cynthia M Furse. "RF coil design for MRI using a genetic algorithm". In: *The Applied Computational Electromagnetics Society Journal (ACES)* (2007), pp. 277–286.

[68] Bjoern Haefner et al. "Fight ill-posedness with ill-posedness: Single-shot variational depth super-resolution from shading". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2018, pp. 164–174.

[69] Zhonghua He et al. “A Time-Harmonic Inverse Design of Uniplanar RF Coil for Unilateral NMR Sensors”. In: *IEEE Sensors Journal* 17.9 (2017), pp. 2696–2702. doi: 10.1109/JSEN.2017.2680455.

[70] Jan S Hesthaven and Stefano Ubbiali. “Non-intrusive reduced order modeling of nonlinear problems using neural networks”. In: *Journal of Computational Physics* 363 (2018), pp. 55–78.

[71] Michael Hofer and Helmut Pottmann. “Energy-minimizing splines in manifolds”. In: *ACM SIGGRAPH 2004 Papers*. 2004, pp. 284–293.

[72] Mark Hollins and Sliman J Bensmaıa. “The coding of roughness.” In: *Canadian Journal of Experimental Psychology/Revue canadienne de psychologie experimentale* 61.3 (2007), p. 184.

[73] Mark Hollins and S Ryan Risner. “Evidence for the duplex theory of tactile texture perception”. In: *Perception & psychophysics* 62.4 (2000), pp. 695–705.

[74] Hugues Hoppe. “Progressive meshes”. In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. 1996, pp. 99–108.

[75] Yixin Hu et al. “Fast tetrahedral meshing in the wild”. In: (2020).

[76] Yixin Hu et al. “Tetrahedral meshing in the wild.” In: *ACM Trans. Graph.* 37.4 (2018), p. 60.

[77] Yixin Hu et al. “TriWild: robust triangulation with curve constraints”. In: *ACM Transactions on Graphics (TOG)* 38.4 (2019), pp. 1–15.

[78] Jingwei Huang, Yichao Zhou, and Leonidas J. Guibas. “ManifoldPlus: A Robust and Scalable Watertight Manifold Surface Generation Method for Triangle Soups”. In: *CoRR* abs/2005.11621 (2020).

[79] Zizhou Huang, Daniele Panozzo, and Denis Zorin. “Optimized shock-protecting microstructures”. In: *arXiv preprint arXiv:2310.08609* (2023).

[80] Phillip Isola et al. “Image-to-image translation with conditional adversarial networks”. In: *arXiv preprint* (2017).

[81] John David Jackson. *Classical Electrodynamics*. 3rd. College Park, MD: American Association of Physics Teachers, 1999.

[82] Alec Jacobson et al. *gptoolbox: Geometry Processing Toolbox*. http://github.com/alecjacobson/gptoolbox. 2021.

[83] Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. “Robust inside-outside segmentation using generalized winding numbers”. In: *ACM Transactions on Graphics (TOG)* 32.4 (2013), pp. 1–12.

[84] Dan Jiao and Jian-Ming Jin. “Fast frequency-sweep analysis of RF coils for MRI”. In: *IEEE transactions on biomedical engineering* 46.11 (1999), pp. 1387–1390.

[85] Yao Jin et al. “A shell space constrained approach for curve design on surface meshes”. In: *Computer-Aided Design* 113 (2019), pp. 24–34.

[86] Tao Ju. “Fixing Geometric Errors on Polygonal Models: A Survey”. In: *J. Comput. Sci. Technol.* 24.1 (2009), pp. 19–29.

[87] Tao Ju. “Robust repair of polygonal models”. In: *ACM Trans. Graph.* 23.3 (2004), pp. 888–895.

[88] Tao Ju et al. “Dual contouring of hermite data”. In: *ACM Trans. Graph.* 21.3 (July 2002), pp. 339–346. ISSN: 0730-0301. DOI: 10.1145/566654.566586.

[89] David LB Jupp. “Approximation to data by splines with free knots”. In: *SIAM Journal on Numerical Analysis* 15.2 (1978), pp. 328–343.

[90] Hongmei Kang et al. “Knot calculation for spline fitting via sparse optimization”. In: *Computer-Aided Design* 58 (2015), pp. 179–188.

[91] Brian Karis. *The Journey to Nanite*. 2022.

[92] Zachi Karni and Craig Gotsman. “Compression of soft-body animation sequences”. In: *Computers & Graphics* 28.1 (2004), pp. 25–34.

[93] Boris Keil et al. “A 64-channel 3T array coil for accelerated brain MRI”. en. In: *Magn. Reson. Med.* 70.1 (July 2013), pp. 248–258.

[94] C Khatri and Calyampudi Radhakrishna Rao. “Effects of estimated noise covariance matrix in optimal signal detection”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 35.5 (1987), pp. 671–679.

[95] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[96] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015.

[97] Alexander Kolesnikov. “Approximation of digitized curves with cubic Bézier splines”. In: *2010 IEEE International Conference on Image Processing*. IEEE. 2010, pp. 4285–4288.

[98] P Konzbul and K Sveda. “Shim coils for NMR and MRI solenoid magnets”. In: *Measurement Science and Technology* 6.8 (Aug. 1995), p. 1116. DOI: 10.1088/0957-0233/6/8/005.

[99] Russell L. Lagore et al. “A 128-channel receive array with enhanced signal-to-noise ratio performance for 10.5T brain imaging”. In: *Magnetic Resonance in Medicine* 93.6 (2025), pp. 2680–2698. DOI: https://doi.org/10.1002/mrm.30476.

[100] Riccardo Lattanzi and Daniel K Sodickson. “Ideal current patterns yielding optimal signal-to-noise ratio and specific absorption rate in magnetic resonance imaging: computational methods and physical insights”. en. In: *Magn. Reson. Med.* 68.1 (July 2012), pp. 286–304.

[101] Riccardo Lattanzi et al. “Approaching ultimate intrinsic signal-to-noise ratio with loop and dipole antennas”. en. In: *Magn. Reson. Med.* 79.3 (Mar. 2018), pp. 1789–1803.

[102] Riccardo Lattanzi et al. “Electrodynamic constraints on homogeneity and radiofrequency power deposition in multiple coil excitations”. en. In: *Magn. Reson. Med.* 61.2 (Feb. 2009), pp. 315–334.

[103] Riccardo Lattanzi et al. “Performance evaluation of a 32-element head array with respect to the ultimate intrinsic SNR”. en. In: *NMR Biomed.* 23.2 (Feb. 2010), pp. 142–151.

[104] Riccardo Lattanzi et al. “Performance evaluation of a 32-element head array with respect to the ultimate intrinsic SNR”. In: *NMR in Biomedicine: An International Journal Devoted to the Development and Application of Magnetic Resonance In vivo* 23.2 (2010), pp. 142–151.

[105] Kai Lawonn et al. “Adaptive and robust curve smoothing on surface meshes”. In: *Computers & graphics* 40 (2014), pp. 22–35.

[106] B.G. Lawrence et al. “A time-harmonic inverse methodology for the design of RF coils in MRI”. In: *IEEE Transactions on Biomedical Engineering* 49.1 (Jan. 2002), pp. 64–71. ISSN: 1558-2531. DOI: 10.1109/10.972841.

[107] Ben G Lawrence et al. “A time-harmonic inverse methodology for the design of RF coils in MRI”. en. In: *IEEE Trans. Biomed. Eng.* 49.1 (Jan. 2002), pp. 64–71.

[108] Ben G Lawrence et al. “An inverse design of an open, head/neck RF coil for MRI”. en. In: *IEEE Trans. Biomed. Eng.* 49.9 (Sept. 2002), pp. 1024–1030.

[109] Hong-Hsi Lee, Daniel K Sodickson, and Riccardo Lattanzi. “An analytic expression for the ultimate intrinsic SNR in a uniform sphere”. en. In: *Magn. Reson. Med.* 80.5 (Nov. 2018), pp. 2256–2266.

[110] Robert Lee and Andreas C Cangellaris. “A study of discretization error in the finite element approximation of wave solutions”. In: *IEEE transactions on antennas and propagation* 40.5 (1992), pp. 542–549.

[111] Jerome Edward Lengyel. “Compression of time-dependent geometry”. In: *Proceedings of the 1999 symposium on Interactive 3D graphics*. 1999, pp. 89–95.

[112] Raphael Linus Levien. *From spiral to spline: Optimal techniques in interactive curve design*. University of California, Berkeley, 2009.

[113] Minchen Li, Danny M Kaufman, and Chenfanfu Jiang. “Codimensional incremental potential contact”. In: *ACM Transactions on Graphics* 40.4 (2021), pp. 1–24.

[114] Minchen Li et al. “Incremental potential contact: intersection-and inversion-free, large-deformation dynamics.” In: *ACM Trans. Graph.* 39.4 (2020), p. 49.

[115] Tzu-Mao Li et al. “Differentiable Monte Carlo Ray Tracing through Edge Sampling”. In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 37.6 (2018), 222:1–222:11.

[116] Weibai Li et al. “Topology optimization of photonic and phononic crystals and metamaterials: a review”. In: *Advanced Theory and Simulations* 2.7 (2019), p. 1900017.

[117] Yifei Li et al. “Fluidic topology optimization with an anisotropic mixture model”. In: *ACM Transactions on Graphics (TOG)* 41.6 (2022), pp. 1–14.

[118] Peter Liepa. “Filling Holes in Meshes”. In: *Eurographics Symposium on Geometry Processing*. Ed. by Leif Kobbelt, Peter Schroeder, and Hugues Hoppe. The Eurographics Association, 2003. ISBN: 3-905673-06-1. DOI: /10.2312/SGP/SGP03/200-206.

[119] Siyou Lin, Zuoqiang Shi, and Yebin Liu. “Fast and Globally Consistent Normal Orientation based on the Winding Number Normal Consistency”. In: *ACM Trans. Graph.* 43.6 (Nov. 2024). ISSN: 0730-0301. DOI: 10.1145/3687895.

[120] Dong C Liu and Jorge Nocedal. “On the limited memory BFGS method for large scale optimization”. In: *Mathematical programming* 45.1 (1989), pp. 503–528.

[121] Songrun Liu, Alec Jacobson, and Yotam Gingold. “Skinning Cubic Bézier Splines and Catmull-Clark Subdivision Surfaces”. In: *ACM Transactions on Graphics (TOG)* 33.6 (2014).

[122] PD Loach and AJ Wathen. “On the best least squares approximation of continuous functions using linear splines with free knots”. In: *IMA journal of numerical analysis* 11.3 (1991), pp. 393–409.

[123] William E. Lorensen and Harvey E. Cline. “Marching cubes: A high resolution 3D surface construction algorithm”. In: *SIGGRAPH Comput. Graph.* 21.4 (Aug. 1987), pp. 163–169. ISSN: 0097-8930. DOI: 10.1145/37402.37422.

[124] Francesco Lucchini et al. “Topology optimization for electromagnetics: A survey”. In: *IEEE Access* 10 (2022), pp. 98593–98611.

[125] Tom Lyche and Knut Mørken. “Knot removal for parametric B-spline curves and surfaces”. In: *Computer Aided Geometric Design* 4.3 (1987), pp. 217–230.

[126] J. N. Lyness and C. B. Moler. “Numerical Differentiation of Analytic Functions”. In: *SIAM Journal on Numerical Analysis* 4.2 (1967), pp. 202–210.

[127] Louise R Manfredi et al. “Natural scenes in tactile texture”. In: *Journal of neurophysiology* 111.9 (2014), pp. 1792–1802.

[128] Rafat Mantiuk et al. “HDR-VDP-2: a calibrated visual metric for visibility and quality predictions in all luminance conditions”. In: *ACM Transactions on graphics (TOG)*. Vol. 30. 4. ACM. 2011, p. 40.

[129] Rodrigo Martín et al. “Using patch-based image synthesis to measure perceptual texture similarity”. In: *Computers & Graphics* 81 (2019), pp. 104–116.

[130] Asif Masood and Muhammad Sarfraz. “Capturing outlines of 2D objects with Bézier cubic approximation”. In: *Image and Vision Computing* 27.6 (2009), pp. 704–712.

[131] Gal Metzer et al. “Orienting point clouds with dipole propagation”. In: *ACM Trans. Graph.* 40.4 (2021), 165:1–165:14. DOI: 10.1145/3450626.3459835.

[132] MITMediaLab. *Vision texture, VisTexdatabase*. 1995. URL: http://vismod.media.mit.edu/vismod/imagery/VisionTexture/ (visited on 01/04/2019).

[133] B. Mohammadi and O. Pironneau. *Applied Shape Optimization for Fluids*. 2nd. Oxford: Oxford University Press, 2009. DOI: 10.1093/acprof:oso/9780199546909.001.0001.

[134] Farzin Mokhtarian, Yoke Khim Ung, and Zhitao Wang. "Automatic fitting of digitised contours at multiple scales through the curvature scale space technique". In: *Computers & Graphics* 29.6 (2005), pp. 961–971.

[135] L Tugan Muftuler, Gang Chen, and Orhan Nalcioglu. "An inverse method to design RF coil arrays optimized for SENSE imaging". en. In: *Phys. Med. Biol.* 51.24 (Dec. 2006), pp. 6457–6469.

[136] Karthik Nadig, William M. Potter, and Walter D. Potter. "Homogeneous RF Coil Design Using a GA". In: *Advanced Research in Applied Artificial Intelligence*. Vol. 7345. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 197–205. ISBN: 978-3-642-31086-7. DOI: 10.1007/978-3-642-31087-4_21.

[137] Ogan Ocali and Ergin Atalar. "Ultimate intrinsic signal-to-noise ratio in MRI". In: *Magnetic resonance in medicine* 39.3 (1998), pp. 462–473.

[138] Ogan Ocali and Ergin Atalar. "Ultimate intrinsic signal-to-noise ratio in MRI". en. In: *Magn. Reson. Med.* 39.3 (Mar. 1998), pp. 462–473.

[139] Michael A Ohliger, Aaron K Grant, and Daniel K Sodickson. "Ultimate intrinsic signal-to-noise ratio for parallel MRI: electromagnetic field considerations". en. In: *Magn. Reson. Med.* 50.5 (Nov. 2003), pp. 1018–1030.

[140] Michael A Ohliger, Aaron K Grant, and Daniel K Sodickson. "Ultimate intrinsic signal-to-noise ratio for parallel MRI: electromagnetic field considerations". In: *Magnetic Resonance in Medicine* 50.5 (2003), pp. 1018–1030.

[141] Julian Panetta et al. “Elastic textures for additive fabrication”. In: *ACM Transactions on Graphics (TOG)* 34.4 (2015), p. 135.

[142] Adam Paszke et al. “Automatic differentiation in pytorch”. In: (2017).

[143] Alvin Penner. “Fitting a cubic Bezier to a parametric function”. In: *The College Mathematics Journal* 50.3 (2019), pp. 185–196.

[144] Andreas Pfrommer and Anke Henning. “The ultimate intrinsic signal-to-noise ratio of loop-and dipole-like current patterns in a realistic human head model”. In: *Magnetic resonance in medicine* 80.5 (2018), pp. 2122–2138.

[145] Michal Piovarči et al. “An Interaction-Aware, Perceptual Model For Non-Linear Elastic Objects”. In: *ACM Transactions on Graphics (Proc. SIGGRAPH)* 35.4 (2016).

[146] Athanasios G Polimeridis et al. “Stable FFT-JVIE solvers for fast analysis of highly inhomogeneous dielectric objects”. In: *Journal of Computational Physics* 269 (2014), pp. 280–296.

[147] Michael Poole and Richard Bowtell. “Novel gradient coils designed using a boundary element method”. In: *Concepts in Magnetic Resonance Part B: Magnetic Resonance Engineering* 31B.3 (2007), pp. 162–175. doi: https://doi.org/10.1002/cmr.b.20091.

[148] Michael Poole et al. “Minimax current density coil design”. In: *Journal of Physics D: Applied Physics* 43.9 (Feb. 2010), p. 095001. doi: 10.1088/0022-3727/43/9/095001.

[149] Cédric Portaneri et al. “Alpha wrapping with an offset”. In: *ACM Trans. Graph.* 41.4 (2022), 127:1–127:22.

[150] Javier Portilla and Eero P Simoncelli. “A parametric texture model based on joint statistics of complex wavelet coefficients”. In: *International journal of computer vision* 40.1 (2000), pp. 49–70.

[151] Michael Rabinovich et al. “Scalable locally injective mappings”. In: *ACM Transactions on Graphics (TOG)* 36.4 (2017), p. 1.

[152] M Raissi, P Perdikaris, and GE Karniadakis. “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”. In: *Journal of Computational Physics* 378 (2019), pp. 686–707.

[153] Sadasiva Rao, Donald Wilton, and Allen Glisson. “Electromagnetic scattering by surfaces of arbitrary shape”. In: *IEEE Transactions on antennas and propagation* 30.3 (1982), pp. 409–418.

[154] MT Homer Reid and Steven G Johnson. “Efficient computation of power, force, and torque in BEM scattering calculations”. In: *IEEE Transactions on Antennas and Propagation* 63.8 (2015), pp. 3588–3598.

[155] A Rogovich et al. “RF coils for MRI applications-a design procedure”. In: *2005 IEEE Antennas and Propagation Society International Symposium*. Vol. 1. IEEE. 2005, pp. 856–859.

[156] Françoise Roméo and DI Hoult. “Magnet field profiling: analysis and correcting coil design”. In: *Magnetic Resonance in Medicine* 1.1 (1984), pp. 44–65.

[157] Günter Rote. “Computing the Fréchet distance between piecewise smooth curves”. In: *Comput. Geom.* 37.3 (2007), pp. 162–174. DOI: 10.1016/j.comgeo.2005.01.004.

[158] Olivier Rouiller et al. “3D-printing spatially varying BRDFs”. In: *IEEE computer graphics and applications* 33.6 (2013), pp. 48–57.

[159] Muhammad Sarfraz and Asif Masood. “Capturing outlines of planar images using Bézier cubics”. In: *Computers & Graphics* 31.5 (2007), pp. 719–729.

[160] Muhammad Sarfraz and MFA Razzak. “An algorithm for automatic capturing of the font outlines”. In: *Computers & Graphics* 26.5 (2002), pp. 795–804.

[161] Mirko Sattler, Ralf Sarlette, and Reinhard Klein. “Simple and efficient compression of animation sequences”. In: *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 2005, pp. 209–217.

[162] Melanie Schmitt et al. “A 128-channel receive-only cardiac coil for highly accelerated cardiac MRI at 3 Tesla”. In: *Magnetic Resonance in Medicine* 59.6 (May 2008), pp. 1431–1439. ISSN: 1522-2594. DOI: 10.1002/mrm.21598.

[163] Philip J Schneider. “An algorithm for automatically fitting digitized curves”. In: *Graphics gems* 1 (1990), pp. 612–626.

[164] Wilfried Schnell et al. “Ultimate signal-to-noise-ratio of surface and body antennas for magnetic resonance imaging”. In: *IEEE Transactions on Antennas and Propagation* 48.3 (2000), pp. 418–428.

[165] Christian Schüller, Daniele Panozzo, and Olga Sorkine-Hornung. “Appearance-mimicking surfaces”. In: *ACM Transactions on Graphics (TOG)* 33.6 (2014), p. 216.

[166] Daniel Scrivener, Ellis Coldren, and Edward Chien. “Winding Number Features for Vector Sketch Colorization”. In: *Comput. Graph. Forum* 43.5 (2024), pp. i–x. DOI: 10.1111/CGF.15141.

[167] Jose E. Cruz Serralles et al. “Rational MRI Coil Design: An Optimization Framework for the Design of Radiofrequency Coils for Magnetic Resonance Imaging”. In: *bioRxiv* (2025). DOI: 10.1101/2025.08.04.668545.

[168] Jose EC Serralles et al. “Rapid parametric optimization of transmit coil arrays: a proof-of-concept”. In: *The International Society for Magnetic Resonance in Medicine*. 2020, p. 4262.

[169] José EC Serrallés et al. “Noninvasive estimation of electrical properties from magnetic resonance measurements via global Maxwell tomography and match regularization”. In: *IEEE Transactions on Biomedical Engineering* 67.1 (2019), pp. 3–15.

[170] Lejun Shao and Hao Zhou. “Curve fitting with Bezier cubics”. In: *Graphical models and image processing* 58.3 (1996), pp. 223–232.

[171] Nicholas Sharp et al. *Polyscope*. www.polyscope.run. 2019.

[172] Jonathan Richard Shewchuk. “Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator”. In: *Workshop on applied computational geometry*. Springer. 1996, pp. 203–222.

[173] Liang Shi et al. “Deep multispectral painting reproduction via multi-layer, custom-ink printing”. In: *SIGGRAPH Asia 2018 Technical Papers*. ACM. 2018, p. 271.

[174] Hang Si. “TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator”. In: *ACM Trans. Math. Softw.* 41.2 (2015), 11:1–11:36.

[175] Ole Sigmund and Salvatore Torquato. “Design of materials with extreme thermal expansion using a three-phase topology optimization method”. In: *Journal of the Mechanics and Physics of Solids* 45.6 (1997), pp. 1037–1067.

[176] Jason Smith and Scott Schaefer. “Bijective parameterization with free boundaries”. In: *ACM Transactions on Graphics (TOG)* 34.4 (2015), pp. 1–9.

[177] Allen Taflove and Korada R Umashankar. “Review of FD-TD numerical modeling of electromagnetic wave scattering and radar cross section”. In: *Proceedings of the IEEE* 77.5 (1989), pp. 682–699.

[178] Wouter M Bergmann Tiest. “Tactual perception of material properties”. In: *Vision research* 50.24 (2010), pp. 2775–2782.

[179] Wouter M Bergmann Tiest and Astrid ML Kappers. “Tactile perception of thermal diffusivity”. In: *Attention, perception, & psychophysics* 71.3 (2009), pp. 481–489.

[180] Cesar Torres et al. “HapticPrint: Designing Feel Aesthetics for Digital Fabrication”. In: *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. ACM. 2015, pp. 583–591.

[181] Davi Colli Tozoni et al. “Cut-Cell Microstructures for Two-scale Structural Optimization”. In: *Computer Graphics Forum*. Vol. 43. 5. Wiley Online Library. 2024, e15139.

[182] R Turner. “A target field approach to optimal coil design”. In: *Journal of Physics D: Applied Physics* 19.8 (Aug. 1986), p. L147. DOI: 10.1088/0022-3727/19/8/001.

[183] Chelsea Tymms, Esther P Gardner, and Denis Zorin. “A Quantitative Perceptual Model for Tactile Roughness”. In: *ACM Transactions on Graphics (TOG)* 37.5 (2018), p. 168.

[184] Chelsea Tymms, Siqi Wang, and Denis Zorin. “Appearance-preserving tactile optimization”. In: *ACM Transactions on Graphics (TOG)* 39.6 (2020), pp. 1–16.

[185] Chelsea Tymms, Denis Zorin, and Esther P Gardner. “Tactile perception of the roughness of 3D-printed textures”. In: *Journal of Neurophysiology* 119.3 (2017), pp. 862–876.

[186] Dmitry Ulyanov et al. “Texture Networks: Feed-forward Synthesis of Textures and Stylized Images.” In: *ICML*. 2016, pp. 1349–1357.

[187] Manushka V Vaidya, Daniel K Sodickson, and Riccardo Lattanzi. “Approaching ultimate intrinsic SNR in a uniform spherical sample with finite arrays of loop coils”. en. In: *Concepts Magn. Reson. Part B Magn. Reson. Eng.* 44.3 (Aug. 2014), pp. 53–65.

[188] Libor Váša and Václav Skala. “Cobra: Compression of the basis for pca represented animations”. In: *Computer Graphics Forum*. Vol. 28. 6. Wiley Online Library. 2009, pp. 1529–1540.

[189] J Fernández Villena et al. “MARIE: A MATLAB-based open dource MRI electromagnetic analysis software”. In: *Biological, Medical Devices, and Systems* (2016), p. 3.

[190] Jorge Fernandez Villena et al. “Fast Electromagnetic Analysis of MRI Transmit RF Coils Based on Accelerated Integral Equation Methods”. In: *IEEE Transactions on Biomedical Engineering* 63.11 (Nov. 2016), pp. 2250–2261. ISSN: 1558-2531. DOI: 10.1109/tbme.2016.2521166.

[191] Jorge Fernández Villena et al. “Fast electromagnetic analysis of MRI transmit RF coils based on accelerated integral equation methods”. In: *IEEE Transactions on Biomedical Engineering* 63.11 (2016), pp. 2250–2261.

[192] Matt Waks et al. “RF coil design strategies for improving SNR at the ultrahigh magnetic field of 10.5T”. In: *Magnetic Resonance in Medicine* 93.2 (2025), pp. 873–888. DOI: https://doi.org/10.1002/mrm.30315.

[193] Shawn W. Walker. “Shape optimization of self-avoiding curves”. In: *J. Comput. Phys.* 311 (2016), pp. 275–298. DOI: 10.1016/j.jcp.2016.02.011.

[194] Thomas SA Wallis et al. “A parametric texture model based on deep convolutional features closely matches texture appearance for humans”. In: *Journal of vision* 17.12 (2017), pp. 5–5.

[195] Kai Wang et al. “A comprehensive survey on three-dimensional mesh watermarking”. In: *IEEE Transactions on Multimedia* 10.8 (2008), pp. 1513–1527.

[196] Siqi Wang et al. “Bézier Spline Simplification Using Locally Integrated Error Metrics”. In: *SIGGRAPH Asia 2023 Conference Papers*. 2023, pp. 1–11.

[197] Siqi Wang et al. “Solid-Shell Labeling for Discrete Surfaces”. In: *SIGGRAPH Asia 2025 Conference Papers*. conditionally accepted. 2025.

[198] Xin Wang et al. “3D printing of polymer matrix composites: A review and prospective”. In: *Composites Part B: Engineering* 110 (2017), pp. 442–458.

[199] Yiqiang Wang et al. “A multi-material level set-based topology and shape optimization method”. In: *Computer Methods in Applied Mechanics and Engineering* 283 (2015), pp. 1570–1586.

[200] Zhou Wang, Eero P Simoncelli, and Alan C Bovik. “Multiscale structural similarity for image quality assessment”. In: *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*. Vol. 2. Ieee. 2003, pp. 1398–1402.

[201] Zhou Wang et al. “Image quality assessment: from error visibility to structural similarity”. In: *IEEE transactions on image processing* 13.4 (2004), pp. 600–612.

[202] Li-Yi Wei and Marc Levoy. “Fast texture synthesis using tree-structured vector quantization”. In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. 2000, pp. 479–488.

[203] Holger Wendland. *Scattered Data Approximation*. Vol. 17. Cambridge Monographs on Applied and Computational Mathematics. Cambridge: Cambridge University Press, 2004.

[204] Florian Wiesinger, Peter Boesiger, and Klaas P Pruessmann. “Electrodynamics and ultimate SNR in parallel MR imaging”. en. In: *Magn. Reson. Med.* 52.2 (Aug. 2004), pp. 376–390.

[205] Graham C. Wiggins et al. “96-Channel receive-only head coil for 3 Tesla: Design optimization and evaluation”. In: *Magnetic Resonance in Medicine* 62.3 (2009), pp. 754–762. DOI: https://doi.org/10.1002/mrm.22028.

[206] Eric C Wong, A Jesmanowicz, and James S Hyde. “Coil optimization for MRI by conjugate gradient descent”. In: *Magnetic resonance in medicine* 21.1 (1991), pp. 39–48.

[207] Xuegang Xin et al. “Inverse design of an organ-oriented RF coil for open, vertical-field, MR-guided, focused ultrasound surgery”. In: *Magnetic Resonance Imaging* 30.10 (2012), pp. 1519–1526. ISSN: 0730-725X. DOI: https://doi.org/10.1016/j.mri.2012.05.009.

[208] Rui Xu et al. “Globally Consistent Normal Orientation for Point Clouds by Regularizing the Winding-Number Field”. In: *ACM Trans. Graph.* 42.4 (July 2023). ISSN: 0730-0301. DOI: 10.1145/3592129.

[209] YahooJAPAN. *Frog*. Modified. 2013. URL: https://www.thingiverse.com/thing:182144.

[210] Ying Yang et al. “A 3D steganalytic algorithm and steganalysis-resistant watermarking”. In: *IEEE transactions on visualization and computer graphics* 23.2 (2017), pp. 1002–1013.

[211] D Yau and S Crozier. “A genetic algorithm/method of moments approach to the optimization of an RF coil for MRI applications—theoretical considerations”. In: *Progress In Electromagnetics Research* 39 (2003), pp. 177–192.

[212] D. Yau and S. Crozier. “A Genetic Algorithm/Method of Moments Approach to the Optimization of an RF Coil for MRI Applications — Theoretical Considerations — Abstract”. In: *Journal of Electromagnetic Waves and Applications* 17.5 (Jan. 2003), pp. 753–754. ISSN: 1569-3937. DOI: 10.1163/156939303322226437.

[213] Pasi Yla-Oijala et al. “Surface and volume integral equation methods for time-harmonic solutions of Maxwell’s equations”. In: *Progress in electromagnetics Research* 149 (2014), pp. 15–44.

[214] Christopher Yu, Henrik Schumacher, and Keenan Crane. “Repulsive Curves”. In: *ACM Transactions on Graphics (TOG)* 40 (2020), pp. 1–21. DOI: 10.1145/3439429.

[215] Ning Yu et al. “Texture Mixer: A Network for Controllable Synthesis and Interpolation of Texture”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[216] Bei Zhang et al. “Performance of receive head arrays versus ultimate intrinsic SNR at 7 T and 10.5 T”. In: *Magnetic resonance in medicine* 92.3 (2024), pp. 1219–1231.

[217] Jiayi Eris Zhang et al. “Complementary dynamics”. In: *arXiv preprint arXiv:2009.02462* (2020).

[218] Lin Zhang et al. “FSIM: a feature similarity index for image quality assessment”. In: *IEEE transactions on Image Processing* 20.8 (2011), pp. 2378–2386.

[219] Richard Zhang et al. “The Unreasonable Effectiveness of Deep Features as a Perceptual Metric”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* June 2018.

[220] Xiaoting Zhang et al. “Thermal-comfort design of personalized casts”. In: *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology.* ACM. 2017, pp. 243–254.

[221] Zhongtian Zheng et al. “Visual-Preserving Mesh Repair”. In: *IEEE Transactions on Visualization & Computer Graphics* 30.09 (2024), pp. 6586–6597.

[222] Qingnan Zhou et al. “Mesh arrangements for solid geometry”. In: *ACM Trans. Graph.* 35.4 (2016), 39:1–39:15. DOI: 10.1145/2897824.2925901.

[223] Yang Zhou et al. “Non-stationary Texture Synthesis by Adversarial Expansion”. In: *ACM Transactions on Graphics (Proc. SIGGRAPH)* 37.4 (2018), 49:1–49:13.

[224] Daniel Zint et al. “Topological Offsets”. In: *arXiv preprint arXiv:2407.07725* (2024).

[225] Peter Zwamborn and Peter M Van Den Berg. “The three dimensional weak form of the conjugate gradient FFT method for solving scattering problems”. In: *IEEE Transactions on Microwave Theory and Techniques* 40.9 (1992), pp. 1757–1766.